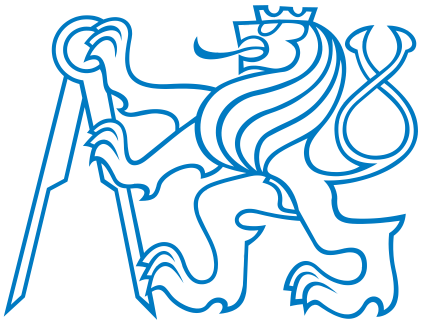
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science



Efficient Modelling and Reformulation of Planning Tasks

Habilitation Thesis

Lukáš Chrpa

Praha, June 2020

Contents

Abstract

This thesis summarises author's research contributions concerning applying Knowledge Engineering techniques in Domain-independent Automated Planning. Domain-independent planning decouples planning task description, provided in some standardised language (e.g. PDDL), and planning engines, generic solvers of planning tasks. Thanks to the International Planning Competitions that are organised since 1998, numerous advanced planning engines have been developed. Efficiency of planning task models (or lack of) has considerable impact on performance of these planning engines. Hence research efforts concerning efficient planning task modelling complement research efforts concerning developing advanced planning techniques.

This thesis discusses techniques for automated planning task reformulation, namely entanglements and macro-operators. Then, the thesis discusses a technique for manual encoding of Domain Control Knowledge into planning task models. Finally, the thesis discusses the planning task modelling process from more general perspective using a case study of task planning for Autonomous Underwater Vehicles.

Acknowledgements

In the first place, I thank my research mentors Roman Barták, my former PhD supervisor at Charles University, and Lee McCluskey, my former line manager at University of Huddersfield. A special thanks goes to Mauro Vallati, my former colleague at University of Huddersfield, with whom I have published more than 30 papers. My thanks also go to other people who inspired me during my research career.

My research was supported by Czech Science Foundation (currently by projects no. 17-17125Y and 18-07252S), UK Engineering and Physical Sciences Research Council, and most recently by an OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics".

Finally, I would like to thank my family and friends for their support during my research career.

Copyright

Chapter 1

Introduction

Automated Planning deals with the problem of finding a (partially ordered) action sequence, a plan, transforming the environment from a given initial state to some required goal state [26]. In a nutshell, Automated Planning is a tool for deliberative reasoning which intelligent entities can use to find strategies (plans) for achieving longer-term goals. There are many successful real-world applications ranging from space and planet observations [1], Urban Traffic Control [34], narrative generation [28] to Machine Tool calibration [39].

Domain-independent planning decouples planning task description, specified in a language such as PDDL [36], and planning engines that generate a plan from the task description (if a plan exists). Planning task consists of a *planning domain model* that gives general description of the environment and actions of a given domain and a *planning problem* that specifies concrete objects, an initial state and a goal. Since domain models capture generalised aspects of domains, it is typical that one domain model can be used for a class of planning tasks.

The advantage of domain-independent planning is its modularity, i.e., the domain model and the planning engine are independent components, that can be easier plug into larger systems. To communicate with the planning component, the system has to be able to generate planning problems in a given language (for example, in PDDL), pass them alongside the domain model to the planning engine and interpret (and execute) generated plans.

Whereas the planning community, driven by International Planning Competitions (IPC)[1], focuses on developing advanced planning techniques for solving planning tasks, the engineering process of developing models of planning tasks, or domain models, is also important, although as shown in the

[1] http://ipc.icaps-conference.org

last International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS) it has not yet received that much attention from the planning community [10]. Efficiency of a planning task model, or a domain model is however a crucial factor considerably affecting performance of planning engines and hence the usability of the model in a real application.

This thesis consists of a collection of papers:

1. introducing techniques for automated planning task reformulation, namely entanglements and macro-operators,

2. introducing a technique for manual encoding of Domain Control Knowledge, specified in a form of finite automata,

3. discussing the planning task modelling process from more general perspective and summarises an experience of the author with the process of knowledge elicitation, domain modelling, and plan execution for the application of task planning for Autonomous Underwater Vehicles (AUVs).

Contribution of the papers in the collection (in Appendix) will be discussed in Chapters 3 and 4.

1.1 Impact

The impact of my research, presented in this thesis, is twofold. Firstly, an improvement of domain-independent planning by (automated) remodelling of planning tasks and, secondly, studying and applying Knowledge Engineering techniques for designing and developing planning domain models for real-world applications.

In analogy to computer programming, how planning tasks are modelled is one of the crucial factors determining efficiency of domain-independent planning engines. Simply said, efficient representation of planning tasks allows planning engines to solve them more quickly (or solve them at all). One possibility is to reformulate domain models such that additional knowledge about the models, which can provide further guidance to planning engines, is encoded within them. Chapter 3 summarises my research achievements concerning planning task reformulation.

To effectively leverage domain-independent planning in real-world applications, models of the environment and actions or domain models, in other words, have to be developed. Knowledge Engineering process concerns domain knowledge elicitation, its formal conceptualisation and domain model

development. Chapter 4 is devoted to describing such a Knowledge Engineering process for the application of task planning for AUVs.

Chapter 2

Preliminaries

The classical (STRIPS) representation considers static and fully observable environment, and deterministic and instantaneous action effects [26]. The environment is described by first-order logic *predicates*. *States* are defined as sets of *atoms*, which are grounded predicates. We say that $o = (name(o), pre(o), del(o), add(o))$ is a *planning operator*, where $name(o) = op\_name(x_1, \ldots, x_k)$ (op_name is an unique operator name and $x_1, \ldots x_k$ are variable symbols (arguments) appearing in the operator) and $pre(o), del(o)$ and $add(o)$ are sets of predicates with variables taken only from $x_1, \ldots x_k$ representing o's precondition, delete, and add effects respectively. *Actions* are grounded instances of planning operators. An action a is *applicable* in a state s if and only if $pre(a) \subseteq s$. Application of a in s (if possible) results in a state $(s \setminus del(a)) \cup add(a)$.

A *planning domain model* $D = (P, O)$ is specified by a set of predicates P and a set of planning operators O. A *planning task* $\Pi = (D, I, G)$ is specified via a domain model D, an initial state I and a set of goal atoms G. Given a planning problem, a *plan* is a sequence of actions such that their consecutive application starting in the initial state results in a state containing all the goal atoms.

Classical planning is the simplest form of Automated Planning and hence it might not be expressive enough to capture physics of the real-world domain. More expressive (yet deterministic) planning involves *durative actions*, i.e., actions whose application takes time, *numeric fluents*, or *continuous action effects* [25]. Non-deterministic planning involves, for instance, actions with *non-deterministic or probabilistic* effects, or *partially-observable* environment [27].

Chapter 3

Planning Task Reformulation

Planning Task reformulation can be leveraged for encoding additional domain knowledge directly into the planning domain model (and/or task description). Whereas a "raw" domain model might accurately capture the physics of the application domain and hence plans might be easier to interpret, it might be difficult to reason with such "raw" models for planning engines. Incorporating additional knowledge into the domain model by effectively reformulating it provides generic planning engines guidance that they can leverage while generating plans.

Additional domain knowledge can be automatically generated, for example, from training plans, or manually encoded by a domain expert. Although the former does not require a domain expert, generated knowledge might have limited impact on planning engines. In this chapter, we present two types of automatically generated domain knowledge – entanglements and macro-operators – and one type of manually specified type of domain knowledge – Transition-based Domain Control Knowledge.

3.1 Entanglements

Entanglements [18, 19] are relations between planning operators and predicates. Entanglements aim to capture the causal relationships characteristic for a given class of planning tasks which if leveraged can reduce the branching factor in the state space. There are two kinds of entanglements, outer and inner entanglements.

3.1.1 Outer Entanglements

Outer entanglements are relations between planning operators and initial or goal atoms which refer to situations where to solve a given planning problem we need only such instances of operators where instances of a certain predicate in an operator's precondition or add effects respectively are present in an initial state or a goal respectively. In BlocksWorld, it can be observed that unstacking blocks only has to occur from their initial positions. In this case an *entanglement by init* will capture that if an atom `(on a b)` is to be achieved for a corresponding instance of operator `unstack(?x ?y)` (`unstack(a b)`), then the atom is an initial atom. Similarly, it may be observed that stacking blocks only has to occur to their goal positions. Then, an *entanglement by goal* will capture that atom `(on b a)` achieved by a corresponding instance of operator `stack(?x ?y)` (`stack(b a)`) is a goal atom. Encoding outer entanglements into planning domains and problems is done by introducing static predicates which allow only instances following conditions of outer entanglements.

The aim of Outer Entanglements is to decrease the number of operator instances that planning engines have to reason with for solving a given planning task. In the BlocksWorld example, we can observe that the original model considers quadratic number of the `unstack` and `stack` operators, i.e., $n^2 - n$ each, where n is the number of blocks, while the "entangled" model consider linear number of these operators, i.e., at most $n - 1$ each. Reducing the number of operators' instances might propagate (although does not have to) into reduction of the number of reachable states. It can be observed that each block can be at most in 4 states: stacked on the initial block, held by the robotic hand, put on the table, and stacked on the goal block. Moreover, unless the initial and goal blocks are the same, once the block is unstacked from the initial position it cannot be returned to it and after the block is stacked on the goal position it can no longer be moved. In other words, the states can become "directional". Such a property might, however, introduce dead-ends, for example when the tower of block is being built from the middle.

For further details and experimental results, see [18].

3.1.2 Inner Entanglements

Inner entanglements are relations between pairs of planning operators and predicates which refer to situations where one operator is an exclusive "achiever" or "consumer" of a predicate to or from another operator. In the BlocksWorld it may be observed that operator `pickup(?x)` achieves predicate `holding(?x)`

exclusively for operator `stack(?x ?y)` (and not for operator `putdown(?x)`), i.e., `pickup(?x)` is *entangled by succeeding* `stack(?x ?y)` with `holding(?x)`. Similarly, it may be observed that predicate `holding(?x)` for operator `putdown(?x)` is exclusively achieved by operator `unstack(?x ?y)` (and not by operator `pickup(?x)`), i.e., `putdown(?x)` is *entangled by preceding* `unstack(?x ?y)` with `holding(?x)`. Encoding inner entanglements into planning domains and problems must ensure "achiever" and "consumer" exclusivity given by these inner entanglements. It is done by using specific predicates, "locks", which prevent executing certain instances of operators in some stage of the planning process. An instance of an operator having a "lock" in its precondition cannot be executed after executing an instance of another operator ("locker") having a "lock" in its negative effects until an instance of some other operator ("releaser") having a "lock" in its positive effects has been executed. For example, a situation where `pickup(?x)` is *entangled by succeeding* `stack(?x ?y)` with `holding(?x)` is modelled such that `pickup(?x)` is a "locker" for `putdown(?x)` and `stack(?x ?y)` is a "releaser" for `putdown(?x)`.

In contrast to Outer Entanglements, Inner Entanglements do not necessarily reduce the number of operators' instances as well as the size of the state space. On the other hand, Inner Entanglements eliminate some transition sequences in the state space (e.g. `pickup(?x)` cannot be followed by `putdown(?x)`). That might, in consequence, lead to state "directionality".

For further details and experimental results, see [19].

3.2 Macro-operators

Macro-operators (macros) encapsulate sequences of ordinary planning operators while being encoded in the same way as ordinary planning operators [32]. Technically, an instance of a macro is applicable in a state if and only if a corresponding sequence of operators' instances is applicable in that state and the result of the application of the macro's instance is the same as the result of application of the corresponding sequence of operators' instances. Macros can be added to the original domain models, which gives the technique the potential of being *planner independent*. Informally speaking, macros represent shortcuts in the state space and hence planning engines can generate plans in fewer steps.

Formally, a macro $o_{i,j}$ is constructed by assembling planning operators o_i and o_j (in that order) as follows. Let Φ and Ψ be mappings between variable symbols (we need to appropriately rename variable symbols of o_i and o_j to construct $o_{i,j}$).

- $pre(o_{i,j}) = pre(\Phi(o_i)) \cup (pre(\Psi(o_j)) \setminus add(\Phi(o_i)))$

- $del(o_{i,j}) = (del(\Phi(o_i)) \setminus add(\Psi(o_j))) \cup del(\Psi(o_j))$
- $add(o_{i,j}) = (add(\Phi(o_i)) \setminus del(\Psi(o_j))) \cup add(\Psi(o_j))$

For a macro to be *sound*, no instance of $\Phi(o_i)$ can delete an atom required by a corresponding instance of $\Psi(o_j)$, otherwise they cannot be applied consecutively. Whereas it is obvious that if a predicate deleted by $\Phi(o_i)$ (and not added back) is the same (both name and variable symbols) as a predicate in the precondition of $\Psi(o_j)$, i.e., $(del(\Phi(o_i)) \setminus add(\Phi(o_i)) \cap pre(\Psi(o_j)) \neq \emptyset)$, then the macro $o_{i,j}$ is unsound, another source of macro unsoundness refers to instantiating two or more macro's variables to a same object. For example, in the Blocks-World domain, a macro pickup-stack(?x ?y) that has (clear ?x)(ontable ?x)(clear ?y)(handempty) in its precondition can be instantiated into pickup-stack(A A) that is applicable if (clear A)(ontable A)(handempty) is true in a current state. However, actions pickup(A) and stack(A A) cannot be applied consecutively because pickup(A) deletes (clear A) which is required by stack(A A) and hence pickup-stack(?x ?y) is unsound. These situations can be easily identified by checking whether the same object substitution leads to the situation in which the first operator deletes a precondition for a second operator. To make the affected macro sound, *inequality constraints* [5] have to be added to macro's precondition (e.g. (not (= ?x ?y)) is added into pickup-stack(?x ?y)'s precondition) [5].

Longer macros, i.e., those encapsulating longer sequences of original planning operators can be constructed iteratively by the above approach.

Besides a clear advantage, that is, providing shortcuts in the search space, macros often have many instances that in consequence increase branching factor as well as memory requirements. Therefore, there exist works that aim at opposite, either eliminate "redundant" actions whose effects can be achieved by sequences of other actions [29], or split operators into simpler ones [2]. That said, it is important that benefits of macros outweigh their drawbacks, which is also known as the *utility problem* [37].

Several recently developed planner-dependent macro generation systems aim at addressing a weakness of a specific planning engine. We can mention the SOL-EP version of Macro-FF [4], or Marvin [21] that generate macros that help to escape local heuristic minima of the well known FF planner [30]. Wizard [38] that is a planner-independent macro generating system that learns macros from training plans by leveraging genetic programming. The learning process incorporates a cross-validation phase in which generated macros are evaluated on a set of test problems being solved by a given planner. Hence, the aim is to learn macros that maximize performance of a given planning engine in a given domain. Such macros can be considered

as planner-specific (despite being generated by a planner-independent technique).

In a planner-independent settings, a frequent occurrence of sequences of actions in training plans usually plays a pivotal role in macro generation systems. Examples of such systems include the work of Chrpa [5] in which macros are learnt by analysing dependencies between actions in training plans. Dulac et al. [24] exploit n-gram algorithm to analyse training plans to learn macros. DBMP/S [31] applies Map Reduce for learning macros from a larger set of training plans. The CA-ED version of MacroFF [4] generates macros according to several pre-defined rules (e.g., the "locality rule") that apply on adjacent actions in training plans.

Chrpa's approach [5] leverages the property of "independent" actions that can be swapped in plans without affecting their validity. This idea was further elaborated by Siddiqui and Haslum in their approach deordering plans into partially ordered action "blocks" such that no ordering constraint between actions can be eliminated [43]. BloMa [14] is a macro learning approach that exploits block deordering [43] such that training plans are deordered into "blocks" that are further combined into "macroblocks" according to several rules describing relationships between blocks. Macroblocks that are frequent in training plans are assembled into macros. The advantage of BloMa is that it can learn useful longer macros, for example, a macro capturing shaking a cocktail and cleaning the shaker afterwards.

MUM [16] aims at learning "instance-wise" macros which, in other words, means that macros learnt by MUM should have a comparable number of instances to the original operators. To do so, MUM exploits Outer Entanglements that when applied on macros reduce the number of macros' instances. Outer Entanglements also serve as a heuristic for macro generation such that operators with the entanglement by init relation go first while operators with the entanglement by goal relation go last. In the ideal case, macros directly connect initial state of an object with its goal state (e.g. pick-move-drop). Frequency of macro occurrence in training plans is a secondary criterion.

Critical Section Macros [15], inspired by Critical Sections in parallel computing in which processes deal with shared resources, capture activities that use limited resources (e.g. a robotic hand). In planning, resource availability and use is represented by mutex predicates, for example, (handempty) and (holding ?x) respectively. Then *locker* and *releaser* operators that locks and releases the resource respectively can be identified. For example, unstack is a locker as it deletes (handempty) and achieves (holding ?x) while putdown is a releaser as it deletes (holding ?x) and achieves (handempty). Straightforwardly, a Critical Section Macro starts with a locker and ends up with a releaser. In between, the macro might contain *users* that have the resource use

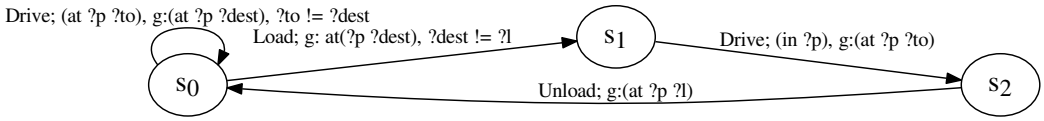


Figure 3.1: Transition-based DCK for our simple logistic domain.

predicate in their preconditions (e.g. a paint operator that paints the block held by the robotic hand), or *gluing operators* that have to be present in the macro for some reason (e.g. a move operator that moves the robotic hand between tables). Critical Section Macros hence capture the whole activities from locking to releasing resources and, consequently, encapsulate the whole period of resource use. As it is often the case that the resource can be locked by multiple objects (e.g. a robotic hand can hold any of the available blocks but at most one at a time), delete-relaxation heuristics, which are widely used in planning, tend to (heavily) underestimate the plan cost as they assume the resource can be used by multiple objects at the same time. Critical Section Macros, in other words, "bypass" the resource use part, therefore, mitigate the discrepancy between delete-relaxed heuristic estimation and the real cost of the plan.

Besides systems that learn macros by training on a set of small problems, there are several systems such as DHG [3] or OMA [17] that extracts planner-independent macros online, i.e., without the training phase. These systems, however, usually underperform the learning ones [17].

3.3 Transition-based Domain Control Knowledge

A domain engineer can leverage *Transition-Based Domain Control Knowledge* (TB-DCK) that is based on Finite-State Automata [7, 6]. TB-DCK represents a "grammar" of solution plans which is, roughly speaking, knowledge about ordering of planning operators in plans. On top of that, TB-DCK allows to define extra preconditions that can be used to restrict applicability of some instances of planning operators that are not useful.

In principle, TB-DCK consists of a set of DCK states and transitions that refer to which planning operators can be applied under specified conditions in a given planning state. To illustrate the concept of TB-DCK, we consider a simple logistic domain, where all locations are connected and packages have to be delivered from their initial locations to their goal locations by a truck that can carry at most one package. We define four predicates to describe the

environment: (at-truck ?l) – the truck is at location ?l; (at ?p ?l) – a package ?p is at location ?l; (in ?p) – a package ?p is in the truck; (empty-truck) – a truck is empty (no package is in it). Then, we define three planning operators: Drive(?from ?to) – the truck moves from the location ?from to the location ?to; Load(?p ?l) – a package ?p is loaded into the truck at the location ?l; and Unload(?p ?l) – a package ?p is unloaded from the truck at the location ?l. We may observe that i) an empty truck has to be moved only to locations where some package is waiting for being delivered, and ii) if a package is loaded to the truck (in its initial location), then the truck has to move to package's goal location, where the package is unloaded. We can encode such an observation in the TB-DCK as depicted in Figure 3.1. In particular, the DCK state s_0 represents that the truck is empty, s_1 represents that the truck has just been loaded with a package, and s_2 represents that the package is ready to be unloaded in its goal location. Hence there are two options while being in the DCK state s_0 - the truck can load a package that has not yet been delivered, or move to a location in which there is a package that has not yet been delivered. In the DCK states s_1 and s_2, there is only one option - move to the goal location of the loaded package, or unload the package, respectively.

TB-DCK consists of DCK states and transitions (as illustrated in Figure 3.1). Each transition is associated with a planning operator and, possibly, additional constraints (e.g. extra preconditions). Conceptually, the action selection that is the core aspect of planning algorithms is amended by selecting an outgoing transition from the current DCK state such that some instance of the associated planning operator is applicable in the current planning state and the additional constraints of the transition are met. Therefore, generated plans remain valid as the action applicability in the current planning state is checked while the action selection is restricted to only possibly useful alternatives.

One of the advantages of TB-DCK is that it can be encoded into a planning task and thus exploited by generic planning engines without their modification. The environment description of a TB-DCK enhanced planning domain model is extended by a "DCK state" predicate and "open goal" predicates (the latter is used for the additional transition constraints). Transitions are encoded as planning operators such that the associated planning operators are extended with the DCK state update as well as the additional constraints. It should be noted that more transitions can be associated with the same planning operators. In our running example, we have two transitions referring to the Drive operator. So, we create two operators, for instance, Drive-empty and Drive-full to reflect two different transitions in the given TB-DCK (see Figure 3.1). For details, see [7, 6].

Chapter 4

Planning Task Modelling

Knowledge Engineering is a discipline that, in a nutshell, deals with elicitation of domain knowledge and its effective representation. In Automated Planning, the Knowledge Engineering process elicits and transform requirements of a (real-world) domain into a planning domain model. The crucial aspect is the *quality* of the developed domain model [35]. As McCluskey et al. [35] argue, a good quality domain model has to be *complete*, i.e., solution plans match solutions in the real-world, *accurate*, i.e., the model accurately represents real-world domain requirements, and *operational*, i.e., planning engines can efficiently reason with the model. The latter two properties – accuracy and operationality – usually go against each other as more accurate models might be difficult for planing engines to reason with.

The key step of the Knowledge Engineering process is a formalised conceptualisation of the domain. In other words, a formal conceptualisation of the domain is a "mid-product" between an informal requirements description and a domain model [42]. That said, a domain expert and a domain model engineer in an iterative process come to an agreement on the environment description as well as the description of planning operators. After a formal conceptualisation of the domain is finished, a planning engineer refines a planning domain model. It is worth noting that the domain model might be of a different level of expressivity (e.g. classical, temporal) that on one side affects model accuracy, i.e., more expressive models are more accurate, while on the other side affects model operationality, i.e., less expressive models are less operational [42].

The following sections provide a case study of the Knowledge Engineering process of developing domain model for an application of task planning for Autonomous Underwater Vehicles (AUVs).

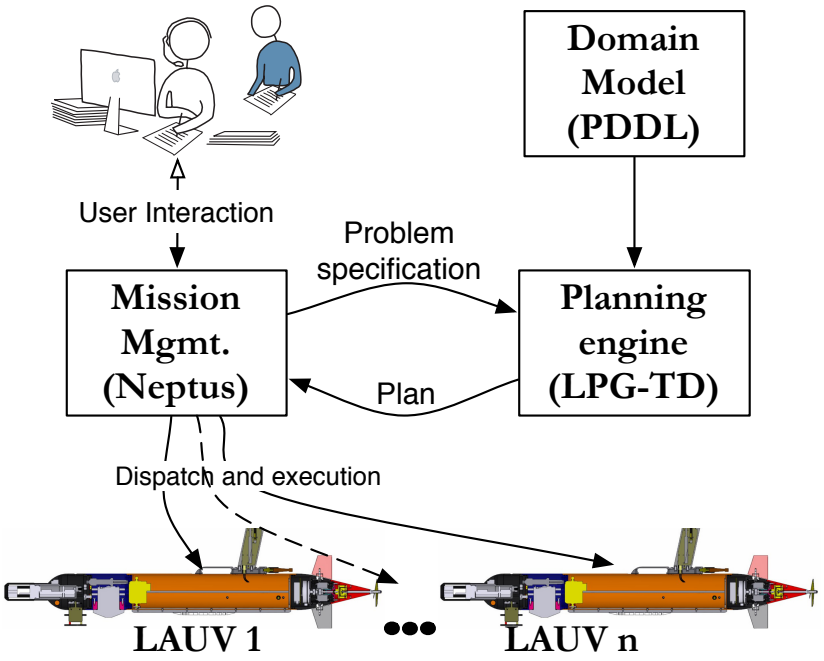


Figure 4.1: A modular architecture of the system [12]

4.1 Planning for AUVs

While operating Autonomous Underwater Vehicles (AUVs), human operators interact with a fleet of vehicles via NEPTUS, a graphical decision-support system with Graphical User Interface and analysis capabilities [23]. NEPTUS, in a nutshell, allows users to view vehicle data and to define behaviours and tasks of the vehicles. NEPTUS is connected via Inter-Module Communication (IMC) Protocol to DUNE that runs on board of each vehicle and is responsible for command execution and gathering of sensory data [40].

In a nutshell, a human operator specifies and executes high level commands in NEPTUS, for example, "move AUV1 to location X", or "sample AUV1 an object Y at location X". However, operating multiple (heterogeneous) AUVs to perform several tasks might be time consuming and error prone for human operators even though the mission might not be very complex.

The idea how to automatise AUV operations is to leverage Automated Planning for generating plans for AUVs such that they eventually complete all tasks specified by human operators [12]. An Automated Planning component can be embedded to the NEPTUS toolchain as depicted in Figure 4.1. Intuitively, the high level commands an operator can specify in NEPTUS

will correspond to actions specified in a planning domain model.

4.2 Requirements

Given a fleet of heterogeneous AUVs where each AUV has different payloads attached to it, a human operator specifies tasks in NEPTUS such that each task consists of an object or area of interest and a payload which has to be used to collect data about the object/area of interest. Each task has to be fulfilled by a single AUV. In other words, no task is collaborative, i.e., requiring two or more AUVs to work simultaneously.

To incorporate dynamic task allocation as well as recovery from task failures, the system has to be able to replan, i.e., to generate a new plan which replaces the old one. In particular, any user change might trigger replanning, however, for practical reason replanning is triggered (if a change occurs) periodically. Task failures can be directly reported to the system that reinserts the failed task back into the system and the task is considered for replanning.

During operations none of the AUVs has to run out of power. Also, two or more AUVs cannot operate at the same location or in the same area. On the other hand, when moving between locations AUVs can be in different depths and hence they should not collide. When operating underwater, AUVs are usually unable to establish reliable communication with the control center. In practice, AUVs might be radio silent for a couple of minutes. Hence, AUVs have to periodically return to their "depots" to establish communication with the control center.

Each AUV can move between two locations if it has enough power. An AUV can sample an object of interest or survey an area of interest if it is in the required location, has enough energy and has a required payload. If an AUV is close to the operation center, it can transfer acquired data there.

Addressing the communication issue can be done by requesting AUVs to return to their "depots" and establish communication with the control center in a given period of time. Specifically, each AUV can be away for at most a given period of time before returning to its "depot" to establish communication and then it can go away again (for at most the given period of time). That said, AUV will complete tasks in "rounds".

The planning engine has to then find a plan that allocates all user-specified tasks to particular AUVs such that the above constraints are met (e.g. AUVs will not run out of power) and the AUVs return to their "safe spots" next to the control center and transmit acquired data.

4.3 Domain Model

Actions were designed such that they reflect the high-level commands in the NEPTUS system. These actions are: *move* – an AUV moves between two locations, *sample* – an AUV samples an object of interest, *survey* – an AUV surveys an area of interest and *communicate* – an AUV transmits collected data to the control center. Environment is described by predicates such as *at* – specifying the position of an AUV, or *task-desc* – specifying what object/area of interest has to be sampled/surveyed by what payload, and by numeric fluents such as *battery-level* – specifying the remaining battery of an AUV. Details about the environment description can be found in works [12, 11].

Each action costs some energy, hence an AUV has to have enough energy to execute the action and after executing it AUV's battery level decreases. Similarly, each action takes time to be executed, to comply with the "maximum away time" constraint, we keep track about how long the vehicle is "away" and whether its "away time" is within the constraint. With regards to *sample* and *survey* actions, an AUV has to have a required payload for the given task. The *move* action is split into three variants – *move-to-sample*, *move-to-survey* and *move-to-depot* – that, roughly speaking, ensures that an AUV goes to locations where it has something to do. The *communicate* action is applicable only when an AUV is in its depot (so it can reliably establish communication with the control center). For the formal description of the domain model, see [12, 11].

4.4 Problem Specification

The planning problem is specified by concrete objects (e.g. AUVs, phenomena, locations), an initial state and a goal. The goal is to have the required data transmitted to the control center and having AUVs back in their depots. The initial state consists of initial locations of AUVs and their depots, locations or areas of phenomena, vehicles' payloads, task descriptions (specified by a human operator in NEPTUS), i.e., which types of payloads is to be used to collect data about the phenomena, distance between the locations, vehicle speed, maximum "away" time, battery levels, and battery consumption values per vehicle/payload.

One possibility is to generate plans that fulfil all specified tasks (at that time) while minimising "make-span" (the total plan execution time) alongside with the number of the *move-to-depot* actions (to minimise the number of AUVs' "rounds"). Such an approach is called the *all-tasks* model [11].

Another possibility is to generate plans only for the next "round" while optimising for the number of completed tasks for that round. Such an approach is called the *one-round* model [11].

4.5 Field Experiment

The models have been evaluated on a "mine-hunting" scenario in Porto Harbour [12, 11]. Three AUVs were used such that in the first stage they were set to perform several survey tasks while in the second stage they were set to perform several sample tasks. Generated plans were successfully executed, hence AUVs successfully completed assigned tasks in both stages.

The results have shown that there are considerable discrepancies between anticipated and actual action durations, especially for the *move* and *survey* actions [12]. The issue has been addressed to a large extent by considering "timed-waypoint" *move* actions that, roughly speaking, allow to adjust vehicle speed to mitigate the differences of the planned and actual time [11].

In longer-term autonomy with none or very limited human intervention, domain models have to adapt themselves according to observation of the environment (e.g. ocean currents) in order to provide accurate plans. That said, domain models have to have configurable parameters (e.g. action duration, energy consumption) that can be automatically modified onboard the vehicle according to data from its current observation [22].

Chapter 5

Conclusions

Knowledge Engineering for Automated Planning is an important discipline for designing and developing effective and efficient planning domain models. Efficiency of domain models can be improved by automatic extraction of additional knowledge such as macro-operators [5, 16, 15] or entanglements [18, 19] and incorporating such knowledge directly into the domain models, in other words reformulating them. Alternatively, additional knowledge can be manually specified (e.g. Transition-based Domain Control Knowledge [7]) and encoded into the domain model. Whereas there exist many techniques supporting classical planning, more expressive planning (e.g. temporal, conformant) is rarely supported at the moment.

From the perspective of design and development of domain models from the scratch, it is important to conceptualise the requirements, elicited from domain experts, into a formal representation that forms a base for the development of the domain model itself [42]. Such a domain modelling process has been applied for task planning for AUVs and tested in a real-world environment [12, 11]. The initial observations showed that planning and plan execution in a real-world environment pose a challenge of uncertainty (e.g. how long the action execution actually takes) that grows with the time (i.e., longer plan execution yields higher uncertainty) as the domain model might not accurately reflect the actual situation. Whereas some of the issues can be addressed by ad-hoc domain specific approaches (e.g. timed-waypoint move actions) [11], a more general (domain-independent) concept would be useful.

5.1 Future Work

In a nutshell, my future work concerns of bridging the gap between domain-independent planning and its use in real-world applications. Planning pro-

vides solutions that are usually more complex than rules provided by experts while still being explainable (e.g. it is known why a given action has been applied). That said, Automated Planning is an essential component of intelligent behaviour and thus is crucial for autonomous and intelligent systems.

However, in reality the environment is rarely static that makes planning and plan execution a challenge as dynamic environment might render plans invalid and, worse, might cause damages to the autonomous agent or robot. Specifically, my focus will be given to models considering exogenous non-deterministic events that might occur without the consent of the agent. Reasoning about unsafe or "dangerous" states, those in which occurrence of events might lead to dead-ends (and possible damage to the agent) [8, 9], is one possible direction towards safer autonomy. Taken from a different perspective, events might account for actions of adversaries that might try to deliberately hinder agent's plans. In such a case, a combination of Automated Planning and Game-theoretical approaches seems to be beneficial to address the issue (a preliminary work incorporating the Double Oracle algorithm into Classical planning to tackle zero-sum game-like scenarios has recently been published [41, 13]).

Another possibility (besides defining exogenous events) is to use self-adaptable domain models. It means, roughly speaking, that the models would contain customisable parameters that can be automatically adjusted according to agent's observations of the environment (that can be done, for example, by machine learning techniques, or by mathematical models). Self-adaptable domain models can be very useful in long-term autonomy, for example, in ocean exploration [22] or Urban Traffic Control [20, 33].

Bibliography

[1] M. Ai-Chang, J. L. Bresina, L. Charest, A. Chase, J. C. Hsu, A. K. Jónsson, B. Kanefsky, P. H. Morris, K. Rajan, J. Yglesias, B. G. Chafin, W. C. Dias, and P. F. Maldague. MAPGEN: mixed-initiative planning and scheduling for the mars exploration rover mission. *IEEE Intelligent Systems*, 19(1):8–12, 2004.

[2] C. Areces, F. Bustos, M. Dominguez, and J. Hoffmann. Optimizing planning domains by automatic action schema splitting. In *Proceedings of ICAPS*, 2014.

[3] G. Armano, G. Cherchi, and E. Vargiu. Automatic generation of macro-operators from static domain analysis. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 955–956, 2004.

[4] A. Botea, M. Enzenberger, M. Müller, and J. Schaeffer. Macro-FF: improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research (JAIR)*, 24:581–621, 2005.

[5] L. Chrpa. Generation of macro-operators via investigation of action dependencies in plans. *Knowledge Eng. Review*, 25(3):281–297, 2010.

[6] L. Chrpa and R. Barták. Enhancing domain-independent planning by transition-based domain control knowledge. In *The 33rd Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG)*, 2015.

[7] L. Chrpa and R. Barták. Guiding planning engines by transition-based domain control knowledge. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016.*, pages 545–548, 2016.

[8] L. Chrpa, J. Gemrot, and M. Pilát. Towards a safer planning and execution concept. In *29th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2017, Boston, MA, USA, November 6-8, 2017*, pages 972–976, 2017.

[9] L. Chrpa, J. Gemrot, and M. Pilát. Planning and acting with nondeterministic events: Navigating between safe states. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 9802–9809, 2020.

[10] L. Chrpa, T. L. McCluskey, M. Vallati, and T. Vaquero. The fifth international competition on knowledge engineering for planning and scheduling: Summary and trends. *AI Magazine*, 38(1):104–106, 2017.

[11] L. Chrpa, J. Pinto, T. S. Marques, M. A. Ribeiro, and J. B. de Sousa. Mixed-initiative planning, replanning and execution: From concept to field testing using AUV fleets. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*, pages 6825–6830, 2017.

[12] L. Chrpa, J. Pinto, M. A. Ribeiro, F. Py, J. B. de Sousa, and K. Rajan. On mixed-initiative planning and control for autonomous underwater vehicles. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*, pages 1685–1690, 2015.

[13] L. Chrpa, P. Rytír, and R. Horcík. Planning against adversary in zero-sum games: Heuristics for selecting and ordering critical actions. In *Proceedings of the Thirteenth International Symposium on Combinatorial Search, SOCS 2020, Online Conference [Vienna, Austria], 26-28 May 2020*, pages 20–28, 2020.

[14] L. Chrpa and F. H. Siddiqui. Exploiting block deordering for improving planners efficiency. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, pages 1537–1543, 2015.

[15] L. Chrpa and M. Vallati. Improving domain-independent planning via critical section macro-operators. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, pages 7546–7553, 2019.

[16] L. Chrpa, M. Vallati, and T. L. McCluskey. Mum: A technique for maximising the utility of macro-operators by constrained generation and use. In *Proceedings of the International Conference on Automated Planning and Scheduling, ICAPS*, pages 65–73, 2014.

[17] L. Chrpa, M. Vallati, and T. L. McCluskey. On the online generation of effective macro-operators. In *Proceedings of IJCAI*, pages 1544–1550, 2015.

[18] L. Chrpa, M. Vallati, and T. L. McCluskey. Outer entanglements: a general heuristic technique for improving the efficiency of planning algorithms. *J. Exp. Theor. Artif. Intell.*, 30(6):831–856, 2018.

[19] L. Chrpa, M. Vallati, and T. L. McCluskey. Inner entanglements: Narrowing the search in classical planning by problem reformulation. *Computational Intelligence*, 35(2):395–429, 2019.

[20] L. Chrpa, M. Vallati, and S. Parkinson. Exploiting automated planning for efficient centralized vehicle routing and mitigating congestion in urban road networks. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 8-12, 2019*, pages 191–194, 2019.

[21] A. Coles, M. Fox, and A. Smith. Online identification of useful macro-actions for planning. In *Proceedings of ICAPS*, pages 97–104, 2007.

[22] A. Dallolio, L. Bertino, L. Chrpa, T. A. Johansen, M. Ludvigsen, K. Orvik, L. H. Smedsrud, J. Sousa, I. B. Utne, and K. Rajan. Long-duration autonomy for open ocean exploration: Preliminary results & challenges. In *Robots in the Wild: Challenges in Deploying Robust Autonomy for Robotic Exploration Workshop*, 2019.

[23] P. S. Dias, R. M. F. Gomes, J. Pinto, G. M. Gonçalves, J. B. de Sousa, and F. M. L. Pereira. Mission planning and specification in the neptus framework. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation, ICRA 2006, May 15-19, 2006, Orlando, Florida, USA*, pages 3220–3225, 2006.

[24] A. Dulac, D. Pellier, H. Fiorino, and D. Janiszek. Learning useful macro-actions for planning with n-grams. In *25th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2013, Herndon, VA, USA, November 4-6, 2013*, pages 803–810, 2013.

[25] M. Fox and D. Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, 20:61–124, 2003.

[26] M. Ghallab, D. Nau, and P. Traverso. *Automated planning, theory and practice*. Morgan Kaufmann, 2004.

[27] M. Ghallab, D. S. Nau, and P. Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016.

[28] P. Haslum. Narrative planning: Compilations to classical planning. *J. Artif. Intell. Res.*, 44:383–395, 2012.

[29] P. Haslum and P. Jonsson. Planning with reduced operator sets. In *Proceedings of AIPS*, pages 150–158, 2000.

[30] J. Hoffmann. FF: the fast-forward planning system. *AI Magazine*, 22(3):57–62, 2001.

[31] T. Hofmann, T. Niemueller, and G. Lakemeyer. Initial results on generating macro actions from a plan database for planning on autonomous mobile robots. In *ICAPS*, pages 498–503, 2017.

[32] R. Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26(1):35–77, 1985.

[33] A. Lindsay, S. Franco, R. Reba, and T. L. McCluskey. Reprocess: Process refinement for improving accuracy in hybrid planning domain models. In *Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*, 2019.

[34] T. L. McCluskey and M. Vallati. Embedding automated planning within urban traffic management operations. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017, Pittsburgh, Pennsylvania, USA, June 18-23, 2017.*, pages 391–399, 2017.

[35] T. L. McCluskey, T. S. Vaquero, and M. Vallati. Engineering knowledge for automated planning: Towards a notion of quality. In *Proceedings of the Knowledge Capture Conference, K-CAP 2017, Austin, TX, USA, December 4-6, 2017*, pages 14:1–14:8, 2017.

[36] D. Mcdermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - The Planning Domain Definition

Language. Technical Report TR-98-003, Yale Center for Computational Vision and Control,, 1998.

[37] S. Minton. Quantitative results concerning the utility of explanation-based learning. In *Proceedings of AAAI*, pages 564–569, 1988.

[38] M. A. H. Newton, J. Levine, M. Fox, and D. Long. Learning macro-actions for arbitrary planners and domains. In *Proceedings of the International Conference on Automated Planning and Scheduling, ICAPS*, pages 256–263, 2007.

[39] S. Parkinson, A. Longstaff, S. Fletcher, A. Crampton, and P. Gregory. Automatic planning for machine tool calibration: A case study. *Expert Syst. Appl.*, 39(13):11367–11377, 2012.

[40] J. Pinto, P. S. Dias, R. Martins, J. Fortuna, E. Marques, and J. Sousa. The LSTS toolchain for networked vehicle systems. In *OCEANS-Bergen, 2013 MTS/IEEE*, pages 1–9. IEEE, 2013.

[41] P. Rytír, L. Chrpa, and B. Bosanský. Using classical planning in adversarial problems. In *31st IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019, Portland, OR, USA, November 4-6, 2019*, pages 1335–1340, 2019.

[42] M. M. S. Shah, L. Chrpa, D. E. Kitchin, T. L. McCluskey, and M. Vallati. Exploring knowledge engineering strategies in designing and modelling a road traffic accident management domain. In *IJCAI*, 2013.

[43] F. H. Siddiqui and P. Haslum. Block-structured plan deordering. In *AI 2012: Advances in Artificial Intelligence - 25th Australasian Joint Conference, Sydney, Australia, December 4-7, 2012. Proceedings*, pages 803–814, 2012.

Appendix A

Selected Papers

L. Chrpa. Generation of macro-operators via investigation of action dependencies in plans. *Knowledge Eng. Review*, 25(3):281–297, 2010.

M. M. S. Shah, L. Chrpa, D. E. Kitchin, T. L. McCluskey, and M. Vallati. Exploring knowledge engineering strategies in designing and modelling a road traffic accident management domain. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI*, 2013.

L. Chrpa, M. Vallati, and T. L. McCluskey. MUM: A technique for maximising the utility of macro-operators by constrained generation and use. In *Proceedings of the International Conference on Automated Planning and Scheduling, ICAPS*, pages 65–73, 2014.

L. Chrpa and F. H. Siddiqui. Exploiting block deordering for improving planners efficiency. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, pages 1537–1543, 2015.

L. Chrpa, J. Pinto, M. A. Ribeiro, F. Py, J. B. de Sousa, and K. Rajan. On mixed-initiative planning and control for autonomous underwater vehicles. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*, pages 1685–1690, 2015.

L. Chrpa and R. Barták. Guiding planning engines by transition-based domain control knowledge. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016.*, pages 545–548, 2016.

L. Chrpa, J. Pinto, T. S. Marques, M. A. Ribeiro, and J. B. de Sousa. Mixed-initiative planning, replanning and execution: From concept to field testing using AUV fleets. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*, pages 6825–6830, 2017.

L. Chrpa, M. Vallati, and T. L. McCluskey. Outer entanglements: a general heuristic technique for improving the efficiency of planning algorithms. *J. Exp. Theor. Artif. Intell.*, 30(6):831–856, 2018.

L. Chrpa and M. Vallati. Improving domain-independent planning via critical section macro-operators. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, pages 7546–7553, 2019.

L. Chrpa, M. Vallati, and T. L. McCluskey. Inner entanglements: Narrowing the search in classical planning by problem reformulation. *Computational Intelligence*, 35(2):395–429, 2019.

A.1 Generation of macro-operators via investigation of action dependencies in plans

L. Chrpa. Generation of macro-operators via investigation of action dependencies in plans. *Knowledge Eng. Review*, 25(3):281–297, 2010.

The Knowledge Engineering Review, Vol. 25:3, 281–297.
doi:10.1017/S0269888910000159

Generation of macro-operators via investigation of action dependencies in plans

LUKÁŠ CHRPA

Faculty of Mathematics and Physics, Department of Theoretical Computer Science and Mathematical Logic, Charles University in Prague, Malostranské náměsti 25, 118 00, Prague 1, Czech Republic
e-mail: chrpa@kti.mff.cuni.cz

Abstract

There are many approaches for solving planning problems. Many of these approaches are based on 'brute force' search methods and they usually do not care about structures of plans previously computed in particular planning domains. By analyzing these structures, we can obtain useful knowledge that can help us find solutions to more complex planning problems. The method described in this paper is designed for gathering macro-operators by analyzing training plans. This sort of analysis is based on the investigation of action dependencies in training plans. Knowledge gained by our method can be passed directly to planning algorithms to improve their efficiency.

1 Introduction

Planning is an important branch of Artificial Intelligence (AI) research. In planning, we define states of 'worlds' described by logical facts or functions and actions (or operators) that can modify these states. The purpose of planning is to generate a sequence of actions that transform the 'worlds' from some initial state to the given goal state.

Despite significant improvement in planning systems in the last few years, many automated planning algorithms are still based on 'brute force' search techniques accommodated with heuristics guiding the planner toward the solution Bonet and Geffner (1999). Hence, an important question is how such knowledge transformable into efficient planning heuristics can be found. Several heuristics are based on the structure of a Planning Graph Blum and Furst (1997). While these heuristics provide good results, an analysis of the Planning Graph does not seem to reveal complete information hidden in the plan structures. One of the most significant works from the past was a solver called REFLECT (Dawson & Siklóssy, 1977), which uses a preprocessing phase to ease its own solving. Specifically, the preprocessing phase consists of detecting incompatible predicates (i.e. predicates that cannot be simultaneously true) and building macro-operators (described in greater depth in Section 3). System PRODIGY (Minton & Carbonell, 1987) focuses on learning search control rules (i.e. logical rules describing relationships between predicates or operators). Search control rules were also applied in a well-known planner called TALPlanner (Kvanström & Magnusson, 2003). A newer approach presented in Hoffmann *et al.* (2004) describes Landmarks—facts that must be true in every valid plan. Another work (Knoblock, 1994) presents a structure called Causal Graph that describes dependencies between state variables. The most recent studies (Gimenez & Jonsson, 2007; Katz & Domshlak, 2007) analyze the Causal Graph with respect to complexity of planning problems. Both the Landmarks and the Causal Graphs are tools based on analyzing literals, giving us useful information about planning domains, but almost no information about the dependencies between actions in plans. One of the

most influential works from the area of action dependencies (McCain & Turner, 1997) defines a language for expressing causal knowledge (previously studied in Geffner (1990) and Lin (1995)) and formalizes the actions in it. One of the newest approaches (Vidal & Geffner, 2006) is based on plan space planning techniques over temporal domains. It gained very good results, especially in parallel planning, because it handles supports, precedences and causal links in a better way. There are other more practically oriented approaches, such as those described by Wu *et al.* (2005), where knowledge is gathered from plans stochastically, and Nejati *et al.* (2006) where learning from expert traces is adapted for acquiring classes of hierarchical task networks (HTN). Finally, papers (Chrpa & Bartak, 2008a, 2008b) define relations describing action dependencies and present methods based on these relations.

Another way to improve the efficiency of planners rests in using macro-actions or macro-operators that represent sequences of primitive actions or operators (related works are discussed in Section 3). In this paper, we provide a method generating macro-operators by investigation of action dependencies in training plans (the method is an extension of the work presented in (Chrpa, 2008)). Our method is used for learning macro-operators from simpler training plans; the learned macro-operators are encoded back into the domains and the primitive operators replaced by the macro-operators are removed from the domains. Such domains can be passed to planners without modifying their code. It means that our method is designed as a supporting tool for arbitrary planners.

The paper is organized as follows. In the next section, we introduce basic notions from the planning theory. Then we discuss related works in the area of macro-operators. After that, we provide a brief theoretical background of the problem of action dependencies in plans and then we describe our method for gathering macro-operators from training plans. Then we present and discuss the formal soundness and time complexity of our method. Finally, we discuss the experimental results of our method, similarities and differences with existing approaches and possible directions of our future research.

2 Preliminaries

Traditionally, AI planning (in state space) deals with the problem of finding a sequence of actions transforming the world from some initial state to a desired state. State s is a set of predicates that are true in s. Action a is a 3-tuple $(p(a), e^-(a), e^+(a))$ of sets of predicates such that $p(a)$ is a set of predicates representing the precondition of action a, $e^-(a)$ is a set of negative effects of action a, $e^+(a)$ is a set of positive effects of action a. Action a is applicable to state s if $p(a) \subseteq s$. If action a is applicable to state s, then new state s' obtained after applying action a is $s' = (s \setminus e^-(a)) \cup e^+(a)$. A planning domain is represented by a set of states and a set of actions. A planning problem is represented by a planning domain, an initial state and a set of goal predicates. A plan is an ordered sequence of actions that lead from the initial state to any goal state containing all of the goal predicates. For a deeper insight in this area, see Ghallab *et al.* (2004).

In this paper, we consider the classical representation of planing problems. This representation allows the definition of planning operators, in which actions are their grounded instances. Our approach supports the Typed STRIPS representation of PDDL (Planning Domain Definition Language).

3 Related works

Macro-operators (macro-actions) represent sequences of primitive operators (actions), but behave as common planning operators (actions). The advantage of using macro-operators is clear—shorter plans are explored to find a solution. However, macro-operators usually have much more instances than primitive operators, which leads to an increased branching factor for search.

One of the oldest approaches, STRIPS (Fikes & Nilsson, 1971), generates macro-actions from all subsequences of plans. It leads to plenty of useless macro-actions. REFLECT (Dawson & Siklóssy, 1977) builds macro-operators from pairs of primitive operators that are applied successively

and share at least one argument. In this case, macro-operators are learned directly from a domain analysis. However, it may lead to the generation of useless macro-operators. FM (McCluskey, 1987) follows the ideas used by STRIPS, but instead of STRIPS, FM compiles learned sequences of operators into one single operator representing the whole sequence of primitive ones. In addition, FM learns b-chunks that help it with instantiating of macro-actions. Even though FM gained a significant improvement against STRIPS, still it produced many useless and too complex macro-operators. MORRIS (Minton, 1985) learns macro-operators for STRIPS from parts of plans appearing frequently or being potentially useful (but having low priority). Macro Problem Solver (MPS), presented in Korf (1985), learns macro-actions only for particular goals. It needs different macro-actions when the problem instances scale or goals are different. MACLEARN (Iba, 1989) generates macro-actions that can 'traverse' from one peak of a particular heuristic function to another peak. A domain-dependently oriented work (Iba, 1985) discusses the usability of macro-operators in puzzle worlds (for instance, Peg Solitaire).

One of the state-of-the-art approaches, MARVIN (Coles & Smith, 2007; Coles *et al*., 2007) learns macro-operators online from action sequences that help FF-based planners to escape plateaus. It also learns macro-operators from plans of reduced versions of the given problems. One of the most outstanding works in the area of macro-actions is Macro-FF (Botea *et al*., 2005), a system for generating macro-operators through the analysis of static predicates. In addition, Macro-FF can learn macro-operators from training plans by analyzing successive actions. Macro-FF is produced in two versions. CA-ED version is designed for arbitrary planners where changing their source code is not necessary and SOL-EP version (planner dependent) where a planner (in this case, FF) is enchanted for handling macro-operators. WIZARD (Newton *et al*., 2007) learns macro-actions from training plans by genetic algorithms. There are defined several genetic operators working over action sequences appearing in training plans. WIZARD is designed for arbitrary planners. DHG (Armano *et al*., 2003, 2005) is able to learn macro-operators from static domain analysis by exploring a graph of dependencies between operators.

Our method is designed for domain-independent planning and for arbitrary planners like the other systems. Macro-operators can be assembled only from operators that are dependent, in terms that one operator provides a predicate (or predicates) to the other operator. It is similar to existing approaches. Nevertheless, there are some differences between our method and the existing approaches. We are able to detect pairs of actions that can be assembled into macro-actions, but the actions do not have to be necessarily successive in training plans. In addition, we are able to update the training plans in such a way that the updated training plans consider generated macro-operators. Therefore, it is not necessary to run the planners again. This can help us with another issue, the removal of unnecessary primitive operators that can be replaced by generated macro-operators. Despite the potential loss of completeness of some planning problems, planners benefit from the removal of primitive operators and the experiments we made on International Planning Competition (IPC) domains did not reveal any problem that became unsolvable. In addition, our method can reveal a suitable set of macro-operators in very little time. A more thorough comparison of our method with the existing ones is done in Section 8.3 (last paragraph).

4 Action dependencies in plans

Action choice is the key part of planning. Plans often contain sequences with dependencies between actions in the sense that one action provides predicates serving as preconditions for the other actions. In this section, we formally describe this dependency relation and present some of its useful features.

Every action needs some predicates to be true before the action is applicable. These predicates are provided by the initial state or by other actions that were performed before. If we have a plan solving a planning problem, we can identify which actions are providing these predicates to other actions that need them as their precondition. The following definition describes this relation formally.

DEFINITION 1.1. Let $\langle a_1, \ldots, a_n \rangle$ *be an ordered sequence of actions. Action* a_j *is straightly dependent on the effects of action* a_i *(denoted as* $a_i \rightarrow a_j$*) if and only if* $i < j$ $(e^+(a_i) \cap p(a_j)) \neq \emptyset$ *and* $(e^+(a_i) \cap p(a_j)) \not\subseteq \bigcup_{t=i+1}^{j-1} e^+(a_t)$.
Action a_j is *dependent on the effect of action* a_i if and only if $a_i \rightarrow {}^*a_j$ where $\rightarrow^*$ is a transitive closure of the relation $\rightarrow$.

The relation of straight dependency on the effects of action (hereinafter straight dependency only) means that $a_i \rightarrow a_j$ holds if some predicate from the precondition of action a_j is provided by action a_i and a_i is the last action before action a_j providing that predicate. Notice that an action may be straightly dependent on more actions (if it has more predicates in the precondition). The relation of dependency on the effects of action (hereinafter dependency only) is a transitive closure of the relation of straight dependency.

Remark 1.2. Negation of the relations of straight dependency and dependency is denoted in the following way:

- $a_i \not\rightarrow a_j$ means that a_j is not straightly dependent on a_i (i.e. $\neg(a_i \rightarrow a_j)$).
- $a_i \not\rightarrow^* a_j$ means that a_j is not dependent on a_i (i.e. $\neg(a_i \rightarrow {}^*a_j)$).

Let us define the complementary notion of action independency. The motivation behind this notion is that two independent actions being adjacent can be swapped in the action sequence without influencing the plan (lemma 1.4 (see below) which has been formally proved in Chrpa and Bartak (2008b).

DEFINITION 1.3. *Let* $\langle a_1, \ldots, a_n \rangle$ *be an ordered sequence of actions. Actions* a_i *and* a_j *(without loss of generality, we assume that* $i < j$*) are independent on the effects (denoted as* $a_i \nleftrightarrow a_j$*) if and only if* $a_i \not\rightarrow {}^*a_j$, $p(a_i) \cap e^-(a_j) = \emptyset$ *and* $e^+(a_j) \cap e^-(a_i) = \emptyset$.

LEMMA 1.4. *Let* $\pi = \langle a_1, \ldots, a_{i-1}, a_i, a_{i+1}, a_{i+2}, \ldots, a_n \rangle$ *be a plan solving planning problem P and* $a_i \nleftrightarrow a_{i+1}$. *Then plan* $\pi' = \langle a_1, \ldots, a_{i-1}, a_{i+1}, a_i, a_{i+2}, \ldots, a_n \rangle$ *also solves planning problem P.*

The symbol for relation of independency on the effects (hereinafter independency only) evokes a symmetrical relation even though, according to Definition 1.3, the relation of independency does not have to be necessarily symmetrical. The reason for using the symmetrical symbol is hidden in the previously mentioned property of the independency relation (lemma 1.4).

Remark 1.5. Since the relations of dependency and independency are not complementary, we define the following symbol:

- $a_i \leftrightarrow a_j$ means that a_j is not independent on a_i (i.e. $\neg(a_i \nleftrightarrow a_j)$).

Computation of the relation of straight dependency is quite straightforward. The idea is based on storing of indices of the last actions that created the particular predicates. Concretely, each predicate p is annotated by $d(p)$, which refers to the last action that created it. We simulate execution of the plan and when action a_i is executed, we find the dependent actions by exploring $d(p)$ for all predicates p in the precondition of a_i. The relation of straight dependency can be naturally represented as a directed acyclic graph, so the relation of dependency is obtained as a transitive closure of the graph Mehlhorn (1984). The relation of independency can be easily computed by checking every pair of actions a_i and a_j $(i < j)$ on satisfaction of the conditions from Definition 1.3. It is straightforward that the time complexity in the worst case is $O(n^2)$ where n represents the length of the sequence of actions (plan).

5 Identifying actions that can be assembled

We obtain a new macro-action by assembling two primitive actions. The result of applying a macro-action to some state is identical to the result of applying the primitive actions in the given

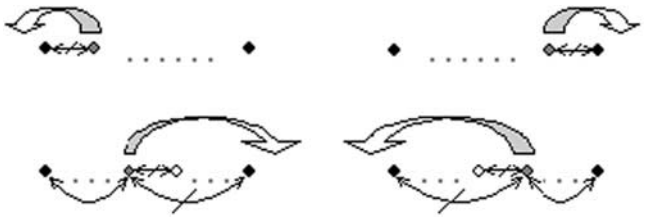


Figure 1 Four different situations for moving the intermediate actions (gray-filled) before or behind one of the boundary actions (black-filled)

order to the same state. A macro-action obtained by assembling of actions a_i and a_j (in this order) will be denoted as $a_{i,j}$, formally:

- $p(a_{i,j}) = p(a_i) \cup (p(a_j) \setminus e^+(a_i))$
- $e^-(a_{i,j}) = (e^-(a_i) \cup e^-(a_j)) \setminus e^+(a_j)$
- $e^+(a_{i,j}) = (e^+(a_i) \cup e^+(a_j)) \setminus e^-(a_j)$

This approach can be easily extended for more actions; see Chrpa *et al.* (2007).

It is clear that we have to decide which actions can be assembled. We can analyze several previously found plans (training plans), where we focus on actions (instances of operators) that are (or can be) often successive. We can analyze the plans by looking for successive actions only. However, in such a case, we may miss many pairs of actions that can be performed successively, but in the plans, there are some other actions placed between them. If the intermediate actions can be moved before or behind the chosen pair of actions without losing plan validity, then we can assemble even non-successive actions. We use the main property of independent actions (can be swapped if adjacent) for detection if a pair of actions can be assembled (we can make them adjacent). To get more insight regarding permutations in plans, see Fox and Long (1999). Figure 1 shows four different situations (actually two situations and their mirror alternatives) for moving the intermediate actions. Clearly, if the intermediate action is adjacent and independent on the boundary action, we can move this action before or behind one of the boundary actions (according to lemma 1.4). If the intermediate action is not independent on one of the boundary actions, then we have to move it only before or behind the other boundary action, which means that this intermediate action must be independent on all actions in between (including the boundary action).

The algorithm (Figure 2) is based on repeated application of the above steps. If all intermediate actions are moved before or behind the boundary actions, then the boundary actions can be assembled (become adjacent). If some intermediate actions remain and none of the steps can be performed, then the boundary actions cannot be assembled. Anyway, if the algorithm returns true (i.e. actions can be assembled), we also obtain lists of action indices representing (intermediate) actions that must be moved before (respectively behind) actions a_i and a_j. Usage of these lists will be explained in the following section.

6 Generating macro-operators

As mentioned earlier, planning domains include planning operators rather than ground actions. Assembling operators rather than actions is more advantageous, because macro-operators can be more easily converted into more complex problems than macro-actions. The idea of detecting such operators, which can be assembled, is based on the investigation of training plans, where we explore pairs of actions (instances of operators) that can be assembled more times.

DEFINITION. 2.1. *Let M be a square matrix where both rows and columns represent all planning operators in the given planning domain. If field $M(k,l)$ contains a pair $\langle N, V \rangle$ such that:*

- N is a number of such pairs of actions a_i, a_j that are instances of k-th and l-th planning operator (in order), $a_i \rightarrow a_j$ and both actions a_i and a_j can be assembled in some example plan.

```
Function DETECT-IF-CAN-ASSEMBLE(IN index i, IN index j, IN independency relation S, OUT list of indices L,
    OUT list of indices R) : returns bool
    D := {k|i < k < j}
    L := R := ∅
    Repeat
        chg := false
        k := min(D)    // min(D) returns the smallest element from D or 0 if D is empty
        If k > 0 and (i, k) ∈ S then
            D := D \ {k}
            chg := true
            L := L ∪ {k}
        EndIf
        k := max(D)    // max(D) returns the greatest element from D or 0 if D is empty
        If k > 0 and (k, j) ∈ S then
            D := D \ {k}
            chg := true
            R := R ∪ {k}
        EndIf
        Z := {x|x ∈ D ∧ (i, x) ∉ S}
        k := max(Z)
        If k > 0,(k, j) ∈ S and ForEach l ∈ D ∧ l > k (k, l) ∈ S holds then
            D := D \ {k}
            chg := true
            R := R ∪ {k}
        EndIf
        Z := {x|x ∈ D ∧ (x, j) ∉ S}
        k := min(Z)
        If k > 0,(i, k) ∈ S and ForEach l ∈ D ∧ l < k (l, k) ∈ S holds then
            D := D \ {k}
            chg := true
            L := L ∪ {k}
        EndIf
    Until not chg
    If D = ∅ then Return true else Return false
EndFunction
```

Figure 2 Algorithm for detecting pairs of actions that can be assembled

In addition, a_i (resp. a_j) cannot be in such a pair with the other instances of l-th (resp. k-th) operator.

- V is a set of variables shared by k-th and l-th planning operators.

Then M is a matrix of candidates.

In other words, the matrix of candidates contains proper pairs of actions (instances of planning operators) for assembling (or becoming macro-actions). The algorithm (Figure 3) constructs the matrix of candidates from the given set of training plans solving the planning problems in the same domain. Computation of the sets of variables that operators share needs to be clarified. For example, in a variant of a well-known BlockWorld domain, there are operators PICK (box, hoist and surface) and DROP (box, hoist and surface). If we decide to make a macro-operator PICK-DROP (consisting of PICK and DROP operators in this order), then we can also see that the box and hoist are always the same (we are picking and dropping the same box with the same hoist in time), and only the surface may differ. Generally, we observe which parameters (objects) are shared by actions and select such parameters that are shared by all pairs of actions (instances of the given operators) that can be assembled.

Now, we explain the purpose of lists L and R that are generated in function DETECT-IF-CAN-ASSEMBLE. If we have to update plans by replacing selected actions by macro-actions (instances of generated macro-operators), then we must also reorder other actions to keep the

```
Procedure CREATE-MATRIX(IN set of plans P, OUT matrix M)
    Set M as empty square matrix
    ForEach π in P do
        Compute D as a relation of straight dependency on actions from π
        Compute S as a relation of independency on actions from π
        ForEach (i, j) ∈ D do
            If DETECT-IF-CAN-ASSEMBLE(i, j, S, L, R) then
                Set k as the id of the operator whose a_i is an instance
                Set l as the id of the operator whose a_j is an instance
                Compute V as a set of arguments that a_i and a_j share
                If M_{k,l} is empty then
                    M_{k,l} :=< 1, V >
                Else
                    ⟨N, OV⟩ := M_{l,k}
                    If a_i resp. a_j are not already selected as a candidate with l-th operator resp. k-th operator then
                        N1 := N + 1 else N1 := N
                    M_{l,k} := ⟨N1, OV ∩ V⟩
                EndIf
            EndIf
        EndForeach
    EndForeach
EndProcedure
```

Figure 3 Algorithm for creating the matrix of candidates for assemblage

```
(:action pickup_stack
    :parameters (?x ?y)
    :precondition (and (clear ?x)(ontable ?x)(handempty)(clear ?y))
    :effect (and (clear ?x)(on ?x ?y)(handempty)
            (not (ontable ?x))(not (holding ?x))(not (clear ?y)) )
  )
```

Figure 4 Example of PICKUP-STACK macro-operator

(training) plans valid. The following approach shows how to reorder actions in plan $\pi = \langle a_1, \ldots, a_n \rangle$, if a pair of selected actions a_i, a_j is assembled into macro-action $a_{i,j}$:

- actions $a_1, \ldots a_{i-1}$ remain in their positions
- actions listed in L are moved (in order) to positions $i, \ldots, +|L|-1$
- macro-action $a_{i,j}$ is added to $i+|L|$-th position
- actions listed in R are moved (in order) to positions $i+|L|+1, \ldots, j-1$
- actions $a_{j+1}, \ldots, a_n$ are moved one position back (to positions $j, \ldots, n-1$)

To generate macro-operators from training plans (in the given domain), we can use the following approach (formally in Figure 5). We create macro-operators repeatedly until no other macro-operator can be created. At first, we have to compute the matrix of candidates from all the training plans (CREATE-MATRIX). Then we select a proper candidate for creating macro-operators (SELECT-CANDIDATE), which means that such a candidate must satisfy certain conditions (which will be explained later). To ensure the soundness of the generated macro-operators, we have to assign inequality constraints for macro-operator arguments. It prevents a possible instantiation of invalid macro-actions if these arguments are set as equal. In Figure 4, we can see an example of the PICKUP-STACK macro-operator. If the arguments are set as equal, we can simply see that such an instance is applicable, but invalid (when unfolded). Inequality constraints can be easily detected by simulation of performance of the operators that are going to be assembled. After a creation of the macro-operator from the selected candidate, we must update all training plans (UPDATE-PLANS), which means that we replace particular pairs

of actions by the corresponding instances of the new macro-operator. UPDATE-PLANS procedure can be easily implemented by application of the previously described approach (reordering actions after assembling) on every pair of actions (instances of the selected operators) in every plan.

Last but not least, the remaining unexplained issue is the function for selecting the proper candidate for assemblage (SELECT-CANDIDATE). We suggested selecting such a candidate that satisfies the following conditions (let $f(O)$ represent the frequency of operator O (how many instances of operator O occur in all the training plans), $a(O)$ represent the arity of operator O (number of arguments of O), $N_{i,j}$ represent the number N in field $M_{i,j}$ of the matrix of candidates and $V_{i,j}$ represent the set of variables shared by i-th and j-th operator):

$$max\left(\frac{N_{i,j}}{f(O_i)}, \frac{N_{i,j}}{f(O_j)}\right) \geqslant b \tag{6.1}$$

$$\frac{N_{i,j}}{\sum_k f(O_k)} \geqslant c \tag{6.2}$$

$$a(O_i) + a(O_j) - |V_{i,j}| \leqslant d \tag{6.3}$$

Condition 6.1 says that we are looking for such operators whose instances usually appear (or can appear) successively. Constant $b \in \langle 0; 1 \rangle$ represents a pre-defined bound that prevents selecting such operators whose instances do not appear successively so often. It is clear that if the bound is too small, many operators may be assembled. It usually causes that generated macro-operators are representing almost the whole training plans, which does not bring any contribution to planners. On the other hand, if the bound is too big, almost no operators may be assembled, which means that the domains may remain unchanged. However, in some cases we are not able to prevent the generation of such macro-operators representing a huge part of some training plan, even though b is set quite big. The reason for this rests in the fact that sometimes only one (or a very few) instances of some operator occur in all the training plans. Almost always, we can find some other action that can be assembled with this instance, because the ratio between the number of candidates (stored in the matrix of candidates) and the frequency of the operator becomes 1. It means that the operator will be certainly selected for assemblage. To prevent this unwanted selection, we can add condition 6.2 allowing only the selection of such operators whose ratio between the number of instances being able to be assembled (stored in $N_{i,j}$) and the number of all actions from all the training plans reaches a predefined constant c.

Another problem we are facing rests in the fact that many planners use grounding. It means that the planners generate all possible instances of operators that are used during planning. However, macro-operators usually have more parameters than primitive operators, which means that macro-operators may have much more instances than primitive operators. To avoid troubles with planners regarding grounding, we should limit the maximum number of parameters for each macro-operator by a pre-defined constant d (condition 6.3). If there are more candidates satisfying all the conditions, then we prefer the candidate with the maximum value of the expression listed in condition 6.1.

We must also decide which macro-operators can be added to the domain and which primitive operators can be removed from the domain. Here, we decided to add every macro-operator whose frequency in the updated training plans is non-zero. Similarly, we decided to remove every primitive operator whose frequency in the updated training plans becomes zero. It is clear that it may cause a possible failure when solving non-training problems. Fortunately, in IPC benchmarks, it does not happen (we did not experience any such problem during the experiments). If for some problem planners fail to find a solution, then it is possible to bring the removed primitive operators back to the domain and run the planners again.

7 Soundness and complexity discussion

We assume that all plans used for analysis by our algorithms are valid. To ensure the validity of the plans, we can simply extend the algorithm for computing the relation of straight dependency by checking the satisfiability of action preconditions. It is quite straightforward that the algorithms for computing the relations of (straight) dependency and independency (sketches of the algorithms are discussed at the end of Section 3) are sound and can be computed in $O(n^2)$ steps (in the worst case), where n represents the length of the input plan. Soundness and time complexity of the other presented algorithms are justified in more detail.

PROPOSITION 3.1. *The algorithm DETECT-IF-CAN-ASSEMBLE (Figure 2) is sound and can be computed in the worst case in $O(l^2)$ steps where l is the number of intermediate actions (actions between a_i and a_j).*

Proof. The idea of the algorithm is based on moving intermediate actions before or behind defined actions. It is clear that a pair of adjacent actions can be assembled into a macro-action (we must follow their order) without loss of validity of the examined plan. The moving of intermediate actions can be done in the four cases (Figure 1), where two of them are a mirror of the other two. Without loss of generality, we prove the soundness and complexity only in two cases (on the left-hand side on Figure 1), because the soundness and complexity of the other cases can be proved analogically. First, if $a_i \nleftrightarrow a_{i+1}$, then by applying lemma 1.4, we can move a_{i+1} before a_i without loss of the plan's validity and it takes a constant time (i.e. $O(1)$). Second, assume that $a_i \leftrightarrow a_k$, $k < j$ and k is the greatest possible value. If $a_k \nleftrightarrow a_l \; \forall l : k < l < j$, then by repetitively applying lemma 1.4, we can move a_k behind a_j also without loss of the plan's validity. It can take at most $O(l)$ steps. The algorithm always terminates because in each run of the loop we remove at least one intermediate action. When no intermediate action remains, the loop ends. It means that the cycle is performed at most l times. Hence, in the worst case the algorithm requires $O(l^2)$ steps to perform. □

PROPOSITION 3.2. *The algorithm CREATE-MATRIX (Figure 3) is sound and can be computed in the worst case in $O(n^4)$ steps where n is the total length of all the training plans.*

Proof. For each training plan, the algorithm explores each pair of actions being in the relation of straight dependency by the algorithm DETECT-IF-CAN-ASSEMBLE (Figure 2), which is sound (proposition 3.1). It is clear that by using this approach, we can build the matrix of candidates consistent with the previously stated conditions. It is also clear that in the worst case we can have $O(n^2)$ relations of straight dependency and the algorithm DETECT-IF-CAN-ASSEMBLE in the worst case can be performed in $O(n^2)$ steps (proposition 3.1—considering that l can be close to n). Summarized, it gives us the time complexity $O(n^4)$ in the worst case. □

THEOREM 3.3. *The algorithm GENERATE-MACRO (Figure 5) is sound and can be computed in the worst case in $O(n^5)$ steps, where n is the total length of all the training plans.*

Proof. From the soundness of the algorithms DETECT-IF-CAN-ASSEMBLE (proposition 3.1) and CREATE-MATRIX (proposition 3.2), we know that each candidate for assemblage represents a pair of actions that can be assembled without loss of the plan's validity. If we generalize it and consider the inequality constraints, then we can simply see that each macro-operator produced by this algorithm is valid. The algorithm also always terminates because in each step of the loop the length of the training plans decreases at least by one, which means that the loop can be performed in the worst case $n - 1$ times. Together with the complexity of the algorithm CREATE-MATRIX (proposition 3.2), it gives us the time complexity $O(n^5)$ in the worst case. □

It is well known that if we add a generated macro-operator into the domain, then the domain remains valid. We can also remove the primitive operators fully replaced by the generated macro-operators. It brings us an improvement of the performance of the planners. However, it may cause an insolvability of some problems that were solvable in original domains. Hopefully, in all tested

```
Procedure GENERATE-MACRO(IN set of plans P, OUT set of macro-operators O)
    O := ∅
    Repeat
        picked := false
        CREATE-MATRIX(P,M)
        If SELECT-CANDIDATE(M,C) then
            picked := true
            ASSIGN-INEQUALITY-CONSTRAINTS(C)
            O := O ∪ {C}
            UPDATE-PLANS(P,C)
        EndIf
    Until not picked
EndProcedure
```

Figure 5 Algorithm for generation of macro-operators

cases it did not happen as we can see in the experiments (Section 8). Despite the high time complexity of our method (in the worst case), the experiments showed that our method is fast (tenths of a second for one run of the GENERATE-MACRO procedure).

8 Experimental evaluation

In this section, we present the experimental evaluation of our method. We compare the performance of the given planners between the original domains and the domains updated by our method. The planning domains and planning problems that we used here are well known from the IPC. We have done the evaluation in the following steps:

- Generate several simpler training plans as an input for our method.
- Generate macro-operators by our method and add them to the domains. Remove such primitive operators that no longer appear in the updated training plans.
- Compare running times for more complex problems between the original domains and the updated domains. The time limit was set to 600 seconds.

We used SATPLAN 2006 (Kautz *et al.*, 2006) and SGPLAN 5.22 (Hsu *et al.*, 2007) both for the generation of the training plans (for the learning phase) and for the comparison of the running times and quality of plans. We also used LAMA (Richter & Westphal, 2008), Filtering and Decomposition for Planning (FDP) (Grandcolas & Pain-Barre, 2007) and LPG-td (Gerevini & Serina, 2002) for the comparison of running times and quality of plans (not for the learning phase).[1] The choice of the planners was motivated by great results that the planners achieved on the (several last) IPCs. Since SATPLAN and FDP cannot handle negative preconditions (which are necessary for representation of inequality constraints), we used a tool called ADL2STRIPS[2] that can produce grounded STRIPS domain from ADL domain.

For the evaluation, we used IPC domains *Blocks*, *Depots*, *Zenotravel*, *Rovers*, *Gripper*, *Satellite* and *Goldminer*.[3]

8.1 Generating macro-operators and updating the domains

As mentioned earlier, the generation of macro-operators depends on pre-defined bounds b, c and d (conditions 6.1, 6.2 and 6.3). The number of training plans for each domain differs from 3 to 6 with respect to their lengths. The average time taken by both SGPLAN and SATPLAN to

[1] The results of LPG are only briefly reported.

[2] Available on IPC4 website.

[3] Can be obtained on `http://ipc.icaps-conference.org`

Table 1 Suggestion of our method—the best results for the particular domains

Domain	Added macro-operators	Removed primitive operators
Blocks	PICKUP-STACK, UNSTACK-STACK, UNSTACK-PUTDOWN	PICKUP, PUTDOWN, STACK, UNSTACK
Depots	LIFT-LOAD, UNLOAD-DROP	LIFT, LOAD, UNLOAD, DROP
Zenotravel	REFUEL-FLY	REFUEL
Rovers	CALIBRATE-TAKE-IMAGE	CALIBRATE, TAKE-IMAGE
Gripper	PICK-MOVE-DROP, MOVE-PICK-MOVE-DROP	MOVE, PICK, DROP
Satellite	SWITCH-ON-CALIBRATE	SWITCH-ON, CALIBRATE, SWITCH-OFF
Gold miner	MOVE-PICKUP-LASER, MOVE-DETONATE-BOMB-MOVE-PICK-GOLD	PICKUP-LASER, PICK-GOLD, DETONATE-BOMB

generate a training plan was (mostly) within tenths of a second.[4] Despite the high (worst-case) time complexity $O(n^5)$ (Theorem 3.3), the average time taken by one run of our method (GENERATE-MACRO procedure) was within tenths of a second.[5]

We used different settings of bounds b, c and d and two different planners (SATPLAN 2006, SGPLAN 5.22) for the generation of the training plans. First, bound d was set to $N+1$ (except for the Satellite domain and Gripper domain for SATPLAN's training plans), where N represents the greatest number of arguments of operators in the particular domain, because we did not want to generate too complicated macro-operators. If bound b was set too low, then many useless macro-operators were generated. We found out that a reasonable value of bound b can be almost in all cases 0.8; only in the Gripper domain (for SATPLAN's training plans), we lowered it to 0.6. Setting bound c was not as definite as setting the other bounds. Usually, the reasonable value was between 0.1 and 0.05, but in the Gripper domain it was set to 0.03. The reason for keeping bound c low (0.03–0.05) rested in the fact that in the Blocks and Gripped domains, all the primitive operators were replaced by generated macro-operators. The choice of a planner for the generation of training plans brought several differences—only in the Blocks domain, it resulted in the same result. In the Depots domain, we were not able to remove some primitive operators when SATPLAN's training plans were used as we did when SGPLAN's training plans were used. In the Zenotravel and Rovers domains, we were not able to learn any suitable set of macro-operators when SATPLAN's training plans were used. Likewise, in the Satellite domain, when SGPLAN's training plans were used. In the Gripper domain, the results of learning differed with respect to planners' strategies—SATPLAN prefers to carry balls in both robotic hands, SGPLAN prefers to carry balls just in one robotic hand. In the Gold Miner domain, the planners preferred different operators, which resulted in slightly different results of learning.

The results of learning (best for the particular domains) are showed in Table 1. We stated only such alternatives that provided the best results in the running times and quality of plans comparison for the particular domains.

8.2 Comparison of running times and quality of plans

In this evaluation, we used SGPLAN 5.22, an absolute winner of the IPC 5, SATPLAN 2006, co-winner of optimal track in the IPC 5, LAMA, winner of the IPC 6 suboptimal track and FDP, participant of the IPC 5 and LPG-td, awarded on the IPC 4. The benchmarks ran on XEON 2.4 GHz, 1GB RAM and Ubuntu Linux. The results are presented in Tables 2 and 3. We chose such problems (in most domains) that were neither so easy nor so hard for the particular planners, because the evaluation of these problems usually tells us the most about the particular domains.

[4] Performed on XEON 2.4GHz, 1GB RAM, Ubuntu Linux.

[5] Performed on Core2Duo 2.66GHz, 4GB RAM, Win XP SP2.

Table 2 Comparison of running times and plans lengths (we assume that macro-actions are unfolded into primitive actions) for SGPLAN (left-hand side) and SATPLAN (right-hand side)

	SGPLAN							SATPLAN					
	Time (in seconds)			Plan length				Time (in seconds)			Plan length		
Problem	orig	upd-SG	upd-SAT	orig	upd-SG	upd-SAT	Problem	orig	upd-SAT	upd-SG	orig	upd-SAT	upd-SG
Blocks14-0	>600.00	0.03	0.03	NA	48	48	Blocks14-0	23.58	3.14	3.14	38	56	56
Blocks14-1	>600.00	0.03	0.03	NA	44	44	Blocks14-1	38.06	3.84	3.84	36	88	88
Blocks15-0	>600.00	0.32	0.32	NA	88	88	Blocks15-0	46.90	7.24	7.24	40	60	60
Blocks15-1	179.84	0.05	0.05	114	54	54	Blocks15-1	45.68	7.63	7.63	52	142	142
depots1817	24.56	15.52	20.71	100	104	94	depots4321	5.24	4.40	2.07	43	41	38
depots4534	>600.00	0.53	54.71	NA	112	110	depots5656	222.42	>600.00	143.33	70	NA	59
depots5656	410.94	0.32	7.70	133	132	82	depots6178	6.82	43.14	26.11	51	50	42
depots7615	8.48	1.88	2.14	98	102	91	depots7654	10.04	25.96	16.45	41	56	39
zeno-5-20a	0.88	0.75	–	98	101	–	depots8715	35.96	46.95	err	50	38	err
zeno-5-20b	1.07	0.77	–	92	97	–	zeno-3-10	3.77	–	4.17	31	–	35
zeno-5-25a	1.74	1.05	–	124	122	–	zeno-5-10	34.07	–	48.19	42	–	38
zeno-5-25b	0.57	0.58	–	117	125	–	zeno-5-15a	92.13	–	30.93	50	–	51
rovers4621	2.31	0.03	–	48	44	–	zeno-5-15b	err	–	err	err	–	err
rovers5624	0.10	0.02	–	52	52	–	rovers4621	182.20	–	>600.00	47	–	NA
rovers7182	4.32	0.12	–	90	91	–	rovers5624	4.30	–	>600.00	62	–	NA
rovers8327	3.53	0.06	–	78	71	–	rovers8327	1.17	–	>600.00	45	–	NA
gripper16	0.05	0.05	1.11	135	135	101	gripper8	>600.00	8.14	0.03	NA	53	71
gripper17	0.06	0.06	1.31	143	143	107	gripper9	>600.00	12.86	0.06	NA	59	79
gripper18	0.06	0.07	1.56	151	151	113	gripper10	>600.00	19.78	0.04	NA	65	87
gripper19	0.06	0.07	1.83	159	159	119	gripper11	>600.00	err	0.07	NA	err	95
gripper20	0.07	0.08	2.13	167	167	125	gripper12	>600.00	err	0.06	NA	err	103
satellite26	3.73	–	29.99	138	–	138	satellite15	82.79	88.25	–	68	70	–
satellite27	4.73	–	13.20	138	–	139	satellite16	>600.00	115.07	–	NA	69	–
satellite28	12.87	–	260.26	193	–	193	satellite17	129.39	127.62	–	74	73	–
satellite29	18.69	–	70.36	195	–	195	satellite18	25.05	24.40	–	44	43	–
satellite30	31.57	–	117.52	231	–	231	satellite19	>600.00	574.46	–	NA	66	–
satellite31	56.65	–	201.36	272	–	272	gminer7×7-06	6.00	5.07	6.34	33	35	34
gminer7×7-06	err	0.01	0.01	NA	33	30	gminer7×7-07	6.08	4.91	5.82	38	38	37
gminer7×7-07	err	0.02	0.01	NA	34	65	gminer7×7-08	3.06	2.08	2.81	25	25	25
gminer7×7-08	err	0.01	0.01	NA	25	26	gminer7×7-09	4.24	3.47	4.26	33	30	29
gminer7×7-09	err	0.01	0.01	NA	29	32	gminer7×7-10	5.96	4.83	6.05	35	35	35
gminer7×7-10	err	0.01	0.02	NA	33	43							

Table 3 Comparison of running times and plans lengths (we assume that macro-actions are unfolded into primitive actions) for LAMA (left-hand side) and FDP (right-hand side)

	LAMA							FDP					
	Time (in seconds)			Plan length				Time (in seconds)			Plan length		
Problem	orig	upd-SG	upd-SAT	orig	upd-SG	upd-SAT	Problem	orig	upd-SG	upd-SAT	orig	upd-SG	upd-SAT
Blocks14-0	0.12	0.08	0.08	84	84	84	Blocks10-1	>600.00	11.82	11.82	NA	34	34
Blocks14-1	0.13	0.06	0.06	52	44	44	Blocks10-2	>600.00	6.57	6.57	NA	34	34
Blocks15-0	0.44	0.10	0.10	144	52	52	Blocks11-0	>600.00	178.31	178.31	NA	36	36
Blocks15-1	0.27	0.14	0.14	112	62	62	Blocks11-1	>600.00	146.24	146.24	NA	34	34
depots1817	>600.00	93.68	>600.00	NA	122	NA	Blocks11-2	>600.00	93.02	93.02	NA	38	38
depots4534	243.61	1.39	9.81	122	67	107	depotprob7512	1.66	0.10	0.37	15	15	15
depots5656	>600.00	0.53	7.70	NA	70	98	depotprob1935	>600.00	11.41	44.18	NA	27	27
depots7615	>600.00	5.71	61.61	NA	77	78	depotprob6512	>600.00	70.24	288.27	NA	30	30
zeno-5-20a	1.22	0.87	–	91	91	–	depotprob1234	>600.00	29.18	147.31	NA	23	21
zeno-5-20b	1.55	0.75	–	83	91	–	zeno-2-4	5.34	7.88	–	11	11	–
zeno-5-25a	2.85	0.98	–	95	105	–	zeno-2-5	42.29	73.01	–	11	11	–
zeno-5-25b	7.18	1.42	–	100	115	–	zeno-2-6	58.18	7.71	–	15	15	–
rovers4621	0.06	0.06	–	47	47	–	zeno-3-6	551.53	>600.00	–	11	NA	–
rovers5624	0.08	0.04	–	50	50	–	gripper8	>600.00	0.04	8.98	NA	71	53
rovers7182	0.23	0.18	–	90	90	–	gripper9	>600.00	0.06	16.91	NA	79	59
rovers8327	0.15	0.10	–	71	77	–	gripper10	>600.00	0.08	32.66	NA	87	65
gripper16	0.05	0.06	3.76	101	135	101	gripper11	>600.00	0.10	55.57	NA	95	71
gripper17	0.05	0.07	4.49	107	143	107	gripper12	>600.00	0.13	92.14	NA	103	77
gripper18	0.06	0.08	5.26	113	151	113	satellite3	1.39	–	0.28	11	–	11
gripper19	0.07	0.08	6.13	122	159	119	satellite4	62.52	–	17.24	17	–	17
gripper20	0.07	0.10	7.02	128	167	125	satellite5	>600.00	–	264.06	NA	–	15
satellite26	3.85	–	7.37	139	–	139	satellite6	>600.00	–	>600.00	NA	–	NA
satellite27	2.80	–	3.63	135	–	139							
satellite28	>600.00	–	11.76	NA	–	194							
satellite29	16.82	–	19.88	190	–	191							
satellite30	72.18	–	32.63	229	–	229							
satellite31	40.46	–	67.47	269	–	272							
gminer7×7-06	0.22	0.04	0.03	170	31	31							
gminer7×7-07	0.04	0.04	0.03	65	34	65							
gminer7×7-08	>600.00	0.03	0.03	NA	25	26							
gminer7×7-09	0.14	0.04	0.03	130	29	32							
gminer7×7-10	0.30	0.04	0.03	176	31	43							

The less complex problems were solved in the updated domains almost as fast as or a bit slower than in the original ones (except for the Rovers domain in SATPLAN's evaluation). The hardest problems (both original and updated) were not solved within the time limit of 600 seconds.

SGPLAN performed well in the original domains on almost all the tested problems except Blocks problems, some Depots problems and Gold Miner problems. Running times in the updated domains were always better except in the Gripper domain, where the running times were slightly worse, and the Satellite domain where the results were significantly worse. The quality of the plans[6] generated in the updated domains was not much worse, however; sometimes, the quality was slightly better and, surprisingly, in one Blocks problem it was more than twice better. The best results SGPLAN were reached in the Blocks domain where the speed-up was quite impressive. The possible reason may rest in the fact that SGPLAN's heuristics (FF-based) do not handle well problems like Blocks or Depots, because the plan quality was significantly better in the updated problems as well. SGPLAN's behavior in the Gold Miner domain was weird, because for all more complex (original) problems, SGPLAN terminated without throwing any error message after about 3 minutes of running.

SATPLAN, unfortunately, did not benefit often from our method. In the Blocks domain, SATPLAN was able to generate plans faster, but at the price of significantly worse quality of plans. However, SATPLAN produced very good results in the updated Gripper domain, where the problems normally unsolvable (in 600 seconds) were solved in a couple of seconds (for the domain updated on the basis of SATPLAN's training plans) or in hundreds of a second (for the domain updated on the basis of SGPLAN's training plans). The reason for that rests in the fact that SATPLAN uses a Planning Graph and each tested problem in the updated Gripper domain can be solved in only two layers. SATPLAN also gained quite good results in the Satellite and Gold Miner domains. Errors thrown by SATPLAN were caused by insufficient memory or a large domain file (produced by the ADL2STRIPS tool).

FDP is a planner based on CSP techniques that guarantees optimal plans. Even though FDP seems to be an ideal candidate for generating training plans for the learning phase, it fails to find a reasonable number of training plans in reasonable time. However, the experiments showed that the performance on the updated domains significantly increased in most of the tested problems. Using macro-operators reduced the depth of the search, which expectedly increased FDP's performance. The worse results gained in the Zenotravel domain was caused by the fact that no macro-action was used in the solutions of the updated problems (except zeno-2-6). An absence of results on the Rovers and Gold Miner domains is caused by the inability of FDP to process the domains descriptions (both for the original and updated ones) correctly.

LAMA is a planner that combines the Causal Graph heuristic and FF-based heuristics. In the Depots, Blocks and Gold Miner domains, the quality of plans was significantly better in updated domains. In addition, the time comparison for the Depots domain showed a significant increase in performance. The results correlate a bit with the results achieved by SGPLAN, because SGPLAN uses FF-based heuristics as well.

We also made experiments with the LPG-td (Gerevini & Serina, 2002) planner. LPG is based on local search techniques. Our experiments showed significantly worse performance in the Blocks and Depots domains; in the other domains, LPG performed almost the same. However, the results of LPG had huge discrepancies (both the running times and the quality of plans) with respect to the selected random seed.

8.3 Additional remarks

The presented results showed an interesting improvement for more complex problems in the domains updated by our method. Even though we used only at most six training plans for each domain (depending on the length of the training plans), we usually gathered enough knowledge for

[6] A ratio of the length of the plans in the original domains and the length of the plans in the updated domains—macro-actions are unfolded into primitive actions.

updating the domains. Even though we removed primitive operators from the original domains, we were able to solve correctly each problem in the updated domains. The reason may be that planning problems from the IPCs usually differ by the number of objects and not by different types of initial states or goals. However, there exist domains (for instance, *Freecell*, *Pipesworld*, *N-puzzle* and *Sokoban*) where our method did not manage to find any reasonable set of macro-operators (in terms, that found macro-operators did not fully replace any primitive operator).

Generated macro-operators used in the comparison were in almost all cases combined only from two primitive operators, except in the Gripper and Gold-Miner domains. Although the construction of more complex macro-operators may reduce the depth of the search, such macro-operators may have much more instances that can cause troubles to planners (increased branching factor).

The success of our method depends on several factors. First, training plans should be optimal (shortest) or nearly optimal, because non-optimal plans may contain flaws (useless actions) that may prohibit the detection of useful macro-operators or useless primitive operators. Second, we have to decide what result of our method (generated macro-operators and removed primitive operators) is the best. We followed the strategy where the particular generated macro-operator replaces at least one primitive operator that is removed from the domain. The experiments showed that our strategy is reasonable and contributive in many cases. Of course, there is a possible improvement that considers planners' specifics and strategies. SGPLAN is a planner that decomposes a problem into subproblems and solves them by other planning techniques, mostly FF-based. LAMA also uses FF-based heuristics and, in addition, Causal Graph heuristics. FF-based planning techniques usually experience difficulties with plateaux. Therefore, if there are such macro-operators that help the FF-based planner to escape plateaux, then the performance of the planner should significantly increase. It has been already studied in Coles and Smith (2007). SATPLAN is a planner that translates Planning Graph into SAT and then uses a SAT solver to solve the problem. Potential success, in this case, mainly rests in the reduction of makespan (i.e. the numbers of layers of the planning graph that must be explored). However, if makespan is reduced only slightly, it may not result in speed-up, because the layers can be much more complex. It also depends on the first appearance of instances of particular macro-operators in the Planning Graph (the later the better).

For most of the older approaches (typically for STRIPS or MPS), it is quite common to generate more complex macro-operators to penetrate the depth of the search as much as possible. Our method is able to generate more complex macro-operators, if bounds b and c are kept lower and bound d is kept higher. However, such macro-operators are very problem-specific, which makes them unusable for a larger scale of problems in the given domain. Systems like PRODIGY or DHG use static domain analysis and do not require training plans for their learning. Some macro-operators learned by these systems may be unnecessary (i.e. instances of these macro-operators usually do not appear in solutions of most of the problems). State-of-the-art systems Marvin or Macro-FF (SOL-EP version) are built on the FF planner. These systems achieved very promising results, but they cannot be applied with other planners. WIZARD and Macro-FF (CA-ED version) are, like our method, designed as a supporting tool for arbitrary planners without changing their code. WIZARD learns macro-operators genetically from training plans, which follows quite a different policy than our method does. The usability of macro-operators is evaluated by the monitoring of running behavior of planners on updated training problems (by the macro-operators). WIZARD, in comparison to our method, reported better results in the Satellite domain or with the LPG planner, but WIZARD spends many hours on the learning phase, whereas our method spends seconds. Macro-FF (CA-ED version) generates macro-operators from an analysis of static predicates, then adds them into the domain and then generates training plans (with the macro-operators). Unlike that, our method generates macro-operators from training plans gathered from the original training problems and does not require to resolve them (by the planners) in their updated form (with macro-operators). The idea, how the usability of macro-operators is evaluated, is quite similar to our method, but a bit simpler—Macro-FF (CA-ED version) picks the n most frequent macro-operators (assembled from two primitive operators).

In addition, our method detects which primitive operators can be removed (with the risk of losing completeness). For example, in the Depots domain, our method and Macro-FF (CA-ED version) found the same macro-operators. Our method, in addition, removed four (resp. two) primitive operators by using SGPLAN (resp. SATPLAN) for generating the training plans. Removing the primitive operators brought much more benefit to the planners' performance and often to the quality of plans.

9 Conclusion

In this paper, we presented a method for generating macro-operators and removing useless primitive operators from the planning domain. The method explores pairs of actions (not necessarily adjacent) that can be assembled in the given training plans. It results both in the detection of suitable macro-operators and primitive operators that can be removed. The method is designed as a supporting tool for arbitrary planners. The presented evaluation showed that using our method is reasonable and can transparently improve the planning process, especially on more complex planning problems. Nevertheless, the results were obtained by evaluation of IPC benchmarks only. Probably, the main disadvantage of IPC benchmarks rests in similarities of the planning problems (the problems differ only in the number of objects), which makes the analysis of plans structures much easier. In real world applications, it may be more difficult to use our method properly (e.g. we need a set of good training plans, etc.). Classification of such problems where we can remove particular primitive operators without loss of the problems' completeness remains an open problem. Furthermore, more complex macro-operators may contain many parameters that may cause big difficulties to planners. We are also investigating possibilities of pruning potentially useless actions (operators' instances), see Chrpa and Bartak (2009).

In future, we should also focus on a possible extension of our method for generating HTNs. Then we can use some HTN planner, for example, SHOP2 (Nau *et al*., 2003). This idea partially follows the idea listed in Nejati *et al*. (2006). In addition, we should investigate more deeply how stochastic data gathered during the execution of our method (like the number of operators in training plans, etc.) can be efficiently used. We should also study action dependencies more from the side of predicates, because it may reveal knowledge that can be used as heuristics for planners.

Acknowledgments

The research is supported by the Czech Science Foundation under contract nos 201/08/0509 and 201/05/H014.

References

Armano, G., Cherch, G. & Vargiu, E. 2005. DHG: a system for generating macro-operators from static domain analysis. In *Proceedings of Artificial Intelligence and Applications (AIA)*, Innsbruck, Austria, 18–23.

Armano, G., Cherch, G. & Vargiu, E. 2003. A parametric hierarchical planner for experimenting abstraction techniques. *Proceedings of IJCAI*, Acapulco, Mexico, 936–941.

Blum, A. & Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* **90**(1–2), 281–300.

Bonet, B. & Geffner, H. 1999. Planning as heuristic search: new results. In *Proceedings of ECP*, Durham, UK, Lecture Notes in Computer Science **1809**, 360–372. Springer.

Botea, A., Enzenberger, M., Muller, M. & Schaeffer, J. 2005. Macro-FF: improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research* **24**, 581–621.

Chrpa, L. 2008. Generation of macro-operators via investigation of actions dependencies in plans. In *Proceedings of KEPS*, Sydney, Australia. `http://ktiml.mff.cuni.cz/~bartak/KEPS2008/`

Chrpa, L. & Bartak, R. 2008a. Looking for planning problems solvable in polynomial time via investigation of structures of action dependencies. In *Proceedings of SCAI*, Stockholm, Sweden. IOS Press, **173**, 175–180.

Chrpa, L. & Bartak, R. 2008b. Towards getting domain knowledge: plans analysis through investigation of actions dependencies. In *Proceedings of FLAIRS*, Coconut Grove, Florida, USA. AAAI Press, 531–536.

Chrpa, L. & Bartak, R. 2009. Reformulating planning problems by eliminating unpromising actions. In *Proceedings of SARA*, Lake Arrowhead, California, USA. AAAI Press, 50–57.

Chrpa, L., Surynek, P. & Vyskocil, J 2007. Encoding of planning problems and their optimizations in linear logic. In *Proceedings of INAP/WLP*. Technical Report 434, Bayerische Julius–Maximilians–Universität Würzburg, 47–58.

Coles, A., Fox, M. & Smith, K. A. 2007. Online identification of useful macro-actions for planning. In *Proceedings of ICAPS*, Providence, RI, USA. AAAI Press, 97–104.

Coles, A. & Smith, K. A. 2007. Marvin: a heuristic search planner with online macro-action learning. *Journal of Artificial Intelligence Research* **28**, 119–156.

Dawson, C. & Siklóssy, L. 1977. The role of preprocessing in problem solving systems. In *Proceedings of IJCAI* **1**, Cambridge, MA, USA, 465–471.

Fikes, R. & Nilsson, L. 1971. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Inteligence* **2**(3–4), 189–208.

Fox, M. & Long, D. 1999. The detection and exploitation of symmetry in planning problems. In *Proceedings of IJCAI*, Stockholm, Sweden, 956–961.

Geffner, H. 1990. Causal theories of nonmonotonic reasoning. In *Proceedings of AAAI*, Boston, Massachusetts, USA. AAAI Press, 524–530.

Gerevini, A. & Serina, I. 2002. LPG: a planner based on local search for planning graphs with action costs. In *Proceedings of AIPS*, Toulouse, France. AAAI Press, 13–22.

Ghallab, M., Nau, D. & Traverso, P. 2004. *Automated Planning, Theory and Practice*. Morgan Kaufmann Publishers.

Gimenez, O. & Jonsson, A. 2007. On the hardeness of planning problems with simple causal graphs. In *Proceedings of ICAPS*, Providence, RI, USA. AAAI Press, 152–159.

Grandcolas, S. & Pain-Barre, C. 2007. Filtering, decomposition and search space reduction for optimal sequential planning. In *Proceedings of AAAI*, Vancouver, British Columbia, Canada. AAAI Press, 993–998.

Hoffmann, J., Porteous, J. & Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research* **22**, 215–278.

Hsu, C.-W., Wah, B. W., Huang, R. & Chen, Y 2007. *SGPlan*. `http://manip.crhc.uiuc.edu/programs/SGPlan/index.html`

Iba, G.A. 1989. A Heuristic approach to the discovery of macro-operators. *Machine Learning* **3**, 285–317.

Iba, G. A. 1985. Learning by discovering macros in puzzle solving. In *Proceedings of IJCAI*, Los Angeles, California, USA, 640–642.

Katz, M. & Domshlak, C. 2007. Structural patterns of tracable sequentialy-optimal planning. In *Proceedings of ICAPS*, Providence, RI, USA. AAAI Press, 200–207.

Kautz, H., Selman, B. & Hoffmann, J 2006. Satplan: planning as satisfiability. In *Proceedings of IPC*. `http://zeus.ing.unibs.it/ipc-5/booklet/deterministic11.pdf`

Knoblock, C. 1994. Automatically generated abstractions for planning. *Artificial Intelligence* **68**(2), 243–302.

Korf, R. 1985. Macro-operators: a weak method for learning. *Artificial Intelligence* **26**(1), 35–77.

Kvanström, J. & Magnusson, M. 2003. TALplanner in the third international planning competition: extensions and control rules. *Journal of Artificial Intelligence Research* **20**, 343–377.

Lin, F. 1995. Embracing causality in specifiing the indirect effects of actions. In *Proceedings of IJCAI*, Montréal, Québec, Canada. AAAI press, 1985–1991.

McCain, N. & Turner, H. 1997. Causal theories of action and change. In *Proceedings of AAAI*, Providence, RI, USA. AAAI press, 460–465.

McCluskey, T. L. 1987. Combining weak learning heuristics in general problem solvers. In *Proceedings of IJCAI*, Milan, Italy, 331–333.

Mehlhorn, K. 1984. *Data Structures and Algorithms 2: Graph Algorithms and NP-completeness*. Springer-Verlag.

Minton, S. 1985. Selectively generalizing plans for problem-solving. In *Proceedings of IJCAI*, Los Angeles, California, USA, 596–599.

Minton, S. & Carbonell, J. G. 1987. Strategies for learning search control rules: an explanation-based approach. In *Proceedings of IJCAI*, Milan, Italy, 228–235.

Nau, D., Au, T., Ilghami, O., Kuter, U., Mudrock, J., Wu, D. & Yaman, F. 2003. SHOP2: an HTN planning system. *Journal of Artificial Intelligence Research* **20**, 379–404.

Nejati, N., Langley, P. & Konik, T. 2006. Learning hierarchical task networks by observation. In *Proceedings of ICML*, Pittsburgh, Pennsylvania, USA, *ACM International Conference Proceeding Series 148*, 665–672.

Newton, M. H., Levine, J., Fox, M. & Long, D. 2007. Learning macro-actions for arbitrary planners and domains. In *Proceedings of ICAPS*, Providence, Rhode Island, USA, 256–263.

Richter, S. & Westphal, M 2008. The LAMA planner using landmark counting in heuristic search In *Proceedings of the 6th IPC*. `http://ipc.informatik.uni-freiburg.de/`

Vidal, V. & Geffner, H. 2006. Branching and Pruning: an optimal temporal POCL planner based on constraint programming. *Artificial Intelligence* **170**(3), 298–335.

Wu, K., Yang, Q. & Jiang, Y. 2005. Arms: action-relation modelling system for learning action models. In *Proceedings of ICKEPS*. `http://scom.hud.ac.uk/scomtlm/competition/papers/paper6.pdf`

A.2 Exploring knowledge engineering strategies in designing and modelling a road traffic accident management domain

M. M. S. Shah, L. Chrpa, D. E. Kitchin, T. L. McCluskey, and M. Vallati. Exploring knowledge engineering strategies in designing and modelling a road traffic accident management domain. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI*, 2013.

Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence

Exploring Knowledge Engineering Strategies in Designing and Modelling a Road Traffic Accident Management Domain

Mohammad M. Shah, Lukáš Chrpa,
Diane Kitchin, Thomas L. McCluskey and **Mauro Vallati**
Department of Informatics
School of Computing and Engineering
University of Huddersfield, United Kingdom
Email:{s.shah, l.chrpa, d.kitchin, t.l.mccluskey, m.vallati}@hud.ac.uk

Abstract

Formulating knowledge for use in AI Planning engines is currently something of an ad-hoc process, where the skills of knowledge engineers and the tools they use may significantly influence the quality of the resulting planning application. There is little in the way of guidelines or standard procedures, however, for knowledge engineers to use when formulating knowledge into planning domain languages such as PDDL. This paper seeks to investigate this process using as a case study a road traffic accident management domain.

Managing road accidents requires systematic, sound planning and coordination of resources to improve outcomes for accident victims. We have derived a set of requirements in consultation with stakeholders for the resource coordination part of managing accidents. We evaluate two separate knowledge engineering strategies for encoding the resulting planning domain from the set of requirements: (a) the traditional method of PDDL experts and text editor, and (b) a leading planning GUI with built in UML modelling tools.

These strategies are evaluated using process and product metrics, where the domain model (the product) was tested extensively with a range of planning engines. The results give insights into the strengths and weaknesses of the approaches, highlight lessons learned regarding knowledge encoding, and point to important lines of research for knowledge engineering for planning.

1 Introduction

Knowledge Engineering for automated planning is the process that deals with the acquisition, formulation, validation and maintenance of planning knowledge, where a key product is the *domain model*. The field has advanced steadily in recent years, helped by a series of international competitions[1], the build up of experience from planning applications, along with well developed support environments (for example, *Europa* [Barreiro *et al.*, 2012], itSIMPLE [Vaquero *et al.*, 2007], GIPO [Simpson *et al.*, 2007]). It is generally accepted that effective tool support is required to build domain models and bind them with planning engines into applications. There have been reviews of such knowledge engineering tools and techniques for AI Planning [Vaquero *et al.*, 2011], and some work was done in comparing tools using sets of "features" for the ICKEPS competitions [Barták *et al.*, 2010]. While these works are illuminating, they are not founded on practice-based evaluation, in part, no doubt, because of the difficulty in setting up evaluations of methods themselves. Given a new planning domain, there is little published research to inform engineers on which method and tools to use in order to effectively *engineer a planning domain model*. This is of growing importance, as domain independent planning engines are now being used in a wide range of applications, with the consequence that operational problem encodings and domain models have to be developed in a standard language such as PDDL. In particular, at the difficult stage of domain knowledge formulation, changing a statement of the requirements into something formal - a PDDL domain model - is still somewhat of a "black art", usually conducted by a team of AI experts using text editors. On the other hand, the use of tools such as itSIMPLE or GIPO, with which experts generate a high level diagrammatic description and *automatically* generate the domain model, have not yet been proven to be more effective than hand coding.

In this paper we explore the deployment of automated planning to assist the management of accident planning, using a set of requirements derived from operational manuals and stakeholder interaction. Moreover, in introducing a new planning domain, we take the opportunity to employ and hence evaluate two separate methods for knowledge formulation: (i) the traditional method of hand-coding by PDDL experts, using a text editor and relying on dynamic testing for debugging; (ii) itSIMPLE [Vaquero *et al.*, 2007], an award-winning GUI, utilising a method and tool support based on the Uni-

[1] for the most recent see http://icaps12.poli.usp.br/icaps12/ickeps

fied Modelling Language (UML). Evaluating these two approaches with respect to qualitative and quantitative measures, gives a range of interesting insights into their strengths and weaknesses for encoding new domains. Evaluation measures used are based on two standard categories in the software engineering literature - process (the method of encoding and debugging the domain model) and product (the domain model, and its use within a planner to produce plans). In particular, we provide a comparison of the operationality of the planning domain models generated through the proposed methods, based on the performance of state-of-the-art domain independent planners.

2 Case Study: Management of Road Traffic Accidents (RTAs)

Road traffic management operations are subject to rising costs, rising public expectations, more complex and demanding goals, and contain a great deal of legacy software. Recent technological advances have in part confounded this by providing more management controls and more surveillance data. The need to look to reducing costs, while maintaining level of service is a high priority. The area of *incident management* on the Road Traffic Network combines the challenge above, while demanding an optimal solution in real time; further, the space of possible states, and configurations of the incident, is far too large to be able to generate concrete plans a priori. Systematic, robust planning with the coordination of human and technical resources is the key to managing these incidents. In particular where the process involves safety of victims and other road users, such as in RTA, the responding agencies need to deliver accident management activities safely and efficiently [Owens *et al.*, 2000].

Our work in producing a set of requirements for the automation of accident management plans has been performed in the context of the EU-funded network *Autonomic Road Transport Support*[2] consisting of both academics and practising transportation engineers. Using contacts through this network, two scientific exchanges with transportation specialists, contributions to transport workshops, and a set of manuals [HA, 2009; Owens *et al.*, 2000; Benesch, 2011], we have elicited a set of requirements for the RTA planning problem.

In the UK, the main responsibility for managing and dealing with an incident lies with the highways agency (HA) that serves that area, as well as the police, ambulance, traffic offers and breakdown services. Part 7 of the UK's Highway's Agency Manual [HA, 2009] is our major source of knowledge. This identifies the service providers responsible for dealing with accidents at an operational level, with police leading co-ordination in and around the scene. The phases of an incident are detection, verification, response, scene management, recovery and restoration. Here we assume that an accident has been detected, and consider the planning element for the subsequent phases, with the overall requirement that the planning function is to provide whoever is leading the incident management with an operational plan for managing services. Incidents are centrally controlled, and there is only one leader at any point in time (though leadership can change, e.g. from the police to the HA). Within this context there are *major* and *critical* levels of incident. The former can be described as disasters; the HA require a Crisis management Team to deal with this. We will concentrate on incidents at critical levels, which consist of single or multiple accidents in a region, and typically may consist of up to 10s of vehicles, requiring several emergency vehicles, within a single region.

[2] www.cost-arts.org

2.1 Initial Domain Analysis

An initial conceptualisation of the RTA domain is described in the following paragraphs.

A *Road Network* is represented by an undirected graph (V, E) where vertices V stand for *locations* and edges E for *roads*. It is useful to effectively abstract the topology of the Road Network, since the Road Network usually considers a region covering several 'clusters', i.e., towns/cities or districts (e.g. see Figure 1), with locations of interest (e.g. Hospitals or Police Stations). We assume that all the locations within a 'cluster' are connected to each other. 'Clusters' are connected only if there is a road between them. *Assets* $X = X_s \cup X_m$ are divided into two categories, *static assets* X_s (e.g. Police Stations, Hospitals, Fire Stations) and *mobile assets* X_m (e.g. Police Cars, Ambulances, Fire Brigades). Let $T \subseteq \mathbb{R}_0^+$ be a set of time-stamps. We define a function loc which for an asset and time-stamp returns the location (or $\perp$ which stands for a situation when the asset is on the way), formally $loc : X \times T \to V \cup \{\perp\}$. Clearly, for every static asset $x \in X_s$ $loc(x, t)$ is constant (i.e. its value is not dependent on the time-stamp). Mobile assets can be moved between locations using roads (i.e. a mobile asset can move from one location to another if and only if these locations are connected by road). Artefacts Y (e.g. accident victims, damaged cars etc.) cannot move freely between locations (unlike mobile assets) but they need a mobile asset (e.g. an ambulance) which can transport them to different locations. We define a function in which for an artefact and time-stamp returns either an asset (static or mobile) an artefact is attached to, or a location an artefact is located if the artefact is not attached to any asset, or $\perp$ if an artefact is being attached or detached from an asset, formally $in : Y \times T \to X \cup V \cup \{\perp\}$. An artefact can be attached to an asset if and only if the artefact is currently not attached to any other asset and the current location of an artefact is the same as the current location of the asset. Similarly, if an artefact is unattached from an asset then its location will be the same as the current location of the asset. Each asset may have a limited *capacity*, i.e., a maximum number of attached artefacts in the same time. We define a function $cap : X \to \mathbb{N}$ referring to an asset capacity. It must hold that $\forall t \in T, \forall x \in X : |\{y \mid y \in Y \wedge in(y, t) = x\}| \leq cap(x)$. Assets and artefacts can also interact with each other in order to modify their characteristic properties. For instance, the police have to confirm an accident or a paramedic has to give first aid to victims before they are taken to hospital. Hence, we define *properties* as sets of values characterising artefacts and/or assets (e.g. accident victims can be waiting for first aid, being aided, aided or delivered to a hospital).

All the above thus specify the environment of the RTA domain. This environment can be modified by (planning)

operators representing types of actions, specified via preconditions (what must be met in order to apply the operator) and effects (what is changed in the environment after applying the operator). We define the following operator families which modify the environment of the RTA domain (we assume that the operator is applied in a time-stamp t and lasts for Δt time).

move(x, l_1, l_2) moves a mobile asset $x \in X_s$ from a location l_1 to a location l_2 ($l_1, l_2 \in V$). As a precondition it must hold that $loc(x,t) = l_1$ and l_1 and l_2 are connected with a road (i.e. $(l_1, l_2) \in E$). An effect of applying the operator is that $loc(x, t + \Delta t) = l_2$ and $\forall t' \in (t, t + \Delta t) : loc(x, t') = \perp$.

attach(y, x, l) attaches an artefact $y \in Y$ to an asset $x \in X$ in a location $l \in V$. As a precondition it must hold that $loc(x,t) = in(y,t) = l, \forall t' \in [t, t+\Delta t] : loc(x,t') = l$ and $|\{y' \mid in(y', t) = x\}| < cap(x)$. An effect of applying the operator is that $in(y, t + \Delta t) = x$, $\forall t' \in (t, t + \Delta t) : in(y, t') = \perp$.

detach(y, x, l) detaches an artefact $y \in Y$ from an asset $x \in X$ in a location $l \in V$. As a precondition it must hold that $\forall t' \in [t, t + \Delta t] : loc(x, t') = l$ and $in(y, t) = x$. An effect of applying the operator is that $in(y, t+\Delta t) = l, \forall t' \in (t, t + \Delta t) : in(y, t') = \perp$.

interact$(e_1, e_2, l, p_1, \ldots, p_6)$ refers to interaction between artefacts or assets $e_1, e_2 \in X \cup Y$ in a location $l \in V$. As a precondition it must hold that $\forall t' \in [t, t + \Delta t] : (loc(e_1, t') = l \Leftrightarrow e_1 \in X) \vee (in(e_1, t') = l \Leftrightarrow e_1 \in Y)$, $(loc(e_2, t') = l \Leftrightarrow e_2 \in X) \vee (in(e_2, t') = l \Leftrightarrow e_2 \in Y)$ and e_1 and e_2 has properties p_1 and p_2 in time t. An effect of applying the operator is that properties of e_1 and e_2 in any $t' \in (t, t + \Delta t)$ are p_3, p_4 respectively and properties of e_1 and e_2 in $t + \Delta t$ are p_5, p_6 respectively.

There are further constraints which must be met. Operator families *attach* and *detach* must not be executed simultaneously for a given asset (i.e., during attaching or detaching an artefact no other artefact can be attached to or detached from the given asset). Also artefacts must have certain properties in order to be attached or detached from the assets (e.g. an accident victim must be stabilised before it is loaded to an ambulance).

Despite a very general scope of the operators' definitions it can illustrate well the main aspects of the RTA domain. Clearly, we may have to consider a 'cluster'-like topology of the Road Network by introducing two 'move' operators, one for moving within a 'cluster' and the other for moving between different 'clusters'. 'Attaching' and 'detaching' artefacts to/from assets must reflect different kinds of artefacts or assets. For instance, accident victims can be 'attached' to ambulances or hospitals, in other words the victim is loaded into the ambulance or is delivered into the hospital. 'Interacting' between assets and/or artefacts captures situations such as giving first aid to accident victims (an ambulance must be at the accident scene), certifying an accident by police (a police car must be at the accident scene), or untrapping accident victims by a fire brigade.

The RTA domain deals with a situation which arises immediately after a traffic accident has been reported. Police must certify and secure the accident scene. Fire brigades must free accident victims trapped in a vehicle, and fire brigades must extinguish any fire at the accident scene. Once victims are released and free of wreckage, paramedics must give first aid to them, and then load them into ambulances and deliver them to hospitals. Tow trucks then deliver damaged cars from the accident scene to garages.

3 The Knowledge Engineering Methods

Having analysed the RTA domain, and conceptualised the requirements, we set up the experiment to evaluate two methods of planning domain knowledge formulation. As with any exercise on evaluating methods, there are problems to do with the effect of human factors such as level of method expertise of the knowledge engineers (so-called extraneous variables). Each method was carried out in parallel over a period of time. There was no time limit fixed a priori. Each team was composed of two experts. All the experts were involved in the requirements phase. The background of all participating in the teams was that all except one (who is in the final year of an AI Planning PhD) had a PhD in AI Planning. The expertise level of the PDDL encoders (method A) was expert, whereas for method B the team leader was competent rather than expert in the use of the itSIMPLE tool.

The aim of the experiment was to evaluate two well known approaches to formulate domain models and problem encodings, within the RTA domain, where the problem was to generate management plans for a particular scenario. The criteria for evaluation are arranged in two broad categories, using inspiration from the software engineering area:

- the process of the formulation: this is the encoding of the conceptualised knowledge taken from the initial domain analysis, source documents and expertise, until it reaches a final form in which it can be input to AI Planning engine(s). Features such as defect identification and removal, the nature of testing, and repeatability/traceability of the process are considered here.
- the product of the formulation: this is the domain model and problem files, with features such as maintainability, size, complexity and operationality (the latter based on the performance and quality of plans produced and range of problems solved where they are consequences of the model rather than the planner).

As an application scenario, we have chosen a region shown in Figure 1, consisting of an area within the UK. For evaluating the methods we used a set of test instances, considering the map shown in Figure 1, in which three accidents happen in Ainley Top, Greetland and Baliff Bridge. Police are required to confirm accidents, number of victims and vehicles involved. The victims are required to be taken as soon as possible to one of the hospitals, and involved broken vehicles are required to be removed from the road and taken to an available garage. Three ambulances, four police cars, two fire brigades and four tow trucks are available. We created *test instances* involving from six to one hundred victims, and from

five to thirty cars, where some victims might be trapped inside cars. We also considered an instance in which the number of available emergency vehicles is doubled, to evaluate the coordination that the different methods encodings are able to achieve. In each case, the formulation proceeded until the method produces a planning application which solves the test instances of accident management as specified above.

We overview each method before we use it, but given space restrictions the reader is encouraged to consult the literature for full details.

3.1 Method A: Hand Encoding

PDDL [McDermott, 1998; Ghallab *et al.*, 1998] is an action-based domain definition language which is inspired by STRIPS [Fikes and Nilsson, 1971] style planning. The core of the PDDL formalism is for expressing the semantics of domain actions, using pre- and post- conditions to describe the applicability and effects of actions. In this method the RTA model was hand encoded by a team composed of two PDDL experts, using a text editor and relying on dynamic testing. In the ad-hoc method of generate and test, the experts iterate over the following steps: (i) encode requirements, (ii) run a set of planners on several easy problems, (iii) evaluate the resulting plans (if any), and (iv) in the case of strange plans (in relation to the RTA domain requirements) find a way to fix the issue. Usually an expert has to iterate several times through the steps above before plans are produced that match the requirements.

3.2 Method B: itSIMPLE

The main goal of GUI tools is to provide knowledge engineers with a systematic way to reduce modelling time and errors. There are a number of tools available in the research community, such as JABBAH [González-Ferrer *et al.*, 2009] and GIPO [Simpson *et al.*, 2007]. itSIMPLE [Vaquero *et al.*, 2007; 2012] is a method and tools environment that enables knowledge engineers to model a planning domain using the UML standards. The main function of itSIMPLE is to take UML's Object Constraint Language as input through state machine diagrams, and translate them into PDDL.

4 Evaluation of the methods

4.1 Execution

In method A, first the PDDL experts manually coded the RTA domain using PDDL and PDDL2.1 [Fox and Long, 2001], and utilised standard planners such as LPG [Gerevini *et al.*, 2003], and SGPlan [Chen *et al.*, 2006]. The experts translated the informal description of the requirements directly into PDDL, without developing any intermediate notation, using their skill and judgement to perform the encoding. After approximately ten iterations of step (i) and (ii), simple plans were generated which matched requirements. At the next step, longer plans were generated for more complex problems; in this case, unusual behaviour was noticed (e.g., a single ambulance was able to carry 10s of victims) and six more cycles of debugging resulted in plans which solved the test instances.

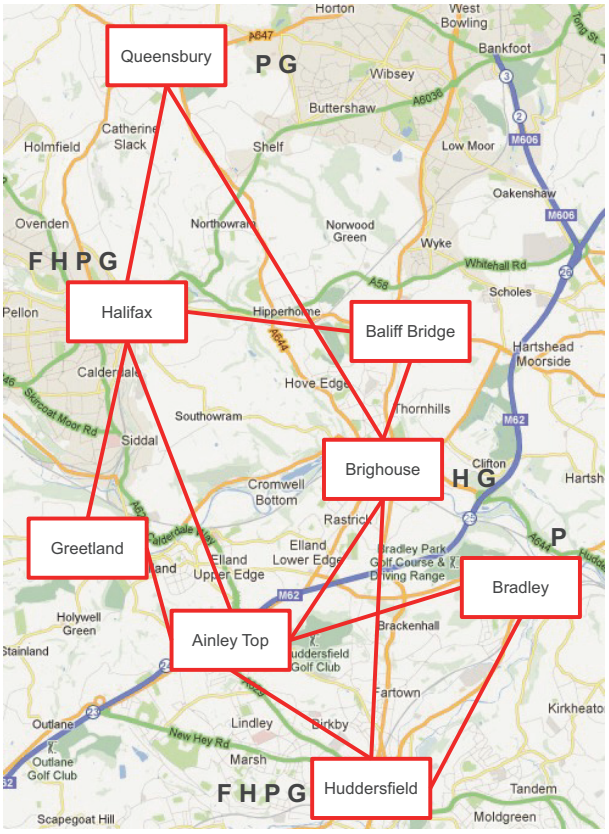


Figure 1: The Road Traffic Domain Model used for empirical analysis. It consists of a portion of the county of West Yorkshire. H, F, P and G respectively stand for Hospital, Fire station, Police station and Garage locations.

The steps in the method B, in general, follow the use of UML in software engineering: (i) design of class diagrams; (ii) definition of state machines; (iii) translation to PDDL; (iv) generation of problem files. A user formulates the requirements by designing several UML diagrams, and automatically generates the corresponding PDDL (or PDDL2.1) domain encoding. While the parameters of the operators and their duration are defined in step (i), in step (ii) the state machine diagrams help the domain modeller to encode pre- and post- condition of operators. In step (iii), we used itSIMPLE to generate both PDDL and PDDL2.1, although it does not cover all the spectrum of timing constraints expressible in PDDL [Vaquero *et al.*, 2012]. The generation of PDDL problem files was done by instantiating objects represented by the previously defined classes, describing their properties in the initial state, and describing the desired properties of objects at the goal state.

The amount of resource to perform the encoding was similar for both the methods. They took approximately 1 person week.

4.2 Process Comparison

Regarding method A, the main issue related with hand coding a real world domain in PDDL is that it tends to be ad-hoc, without the direction or static checks that a tool supported method would impose. The encoding is left to the skill and judgement of the experts that are working on it. This lack of structure leads to domain models that, even if representing the same real world application, are different and hard to understand if developed and maintained by different experts. Moreover, the process of this method is difficult to replicate while everything is left to the sensitivity and to the knowledge of experts. Since the itSIMPLE tool is designed for supporting a disciplined design cycle and for supporting the transition of requirements to formal specifications, the process is more clearly defined and not difficult to repeat.

With respect to bug identification and removal, in the hand-coded method, all but syntactic bugs were dealt with by dynamic testing of the model. Most of the development time was spent in dynamic testing: analysing produced plans, identifying bugs and removing them from the model. While hand encoding a domain, usually many issues are noticed only by carefully reading the generated plans. One example of bug identification is when the team of method A noticed broken vehicles were being delivered to hospitals, instead of garages. Removing the bug in this case amounted to adding further constraints to the operators. On the other hand, most of the time spent with itSIMPLE was in designing classes of objects and defining legal interactions between them. After that, only a relatively short time is required for debugging. However, we found that where debugging was required, it was initiated through dynamic testing: while the structure of the model helps in its development and maintenance, it is the failure of a planning engine to solve a goal which alerts the developer to the presence of a bug, in most cases. This is perhaps a failing peculiar to itSIMPLE, as there are systems with stronger static tests (such as GIPO [Simpson *et al.*, 2007]) which are capable of identifying bugs at an earlier stage than dynamic testing. The need for static tests is reduced, however, given the structure imposed by the UML method; additionally this helps determine the completeness of the model, in terms of classes and finite state machines. Also, itSIMPLE's automated generation of the PDDL model, much like compilation of a high level language into a low level language, has the benefit of eliminating human errors in encoding details. The tool offers the modellers a range of third party planners for generating plans, along with features such as plan analysis, where the plan generated can be viewed graphically.

There are advantages in using hand-coding over using tool supported environments: the development of environments tends to lag behind in the use of expressive modelling languages. itSIMPLE, although being continuously developed, has some limitations in the type of PDDL that it can generate. Also, some details such as parameter associations and metrics are only possible to encode using dialog boxes within GUI-based tools, which hamper their ease of use.

4.3 Product Comparison

For comparing the domain models generated by methods A and B, we selected a subset of the metrics suggested by Roberts and Howe in [Roberts and Howe, 2007]. In this work they described some techniques for predicting the performance of domain-independent planners by evaluating a set of metrics related to both the domain model and the planning problem. Since we are comparing planning domain models for understanding their quality, which depends also on the performance of the planners that will solve the problems, such a set of metrics could give some interesting insights. We considered also the number of lines, which could give a very intuitive idea of the complexity of the models. The results of this comparison are shown in Table 1, metrics considered are the number of types, predicates and operators, the mean number of parameters per operator, the mean number of pos/neg preconditions and the mean number of pos/neg effects.

Metric	**PDDL**		**PDDL2.1**	
	A	**B**	**A**	**B**
# types	19	22	19	22
# predicates	16	18	16	16
# operators	12	14	12	12
mean parameters	3.1	3.2	3.1	3.2
mean precond+	4.3	3.8	4.3	4.0
mean precond-	0.2	0.4	0.2	0.4
mean eff+	1.7	1.4	1.7	1.6
mean eff-	1.9	1.2	1.9	1.3
# lines	225	263	248	259

Table 1: The values of the metrics selected for comparing the domain models generated by methods A and B.

We found that the iterative process in Method A led to an over-constrained domain encoding. Many of the constraints, added in the form of pre- and/or post- conditions, were built up incrementally during de-bugging in an ad-hoc fashion, in order to avoid unwanted behaviours. The resulting model is complex and hard to read and understand compared to the model developed using method B. That method A leads to a constrained model is confirmed by the higher mean number of positive preconditions and effects. This is not noticeable by the number of lines of the files because method B invites to use many different types, as usual in KE approaches, which are not listed in a very compact way. The PDDL domain model generated by method B has 2 more actions than the method A one; these operators are related to the untrapping people and extinguish fire tasks and are used for avoiding that the same fire brigade extinguishes several fires or untraps several people at the same time. The method A model exploits a "trick": the PDDL experts added the same predicate as a positive and negative effect of the operator, which avoids the simultaneous execution of actions instantiated with similar parameters. Although these kind of tricks are commonly used by experts, their impact on the performance of the planners have not been studied, and moreover they make the domain model harder to read and understand.

We observed that the structured and principled process of encoding the requirements in Method B led to domain encodings that are clear and easy to understand. Moreover, we found that the UML documentation is useful in maintenance, as it helps trace the encoding to the initial requirements. The main difference between PDDL and PDDL2.1 encodings is

LPG						
Instance	**CPU time**		**# Actions**		**Duration**	
	A	**B**	**A**	**B**	**A**	**B**
6P, 5V, 1T	0.03	0.03	97	90	28	29
30P, 10V, 2T, 1F	0.5	0.3	318	317	90	108
100P, 30V, 5T, 3F	35.6	22.8	1015	1001	350	311
100P, 30V, 5T, 3F *	62.4	37.9	1033	988	254	246
SGPLAN						
Instance	**CPU time**		**# Actions**		**Duration**	
	A	**B**	**A**	**B**	**A**	**B**
6P, 5V, 1T	0.12	0.09	95	96	95	96
30P, 10V, 2T, 1F	0.36	0.59	324	338	324	338
100P, 30V, 5T, 3F	1.25	1.50	998	1018	998	1018
100P, 30V, 5T, 3F *	2.85	3.60	993	1068	993	1068

LPG						
Instance	**CPU time**		**# Actions**		**Duration**	
	A	**B**	**A**	**B**	**A**	**B**
6P, 5V, 1T	0.04	0.03	94	91	82	76
30P, 10V, 2T, 1F	0.7	0.3	317	321	198	220
100P, 30V, 5T, 3F	57.8	25.2	1000	1012	530	526
100P, 30V, 5T, 3F *	86.0	42.1	1025	1029	315	330
SGPLAN						
Instance	**CPU time**		**# Actions**		**Duration**	
	A	**B**	**A**	**B**	**A**	**B**
6P, 5V, 1T	0.11	0.13	88	97	117	141
30P, 10V, 2T, 1F	0.48	0.54	331	332	528	415
100P, 30V, 5T, 3F	1.45	1.81	1006	1048	1634	1115
100P, 30V, 5T, 3F *	3.64	4.40	1000	1062	1543	1322

Table 2: For every instance, the CPU time (seconds), the number of actions and the duration of plans generated by LPG and SGPlan on domains encoded using methods A and B. The upper table refers to PDDL encodings, the lower to PDDL2.1. Instances are described by the number of victims (P), the number of vehicles involved (V), the number of victims trapped (T) and the number of cars on fire (F); * indicates that the number of available emergency vehicles is doubled.

that PDDL plans, since actions are instantly completed, are a very compact version of the PDDL2.1 ones. The simpler encoding is not very realistic, however, as emergency vehicles are used without taking into account their distance from the accident location, since distance cannot be described in the simpler encoding.

To compare the operationality of the products, we investigated the performance achieved by planning systems on the models generated exploiting by methods A and B. We ran LPG and SGPlan on a set of test instances using the different models. We selected them due to their ability to handle durative actions and negative preconditions, which are both used in the generated domains, because they are readily available and performed well at IPCs. The results of the experiment are shown in Table 2 in terms of CPU time, number of actions and plans duration.

These results indicate that LPG with the hand written domain models needs more CPU time, both in PDDL and PDDL2.1, than with the models generated through method B. In the number of actions and duration of the plans there are no significant differences. The performance of LPG while exploiting method B models is very interesting; for generating good quality solutions, it requires significantly less CPU time.

On the other hand, SGPlan displays a very different performance profile compared to LPG. In this case, the domains encoded by method B slow down the plan generation process, but method B encodings lead to plans with significantly shorter makespan when generated by LPG. While SGPlan is faster than LPG at plan generation, it was not effective at exploiting the parallelisation of actions in solution plans, which unlike in LPG's performance, resulted in plans with a high makespan.

5 Conclusions and Future Work

In this paper we have developed requirements for a new planning domain, the RTA domain, addressing the problem of managing emergency situations in road traffic accidents. We have elicited a set of requirements, and used domain analysis to make precise and unambiguous relevant features for the planning problem. We then described two methods used for formulating requirements into domain models, and set up an evaluation experiment where they were used to design and create RTA domain models. Special attention was given to knowledge engineering aspects such as how long it takes to create a model or which tools can be used to verify the model. We observed that creating different models does not take very different amounts of time (taking into account the developers' expertise). We also noticed that most of the existing domain-independent planners do not support many features required for modelling real world situations: i.e., negative preconditions and durative actions. This is, clearly, a big limitation for their application. The main outcome from our work to feed back to tools developers is to provide facilities to couple planning engines and formulation tools (see [Shah *et al.*, 2013] for more details of such lessons learned).

As for the comparison, we can conclude that using itSIMPLE (method B) achieved superior results in both process and product metrics. From the process point of view, method B is easier to replicate and does not require high expertise in planning languages. From the product point of view, models are clearer to read, understand, and easier to maintain using method B. Moreover, the domain model produced with it led to a better performance from both the selected planners, even if on different metrics: LPG is significantly faster in plan generation, and SGPlan generates better quality plans, using method B's domain model.

Given the fact that different planners exploit different search techniques, they could have very different performance on the same domain encoding, as shown in our experimental analysis. The strategy that we suggest, that is derived from the experience gathered in this work, is (i) to define a metric to be optimized, (ii) select a (set of) planner(s) which handle the required features, (iii) test the planners on some easy instances, and (iv) selecting the planners, or the set of planners, which achieves the best results w.r.t. the predefined metric.

Future work will involve a simulation framework for eval-

uating plan execution, where we can couple model design and plan generation more tightly. This may reveal opportunities for improving domain models in general, and the RTA model in particular. We are also interested in simulating more complex road accidents, with blocked roads or accidents occurring in locations difficult to reach (e.g. on narrow roads). Moreover, we should consider more expressive approaches, for instance, PDDL+ [Howey *et al.*, 2004], capturing features of continuous planning since it might produce more robust system working in real-time and be able to react to unexpected events. Another interesting area might be to compare our centralised approach to using a multi-agent approach which moves the problem from centralized to a distributed point of view.

References

[Barreiro *et al.*, 2012] J. Barreiro, M. Boyce, M. Do, J. Frank, M. Iatauro, T. Kichkayloz, P. Morrisy, J. Ong, E. Remolina, T. Smith, and D. Smith. EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization. In *Proceedings of the 22nd International Conference on Automated Planning & Scheduling (ICAPS-12) – The 4th International Competition on Knowledge Engineering for Planning and Scheduling*, 2012.

[Barták *et al.*, 2010] R. Barták, S. Fratini, and T.L. McCluskey. The third competition on knowledge engineering for planning and scheduling. *AI Magazine*, 31:95 – 98, 2010.

[Benesch, 2011] Benesch. *Traffic Incident Management Operations Guidelines*. Iowa Department of Transportation, Faderal Highway Administration, 1200 New jersey Avenue, SE, Washington, DC 20590, March 24 2011.

[Chen *et al.*, 2006] Y. Chen, B.W. Wah, and C. Hsu. Temporal planning using subgoal partitioning and resolution in SGPlan. *Journal of Artificial Intelligence Research (JAIR)*, 26:323–369, 2006.

[Fikes and Nilsson, 1971] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Fox and Long, 2001] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains . In *Technical Report, Dept of Computer Science, University of Durham*, 2001.

[Gerevini *et al.*, 2003] A. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research (JAIR)*, 20:239 – 290, 2003.

[Ghallab *et al.*, 1998] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[González-Ferrer *et al.*, 2009] A. González-Ferrer, J. Fernández-Olivares, and L. Castillo. JABBAH: A java application framework for the translation between business process models and htn. In *Working notes of the 19th International Conference on Automated Planning & Scheduling (ICAPS-09) – Proceedings of the 3rd International Competition on Knowledge Engireeng for Planning and Scheduling(ICKEPS)*, pages 28–37, 2009.

[HA, 2009] HA. *Highways Agency Network Management Manual*. Highways Agency, Network Services, Network Management Policy Team, City Tower, Piccadilly Plaza, MANCHESTER M1 4BE, 5.10 edition, July 2009.

[Howey *et al.*, 2004] R. Howey, D. Long, and M. Fox. Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *Proceedings of the Sixteenth International Conference on Tools with Artificial Intelligence*, pages 294 – 301, 2004.

[McDermott, 1998] J. McDermott. 1998 AIPS planning competiton. ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html, 1998.

[Owens *et al.*, 2000] N. Owens, A. Armstrong, P. Sullivan, C. Mitchell, D. Newton, R. Brewster, and T. Trego. Traffic Incident Management Handbook. Technical Report Office of Travel Management, Federal Highway Administration, 2000.

[Roberts and Howe, 2007] M. Roberts and A. Howe. Learned models of performance for many planners. In *Proceedings of the ICAPS-07 Workshop of AI Planning and Learning (PAL)*, 2007.

[Shah *et al.*, 2013] M.M. Shah, L. Chrpa, F. Jimoh, D. Kitchin, T.L. McCluskey, S. Parkinson, and M. Vallati. Knowledge engineering tools in planning: State-of-the-art and future challenges. In *Submitted to the Knowledge Engineering for Planning and Scheduling workshop – The 23rd International Conference on Automated Planning & Scheduling (ICAPS-13)*, 2013.

[Simpson *et al.*, 2007] R.M. Simpson, Diane E. Kitchin, and T.L. McCluskey. Planning domain definition using gipo. *Knowledge Engineering Review*, 22(2):117–134, June 2007.

[Vaquero *et al.*, 2007] T.S. Vaquero, V. Romero, F. Tonidandel, and J.R. Silva. itSIMPLE2.0: An integrated tool for designing planning domains. In *Proceedings of the 17th International Conference on Automated Planning & Scheduling (ICAPS-07)*, pages 336–343. AAAI Press, 2007.

[Vaquero *et al.*, 2011] T.S. Vaquero, J.R. Silva, and J.C. Beck. A brief review of tools and methods for knowledge engineering for planning & scheduling. In *Proceedings of the Knowledge Engineering for Planning and Scheduling workshop – The 21th International Conference on Automated Planning & Scheduling (ICAPS-11)*, 2011.

[Vaquero *et al.*, 2012] T.S. Vaquero, R. Tonaco, G. Costa, F. Tonidandel, J.R. Silva, and J.C. Beck. itSIMPLE4.0: Enhancing the modeling experience of planning problems. In *System Demonstration – Proceedings of the 22nd International Conference on Automated Planning & Scheduling (ICAPS-12)*, 2012.

A.3 MUM: A technique for maximising the utility of macro-operators by constrained generation and use

L. Chrpa, M. Vallati, and T. L. McCluskey. MUM: A technique for maximising the utility of macro-operators by constrained generation and use. In *Proceedings of the International Conference on Automated Planning and Scheduling, ICAPS*, pages 65–73, 2014.

Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling

MUM: A Technique for Maximising the Utility of Macro-Operators by Constrained Generation and Use

Lukáš Chrpa and **Mauro Vallati** and **Thomas Leo McCluskey**
PARK Research Group
School of Computing and Engineering
University of Huddersfield
{l.chrpa, m.vallati, t.l.mccluskey}@hud.ac.uk

Abstract

Research into techniques that reformulate problems to make general solvers more efficiently derive solutions has attracted much attention, in particular when the reformulation process is to some degree solver and domain independent. There are major challenges to overcome when applying such techniques to automated planning, however: reformulation methods such as adding macro-operators (macros, for short) can be detrimental because they tend to increase branching factors during solution search, while other methods such as learning entanglements can limit a planner's space of potentially solvable problems (its coverage) through over-pruning. These techniques may therefore work well with some domain-problem-planner combinations, but work poorly with others.

In this paper we introduce a new learning technique (MUM) for synthesising macros from training example plans in order to improve the speed and coverage of domain independent automated planning engines. MUM embodies domain independent constraints for selecting macro candidates, for generating macros, and for limiting the size of the grounding set of learned macros, therefore maximising the utility of used macros. Our empirical results with IPC benchmark domains and a range of state of the art planners demonstrate the advance that MUM makes to the increased coverage and efficiency of the planners. Comparisons with a previous leading macro learning mechanism further demonstrate MUM's capability.

Introduction

A fundamental problem solving technique is to reformulate a problem to make it easier to solve. In automated planning, where solution generation is known to be hard in general, techniques that reformulate planning domains have the potential to increase the speed of solution plan generation, and increase coverage, that is the number of planning problems that can be solved within some resource constraint. Where the reformulation involves encoding knowledge directly into the same language in which the problem definition is encoded, then planning engines do not need to be modified in order to exploit them.

Macro-operator generation is a well known technique for encapsulating sequences of original operators, so that they can be stored and used in future planning problems. Macro-operators (macros, for short) can be encoded in the same format as original operators and therefore can be used to reformulate planning problem definitions. The idea of using macros in planning dates back to 1970s where, for example, it was applied in STRIPS (Fikes and Nilsson 1971) and REFLECT (Dawson and Siklóssy 1977). More recently, systems such as MacroFF CA-ED version (Botea et al. 2005) or WIZARD (Newton et al. 2007) are able to extract macros and reformulate the original domain model, such that standard planning engines can exploit them.

On the other hand, macros can also be exploited by specifically enhanced algorithms. This is the case for MacroFF SOL-EP version (Botea et al. 2005) which is able to exploit offline extracted and ranked macros, and Marvin (Coles, Fox, and Smith 2007) that generates macros online by combining sequences of actions previously used for escaping plateaus. Such systems can efficiently deal with drawbacks of specific planning engines, in this case the FF planner (Hoffmann and Nebel 2001), however, their adaptability for different planning engines might be low.

Another type of additional knowledge that can be encoded into domain/problem definitions and has been exploited in classical planning are entanglements, relations between planning operators and predicates (Chrpa and McCluskey 2012). Outer entanglements (Chrpa and Barták 2009), one of the types of entanglements, capture causal relations between planning operators and initial or goal predicates which are used to prune some unpromising instances of planning operators. Deciding outer entanglements is, however, PSPACE-complete in general (Chrpa, McCluskey, and Osborne 2012), therefore they are extracted by an approximation algorithm which generally does not preserve completeness (solvability might be lost if incorrect entanglements are applied).

In this paper we introduce MUM, a new learning technique for synthesising macros from training examples in order to improve the speed and coverage of domain independent planning engines which input encodings in classical PDDL. MUM utilises constraints which are created through the generation of outer entanglements, then uses these entanglements for selecting macro candidates, for generating macros, and for limiting the size of the ground-

ing set of learned macros. There have been approaches which utilise macros and outer entanglements independently (Chrpa 2010a), however, in contrast to this, MUM is designed to directly exploit knowledge related to outer entanglements throughout the process of macro learning and use, thus maximising their utility. Also, MUM preserves completeness since outer entanglements are applied only on generated macros, so the original operators remain intact. We present an empirical evaluation on IPC benchmarks (from the IPC-7 learning track) using a range of 6 planners and 9 domains. Our empirical results demonstrate the advance that MUM makes to the increased coverage and efficiency of the planners, and this improvement is apparent across planners and domains. Comparisons with a previous leading macro learning mechanism called WIZARD (Newton et al. 2007), further demonstrate MUMs capability.

Background and Related Work

Classical planning (in state space) deals with finding a sequence of deterministic actions to transform the fully observable environment from some initial state to a desired goal state (Ghallab, Nau, and Traverso 2004).

In the set-theoretic representation *atoms*, which describe the environment, are propositions. *States* are defined as sets of propositions. *Actions* are specified via sets of atoms defining their preconditions, negative and positive effects (i.e., $a = (pre(a), eff^-(a), eff^+(a))$). An action a is *applicable* in a state s if and only if $pre(a) \subseteq s$. Application of a in s (if possible) results in a state $(s \setminus eff^-(a)) \cup eff^+(a)$.

In the classical representation atoms are predicates. A *planning operator* $o = (name(o), pre(o), eff^-(o), eff^+(o))$ is a generalised action (i.e. an action is a grounded instance of the operator), where $name(o) = op\_name(x_1, \ldots, x_k)$ (*op_name* is an unique operator name and $x_1, \ldots x_k$ are variable symbols (arguments) appearing in the operator) and $pre(o), eff^-(o)$ and $eff^+(o)$ are sets of (ungrounded) predicates with variables taken only from $x_1, \ldots x_k$.

A *planning domain model* is specified by a set of predicates and a set of planning operators. A *planning problem definition* is specified via a domain model, initial state and set of goal atoms. A *plan* is a sequence of actions. A plan is a *solution* of a planning problem if and only if a consecutive application of the actions in the plan (starting in the initial state) results in a state, where all the goal atoms are satisfied. An important class of predicates in this paper are static predicates. For a given domain model, a predicate is called *static* if it is not present in the effects of any operator.

Macro-operators

Macro learning and use has been studied for several decades (Dawson and Siklóssy 1977; Korf 1985; Botea et al. 2005; Fikes and Nilsson 1971). Here we are concerned with macros that are encoded in the same way as ordinary planning operators but encapsulate sequences of planning operators. This gives the technique the potential of being *planner independent* as well as being domain independent. In the well known BlocksWorld domain (Slaney and Thiébaux 2001), we can observe, for instance, that the operator unstack(?x,?y) is often followed by the operator putdown(?x). Hence, it might be reasonable to create a macro unstack-putdown(?x,?y) which moves a block ?x from top of a block ?y directly to the table, bypassing the situation where the block ?x is held by the robotic hand. Formally, a macro $o_{i,j}$ is constructed by assembling planning operators o_i and o_j (in that order) in the following way:

- $pre(o_{i,j}) = pre(o_i) \cup (pre(o_j) \setminus eff^+(o_i))$
- $eff^-(o_{i,j}) = (eff^-(o_i) \setminus eff^+(o_j)) \cup eff^-(o_j)$
- $eff^+(o_{i,j}) = (eff^+(o_i) \setminus eff^-(o_j)) \cup eff^+(o_j)$

For a macro to be sound o_i must not delete any predicate required by o_j. If soundness is violated then corresponding instances of o_i and o_j cannot be applied consecutively.

Longer macros, i.e., those encapsulating longer sequences of original planning operators can be constructed by this approach iteratively.

Macros can be understood as 'shortcuts' in the state space. This property can be useful since by exploiting them it is possible to reach the goals in fewer steps. However, the number of instances of macros is often higher than the number of instances of the original operators, because they usually have a large set of parameters, that derives from the parameters of the operators that are encapsulated. This increases the branching factor in the search space, which can slow down the planning process and, moreover, increase the memory consumption. Therefore, it is important that benefits of macros outweigh their drawbacks. This problem is known as the *utility problem* (Minton 1988).

Learning Macros

Macro learning techniques often follow the following rules, when a macro $o_{i,j}$ is being created from operator o_i and o_j (in that order):

1) o_i adds a predicate needed in the preconditions of o_j
2) $o_{i,j}$ is not complex
3) the number of learned macros is small

Rule 1) refers to a sort of coherency between o_i and o_j. Theoretically, it is possible to generate a macro from independent operators, however, in practice it is not a very useful approach because then the number of possible instances of the macro can be very high and, moreover, there may not be a clear motivation for executing such independent operators in a specific sequence. Rules 2) and 3) ameliorate the utility problem: complex macros can have many groundings which are likely to introduce overheads which outweigh the benefits of macro use, and similarly, generation of many macros may bring the same shortcomings. Recent related work confirms that the better option is to generate a few and shorter macros rather than many or longer macros. For macros which are generated to have the same format as original operators, and thus are potentially planner independent, relevant state-of-the-art systems include MacroFF (Botea et al. 2005), Wizard (Newton et al. 2007) and the system developed by Chrpa (2010b). The CA-ED version of MacroFF uses a component abstraction technique for learning macros.

In brief, abstract components are sets of static predicates referring to 'localities'. Static predicates provide a kind of matching between objects of different or the same types. An abstract component consists of such static predicates that, informally, can group objects into a single component. For instance, in the Depots domain, each hoist is placed at some location. Hence, at(?hoist, ?place) can form an abstract component since a hoist can be only at one place, in other words, we can have a mapping from the set of hoists to the set of places (locations). On the other hand, supports(?camera, ?mode) cannot form an abstract component since a camera might support more than one mode as well as a specific mode can be supported by more than one different cameras. So, we cannot have any mapping from the set of cameras to the set of modes (or the other way round). Abstract components are used to check the *locality rule* of a generated macro, that is, whether static predicates in a macro's precondition belong to the same abstract component. By pruning macros containing cycles and limiting numbers of arguments or predicates in macro preconditions, MacroFF is able to eliminate complex macros. Limiting the number of learned macros is done by selecting the n most used macros in training plans (which are solutions of simple problems).

In Chrpa's approach, macros are learned from training plans by considering both adjacent actions, and non-adjacent actions which can be made adjacent by permutating the training plans (clearly the permutations considered must preserve the soundness of the plan). Macros are generated according to several criteria such as whether instances of one operator frequently follows (or precedes) instances of the other operator, and whether the number of the macro's arguments is small. No limit on how many macros can be generated is given a priori, but the priority is given to macros that could replace some original operators. Removing original operators is, however, incomplete in general, although it has been empirically shown on IPC benchmarks that solvability is lost very rarely (Chrpa 2010b). Whereas IPC benchmarks differ by number of objects, initial and goal situations often are found to be similar in structure.

Wizard is based on an evolutionary method carried out in a training phase, which computes macros by combining operators using a genetic algorithm. This approach allows Wizard to generate macros that refer to sequences of actions that do not appear in the considered training plans. Generated macros are then cross-validated on training problems and the fitness of the macros, i.e. their usefulness, is determined according to the performance of planners. Although macros generated by Wizard have shown to have good quality, learning time is often very high (typically tens of hours).

Outer Entanglements

Outer Entanglements are relations between planning operators and initial or goal predicates, and have been introduced as a tool for eliminating potentially unnecessary instances of these operators (Chrpa and McCluskey 2012). In the BlocksWorld domain[1] (Slaney and Thiébaux 2001) we may observe, for example, that *unstacking* blocks only occurs from their initial positions. In this case an *entanglement by init* will capture that if a predicate onblock(a,b) is to be achieved for a corresponding instance of operator unstack(?x,?y) (unstack(a,b)), then the predicate is present in the initial state. Similarly, it may be observed that *stacking* blocks only occurs to their goal position. Then, an *entanglement by goal* will capture that a predicate onblock(b,a) achieved by a corresponding instance of operator stack(?x,?y) (stack(b,a)) is the goal one. Such an observation is depicted in Figure 1. Outer entanglements are defined as follows.

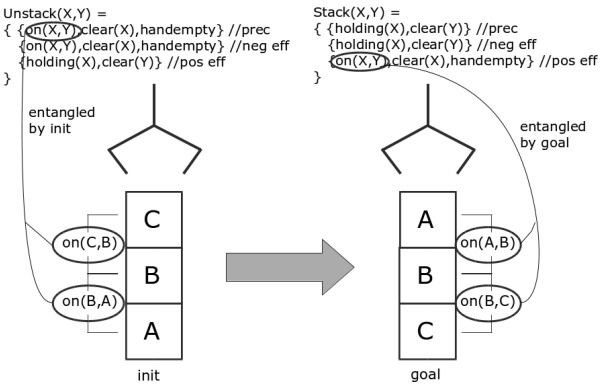


Figure 1: An illustrative example of outer entanglements.

Definition 1. *Let P be a planning problem, where I is the initial situation and G is the goal situation. Let o be a planning operator and p be a predicate (o and p are defined in the domain model of P). We say that operator o is* ***entangled by init (resp. goal)*** *with predicate p in P if and only if $p \in pre(o)$ (resp. $p \in eff^+(o)$) and there exists a plan π that is a solution of P and for every action $a \in \pi$ which is an instance of o and for every grounded instance p_{gnd} of the predicate p it holds: $p_{gnd} \in pre(a) \Rightarrow p_{gnd} \in I$ (resp. $p_{gnd} \in eff^+(a) \Rightarrow p_{gnd} \in G$).*
Henceforth, entanglements by init and goal are denoted as ***outer entanglements***. ■

Outer entanglements can be used to prune potentially unnecessary instances of planning operators. Given the example of the BlocksWorld domain (see Figure 1), we can see that since the unstack operator is entangled by init with the on predicate only the instances unstack(b,a) and unstack(c,b) follow the entanglement conditions and then we can prune the rest of unstack's instances because they are not necessary to find a solution plan. Similarly, we can see that since the stack operator is entangled by goal with the on predicate only the instances stack(a,b) and unstack(b,c) follow the entanglement conditions and then we can prune the rest of stack's instances. Usefulness of such pruning can be demonstrated in the following way. Given n blocks we can have at most $n \cdot (n-1)$ instances of stack or unstack (we do not consider instances when the block is unstacked from or stacked on itself – e.g stack(a,a)). Considering outer entanglements we can have at most $n-1$ instances of stack or unstack (we consider situations where at most one block

[1] no space restriction on the table is considered

can be stacked on the top of another block and no block can be stacked on more than one block). In summary, while in the original setting the number of instances grows quadratically with the number of blocks, considering outer entanglements reduces the instances growth to linear.

Reformulating Planning Problems by Outer Entanglements

Outer entanglements are directly encoded into a problem definition, so, similarly to the kind of macros we consider, are used as a problem reformulation technique. The way outer entanglements are encoded is inspired by one of their properties: given a static predicate p_s, an operator o is entangled by init with p_s if and only if $p_s \in pre(o)$ (Chrpa and Barták 2009). Introducing supplementary static predicates and putting them into preconditions of operators in the outer entanglement relation (both init and goal) will *filter* instances of these operators which do not follow the entanglement conditions. Formally, let P be a planning problem, I be its initial state and G its goal situation. Let an operator o be entangled by init (resp. goal) with a predicate p. Then the problem P is reformulated as follows (Chrpa and McCluskey 2012):

1. Create a static predicate p' (not already defined in the domain model of P) having the same arguments as p and add p' to the domain model of P.
2. Modify the operator o by adding p' into its precondition. p' has the same arguments as p which is in precondition (resp. positive effects) of o.
3. Add instances of p' which correspond to instances of p in I (resp. in G) into I.

Detecting Outer Entanglements

Deciding outer entanglements is PSPACE-complete in general, i.e., as hard as planning itself (Chrpa, McCluskey, and Osborne 2012). Detecting outer entanglements is thus done by using an approximation method which finds the entanglements in several training plans, solution of simpler planning problem, and assumes these entanglements hold for the whole class of planning problems defined in the same domain (e.g. IPC benchmarks) (Chrpa and McCluskey 2012). Although this approximate method may result in an incorrect assumption, it has been shown empirically using IPC benchmarks that it occurs very rarely, though whether the technique would show the same success rate in 'real-world' problems is an open question.

The method proceeds by iterating through the training plans counting how many times for an (operator, predicate) pair the outer entanglement conditions are violated. Because training plans might consist of 'flaws' (e.g. redundant actions) a *flaw ratio* η is used to allow some percentage of errors (i.e. when the entanglement conditions are violated). An entanglement between an operator and a predicate is considered true if the number of the operator's instances is non-zero and the ratio between errors and the number of the operator's instances is smaller or equal to the flaw ratio. Of course, having the flaw ratio greater than zero might result in detecting incorrect entanglements even for training problems. Hence, these entanglements must be validated on the training problems and if some of the problems become unsolvable then the flaw ratio is decreased and the method is run again.

The MUM Technique: Combining Outer Entanglements and Macros

The general idea of MUM is to utilise outer entanglements both in the macro generation and the macro use phase, in order to constrain which macros are generated, and the number of instances of macros in their use. Adding macros into a domain model does not affect completeness since macros can be understood as 'shortcuts' in the state space. Because deciding outer entanglements is intractable in general, an approximation method is used to extract them. However, there is no guarantee that all the extracted entanglements are valid. In other words, applying incorrect entanglements leads to losing solvability of some problems. If some instances of a macro are removed due to 'incorrect entanglements', completeness is not affected since the corresponding sequence of instances of original operators can be used instead. Hence, applying entanglements only on macros (and not on the original operator set) ensures completeness is preserved even in the case where some of the entanglements are incorrect.

It is of course possible to learn macros and outer entanglements separately, and use them both to reformulate planning problems as distinct techniques. While using such an approach can bring promising results (Chrpa 2010a), the problems of completeness not being preserved, or macros having too many instances, remain. In contrary to this, MUM exploits outer entanglements directly during the macro learning process, so we believe that it will lead to generating better quality macros. Following these insights, a high level design of MUM is as follows:

1) Learn outer entanglements
2) Macro Generation: learn macros by exploiting the knowledge of outer entanglements
3) Macro Use: reformulate a problem definition with learned macros and their supporting outer entanglements

One useful result we can use in step 2), is that macros can inherit outer entanglements from original operators as follows (Chrpa 2010a):

- $o_{i,j}$ is entangled by init with p iff $p \in pre(o_{i,j})$ and o_i or o_j is entangled by init with p.
- $o_{i,j}$ is entangled by goal with p iff $p \in eff^{+}(o_{i,j})$ and o_i or o_j is entangled by goal with p.

Estimating the potential number of instances of an operator

Inspired by abstract components used by MacroFF (Botea et al. 2005) to determine 'locality' of objects of different types, we propose an idea of operator *argument matching* which can be used to estimate the number of operator instances. Static predicates which have at least two arguments denote relations between objects. For example, in the well known

Depots domain, the static predicate at(?hoist,?place) provides a relation between hoists and places. In particular, a hoist can be exactly at one place. This information can be useful for estimating how many reachable instances a planning operator (or macro) can have. For example, Lift(?hoist,?crate,?surface,?place) can hypothetically have $\#hoists \cdot \#crates \cdot \#surfaces \cdot \#places$ instances. Knowing that the static predicate at(?hoist,?place) is in the precondition of Lift we can deduce that the number of Lift's instances is bounded by $\#hoists \cdot \#crates \cdot \#surfaces$ because the number of at's instances in the initial state is bounded by $\#hoists$.

Recall how outer entanglements are encoded, that is by introducing supplementary static predicates. If these introduced static predicates have at least two arguments, then they can be exploited in the same way as other static predicates in order to estimate operator instances. It holds that if $p \in pre(o)$ is static, then o is entangled by init with p (Chrpa and Barták 2009). In other words, static predicates are special cases of entanglement by init relations. If Lift(?hoist,?crate,?surface,?place) is entangled by init with a predicate on(?crate,?surface), then a static predicate on'(?crate,?surface) is created and put into Lift's precondition. For each problem, its initial state I is modified to I' by adding instances of on' such that $\forall x, y : on'(x, y) \in I' \Leftrightarrow on(x, y) \in I$.

A similar modification takes place in the case where Lift(?hoist,?crate,?surface,?place) is entangled by init with a predicate at(?crate,?place). We can observe that the number of instances of on'(?crate,?surface) as well as the number of instances of at'(?crate,?place) (a supplementary static predicate derived from at(?crate,?place)) is bounded by the number of crates ($\#crates$). Straightforwardly, at most one crate can be stacked on a given surface and a crate can be in at most one place. With the knowledge of these entanglement relations involving the Lift operator, the estimation of the number of Lift's instances can be modified to $\mathcal{O}(\#crates)$.

Static predicates and outer entanglements provide matching between arguments of planning operators. For every operator we can construct an *argument matching graph* (or a *simple argument matching graph* if only static predicates are involved) as in the following formal definition:

Definition 2. *Let $o(x_1, \ldots, x_n)$ be a planning operator where $x_1, \ldots, x_n$ are its arguments.*
*Let $G = (N, E)$ be an undirected graph such that $N = \{x_1, \ldots, x_n\}$ and $(x_i, x_j) \in E$ if and only if $x_i \neq x_j$ and there is a static predicate p such that $p \in pre(o)$ and $x_i, x_j \in args(p)$. Then, we denote G as the **simple argument matching graph** of o.*
*Let $G = (N, E)$ be an undirected graph such that $N = \{x_1, \ldots, x_n\}$ and $(x_i, x_j) \in E$ if and only if $x_i \neq x_j$ and there is a predicate p such that $x_i, x_j \in args(p)$ and o is entangled by init or goal with p. Then, we denote G as the **argument matching graph** of o.*

Henceforth, we will assume that the number of instances of predicates having at least one argument in the initial/goal state of a typical planning problem is bounded by $\mathcal{O}(n)$ (n stands for the number of objects). This assumption is made according to the observation of standard planning benchmarks and will be used as a heuristic estimation of numbers of operator or predicate instances. Of course, this assumption might not be always true, hence a predicate which does not follow the assumption does not have to be considered when constructing the (simple) argument matching graph.

Using the previous assumption, the (simple) argument matching graph of an operator o can be therefore used to estimate the number of o's instances. Let n be the number of objects defined in some planning problem, o be a planning operator and c be the number of components of the (simple) argument matching graph of o. Then, the number of o's instances is $\mathcal{O}(n^c)$. This result will be useful for estimating impact of generated macros on branching factor.

Algorithm 1 Our method for generating macros

```
1:  macros := {}
2:  repeat
3:     candidates := ComputeMacroCandidates()
4:     candidates := SortMacroCandidates(candidates)
5:     candSelected := false
6:     while not candSelected and not Empty(candidates)
       do
7:        cand := PopFirst(candidates)
8:        mcr := GenerateMacro(cand)
9:        if not Uninformative(mcr) and not Repetitive-
          ness(mcr) and AMGComponentCheck(mcr) then
10:          candSelected := true
11:       end if
12:    end while
13:    if candSelected then
14:       UpdatePlans(mcr)
15:       macros := macros ∪{mcr}
16:    end if
17: until no more macros have been generated or the no. of
    generated macros has reached a prescribed limit
18: FilterGeneratedMacros(macros)
```

Generating Macros

Algorithm 1 describes how MUM generates macros (it is the detail of step 2 in the high level design of MUM provided earlier in the text). Computing macro candidates (Line 3) is done according to Chrpa's approach (2010b). This approach considers two actions adjacent not only in the original plans but also their potential permutations. Analogously to Macro-FF, considered actions are related, i.e., one achieves a predicate (or predicates) required by the other's precondition. Plan permutations are computed according to the property of 'action independence' which allows the swapping of adjacent actions in plans following this property. We shall see later that the outer loop (Line 2) enables the possibility that generated macros are not of restricted length (that is, they can encapsulate more than two original operators).

Considering the sorting procedure in Line 4, let o_i and o_j be two operators making up a macro candidate, and $o_{i,j}$ be the macro generated by assembling o_i and o_j (in this order).

We say that an operator o has a *relational entanglement* by init (goal) if o is entangled by init (goal) with a non-static predicate p which has at least two arguments. The macro candidates are then sorted (Line 4) as follows. The macro candidates are ranked according to the following conditions.

(1) o_i has a relational entanglement by init

(2) o_j has a relational entanglement by goal

If both (1) and (2) are satisfied then the macro candidate is put to the top rank, if either (1) or (2) is satisfied then the candidate is put into the middle rank, otherwise the candidate is put into the bottom rank. If more than one candidate lies within the same rank, then those whose instances of corresponding operators can become macros (macro-actions) in the training plans more times are preferred.

The reason for this ranking is that if both (1) and (2) are satisfied, then the macro is supposed to modify the state of an object (or objects) directly from its (their) initial to its (their) goal state. Such a macro is, from our point of view, very useful. A good example is the macro Pick-Move-Drop in the Gripper domain. Similarly, if only one of (1) and (2) is satisfied, the macro is supposed to start from the initial object state or to reach the goal object state. In other words, in the Gripper domain, we prefer a potential macro Move-Drop before a potential macro Move-Pick.

Within the inner loop, the macro candidates are checked in the given order whether they meet three criteria (Line 9), i.e., whether the macro is *uninformative*, *repetitive* and its number of possible instances remains within the same order as for the original operators (or macros) the macro is assembled from (the *AMGComponentCheck*). A potential macro $o_{i,j}$ is uninformative if and only if $\mathit{eff}^+(o_{i,j}) \subseteq \mathit{pre}(o_{i,j})$. Clearly, such a macro is of no utility since its application will not create any new atom (predicate). Repetitiveness of the potential macro is determined by repeating subsequences of original operators, for example, Move-Move or Lift-Load-Lift-Load. The last check, *AMGComponentCheck*, refers to whether the potential macro will lead to a significant increase of the branching factor during search. For this, we have to construct the argument matching graphs of o_i, o_j and $o_{i,j}$. For original operators which will be intact in the reformulated domain models, the simple argument matching graphs will be considered. Let $comp(o)$ be the number of components of the (simple) argument matching graph of o (depends whether o is an original operator or a macro). The *AMGComponentCheck* succeeds if and only if $comp(o_{i,j}) \leq comp(o_i)$ or $comp(o_{i,j}) \leq comp(o_j)$. Failure of the *AMGComponentCheck* means that the number of instances of the potential macro $o_{i,j}$ can be significantly higher than the number of instances of o_i and o_j, which is undesirable.

If all the criteria (Line 9) are met, then the macro is considered and the training plans are updated by replacing the instances of macro candidates by the new macro (for details, see (Chrpa 2010b)). If no macro candidate has met all the criteria, or the number of generated macros has reached the limit, then the main loop (Lines 6–17) terminates.

Finally, the generated macros are filtered (Line 18) according to following conditions. If $comp(o_{i,j}) > comp(o_i)$ or $comp(o_{i,j}) > comp(o_j)$, then $o_{i,j}$ is removed. Note that the *AMGComponentCheck* used in the generation phase is a weaker condition than this, and allows situations where $\min(comp(o_i), comp(o_j)) < comp(o_{i,j}) \leq \max(comp(o_i), comp(o_j))$. The reason is that $o_{i,j}$ may be used as a 'building block' for a longer macro which can be very useful (for illustration, see the example below). In the case where one or both of the components of a new macro m_l is also a macro (call it m_s), it is desirable to keep at most one of these. m_l does not pass the filter test of Line 18 if $comp(m_l) > comp(m_s)$. m_l is also filtered if $comp(m_l) = comp(m_s)$ and the number of m_l's instances is not greater than the number of m_s's instances in the updated training plans. Otherwise m_l is kept and m_s is filtered out.

	FF	LAMA	LPG	Mp	Probe	Tot
Barman	–	–	–	–	–	–
Blocks	–	2	–	–	3	3
Depots	2	–	–	2	2	2
Gripper	1	1	1	1	1	1
Parking	–	–	–	–	–	–
Rovers	2	2	1	2	1	4
Satellite	1	2	1	1	2	2
Spanner	2	1	2	1	1	3
TPP	1	1	1	1	1	2

Table 1: Number of macros extracted by MUM on a given domain for a single planner and the total number of different macros per domain (Tot).

As an example to demonstrate how our method works we take the Gripper domain model (the version from the IPC-7 learning track). We identified three outer entanglements: Pick is entangled by init with at(?r,?room) and free(?r,?g) and Drop is entangled by goal with at(?r,?room). The numbers of components of the simple argument matching graphs are as follows: $comp(move) = 3$ and $comp(pick) = comp(drop) = 4$. After calculating the macro candidates we selected at first a potential macro Pick-Drop since it is the highest candidate after sorting. This macro is, however, uninformative since Pick and Drop are reversing each other's effects in this case. Then, a potential macro Move-Drop is selected. It passed all the checks, $comp(move\text{–}drop) = 4$ (we consider the non-simple argument matching graph), so the macro is considered and the training plans are updated by replacing corresponding instances of Move and Drop with the new macro Move-Drop. Iterating around the outer repeat loop of Algorithm 1, macro-candidates are recalculated using the re-represented training plans. After the candidates have been sorted, the potential macro Pick-Move-Drop is selected. It passed all the checks, where $comp(pick\text{–}move\text{–}drop) = 2$, hence the macro is considered and the training plans are updated. Furthermore, two more macros Move-Pick-Move-Drop and Pick-Move-Drop-Move-Pick-Move-Drop such that $comp(pick\text{–}move\text{–}drop\text{–}move\text{–}pick\text{–}move\text{–}drop) = comp(move\text{–}pick\text{–}move\text{–}drop) = 3$ are considered. Since the prescribed limit of considered macros was

	Solved		# Fastest		IPC score	
	O	M	O	M	O	M
FF	1	36	0	36	0.5	36.0
LAMA	39	116	5	113	30.4	115.7
LPG	99	86	89	24	97.9	67.1
Mp	8	41	2	41	6.9	41.0
Probe	80	86	20	75	63.4	84.3
Total	227	**365**	116	**289**	199.1	**344.1**

Table 2: Number of solved problems, number of problems solved faster and IPC score achieved by considered planners on all the benchmark problems, while exploiting the original formulation (O) or the formulation extended with MUM macros (M), whenever available.

4, which is reached, we proceed to the filtering stage. The macros Move-Pick-Move-Drop and Pick-Move-Drop-Move-Pick-Move-Drop are pruned since their common 'submacro' Pick-Move-Drop has fewer components of argument matching graph. The macro Pick-Move-Drop has fewer components than the original operator Pick and the macro Move-Drop. Hence, the macro Pick-Move-Drop is kept while its 'submacro' Move-Drop is pruned. After the filtering stage only Pick-Move-Drop remains which is is then added (with entanglements) into the original Gripper domain model.

Experimental Analysis

The aims of this experimental analysis were to (a) evaluate the effectiveness of the MUM system with respect to increasing the speed and coverage of plan generation over a range of domains and planning engine combinations, and (b) to compare it to a similar state-of-the-art method. For this purpose, we use the well-known benchmark instances used in the learning track of the last International Planning Competition (IPC-7) (Coles et al. 2012) that was held in 2011. Such problems are from the following domains: Barman, BlocksWorld (BW), Depots, Gripper, Parking, Rovers, Satellite, Spanner and TPP. As benchmarking planners we chose Metric-FF (Hoffmann 2003), LPG-td (Gerevini, Saetti, and Serina 2003), LAMA-11 (Richter and Westphal 2010; Richter, Westphal, and Helmert 2011), Mp (Rintanen 2012) and Probe (Lipovetzky and Geffner 2011). All the planners successfully competed in the IPCs and exploit different techniques for finding satisficing plans. A runtime cutoff of 900 CPU seconds (15 minutes, as in learning tracks of IPC) was used for both learning and testing runs. All the experiments were run on 3.0 Ghz machine CPU with 4GB of RAM. In this experimental analysis, IPC score as defined in IPC-7 are used. For a planner $\mathcal{C}$ and a problem p, $Score(\mathcal{C}, p)$ is 0 if p is unsolved, and $1/(1 + \log_{10}(T_p(\mathcal{C})/T_p^*))$, where $T_p(\mathcal{C})$ is the cpu time needed by planner $\mathcal{C}$ to solve problem p and T_p^* is the cputime needed by the best considered planner, otherwise. The IPC score on a set of problems is given by the sum of the scores achieved on each considered problem.

As the training set for learning macros for a given planner on a specific domain consists of about 5-8 simpler problems that were generated using the problem generators provided by the organisers. For a single planner and a single domain, the learning process required a couple of seconds including generating the training plans and the execution of MUM.

Results of Macro Generation: In Table 1 the number of macros generated by MUM for a specific planner, on a given domain, is shown. Generally, the results show a good spread across planners and domains. Also, the total number of macros extracted per domain is usually lower than the prescribed limit of 4. Macros are often of length of 2 or 3, occasionally 4, and in the Spanner domain one macro is of length 5. Often the plans generated by different planners have similar structure, reflecting the domain's structure, resulting in the opportunity to generate only a low number of different macros. In Barman and Parking MUM did not generate any macros. In Barman we observe that there is not a sequence of operators which is frequently used in a plan, since the ways for generating different cocktails differ quite significantly, hence the reason why no macros were generated. In Parking, no outer entanglements were extracted, therefore potential macros had more possible instances (according to the number of components of their argument matching graphs), and so were filtered out. In BW and Depots, MUM did not generate macros for some planners because the training plans produced by these planners were of low quality (they were too long) which prevented MUM extracting useful outer entanglements at the start of the process. Because of that, potential macros were not constrained enough (the number of components of the argument matching graph was high), so they were filtered out.

Results of Macro Use: Cumulative results of the evaluation are presented in Table 2, with the performances exploited in the domain model enhanced with macros, compared to the original problem formulation. Values are computed only in domains in which MUM was able to find a set of macros for the given planner. The results show that, in general, exploiting the domain model extended with the extracted macros leads to a performance improvement in terms of number of solved problems, number of problems solved faster and IPC score (that is, improved coverage and speed). The exception to this are all the domains using the LPG planner. To investigate why, we ran LPG with entanglement-supported macros generated by other planners, and observed that the performance of LPG was usually better than the ones achieved on the original domain model. Hence, we postulate this behaviour is caused by (i) low quality solutions found for training problems by LPG (ii) the use of a local search algorithm in LPG being oversensitive to even a marginal increase of the branching factor caused by added macros.

Results of Comparison with Wizard: In order to evaluate the effectiveness of MUM with regards to the related state-of-the-art techniques for planner-independent generation of macros, we compared it to Wizard (Newton et al. 2007). For training Wizard about 90 problems per domain were used, divided in three sets accordingly to their complexity (assessed by the number of involved objects). Default parameters configuration was used. For a single planner, the process of generating macros on all the considered

	Solved			# Fastest			IPC score		
Barman	O	W	M	O	W	M	O	W	M
FF	0	–	–	0	–	–	0.0	–	–
LAMA	0	–	–	0	–	–	0.0	–	–
LPG	0	–	–	0	–	–	0.0	–	–
Mp	0	0	–	0	0	–	0.0	0.0	–
Probe	1	1	–	1	0	–	**1.0**	0.9	–
BW	O	W	M	O	W	M	O	W	M
FF	0	–	–	0	–	–	0.0	–	–
LAMA	18	0	27	2	0	25	13.2	0.0	**26.8**
LPG	20	30	–	0	30	–	9.2	**30.0**	–
Mp	0	0	–	0	0	–	0.0	0.0	–
Probe	20	18	30	0	0	30	13.6	11.1	**30.0**
Depots	O	W	M	O	W	M	O	W	M
FF	1	2	4	0	0	4	0.5	1.1	**4.0**
LAMA	3	–	–	3	–	–	3.0	–	–
LPG	13	–	–	13	–	–	13.0	–	–
Mp	6	5	19	1	2	17	4.7	3.9	**18.6**
Probe	30	30	30	1	0	29	23.0	23.6	**29.9**
Gripper	O	W	M	O	W	M	O	W	M
FF	0	–	25	0	–	25	0.0	–	**25.0**
LAMA	0	–	26	0	–	26	0.0	–	**26.0**
LPG	10	–	22	10	–	12	10.0	–	**18.0**
Mp	0	0	0	0	0	0	0.0	0.0	0.0
Probe	0	–	0	0	–	0	0.0	–	0.0
Parking	O	W	M	O	W	M	O	W	M
FF	7	0	–	7	0	–	**7.0**	0.0	–
LAMA	4	–	–	4	–	–	4.0	–	–
LPG	0	–	–	0	–	–	0.0	–	–
Mp	0	–	–	0	–	–	0.0	–	–
Probe	3	–	–	3	–	–	3.0	–	–
Rovers	O	W	M	O	W	M	O	W	M
FF	0	–	0	0	–	0	0.0	–	0.0
LAMA	6	–	29	0	–	29	4.1	–	**29.0**
LPG	28	–	27	26	–	2	**27.9**	–	21.1
Mp	1	–	0	1	–	0	**1.0**	–	0.0
Probe	20	–	11	19	–	1	**20.0**	–	9.4
Satellite	O	W	M	O	W	M	O	W	M
FF	0	0	7	0	0	7	0.0	0.0	**7.0**
LAMA	2	–	18	0	–	18	1.9	–	**18.0**
LPG	30	–	9	30	–	0	**30.0**	–	4.0
Mp	0	0	0	0	0	0	0.0	0.0	0.0
Probe	0	–	0	0	–	0	0.0	–	0.0
Spanner	O	W	M	O	W	M	O	W	M
FF	0	–	0	0	–	0	0.0	–	0.0
LAMA	0	–	0	0	–	0	0.0	–	0.0
LPG	30	–	26	22	–	8	**29.2**	–	22.0
Mp	0	20	20	0	0	20	0.0	18.7	**20.0**
Probe	0	–	0	0	–	0	0.0	–	0.0
TPP	O	W	M	O	W	M	O	W	M
FF	0	–	0	0	–	0	0.0	–	0.0
LAMA	13	13	16	3	0	15	11.2	10.8	**15.9**
LPG	1	–	2	1	–	2	0.8	–	**2.0**
Mp	1	8	2	0	7	1	0.9	**8.0**	1.9
Probe	10	6	15	0	0	15	6.8	3.4	**15.0**

Table 3: Number of solved problems, number of problems solved faster and IPC score achieved by considered planners on the benchmark domains, while exploiting the original formulation (O), the formulation extended with macros extracted by Wizard (W) or the formulation extended with macros extracted by MUM (M).

domains took between two and ten cpu-time days. Table 3 shows the detailed results achieved by each planner on the considered domains, while exploiting the original domain formulation, the domain model extended with macros found by Wizard and the formulation that includes the entanglement-supported macros extracted by MUM. Since Wizard is able to provide at most three different sets of macro operators, resulting from the execution of different sets of genetic operators, in Table 3 we reported only the results achieved by the best one. In the results, Wizard is not able to generate macros in more cases (domain, planner) than our method. We also observed that wizard macros are usually 'long', in the sense that are composed by several operators. Moreover, some macros generated by Wizard seem somewhat counter-intuitive and therefore they do not seem to be very useful, although they might have some effects to, for instance, delete-relaxation based heuristics (e.g. FF). According to the results shown in Table 3, we can derive that on the considered domains: (i) MUM is able to generate macros for most of the considered planners and domains; (ii) MUM is usually able to generate macros that improve the performance of the given planner; (iii) the macros generated by MUM have a greater impact on planners' performance than Wizard's.

Although MUM uses Chrpa(2010b)'s technique of investigating action dependencies and independencies to identify candidates for becoming macros, MUM's criteria for selecting which candidates are suitable is more sophisticated, resulting in an improved set of generated macros. For example, in the Gripper domain MUM extracted a useful macro Pick-Move-Drop while Chrpa's technique did not extract any macro. Applying entanglements on macros in an ad-hoc way (Chrpa 2010a) has some drawbacks, in particular a macro learning technique may not extract any macros, or it may not be possible to apply entanglements to extracted macros. A detailed empirical study is planned for future work.

One unwelcome side-effect of the use of macros is that their exploitation could have a negative impact on the quality of solution plans. In our experimental analysis we did not observe a significant difference between the quality of plans found by exploiting the original domain formulation and the domain model extended with macros; the average number of actions of plans is 416 in the former case and 422 in the latter. Often, the difference between length of plans is within 10%. In some cases the solutions using macros were of much better quality (e.g. where the planner was LPG, and the domain was Gripper) and in other cases the solutions using macros were of worse quality (e.g. where the planner was LAMA and the domain was BW). In summary, the macros generated using MUM are able to deliver an impressive improvement in the runtime of planners without significantly decreasing the quality of solutions.

Conclusions

In this paper we presented MUM, a technique for learning macros while maximising their utility by keeping them informative, though constrained. The number of possible instantiation of macros is limited by using outer entanglements

in their generation and use. Moreover, MUM preserves completeness since only instances of macros are pruned. Macros generated by MUM generally improve the performance of planning engines by reducing the time to generate plans and increasing the number of solved problems, on the benchmark instances of the learning track of IPC-7. Also, macros generated by MUM have achieved impressive results in comparison to the macro learning system WIZARD (Newton et al. 2007) and we have evidence to show that this is not at the cost of poor quality solutions. Our experiments also give us useful insights into the current limitations of MUM: poor quality solutions used in training (as is the case with LPG) led to reduced performance, and in 2 of the 9 domains (Barman and Parking) no useful macro could be generated using our technique.

For future work, we are interested in investigating possibilities of learning problem- (or problem class-) specific macros. Inspired by work of Alhossaini and Beck (2013) which selects appropriate problem-specific macros from sets of macros learnt by existing approaches (e.g Wizard) we believe that implementing similar ideas on MUM will result in further improvements. We believe that extending our technique to support, for instance, ADL features (e.g. conditional effects) or durative actions (temporal planning), which is planned in future, will bring improvements to such kinds of planning techniques.

Acknowledgements

The research was funded by the UK EPSRC Autonomous and Intelligent Systems Programme (grant no. EP/J011991/1). The authors would like to acknowledge the use of the University of Huddersfield Queensgate Grid in carrying out this work.

References

Alhossaini, M. A., and Beck, J. C. 2013. Instance-specific remodelling of planning domains by adding macros and removing operators. In *Proceedings of SARA*, 16–24.

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-FF: improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research (JAIR)* 24:581–621.

Chrpa, L., and Barták, R. 2009. Reformulating planning problems by eliminating unpromising actions. In *Proceedings of SARA*, 50–57.

Chrpa, L., and McCluskey, T. L. 2012. On exploiting structures of classical planning problems: Generalizing entanglements. In *Proceedings of ECAI*, 240–245.

Chrpa, L.; McCluskey, T. L.; and Osborne, H. 2012. Reformulating planning problems: A theoretical point of view. In *Proceedings of FLAIRS*, 14–19.

Chrpa, L. 2010a. Combining learning techniques for classical planning: Macro-operators and entanglements. In *Proceedings of ICTAI*, volume 2, 79–86.

Chrpa, L. 2010b. Generation of macro-operators via investigation of action dependencies in plans. *Knowledge Engineering Review* 25(3):281–297.

Coles, A.; Coles, A.; Olaya, A. G.; Jiménez, S.; Lòpez, C. L.; Sanner, S.; and Yoon, S. 2012. A survey of the seventh international planning competition. *AI Magazine* 33:83–88.

Coles, A.; Fox, M.; and Smith, A. 2007. Online identification of useful macro-actions for planning. In *Proceedings of ICAPS*, 97–104.

Dawson, C., and Siklóssy, L. 1977. The role of preprocessing in problem solving systems. In *Proceedings of IJCAI*, 465–471.

Fikes, R., and Nilsson, N. J. 1971. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3/4):189–208.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research (JAIR)* 20:239 – 290.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning, theory and practice*. Morgan Kaufmann Publishers.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research (JAIR)* 14:253–302.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal Artificial Intelligence Research (JAIR)* 20:291–341.

Korf, R. 1985. Macro-operators: A weak method for learning. *Artificial Intelligence* 26(1):35–77.

Lipovetzky, N., and Geffner, H. 2011. Searching for plans with carefully designed probes. In *Proceedings of ICAPS*, 154–161.

Minton, S. 1988. Quantitative results concerning the utility of explanation-based learning. In *Proceedings of AAAI*, 564–569.

Newton, M. A. H.; Levine, J.; Fox, M.; and Long, D. 2007. Learning macro-actions for arbitrary planners and domains. In *Proceedings of ICAPS*, 256–263.

Richter, S., and Westphal, M. 2010. The LAMA planner: guiding cost-based anytime planning with landmarks. *Journal Artificial Intelligence Research (JAIR)* 39:127–177.

Richter, S.; Westphal, M.; and Helmert, M. 2011. Lama 2008 and 2011. In *Booklet of the 7th International Planning Competition*.

Rintanen, J. 2012. Engineering efficient planners with SAT. In *Proceedings of ECAI*, 684–689.

Slaney, J., and Thiébaux, S. 2001. Blocks world revisited. *Artificial Intelligence* 125(1-2):119–153.

A.4 Exploiting block deordering for improving planners efficiency

L. Chrpa and F. H. Siddiqui. Exploiting block deordering for improving planners efficiency. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, pages 1537–1543, 2015.

Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

Exploiting Block Deordering for Improving Planners Efficiency

Lukáš Chrpa
PARK Research group
School of Computing & Engineering
University of Huddersfield

Fazlul Hasan Siddiqui
NICTA Optimisation Research Group
Research School of Computer Science
The Australian National University, Australia

Abstract

Capturing and exploiting structural knowledge of planning problems has shown to be a successful strategy for making the planning process more efficient. Plans can be decomposed into its constituent coherent subplans, called *blocks*, that encapsulate some effects and preconditions, reducing interference and thus allowing more deordering of plans. According to the nature of blocks, they can be straightforwardly transformed into useful macro-operators (shortly, "macros"). Macros are well known and widely studied kind of structural knowledge because they can be easily encoded in the domain model and thus exploited by standard planning engines.

In this paper, we introduce a method, called BLOMA, that learns domain-specific macros from plans, decomposed into "macro-blocks" which are extensions of blocks, utilising structural knowledge they capture. In contrast to existing macro learning techniques, macro-blocks are often able to capture high-level activities that form a basis for useful longer macros (i.e. those consisting of more original operators). Our method is evaluated by using the IPC benchmarks with state-of-the-art planning engines, and shows considerable improvement in many cases.

1 Introduction

Capturing and exploiting structural knowledge of planning problems has shown to be a successful strategy for improving efficiency of the planning process. A well-known technique for encapsulating sequences of operators, macro-operators ("macros", for short), is a good example of such structural knowledge due to possibility to encode macros in the same format as original operators, so they can be used in a planner-independent way. Macros date back to 1970s where they were used, for example, in STRIPS [Fikes and Nilsson, 1971] and REFLECT [Dawson and Siklóssy, 1977]. Korf (1985) has shown that using macros can reduce the problem complexity in some cases which gives a good motivation for their use. Hence, many successful macro learning techniques have been developed in recent years (see the following section).

Analysis of training plans, usually solutions of simpler problems, is a key step in the macro learning process. Totally ordered plans, however, often "hide" some promising candidates for macros, since corresponding actions are not adjacent. Chrpa (2010) proposed a technique that as macro candidates considered actions non-adjacent in a given plan but adjacent in some of its permutations. Recently, a technique that decomposes plans into its constituent coherent subplans, *blocks* [Siddiqui and Haslum, 2012], was developed, and successfully applied in post-processing based plan quality optimisation [Siddiqui and Haslum, 2013]. Blocks can reduce interference between actions and thus allowing more deordering of plans, hence blocks can be exploited in the form of macros.

In this paper, we introduce *macro-blocks* that extend the blocks by considering relations between them in order to provide more promising macro candidates. In some domains, macro-blocks can capture longer subplans representing important activities. Then, we introduce a method, called BLOMA, that from macro-blocks achieved by decomposition of training plans extracts domain-specific macros. Given the property of macro-blocks, BLOMA can generate useful longer macros in problems whose structure relies on repetitive application of a larger sets of actions. Traditional macro learning techniques that are based on "operator chaining" approaches (i.e. assembling operators one by one) are often not able to find such long macros. BLOMA is evaluated by using the IPC benchmarks with state-of-the-art planning engines, and shows considerable improvement in many cases.

2 Related Work

Recent macro learning systems can be categorised as planner-independent or planner-specific. The CA-ED version of MacroFF [Botea *et al.*, 2005] generates macros according to several pre-defined rules (e.g., the "locality rule") and by exploring adjacent actions in plans. SOL-EP version of MacroFF learns macros from training plans and uses them for improving the performance of the FF planner [Hoffmann and Nebel, 2001]. Marvin [Coles *et al.*, 2007], which is built on top of FF, combines offline and online macro generating techniques in order to escape plateaus in heuristics landscape. Wizard [Newton *et al.*, 2007] learns macros by exploiting genetic programming. Alhossaini and Beck (2013) proposed a technique for efficient selection of problem-specific

macros from a set of macros learnt by some of the existing techniques. A more recent macro learning method, called MUM [Chrpa *et al.*, 2014], is based on Chrpa's (2010) method considering non-adjacent actions in training plans, and exploits outer entanglements [Chrpa and McCluskey, 2012] as a heuristics for limiting the number of potential instances of macros.

Several works go in the opposite direction. Haslum and Jonsson (2000) proposed a method that identifies and removes "redundant actions" (i.e. actions whose effects can be achieved by sequences of other actions). Areces *et al.* (2014) proposed a technique for decomposing more complex operators into simpler ones.

3 Background

AI planning deals with finding a sequence of actions transforming the environment from an initial state to a desired goal state [Ghallab *et al.*, 2004].

In the classical (STRIPS) representation the environment is described by *predicates*. *States* are defined as sets of grounded predicates. We say that $o = (\mathit{name}(o), \mathit{pre}(o), \mathit{del}(o), \mathit{add}(o))$ is a *planning operator*, where $\mathit{name}(o) = \mathit{op\_name}(x_1, \ldots, x_k)$ (*op_name* is an unique operator name and $x_1, \ldots x_k$ are variable symbols (arguments) appearing in the operator) and $\mathit{pre}(o), \mathit{del}(o)$ and $\mathit{add}(o)$ are sets of (ungrounded) predicates with variables taken only from $x_1, \ldots x_k$ representing o's precondition, delete, and add effects respectively. *Actions* are grounded instances of planning operators. An action a is *applicable* in a state s if and only if $\mathit{pre}(a) \subseteq s$. Application of a in s (if possible) results in a state $(s \setminus \mathit{del}(a)) \cup \mathit{add}(a)$.

A *planning domain model* is specified by a set of predicates and a set of planning operators. A *planning problem* is specified via a domain model, initial state and set of goal predicates. Given a planning problem, a *plan* is a sequence of actions such that their consecutive application starting in the initial state results in a state containing all the goal predicates.

3.1 Macro-operators

Macros can be encoded in the same way as ordinary planning operators, but encapsulate sequences of planning operators. This gives the technique the potential of being *planner independent*. Formally, a macro $o_{i,j}$ is constructed by assembling planning operators o_i and o_j (in that order) in the following way (o_i and o_j may share some arguments)[1]:

- $\mathit{pre}(o_{i,j}) = \mathit{pre}(o_i) \cup (\mathit{pre}(o_j) \setminus \mathit{add}(o_i))$
- $\mathit{del}(o_{i,j}) = (\mathit{del}(o_i) \setminus \mathit{add}(o_j)) \cup \mathit{del}(o_j)$
- $\mathit{add}(o_{i,j}) = (\mathit{add}(o_i) \setminus \mathit{del}(o_j)) \cup \mathit{add}(o_j)$

Macros can be understood as 'shortcuts' in the state space. This property can be useful since by exploiting them it is possible to reach the goals in fewer steps, or to escape local heuristic minima. Macros, however, have often a high number of instances and thus they might considerably increase the size of grounded problem representation and be memory demanding. Therefore, it is important that benefits of macros exploitation outweigh their drawbacks. This problem is known as the *utility problem* [Minton, 1988].

[1]Longer macros, i.e., those encapsulating longer sequences of original planning operators can be constructed by this approach iteratively.

3.2 Outer Entanglements

Outer Entanglements are relations between planning operators and initial or goal predicates, and have been introduced as a technique for eliminating potentially unnecessary instances of these operators [Chrpa and Barták, 2009; Chrpa and McCluskey, 2012]. Such a technique is especially useful for limiting the number of instances of macros [Chrpa *et al.*, 2014].

We say that an operator is *entangled by init* (resp. *entangled by goal*) with a predicate, if there exists a plan where all the operator's instances require (resp., produce) instances of the predicate that correspond to initial (resp., goal) ones.

In the BlocksWorld domain [Slaney and Thiébaux, 2001], the operator unstack is entangled by init with the predicate on, since *unstacking* blocks is necessary only from their initial positions. Similarly, the operator stack is entangled by goal with the predicate on, since *stacking* blocks is necessary only to their goal position.

Outer entanglements have been used as a reformulation technique, as they can be directly encoded into a domain and problem model. The way outer entanglements are encoded is inspired by one of their properties: given a static predicate p_s[2], an operator o is entangled by init with p_s if and only if $p_s \in \mathit{pre}(o)$ [Chrpa and Barták, 2009]. Operators involved in the outer entanglement relation with a non-static predicate are modified by putting a "static twin" of the predicate into the precondition. Instances of the "static twin" corresponding with initial or goal instances of the predicate are added into the initial state. For detailed description of the reformulation approach, see [Chrpa and McCluskey, 2012]

3.3 Plan Deordering and Block Decomposition

A *partially ordered plan* (POP) is a tuple $(\mathcal{A}, \prec)$, where $\mathcal{A}$ is the set of plan actions and $\prec$ is a strict partial order on $\mathcal{A}$. $\prec^+$ denotes the transitive closure of $\prec$. A *linearization* of $(\mathcal{A}, \prec)$ is a total ordering of the actions in $\mathcal{A}$ that respects $\prec$. A POP provides a compact representation for multiple linearizations. We assume that every ordering constraint, $a_i \prec a_j$, in $(\mathcal{A}, \prec)$ must have at least one necessary reason, denoted by $\mathrm{Re}(a_i \prec a_j)$, for not violating $a_i \prec a_j$. Violating $a_i \prec a_j$ causes an action precondition to be unsatisfied before its execution in some linearizations of $(\mathcal{A}, \prec)$. Necessary reasons (with respect to an atom m) are of four types: $\mathrm{PC}(m)$ (producer–consumer of m), $\mathrm{CD}(m)$ (consumer–deleter of m), $\mathrm{DP}(m)$ (deleter–producer of m), and $\mathrm{DK}(m)$ (deleter–knight of m). Note that an ordering constraint can have several associated reasons of the same type but referring to different atoms. $\mathrm{PC}(m) \in \mathrm{Re}(a_i \prec a_j)$ states that a_i produces an atom m that a_j consumes. This relation is usually called a *causal link* from a_i to a_j for m [McAllester and Rosenblitt, 1991], and denoted by a triple $\langle a_i, m, a_j \rangle$. A causal link $\langle a_i, m, a_j \rangle$ is threatened if there is

[2]A predicate is static if not present in effects of any operator

any possibility of m being deleted and there is no producer to reproduce it before the execution of a_j. $\mathrm{CD}(m)$ states that a_j deletes an atom m that a_i consumes. Therefore, unless a_j is ordered after a_i, m could be unsatisfied before the execution of a_i in some linearizations of $(\mathcal{A}, \prec)$. $\mathrm{DP}(m)$ states that a_i deletes an atom m that a_j produces, and that is consumed by some action a_k with $a_j \prec a_k$. Therefore, unless a_i is ordered before a_j, m could be unsatisfied before the execution of a_k in some linearizations of $(\mathcal{A}, \prec)$. Note that it is not necessary to order a producer and deleter if no step that may occur after the producer in the plan depends on the produced atom. Finally, $\mathrm{DK}(m)$ states that a_i deletes an atom m that a_j produces, and a_i threats a causal link $\langle a_x, m, a_y \rangle$ in some linearizations of $(\mathcal{A}, \prec)$ unless a_i is ordered before a_j. Hence, a_j acts as a white knight to $\langle a_x, m, a_y \rangle$.

The validity of a POP can be defined in two equivalent ways: (1) a POP is valid iff every linearisation of its actions is a valid sequential plan, under the usual STRIPS execution semantics; and (2) a POP is valid if every action precondition is supported by an unthreatened causal link.

A *block* [Siddiqui and Haslum, 2012] is a constituent coherent subplan, i.e., a subset of actions, that must not be interleaved with actions outside the block.

Definition 1. *Let $(\mathcal{A}, \prec)$ be a partially ordered plan. A **block** w.r.t. $\prec$ is a subset $b \subset \mathcal{A}$ of actions such that for any two actions $a, a' \in b$, there exists no action $a'' \in (\mathcal{A} - b)$ such that $a \prec^+ a'' \prec^+ a'$.*

Blocks, like ordinary actions, have preconditions, add, and delete effects that are a subset of the union of those of its constituent actions. This enables blocks encapsulating some effects and preconditions, reducing interference, and thus allowing more deordering of plans.

A *decomposition* of a plan into blocks is recursive, i.e., a block can be wholly contained in another. However, blocks cannot be partially overlapping. The semantics of a partially ordered block decomposed plan is defined by restricting its linearisations (for which it must be valid) to those that respect the block decomposition, i.e., that do not interleave actions from disjoint blocks.

Deordering converts a sequential plan into a POP, but the conventional deordering approach restricts the deordering to only the cases where individual actions are independent and thus non-interfering. *Block deordering* [Siddiqui and Haslum, 2012] eliminates that restriction by forming blocks which allows to remove some ordering constraints. It enables deordering in many cases where it is impossible in the standard interpretation of plans. Maximising such deordering helps to exhibit the plan structure more clearly.

The block deordering procedure [Siddiqui and Haslum, 2012] automatically finds a block decomposition of a plan that maximises deordering of a partially ordered plan. This procedure works in a check-and-remove fashion: Firstly, it checks the reasons ($\mathrm{PC}, \mathrm{CD}, \mathrm{DP}$, and DK) behind every necessary ordering $a_i \prec a_j$ within the current plan structure, and forms two blocks b_i and b_j with the initial element a_i and a_j respectively. Then the blocks gradually expand in opposite directions picking steps one after another from the plan structure until those reasons (and newly added reasons) behind the ordering no longer exist (due to the encapsulation within the blocks) or the expansion has reached the boundary. If no reason is left at the end, the ordering is removed as well, and this process is repeated until all the necessary orderings have been checked or the allotted time is up. As a simple example, Figure 1(i) shows a sequential plan for a small BlocksWorld problem. This plan cannot be deordered into a conventional POP, because each action in plan has a reason to be ordered next to another. Block deordering, however, is able to break the ordering $(a_2 \prec a_3)$ by removing the only reason PC(handempty) based on the formation of two blocks b_1 and b_2 as shown in (ii). Neither of the two blocks delete or add the atom "handempty" (though it is a precondition of both). This removes the interference between them, and allows the two blocks to be executed in any order but not in an iterleaving way. Therefore, the possible linearisations of the block decomposed partially ordered plan are only (a_1, a_2, a_3, a_4) and (a_3, a_4, a_1, a_2).

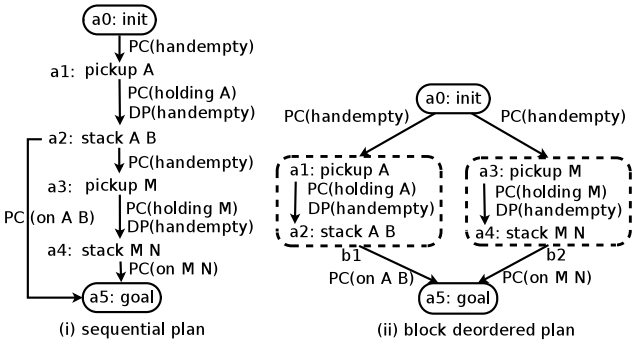


Figure 1: A sequential plan and its block deordering. Precedences are labelled with their reasons: producer–consumer (i.e., a causal link), denoted $\mathrm{PC}(p)$; deleter–producer, denoted $\mathrm{DP}(p)$; and consumer–deleter, denoted $\mathrm{CD}(p)$.

The nature of blocks is very similar to macros. Since block deordering tends to produce blocks that localise interactions between actions as much as possible, they often capture some coherent activities that can form a basis for useful macros. In addition, the block deordering algorithm returns also a justification for correctness of the block ordering, by labelling ordering constraints with their reasons (as mentioned above).

4 Macro-block Formulation

We extend the blocks to *macro-blocks* by considering different relations between them (as defined below) in order to provide larger subplans revealing important structural properties that often capture more complex activities that are frequently used in plans (e.g. mixing a cocktail and cleaning the shaker afterwards – as observed in the Barman domain).

Having a block deordered plan, we define relations of *immediate predecessor and immediate successor* of a block. Ordering between blocks is determined by any of the necessary reasons ($\mathrm{PC}, \mathrm{CD}, \mathrm{DP}$, and DK) as stated in Section 3.3. Let $\mathrm{IP}(b)$ be a set of blocks being ordered immediately before b with respect to the transitively reduced ordering. Also, let $\mathrm{IS}(b)$ be a set of blocks that are ordered immediately af-

Algorithm 1 Computing extended blocks.

1: $\mathrm{B_{ext}} \leftarrow \mathrm{B_{basic}}$
2: **while** $\exists b_i, b_j \in \mathrm{B_{ext}} : \mathrm{IP}(b_j) = \{b_i\}, \mathrm{IS}(b_i) = \{b_j\}$ **do**
3: $\quad \mathrm{B_{ext}} \leftarrow \mathrm{B_{ext}} \cup \{b_i \cdot b_j\} \setminus \{b_i, b_j\}$
4: **end while**

ter b with respect to the transitively reduced ordering. The *causal followers* of a producer a_p with respect to an atom m, $\mathrm{CF}_{\langle m,a_p\rangle}$, are a set of actions $\{a_p, a_j, ..., a_k\} \setminus \{a_\mathrm{I}, a_\mathrm{G}\}$ (a_I and a_G are special "init" and "goal" actions respectively, where a_I produces initial atoms, and a_G consumes the goal atoms) such that $\{\langle a_p, m, a_j\rangle, ..., \langle a_p, m, a_k\rangle\}$ are the causal links. For example, the atom $m =$ (holding A) in the block deordered plan of Figure 1 is associated with one causal link: $\langle a_1, m, a_2\rangle$, which form the causal followers $\mathrm{CF}_{\langle(\text{holding A}),a_1\rangle} = \{a1, a2\}$. The *causal follower blocks* involve a set of blocks related to the given causal link rather than a set of actions. In particular, the causal follower blocks for an atom m and its producer a_p are defined as $\mathrm{CFB}_{\langle m,a_p\rangle} = \{b \mid b \in B, \mathrm{CF}_{\langle m,a_p\rangle} \cap b \neq \emptyset\}$ (B is a set of blocks). For example, the causal follower blocks of the atom $m =$ (holding A) and its producer a_1 in the block deordered plan of Figure 1, are $\mathrm{CFB}_{\langle(\text{holding A}),a_1\rangle} = \langle\{b1\}\rangle$, since all the actions of $\mathrm{CF}_{\langle(\text{holding A}),a_1\rangle}$ are captured by the block $b1$.

Let $\mathrm{B_{basic}}$ be the set of blocks acquired by applying the block deordering approach. *Extended blocks* $\mathrm{B_{ext}}$ are generated iteratively from the basic blocks as depicted in Algorithm 1. In other words, extended blocks encapsulate non-branching subsequences of blocks, and therefore, can better capture some non-trivial activities. It should be noted that if no deordering is possible, $\mathrm{B_{ext}}$ will consist of a single (extended) block encapsulating the whole plan.

Macro-blocks are constructed according to the following eight rules we have specified. Each rule is applied over both sets of basic ($\mathrm{B_{basic}}$) and extended blocks ($\mathrm{B_{ext}}$). The macro-block construction rules are as follows:

$\mathrm{R1} : \langle b\rangle$	$\mathrm{R5} : \langle \mathrm{R4}, \mathrm{R2}\rangle$
$\mathrm{R2} : \langle \mathrm{IP}(b), b\rangle$	$\mathrm{R6} : \langle \mathrm{R4}, \mathrm{R3}\rangle$
$\mathrm{R3} : \langle b, \mathrm{IS}(b)\rangle$	$\mathrm{R7} : \langle \mathrm{R4}, \mathrm{R4}\rangle$
$\mathrm{R4} : \langle \mathrm{IP}(b), b, \mathrm{IS}(b)\rangle$	$\mathrm{R8} : \mathrm{CFB}_{\langle m,a_p\rangle}$

The above rules can be divided into three groups, namely the primary rules (R1 to R4), the secondary rules (R5 to R7), and the causal rule (R8). Applied to all blocks in $\mathrm{B_{basic}} \cup \mathrm{B_{ext}}$, the primary, secondary, and causal rules can produce duplicates; of course, only unique macro-blocks are kept. The primary rules generate macro-blocks considering basic and extended blocks and their immediate predecessors and successors. Secondary rules chain exactly two macro-blocks generated by primary rules. The causal rule (R8) constructs macro-blocks based on causal links, specifically for each atom m and its producer a_p. These macro-blocks are not limited to any fixed length. The resultant macro-block, after applying any rule, may not capture some intermediate blocks, i.e., blocks that are ordered in between. Assume that $\mathrm{IS}(b_x) = \mathrm{IP}(b_y) = \{b_p, b_q\}$, and b_p and b_q are unordered. Applying R4 over b_p results in a set of blocks that do not contain the intermediately placed block b_q. We incorporate the intermediate blocks after applying each rule, i.e., b_q will be considered in a macro-block constructed from b_p by applying R4. If intermediate blocks are not considered, then macro-blocks do not represent consistent subplans.

Algorithm 2 The high-level design of BLOMA

1: $\Pi \leftarrow$ GenerateTrainingPlans()
2: $\mathcal{B} \leftarrow$ GenerateMacroBlocks(Π)
3: $M \leftarrow$ ExtractMacros($\mathcal{B}$)
4: EnhanceDomain(M)
5: $\Pi \leftarrow$ GenerateTrainingPlans()
6: FilterMacros(Π)
7: LearnEntanglements(Π)

5 Generating Macros from Macro-Blocks

Blocks aim to remove some of the ordering constraints (see Section 3.3), and thus reduce some interference between actions. Extended blocks put together blocks that have "no other alternative", in other words, blocks that are strictly consecutive in block deordered plans. Extended blocks are thus supposed to capture larger activities (e.g. shaking a cocktail and cleaning the shaker). Macro-blocks encapsulate both basic and extended blocks and relations between them. Considering relations between these blocks is somewhat related to the "chaining" approaches utilised by most of the existing macro learning techniques. In short, macro-blocks thus capture structural knowledge in form of coherent subplans that frequently occur in plans. These subplans can be linearised and thus can be exploited in the form of macros.

Algorithm 2 depicts the high-level implementation of BLOMA. Firstly, given a set of training planning problems (simpler but not trivial), training plans are generated. Then, BLOMA processes the training plans and generates macro-blocks as described in Section 4. Macro-blocks that consist of more than one action are linearised and then assembled into macros as described in Section 3.1. Macros that appear frequently in macro-blocks are added into the domain model. Then, training plans are re-generated using the macro enhanced domain model. Macros that do not appear or appear infrequently in the re-generated training plans are filtered out. In fact, we let the planner "decide" which macros are useful for it. The same strategy was used by MacroFF [Botea *et al.*, 2005]. As a final step of BLOMA macro-specific entanglements (i.e. those where only macros are involved) are learnt. Entanglements are learnt by checking whether for each operator and related predicates the entanglement conditions are satisfied in all the training plans. Some error rate (typically 10%) is allowed. For details, see [Chrpa and McCluskey, 2012]. As MUM does [Chrpa *et al.*, 2014] BLOMA uses entanglements for efficient pruning of unnecessary instances of macros.

Whether a macro candidate or macro is "frequent" is determined relatively. Let $f^b(m)$ be a number of occurrences of a macro candidate m in the set of macro-blocks $\mathcal{B}$. Then, a macro candidate $m \in M$ (M is the set of macros) is considered as frequent if $f^b(m) \geq p_b \max_{x \in M} f^b(x)$, where $0 < p_b \leq 1$. Similarly, let $f^p(o)$ be a number of occurrences of an operator or macro o in macro enhanced train-

ing plans Π. Then, a macro m is considered as frequent if $f^p(m) \geq p_p \max_{x \in O} f^p(x)$, where $0 < p_p \leq 1$ and O is a set of operators (including macros) defined in the planning domain. Clearly, setting p_b, p_p too high might cause filtering some useful macros out, while setting them too low might cause keeping useless macros.

Using basic blocks leads to generating a subset of macros that can be generated by using extended blocks, which often results in failing to generate any macro. Although exploiting extended blocks yields to a relatively small number of considered macro candidates, they often provide a basis for good quality macros. This is because extended blocks often capture complex single activities. Such macros are usually not generated by "chaining-based" approaches. Macro-blocks, on the other hand, provide a larger number of suitable macro candidates than extended blocks, since R2-R8 allow weaker forms of block chaining. R2-R8 in fact incorporate a variant of "chaining" approaches which might lead into generating similar macros as the existing techniques do.

Following the above observations, BLOMA works in a two-phased way. Initially, BLOMA tries to generate macros from extended blocks. If no macro is generated, then BLOMA uses macro-blocks to generate macros.

6 Experimental Analysis

We experimentally evaluated BLOMA in order to demonstrate how it improves against the original and MUM enhanced domain and problem models. We used all the domains from the learning track of IPC-7. The results are analysed in terms of providing insights into possible impact of generated macros to the planning process.

6.1 Learning

We generated 6 training problems for each domain. The training problems were rather simple but not trivial, so the plan length was mostly within 40-80 steps. For learning, we have used 4 state-of-the-art planners that accommodate various planning techniques: LAMA [Richter and Westphal, 2010], MpC [Rintanen, 2014], Probe [Lipovetzky *et al.*, 2014] and Mercury [Katz and Hoffmann, 2014]. For each domain, we considered that planner that generated the best quality (shortest) training plans. One plan was considered per each training problem. The parameters p_b and p_p were both set to 0.5. The learning process took from couple of seconds to couple of minutes, where generating training plans consume the majority of the learning time.

6.2 Comparison

We use IPC score as defined in the learning track of IPC-7 [Coles *et al.*, 2012]. For an encoding e of a problem p, $IPC(p, e)$ is 0 if p is unsolved in e, and $1/(1 + \log_{10}(T_{p,e}/T^*_p))$, where $T_{p,e}$ is the CPU-time needed to solve p in e and T^*_p is the smallest CPU-time needed to solve p in any considered encodings, otherwise. As in the learning track, the time limit was 15 minutes per problem. All the experiments were run on Intel Xeon 2.53 Ghz with 2GB of RAM, CentOS 6.5. Apart of the four planners considered for learning, we also used Yahsp3 [Vidal, 2014] and Bfs-f [Lipovetzky *et al.*, 2014].

	Coverage			Δ IPC	
	O	M	B	M	B
Barman					
Lama	0	-	30	-	+30.0
Mercury	23	-	30	-	+9.8
MpC	0	-	0	-	0.0
Probe	3	-	22	-	+19.7
Yahsp	0	-	11	-	+11.0
Bfs-f	30	-	30	-	-6.5
BlocksWorld					
Lama	23	-	25	-	+3.3
Mercury	19	-	8	-	-11.1
MpC	0	-	0	-	0.0
Probe	24	-	25	-	+3.5
Yahsp	28	-	22	-	-7.5
Bfs-f	0	-	10	-	+10.0
Depots					
Lama	0	2	2	+2.0	+2.0
Mercury	0	0	0	0.0	0.0
MpC	18	24	24	+8.6	+8.6
Probe	30	30	30	+4.2	+4.2
Yahsp	21	20	20	+1.0	+1.0
Bfs-f	4	21	21	+17.8	+17.8
Gripper					
Lama	0	30	30	+30.0	+30.0
Mercury	0	4	4	+4.0	+4.0
MpC	0	0	0	0.0	0.0
Probe	0	5	5	+5.0	+5.0
Yahsp	0	0	0	0.0	0.0
Bfs-f	0	0	0	0.0	0.0
Parking					
Lama	3	-	0	-	-3.0
Mercury	6	-	3	-	-3.1
MpC	5	-	0	-	-5.0
Probe	3	-	4	-	+1.2
Yahsp	0	-	4	-	+4.0
Bfs-f	5	-	5	-	-0.9
Rovers					
Lama	27	29	25	+3.6	-3.3
Mercury	24	26	29	+6.7	+9.5
MpC	5	5	5	-0.1	0.0
Probe	28	27	19	-1.0	-11.9
Yahsp	30	30	30	+0.6	-0.4
Bfs-f	0	0	0	0.0	0.0
Satellite					
Lama	3	26	18	+23.7	+15.7
Mercury	19	11	13	-10.7	-9.2
MpC	1	0	0	-1.0	-1.0
Probe	0	2	0	+2.0	0.0
Yahsp	16	27	16	+4.7	-6.8
Bfs-f	0	0	0	0.0	0.0
Spanner					
Lama	0	0	-	0.0	-
Mercury	0	0	-	0.0	-
MpC	30	30	-	+1.7	-
Probe	0	0	-	0.0	-
Yahsp	0	0	-	0.0	-
Bfs-f	0	0	-	0.0	-
TPP					
Lama	16	15	17	-2.0	+1.0
Mercury	19	16	20	-4.0	+2.7
MpC	9	11	20	+2.0	+14.0
Probe	12	14	17	+2.2	+7.1
Yahsp	30	30	30	-0.7	+0.4
Bfs-f	15	15	8	-0.7	-8.3

Table 1: Comparison between original (O), MUM (M) and BLOMA (B) on the IPC-7 learning track domains. "-" stands for "no macros found". Δ IPC stands for a difference of IPC score of the original and the corresponding macro enhanced encodings.

Table 1 presents the results of BLOMA in comparison to the original problem encodings and macros generated by MUM. BLOMA and MUM generated the same sets of macros in Depots and Gripper. By exploiting extended blocks, macros have been generated in Barman, BW, Rovers and TPP. In Spanner, no macro has been generated. Positive results have been achieved in Barman, Depots, Gripper and TPP. In the rest of domains the results were rather mixed.

Mixed results point to the fact that different planning techniques have often a different "response" to macros. This observation is, of course, not very surprising. Macros are often not supportive if the original problems are solved quickly (in a few seconds) since macros are more demanding in the pre-processing stage. Letting a planner learn macros for itself is, however, occasionally helpful, although in the Satellite domain, MpC learnt a good macro for itself. On the other hand, when training plans are of poor quality, generated macros are very poor as well. For example, in TPP, Probe learnt a very poor macro.

In Barman, the success of BLOMA rests in finding an 8-step long macro that captures an important activity – shaking a cocktail, pouring it into a shot and cleaning the shaker afterwards. In Gripper, BLOMA (as well as MUM) found a useful 3-step long macro that directly delivers an object from its initial to its goal location. In other domains, BLOMA found only 2-step macros capturing only partial activities (e.g. loading a truck). In TPP, BLOMA generated a "recursive" macro drive-drive that a bit surprisingly contributed considerably to MpC and Probe's performance.

Apart from Barman, we identified other domains, namely Scanalyzer, Storage and the IPC-2 version of Gripper, where BLOMA found longer macros. Most problems from these domains (all problems in the IPC-2 Gripper domain) are easy to solve, that is, the planners needed at most a few seconds. With higher pre-processing requirements for macros, there is no space for improvement, although the performance was rarely considerably worse. In Storage, the learnt macro helped Probe to solve 7 more problems, Bfs-f and LAMA to solve 2 more problems, however, the macro caused MpC to solve 6 less problems. In Scanalyzer, the learnt macro was specific for "2-cycle problems" [Helmert and Lasinger, 2010]. While considering the macro, MpC solved 1 more problem and Mercury has better performance on harder problems. On the contrary, Bfs-f solved 1 less problem.

6.3 Discussion

As discussed before, BLOMA is particularly useful in domains where longer macros capturing important activities can be identified. Traditional "chaining-based" approaches (e.g. MUM) often fail in these occasions. In other domains, BLOMA performs with mixed results.

Number of instances of macros might cause issues with memory consumption as well as might make pre-processing more difficult. A typical example is the Parking domain that represents a combinatorial problem of re-arranging cars in a parking lot. Therefore, macros despite being frequently used in plans might have detrimental effect on performance. From this perspective, approaches such as MUM that consider only macros with a relatively small number of instances are efficient. However, as showed in this paper, they are not able to generate longer macros that despite a larger number of instances can be very beneficial.

On the other hand, we have observed that in some cases macros have arguments that are not necessary to be kept explicitly, so the number of instances might be reduced considerably. In particular, if a state of some object remains the same, i.e., no predicates containing this object are added or deleted, then it is not necessary to keep this object as an argument of the macro. For example, having a macro that represents an activity of delivering a package by a truck and returning the truck to the place of package's origin. The truck in fact does not change its state after applying the macro. Of course, some truck must be in the place of package's origin which has to be reflected in macro's precondition. It can be done by using existential quantifiers. Although they are supported in PDDL, many planning engines do not support such a feature.

Impact of particular macros depends on a planning technique exploiting them. In Depots, macros bypass situations where a crate is held by a hoist. It is well known that a hoist can hold at most one crate at time, however, delete-relaxed heuristics ignores such a constraint which might lead into having many local minima in heuristic landscape [Hoffmann, 2011]. Macros that reduce the complexity of Planning Graph (e.g. smaller number of layers, less mutexes) seem to be beneficial for techniques such as those incorporated in MpC as observed in Depots and TPP. We believe that classifying macros according to their features will be helpful for selecting planner-specific macros that will be tailored for a given planning technique.

7 Conclusions

In this paper, we presented BLOMA that learns planner-independent macros from macro-blocks, which can be extracted from training plans, encapsulating constituent coherent subplans. Such an approach is beneficial especially for domains (e.g. Barman), where an important activity repeatedly applied in plans can be encapsulated by (longer) macros. We have shown empirically that BLOMA can considerably improve the performance in many cases.

There are two major limitations. Firstly, a higher number of macros' instances might be detrimental to the planning process. However, all the arguments of macros do not have to be always explicitly defined. To address this we have to use existential quantifiers in macros' preconditions, however, such a feature is not widely supported by planning engines. Alternatively, we might learn HTN methods rather than macros. Secondly, some macros are not supportive for certain planning techniques (e.g. they might "jump" into local heuristics minima). Hence, classifying macros according to their features might reveal what kind of macros has positive/negative impact on certain planning techniques.

Acknowledgements The research was funded by the UK EPSRC Autonomous and Intelligent Systems Programme (grant no. EP/J011991/1). The authors would like to acknowledge the use of the University of Huddersfield Queensgate Grid in carrying out this work. NICTA is funded by the Australian

Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program

References

[Alhossaini and Beck, 2013] Maher A. Alhossaini and J. Christopher Beck. Instance-specific remodelling of planning domains by adding macros and removing operators. In *Proceedings of SARA*, pages 16–24, 2013.

[Areces *et al.*, 2014] Carlos Areces, Facundo Bustos, Martín Dominguez, and Jörg Hoffmann. Optimizing planning domains by automatic action schema splitting. In *Proceedings of ICAPS*, 2014.

[Botea *et al.*, 2005] A. Botea, M. Enzenberger, M. Müller, and J. Schaeffer. Macro-FF: improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research (JAIR)*, 24:581–621, 2005.

[Chrpa and Barták, 2009] L. Chrpa and R. Barták. Reformulating planning problems by eliminating unpromising actions. In *Proceedings of SARA*, pages 50–57, 2009.

[Chrpa and McCluskey, 2012] Lukás Chrpa and Thomas Leo McCluskey. On exploiting structures of classical planning problems: Generalizing entanglements. In *Proceedings of ECAI*, pages 240–245, 2012.

[Chrpa *et al.*, 2014] Lukás Chrpa, Mauro Vallati, and Thomas Leo McCluskey. Mum: A technique for maximising the utility of macro-operators by constrained generation and use. In *Proceedings of ICAPS*, pages 65–73, 2014.

[Chrpa, 2010] L. Chrpa. Generation of macro-operators via investigation of action dependencies in plans. *Knowledge Engineering Review*, 25(3):281–297, 2010.

[Coles *et al.*, 2007] A. Coles, M. Fox, and A. Smith. Online identification of useful macro-actions for planning. In *Proceedings of ICAPS*, pages 97–104, 2007.

[Coles *et al.*, 2012] Amanda Coles, Andrew Coles, Angel G. Olaya, Sergio Jiménez, Carlos L. Lòpez, Scott Sanner, and Sungwook Yoon. A survey of the seventh international planning competition. *AI Magazine*, 33:83–88, 2012.

[Dawson and Siklóssy, 1977] C. Dawson and L. Siklóssy. The role of preprocessing in problem solving systems. In *Proceedings of IJCAI*, pages 465–471, 1977.

[Fikes and Nilsson, 1971] Richard Fikes and Nils J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208, 1971.

[Ghallab *et al.*, 2004] M. Ghallab, D. Nau, and P. Traverso. *Automated planning, theory and practice*. Morgan Kaufmann, 2004.

[Haslum and Jonsson, 2000] Patrik Haslum and Peter Jonsson. Planning with reduced operator sets. In *Proceedings of AIPS*, pages 150–158, 2000.

[Helmert and Lasinger, 2010] Malte Helmert and Hauke Lasinger. The scanalyzer domain: Greenhouse logistics as a planning problem. In *ICAPS*, pages 234–237, 2010.

[Hoffmann and Nebel, 2001] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research (JAIR)*, 14:253–302, 2001.

[Hoffmann, 2011] J. Hoffmann. Analyzing search topology without running any search: On the connection between causal graphs and h+. *Journal Artificial Intelligence Research (JAIR)*, 41:155–229, 2011.

[Katz and Hoffmann, 2014] Michael Katz and Joerg Hoffmann. Mercury planner: Pushing the limits of partial delete relaxation. In *The Eighth IPC. Description of Participant Planners of the Deterministic Track*, pages 43–47, 2014.

[Korf, 1985] R.E. Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26(1):35–77, 1985.

[Lipovetzky *et al.*, 2014] Nir Lipovetzky, Miquel Ramirez, Christian Muise, and Hector Geffner. Width and inference based planners: Siw, bfs(f), and probe. In *The Eighth IPC. Description of Participant Planners of the Deterministic Track*, pages 6–7, 2014.

[McAllester and Rosenblitt, 1991] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proc. 9th National Conference on Artificial Intelligence*, 1991.

[Minton, 1988] Steven Minton. Quantitative results concerning the utility of explanation-based learning. In *Proceedings of AAAI*, pages 564–569, 1988.

[Newton *et al.*, 2007] M. A. H. Newton, J. Levine, M. Fox, and D. Long. Learning macro-actions for arbitrary planners and domains. In *Proceedings of ICAPS*, pages 256–263, 2007.

[Richter and Westphal, 2010] S. Richter and M. Westphal. The LAMA planner: guiding cost-based anytime planning with landmarks. *Journal Artificial Intelligence Research (JAIR)*, 39:127–177, 2010.

[Rintanen, 2014] Jussi Rintanen. Madagascar: Scalable planning with sat. In *The Eighth IPC. Description of Participant Planners of the Deterministic Track*, pages 66–70, 2014.

[Siddiqui and Haslum, 2012] F.H. Siddiqui and P. Haslum. Block-structured plan deordering. In *AI 2012: Advances in Artificial Intelligence (Proc. 25th Australasian Joint Conference*, volume 7691 of *LNAI*, pages 803–814, 2012.

[Siddiqui and Haslum, 2013] Fazlul Hasan Siddiqui and Patrik Haslum. Plan quality optimisation via block decomposition. In *Proceedings of IJCAI 2013*, 2013.

[Slaney and Thiébaux, 2001] J. Slaney and S. Thiébaux. Blocks world revisited. *Artificial Intelligence*, 125(1-2):119–153, 2001.

[Vidal, 2014] Vincent Vidal. Yahsp3 and yahsp3-mt in the 8th international planning competition. In *The Eighth IPC. Description of Participant Planners of the Deterministic Track*, pages 64–65, 2014.

A.5 On mixed-initiative planning and control for autonomous underwater vehicles

L. Chrpa, J. Pinto, M. A. Ribeiro, F. Py, J. B. de Sousa, and K. Rajan. On mixed-initiative planning and control for autonomous underwater vehicles. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*, pages 1685–1690, 2015.

2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Congress Center Hamburg
Sept 28 - Oct 2, 2015. Hamburg, Germany

On Mixed-Initiative Planning and Control for Autonomous Underwater Vehicles

Lukáš Chrpa*, José Pinto†, Manuel A. Ribeiro†, Frédéric Py†, João Sousa†, Kanna Rajan†
*PARK Research Group, School of Computing and Engineering, University of Huddersfield, UK
†LSTS, Faculty of Engineering, University of Porto, Portugal
l.chrpa@hud.ac.uk,{zepinto,maribeiro,jtasso,kanna.rajan}@fe.up.pt, fredpy@gmail.com

***Abstract*— Supervision and control of Autonomous underwater vehicles (AUVs) has traditionally been focused on an operator determining a priori the sequence of waypoints of a single vehicle for a mission. As AUVs become more ubiquitous as a scientific tool, we envision the need for controlling multiple vehicles which would impose less cognitive burden on the operator with a more abstract form of human-in-the-loop control. Such mixed-initiative methods in goal-oriented commanding are new for the oceanographic domain and we describe the motivations and preliminary experiments with multiple vehicles operating simultaneously in the water, using a shore-based automated planner.**

I. Introduction

Autonomous Underwater Vehicles (AUVs) have made steady gains as tools for scientific observations and security applications in recent years. They have shown their utility in both benthic [1] and upper water column exploration [2], [3].Typically they have been deployed as single vehicles controlled by a single operator.

As these robots have become more affordable and robust, it is becoming viable to own and operate multiple vehicles, sometimes simultaneously. The LSTS laboratory, for instance, participates in an annual exercise where multi-vehicle operation and control are the key emphasis for security and upper water exploration [4]. These experiments target coordinated observations primarily to look for features such as frontal zones, blooms and plumes in the coastal ocean [5]. Networked heterogeneous robotic vehicles can be applied to complex scenarios where mobile sensing nodes can be scheduled according to their specific capabilities towards a common goal. However, in order for tasking vehicles appropriately, they must be aware of the state of the entire system and any vehicle-specific capabilities and parameters.

Our experiments in coordination and control, have revolved on the use of advanced decision-support tools in the LSTS toolchain [6] often with onboard machine intelligence to synthesize actions with continuous interleaved planning and execution. Our AUV platform is the Light AUV (LAUV) [7], an advanced and robust vehicle with an open source (and open) architecture (Fig. 1).

In these experiments, decision support, situational awareness, control, planning, data visualization and archiving is provided by software including NEPTUS, with DUNE providing low-level functional control and IMC the middleware message-passing mechanism. Task planning on NEPTUS involves either sending waypoints

Fig. 1. The human-portable LSTS Light Autonomous Underwater Vehicle (LAUV) [7] is a multi-use vehicle for benthic and upper water-column exploration.

to the AUV when on the surface, or more recently, sending abstract plans to the vehicle formulated as goals for onboard plan synthesis [4]. This has proved adequate for commanding a single vehicle. However, as we move towards mixed-mode multi-vehicle operations at sea, the need arises to use abstraction for goal formulation. We envision that a single NEPTUS user would rapidly plan a set of goals for a collection of vehicles in a mixed-initiative formulation [8], and then dispatch these goals for execution for AUVs. The goals would initially be targeted for distinct tasks on individual vehicles with the aim of obtaining field experience on such multi-vehicle operationand then gravitate towards coordinated experiments with temporal and task constraints involving complex task handoffs between AUVs and in the near future UAVs.

In this paper, we describe the first steps towards such a mixed-initiative planning and control tool-set that combines "high-level" goal-oriented mission planning and "low-level" control of the vehicles for AUVs only. In particular, the operator generates multiple tasks in NEPTUS which encodes them as a planning problem that is solved by a domain-independent planning engine and the plan is then dispatched to every vehicle as a sequence of tasks to perform. Control-loop closure occurs onboard; however task closure for now is done visually by the operator on NEPTUS who can observe the vehicle behavior in real-time using acoustic modems for communicating AUV states. We have evaluated our approach on a mine-hunting scenario with multiple AUVs with different payloads. To the best of our knowledge this is the first work in this domain, which use mixed-initiative automated planning and control methods on shore.

The paper is organized as follows. Section II situates this work in the context of previous efforts, Section III provides background material, with Section IV the core of the paper, highlighting the planning technique.

978-1-4799-9994-1/15/$31.00 ©2015 IEEE

Results from experiments are highlighted in Section V and we conclude with future work in Section VI.

II. Related Work

In the oceanographic domain, command and control of a vehicle is typically waypoint based with incremental waypoint achievement as a means to identify goal achievement. Our previous work is among the few to include automated planning in this harsh domain where we have demonstrated embedded *continuous* planning and execution in the context of scientific upper water-column exploration on a single AUV [9], [10], [11]. Recent work [12] shows more interest along similar lines. Multi-vehicle planning in the oceanographic context has continued to revolve along waypoint based low-level control [13]. Multiple glider deployments [14] have also been demonstrated, again using simple waypoint-based methods. Further, our work is informed by past efforts in mixed-initiative command/control in the space domain; the MAPGEN system continues to command the rover *Opportunity* on the surface of Mars using automated planning methods [8] also used in this work. While we have demonstrated a form of mixed-initiative control on AUVs, this has been in the context of a single AUV while tracking gradients [11] and where automated decision-making was onboard the vehicle. To the best of our knowledge using automated planning as a means to provide abstraction in control over multiple vehicles in the oceanographic domain is novel.

III. Background & Motivation

LSTS, has developed a set of software tools to support the operation of heterogeneous vehicle networks [15]. Operators interact with vehicles via NEPTUS, a graphical decision-support system with visualization and analysis capabilities that allows users to view all incoming vehicle data in real-time, to define vehicle objectives and to supervise their execution.

Embedded on the vehicles, the DUNE executive manages localization, path-planning, data acquisition and command execution on different types of vehicles and other embedded systems like data loggers and communication gateways. IMC is the middle-ware used for conjoining all the sensory data with the various platforms. DUNE has an in-built executive which is capable of parsing and executing script-based plans. The language used for describing plans is graph-based where each node represents a parameterized behavior and transitions between nodes occur in response to events such as behavior termination or failures. Moreover, DUNE allows the specification of configuration parameters, like payload settings, on transitions. This simple plan specification has been used extensively for deterministic behaviors. In order to allow other types of behavior definition and execution, an API was devised that allows external modules to guide vehicles and control their payload configuration.

In this work, we describe a shore based automated planner which connects to the NEPTUS API for mixed-initiative command and control. Automated Planning

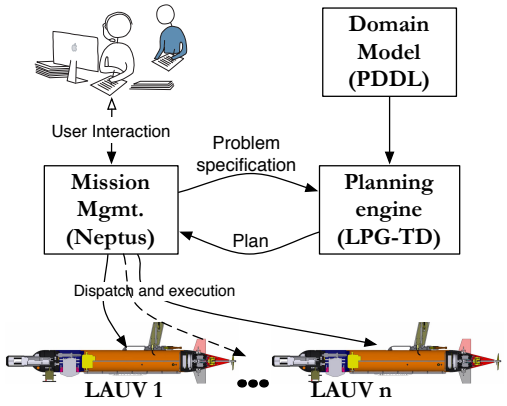


Fig. 2. A modular architecture of the system.

deals with a problem of finding a sequence of actions that transform the environment from a given initial state to a desired goal state [16]. The simplest form of automated planning works in deterministic, fully observable and static environment, where effects of actions are instantaneous. Temporal planning, which is the focus of our interest, allows "durative actions" whose planning and execution makes explicit use of, and reasons with, the notion of time.

We use PDDL 2.1 [17] for representing our planning problems. The environment is described by predicates and numeric functions. Actions are specified via pre-conditions, effects and the duration of their execution. Preconditions are sets of logical expressions that must be true in order to have the actions executable. In PDDL, these expressions can take place prior to starting action execution, prior to finishing action execution, or over the whole time period when the action is executed. Effects are sets of literals and function assignments that become true when the action is executed. In PDDL, effects can occur just after starting, or just after finishing action execution.

The PDDL representation allows us to specify domain models and problem specifications separately. Usually, one domain model is used for a class of problems. In particular, a domain model consists of predicate and function descriptions and action definitions, while a problem specification consists of definition of objects, an initial state and a set of goal conditions.

In domain-independent planning, planning engines and domain models are decoupled. A planning engine can deal with various domain models, and a domain model can be accepted by various planning engines. Therefore, if the domain model is modified, there is no reason to modify the planning engine; equally it is possible to replace one planning engine with another without changing the domain model. As shown in Fig. 2, a user specifies the mission in a control system like NEPTUS, which then automatically generates planning problem description that is accepted by the planning engine. Given the domain model and problem specification, the planning engine returns a plan (if one exists) to NEPTUS which in turn, distributes the plan among the vehicles, where it is executed. Since the planning engine is used as a "black-box" we have extended NEPTUS in

order to generate problem specification in PDDL as well as to process PDDL-compliant plans.

Specifying problems in PDDL is relatively straightforward. For example, predicates are defined in the form of `(at noptilus1 task1-loc)` and function assignments are defined in form of `(= (battery-use noptilus1) 50)`. A plan action is in the form `48.0: (MOVE XPLORE1 XPLORE1-DEPOT T01-LOC) [1365.000]`, where the first number refers to the time-stamp when the action is executed, while the second refers to the duration of action execution. Every action accepts a single vehicle as a parameter.

In this work we use **LPG-TD** [18] as the planning engine. **LPG-TD** is based on a local search in Planning Graphs [16] which allows executing actions that do not interfere with each other in each plan step. While it is a mature planner, most state-of-the-art planning engines either do not support required features (such as numeric functions or durative actions) nor do they scale well. For instance, while both **Optic** [19] and **EUROPA**$_2$ [20] offer richer representations, our experimentation with these demonstrated that their performance did not scale well for problems we envision for such mixed-initiative interaction.

IV. Goal-oriented Mission Planning

In short, automated planning is used to decide what tasks, which are specified by a user, are performed by which AUV and when. Various constraints are considered (e.g. battery limits).

We have conceptualized mission requirements in the form of a domain model specification. This conceptualization is divided into three categories: object types, predicates and functions, and actions similar to work of Shah et al. [21]. Object types refer to classes of objects that are relevant for the planning process such as: *vehicle* (V), *payload* (P), *phenomenon* (X), *task* (T), *location* (L). By "phenomenon" we mean a target object or area of interest, where we assume that such phenomena are deterministic and static. Tasks are considered as atomic, to be fulfilled by a single vehicle.

Predicates and functions describe states of the environment. In particular, predicates represent relationships between objects, and functions refer to quantity of resources related to the objects. In our case, we have defined: $at \subseteq V \times L$ – a location of the vehicle, $base \subseteq V \times L$ – a location of the vehicle's *depot*, i.e. known positions where a vehicle is expected to be safe when on the surface, $has \subseteq V \times P$ – whether a payload is attached to the vehicle, $at\text{-}phen \subseteq X \times L$ – a location of the phenomenon, $task \subseteq T \times X \times P$ – which describes a task of getting data about a phenomenon from a specific payload, $sampled \subseteq T \times V$ – whether data of a given task has been acquired by the vehicle, $data \subseteq T$ – whether the task data has been acquired from the vehicle, $dist : L \times L \to \mathbb{R}^+$ – a distance between two locations, $speed : V \to \mathbb{R}^+$ – speed over ground of the vehicle, $battery\text{-}level : V \to \mathbb{R}_0^+$ – the amount of energy in a vehicle's battery, $battery\text{-}use : V \cup P \to \mathbb{R}^+$ – battery consumption per distance unit (moving a vehicle) or per time unit (using a payload). We currently make an assumption of linear energy use both for moving or using a payload.

Actions when executed modify the environment according to their effects. We have specified 4 actions (we denote t_s as time when an action is executed, and t_e as time when an action execution ends):

- *move*(v, l_1, l_2) – the vehicle v moves from its location of origin l_1 to its desination location l_2. As a precondition is must hold that in t_s: $(v, l_1) \in at$, $battery\text{-}level(v) \geq dist(l_1, l_2) * battery\text{-}use(v)$; and in t_e: $\neg\exists v_x \neq v : (v_x, l_2) \in at$. The effect is that in t_s: $(v, l_1) \notin at$, and $battery\text{-}level(v) = battery\text{-}level(v) - dist(l_1, l_2) * battery\text{-}use(v)$, and in t_e: $(v, l_2) \in at$
- *sample*(v, t, x, p, l) – the vehicle v samples a phenomenon x by the payload p. As a precondition it must hold that in t_s: $battery\text{-}level(v) \geq (t_e - t_s) * battery\text{-}use(p)$, and in $[t_s, t_e]$: $(v, l) \in at$, $(x, l) \in at\text{-}phen$, $(v, p) \in has$ and $(t, x, v) \in task$. The effect is that in t_s: $battery\text{-}level(v) = battery\text{-}level(v) - (t_e - t_s) * battery\text{-}use(p)$, and in t_e: $(t, v) \in sampled$.
- *survey*(v, t, x, p, l_1, l_2) – the vehicle v surveys the area (between l_1 and l_2) of a phenomenon x occurrence by the payload p. As a precondition is must hold that in t_s: $(v, l_1) \in at$, $battery\text{-}level(v) \geq dist(l_1, l_2) * battery\text{-}use(v) + (t_e - t_s) * battery\text{-}use(p)$, and in $[t_s, t_e]$: $(x, l_1) \in at\text{-}phen$, $(x, l_2) \in at\text{-}phen$, $(v, p) \in has$ and $(t, x, v) \in task$. Also, no other vehicle can perform the survey action over the phenomenon x in $[t_s, t_e]$. The effect is that in t_s: $(v, l_1) \notin at$, and $battery\text{-}level(v) = battery\text{-}level(v) - (dist(l_1, l_2) * battery\text{-}use(v) + (t_e - t_s) * battery\text{-}use(p))$, and in t_e: $(v, l_2) \in at$, $(t, v) \in sampled$.
- *collect-data*(v, t, l) – the data associated with a task t is collected by vehicle v. As a precondition it must hold that in $[t_s, t_e]$: $(v, l) \in at$, $(v, l) \in base$ and $(t, v) \in sampled$. The effect is that in t_e: $t \in data$.

As an example, the `Sample` action is encoded as depicted in Fig. 3.

The user specifies mission tasks in NEPTUS's front-end by pointing to the locations/area of the phenomena

```
(:durative-action sample
 :parameters (?v - vehicle ?l - location
    ?t -task ?o - phenomenon ?p - payload)
 :duration (= ?duration 60)
 :condition (and (over all (at-phen ?o ?l))
   (over all (task ?t ?o ?p))
   (over all (at ?v ?l))
   (over all (having ?p ?v))
   (at start (>= (battery-level ?v)
             (* (battery-use ?p) 60))))
 :effect (and (at end (sampled ?t ?v))
   (at start (decrease (battery-level ?v)
             (* (battery-use ?p) 60)))) )
```

Fig. 3. The `Sample` action in PDDL.

the user wants to observe and by selecting payloads the user wants to use for obtaining data. AUVs that are connected to NEPTUS are considered as operational and could be used for a mission. NEPTUS then encodes the information about vehicles and tasks into PDDL, and as a goal sets to acquire all the data specified by the tasks.

The planner decides which vehicle does which task and in which order the tasks are to be performed. Plans follow constraints specified in the action descriptions, i.e., collision avoidance (at most one vehicle can be in one location) and energy constraints (vehicles must not run out of energy before finishing all the tasks) with plans are optimized for total mission time. If the planner is unable to find a plan, the user is notified of plan failure, requiring her/him to iteratively relax constraints. An illustrative example follows:

Example:: An operator needs to plan two AUVs (#'s 1 & 2). #1 has multi-beam, #2 has the downward looking camera, and each of the vehicles has a side-scan sonar. Then, the operator specifies three tasks; two scope out areas of interest, where one has to be surveyed by side-scan and the second by multibeam, while the third refers to a location of another phenomenon that has to be sampled by camera. The planner then assigns the multibeam related task to be performed by #1 and the camera task to be performed by #2. The task where side-scan sonar is required might be performed by both AUVs. The planner decides to allocate it to #2 since it takes less time than #1. However at plan time, it is determined that #2 does not have enough energy; the task therefore gets assigned to #1. When neither of the AUVs has enough energy, the planner does not return any plan, since this last task cannot be allocated.

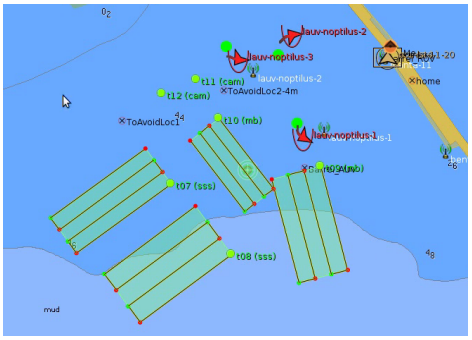
V. Experimental Evaluation

For system evaluation we used a mine-hunting scenario with multiple AUVs with different payloads. Typically, AUVs for such a scenario are equipped with side-scan sonars to image the sea floor, an acoustic altimeter and a high-resolution camera. After side-scan images are inspected by an operator, a set of objects of interest or *contacts* and their locations are determined and the AUVs are dispatched to identify those objects with high resolution cameras to be ground-truthed.

For sea-floor surveys, AUVs typically need to travel while maintaining a constant altitude over ground. This is primarily because for each sonar range and frequency configuration, there is an optimal distance at which the seafloor can be sampled to derive the appropriate resolution of the bathymetry. As a result, planned trajectories must be in accordance to the selected side-scan sonar configurations. Mine-hunting operations are also usually executed with a single AUV or using homogeneous vehicles as plans created manually by operators need to be adapted to each individual vehicle hardware configuration, periodicity of surfacing, side-scan sonar configurations, planned depths, etc.

Conceptually, the use of heterogeneous vehicles for such an application reduces the overall operation time, because some of the vehicles may be better suited for surveying large areas while others may have higher resolution sensors and/or navigation in order to take high-resolution images of detected contacts. However this results in a high cognitive burden on the operator. Moreover, manual planning for AUVs also requires a number of safety procedures that can add to operator overload when using multiple heterogeneous vehicles. Such a scenario is therefore ripe for our evaluation using automated planning.

(a) The deployment area, Leixões harbor, Porto.



(b) Detail of depots and objectives from Fig. 4(a) defined for phase one of the experiment.

Fig. 4. Experiment location and transit lines of the vehicles.

A. Field deployment

We used 3 LAUV's (Noptilus-1, -2 and -3) in our experiments. All vehicles have different payload configurations but similar navigation accuracy using the same Inertial Navigation System (INS). Noptilus-1 and Noptilus-3 carry lower-resolution Imagenex side-scan sonars and Imagenex DeltaT Multibeam sensors. Noptilus-3 and Noptilus-1 both carry underwater high-resolution cameras and Noptilus-2 carries only an Edgetech side-scan sonar. Noptilus-1 carries an RBR CTD (conductivity/temperature/depth) probe while other vehicles carry a sound velocity sensor, used primarily for correcting sonar measurements.

The deployment area was inside the Leixões harbor in Porto (Fig. 4(a)). The base station was located on top of a pier in the harbor with easy access to the water. The vehicles were deployed from near the base station and tele-operated to the operational area shown in the figure. Inside the operational area, the vehicles were placed in known and safe positions at the surface, which we call *depots*, where communications with the base station was viable. The experiment was then divided in two phases. In the first phase the vehicles were used to survey the

Vehicle	Action	Average Time Difference	Standard deviation
Noptilus-1	move	47,80	49,11
	survey	23,15	23,26
	sample	1,33	0,58
	communicate	0,16	0,17
Noptilus-2	move	39,57	35,66
	survey	107,88	141,10
	sample	N/A	N/A
	communicate	0,25	0,07
Noptilus-3	move	59,90	57,05
	survey	24	0,00
	sample	9,57	13,64
	communicate	0,11	0,16

TABLE I

DIFFERENCE BETWEEN PLANNED AND EXECUTION TIME FOR THE THREE AUVS (N/A IMPLIES A GIVEN ACTION WAS NOT IN A PLAN).

operational area while in the second, to identify and sample in points of interest in the data acquired in the earlier phase [1].

In phase one, the operator defined a set of areas of interest to be surveyed using side-scan or multibeam sonar sensors. Fig. 4(b) is a NEPTUS console view taken during the experiment after the operator specified the survey area. In order to task the vehicles, the operator requests NEPTUS to generate the plans for the vehicles which will result in translating the current world state to PDDL and generating a set of solutions using **LPG-TD**. The best solution is selected automatically based on overall execution time. As a result a set of scripted plans is generated and available for the operator to visualize and simulate. The operator can visually verify the plan and change the objectives and regenerate a solution accordingly if needed.

When the operator has validated the plan, s/he sends the plans to the respective vehicles and order its execution. After the vehicles perform this initial survey, all side-scan data is downloaded to the base station and inspected by the operator in determining contacts in the sampled regions.

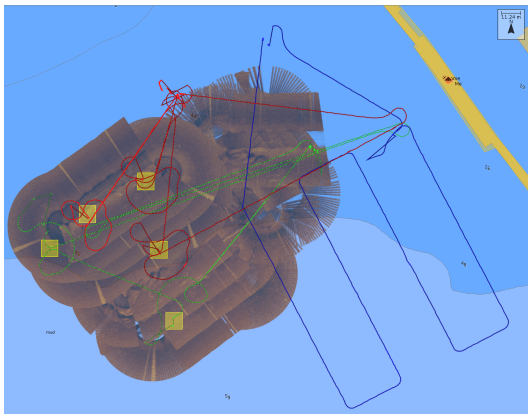
In phase two, a new set of objectives is then defined by the operator in order to get a closer view of these potential contacts. For some contacts it is likely that the operator not only requests a camera inspection but also CTD sampling in order to augment identification. The deployment ends when all vehicles return to their respective depots after completing objectives.

B. Experimental Data and Analysis

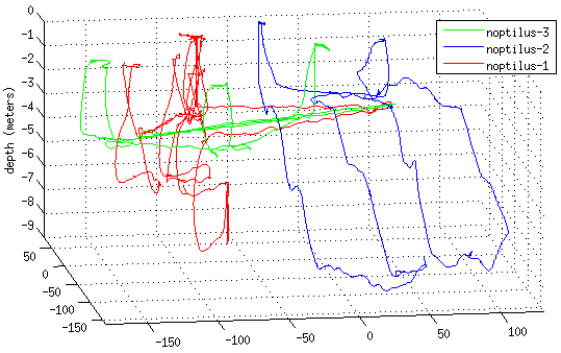
Fig. 5(a) shows a visualization of side-scan data (in brown) that was acquired on the first phase of the experiment showing complete coverage of the survey area. Moreover, the contacts identified by the operator at the end of this phase are indicated with yellow markers. In the second phase of the experiment, these contacts were visited by the vehicles and the tracks performed by the vehicles in phase two are overlayed within NEPTUS.

The behavior of the vehicles mapped well into the specification of the operator; however the action durations had discrepancies compared to the predicted

[1] https://www.youtube.com/watch?v=qWHXRek8_so



(a) Vehicle trajectory in the second phase of the experiment, overlayed on top of side-scan data (brown) and identified contacts (yellow).



(b) 3D view of the vehicle trajectories for the second phase of the experiment with trajectories from three AUVs.

Fig. 5. Results from phase two of the experiment showing side-scan and trajectories of the AUVs.

plan times. Table I shows the average difference between predicted and actual times for executing different actions. The results show that executing *communicate* and *sample* actions is more accurate than *survey* and *move*. This occurs because *communicate* and *sample* in particular, generate timed actions while the other actions generate a behavior that requires the vehicle to move between different locations whose completion time depends on environment conditions such as water currents, sea floor roughness and time to achieve depth. The roughness of the terrain can be seen in Fig. 5(b) where the trajectories of the vehicles are plotted in 3D. To give an example, the vehicle takes longer to reach the depth required to take a camera image if the location is in a deeper point of the operational area and this depth is not modeled a priori in the planner.

For movement actions, DUNE also appended a surfacing behavior. This was added in order to re-acquire GPS positions and therefore improve the quality of localization for upcoming samples and surveys. This is important, since it may require the vehicle to stay underwater for a substantial period of time with consequent localization errors. However this was not reflected in the planning domain model. Since the generated plans do not require time coordination of the vehicles, this did not impact the end result from the operator perspective.

C. Discussion

Incorporating domain-independent planning, i.e., a PDDL domain model and the **LPG-TD** planner, into NEPTUS does not introduce any serious engineering challenges. Our domain model scales well and **LPG-TD** solves problems with 20 tasks in at most a few seconds. However, the main drawback of our approach is that durations of the actions must be determined a priori, which as shown leads into discrepancies between planned and execution time. Although this is not a major issue for scenarios like ours, more complex applications requiring synchronisation of multiple vehicles might be considerably impacted by such discrepancies accumulating delays. We believe that considering optimistic, realistic and pessimistic estimations of action durations, and then evaluating corresponding plans might help to reduce discrepancies between plan and execution time.

VI. Discussion & Future Work

While the field experimentation validated our preliminary approach for real-world multi-vehicle applications, scenarios that require time coordination among vehicles, the DUNE executive lacks in-situ plan adaptation. One trivial approach to solve this problem is to use conservative plans with padded time between actions and an executive that supports timing of actions by advancing to the next action only when its start time has arrived. Such a strategy has indeed been tested with the Networked Vehicle Language (**NVL**) [22].

Moreover, if we consider that the vehicle is connected with other systems only during *communicate* actions, a more advanced executive may deterministically adjust the plan between *communicate* actions. Such plan adjustments may include throttling vehicle speed, changing the traveled path or stopping at the surface, as long as the *communicate* actions, where the vehicles synchronize with other parts of the network, is correctly timed.

Limitations in underwater communication also makes it crucial that part of the temporal execution of the plan for each vehicle is managed locally ensuring local adaptability of vehicle behavior. These problems are precisely where the applicability of a temporal constrained-based planning/execution framework like **T-REX** [9], [10], [11] can be tackled in the near future. While **T-REX** has already been integrated on our LAUVs, coupling multiple **T-REX** enabled vehicles to a shore-based mixed-initiative system as in this paper, is future work. A likely challenge to address in that eventuality, will be the interaction of local and autonomous decision-making with the plan for the ensemble of robots.

In conclusion, we described our work in mixed-initiative control, with a shore-based automated planner synthesizing goals to command multiple AUVs. Abstraction in control is an important objective of this work primarily as a means to control multiple heterogeneous vehicles for distributed problem solving in real-world environments. Finally, we have demonstrated this work by coalescing and abstracting control functions with *one* operator using a well established command/control methodology used in recent field experiments.

Acknowledgements: Rajan and Py are supported by ONR Grant # N00014-14-1-0536 on 'Integrated Autonomy for Long Duration Operations'.

References

[1] D. Yoerger, M. Jakuba, A. Bradley, and B. Bingham, "Techniques for Deep Sea Near Bottom Survey Using an Autonomous Underwater Vehicle," *The International Journal of Robotics Research*, vol. 26, no. 1, pp. 41–54, 2007.

[2] J. P. Ryan, S. Johnson, A. Sherman, K. Rajan, F. Py, H. Thomas, J. Harvey, L. Bird, J. Paduan, and R. Vrijenhoek, "Mobile autonomous process sampling within coastal ocean observing systems," *Limnology & Oceanograhy: Methods*, vol. 8, pp. 394–402, 2010.

[3] J. Gottlieb, R. Graham, T. Maughan, F. Py, G. Elkaim, and K. Rajan, "An Experimental Momentum-based Front Detection for Autonomous Underwater Vehicles," in *ICRA*, 2012.

[4] M. Faria, J. Pinto, F. Py, J. Fortuna, H. Dias, R. M. F. Leira, T. A. Johansen, J. B. Sousa, and K. Rajan, "Coordinating UAVs and AUVs for Oceanographic Field Experiments: Challenges and Lessons Learned," in *ICRA*, 2014.

[5] J. Gonzalez and et.al, "AUV Based Multi-vehicle Collaboration: Salinity Studies in Mar Menor Coastal Lagoon," in *IFAC Workshop on Navigation, Guidance and Control of Underwater Vehicles (NGCUV)*, Porto, Portugal, May 2012.

[6] J. Pinto, P. Calado, J. Braga, P. Dias, R. Martins, E. Marques, and J. Sousa, "Implementation of a control architecture for networked vehicle systems," in *Proc. IFAC Workshop on Navigation, Guidance and Control of Underwater Vehicles*, 2012.

[7] A. Sousa, L. Madureira, J. Coelho, J. Pinto, J. Pereira, J. Sousa, and P. Dias, "LAUV: The man-portable autonomous underwater vehicle," in *Navigation, Guidance and Control of Underwater Vehicles*, vol. 3, no. 1, 2012, pp. 268–274.

[8] J. Bresina, A. Jonsson, P. Morris, and K. Rajan, "Activity Planning for the Mars Exploration Rovers," in *ICAPS*, 2005.

[9] F. Py, K. Rajan, and C. McGann, "A Systematic Agent Framework for Situated Autonomous Systems," in *AAMAS*, 2010.

[10] K. Rajan and F. Py, "T-REX: Partitioned Inference for AUV Mission Control," in *Further Advances in Unmanned Marine Vehicles*, G. N. Roberts and R. Sutton, Eds. The Institution of Engineering and Technology (IET), August 2012.

[11] K. Rajan, F. Py, and J. Berreiro, "Towards Deliberative Control in Marine Robotics," in *Autonomy in Marine Robots*, M. Seto, Ed. Springer Verlag, 2012.

[12] M. Cashmore, M. Fox, T. Larkworthy, D. Long, and D. Magazzeni, "Auv mission control via temporal planning," in *ICRA*, 2014, pp. 6535–6541.

[13] J. Almeida, C. Silvestre, and A. Pascoal, "Cooperative control of multiple surface vessels in the presence of ocean currents and parametric model uncertainty," *International Journal of Robust and Nonlinear Control*, vol. 20, no. 14, pp. 1549–1565, 2010.

[14] O. Schofield, J. Kohut, D. Aragon, L. Creed, J. Graver, C. Haldeman, J. Kerfoot, H. Roarty, C. Jones, D. Webb, and S. Glenn, "Slocum Gliders: Robust and ready," *J. Field Robotics*, vol. 24, pp. 473–485, 2007.

[15] J. Pinto, P. S. Dias, R. Martins, J. Fortuna, E. Marques, and J. Sousa, "The LSTS toolchain for networked vehicle systems," in *OCEANS-Bergen, 2013 MTS/IEEE*. IEEE, 2013, pp. 1–9.

[16] M. Ghallab and D. Nau and P. Traverso, *Automated Planning Theory and Practice*. Elsevier Science, 2004.

[17] M. Fox and D. Long, "PDDL2.1: an extension to PDDL for expressing temporal planning domains," *J. Artif. Intell. Res. (JAIR)*, vol. 20, pp. 61–124, 2003.

[18] A. Gerevini, A. Saetti, and I. Serina, "Planning through stochastic local search and temporal action graphs in LPG," *J. Artif. Intell. Res. (JAIR)*, vol. 20, pp. 239–290, 2003.

[19] J. Benton, A. J. Coles, and A. Coles, "Temporal planning with preferences and time-dependent continuous costs," in *ICAPS*, 2012.

[20] J. Frank and A. Jónsson, "Constraint-based Attribute and Interval Planning," *Constraints*, vol. 8, no. 4, pp. 339–364, oct 2003.

[21] M. M. S. Shah, L. Chrpa, D. E. Kitchin, T. L. McCluskey, and M. Vallati, "Exploring knowledge engineering strategies in designing and modelling a road traffic accident management domain," in *IJCAI*, 2013.

[22] E. R. B. Marques, M. A. Ribeiro, J. Pinto, J. Sousa, and F. Martins, "Towards programmable coordination of unmanned vehicle networks," in *Navigation, Guidance and Control of Underwater Vehicles*, 2015.

A.6 Guiding planning engines by transition-based domain control knowledge

L. Chrpa and R. Barták. Guiding planning engines by transition-based domain control knowledge. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016.*, pages 545–548, 2016.

Proceedings, Fifteenth International Conference on
Principles of Knowledge Representation and Reasoning (KR 2016)

Guiding Planning Engines by Transition-Based Domain Control Knowledge

Lukáš Chrpa
PARK Research Group
School of Computing & Engineering
University of Huddersfield

Roman Barták
Constraint & Logic Programming Research Group
Faculty of Mathematics and Physics
Charles University in Prague

Abstract

Domain-independent planning requires only to specify planning problems in a standard language (e.g. PDDL) in order to utilise planning in some application. Despite a huge advancement in domain-independent planning, some relatively-easy problems are still challenging for existing planning engines. Such an issue can be mitigated by specifying Domain Control Knowledge (DCK) that can provide better guidance for planning engines.

In this paper, we introduce *transition-based DCK*, inspired by Finite State Automata, that is efficient as demonstrated empirically, planner-independent (can be encoded within planning problems) and easy to specify.

Introduction

Despite a huge advancement of domain-independent planning engines, they might still struggle with problems that are easy to solve for a domain-dependent approach because "raw" specification of these problem might not be very informative for domain-independent planning engines. This issue can be (to some extent) addressed by specifying Domain Control Knowledge (DCK) that can guide the search that planning engines perform. DCK can be specified, for instance, in form of Control rules (Minton and Carbonell 1987), Hierarchical Task Networks (HTNs) (Georgievski and Aiello 2015), and Macro-operators (Korf 1985). DCK can be exploited by specifically tailored planning engines such as TALPlanner (Kvarnström and Doherty 2000) for Control Rules, and SHOP2 (Nau et al. 2003)) for HTNs. Some kinds of DCK such as macro-operators can also be (automatically) encoded into planning problems and thus any standard planning engine can exploit such DCK.

In this paper, we introduce *transition-based Domain Control Knowledge* that is inspired by Finite State Automata. Roughly speaking, transition-based DCK represents knowledge about dependencies between planning operators that is used to restrict the number of their instances that can be applied in each step of the planning process. We will show that transition-based DCK can be directly encoded into planning problems, so standard planning engines can reason with it. Like Finite State Automata, transition-based DCK can be specified in a schematical way, which we believe is easy to use for domain engineers. To demonstrate the efficiency of transition-based DCK we use six benchmark domains and six state-of-the-art domain-independent planning engines.

Classical Planning

Classical Planning deals with finding a partially or totally ordered sequence of actions transforming the static and fully observable environment from an initial state to a desired goal state (Ghallab, Nau, and Traverso 2004).

In the classical representation, the environment is described by *predicates* that are constructed in form $\mathit{pred\_name}(x_1, \ldots, x_k)$ such that *pred_name* is a unique predicate name and $x_1, \ldots x_k$ are predicate arguments, where each argument is either a variable symbol or a constant. *Planning states* are defined as sets of grounded predicates. We say that $o = (\mathit{name}(o), \mathit{pre}^+(o), \mathit{pre}^-(o), \mathit{eff}^-(o), \mathit{eff}^+(o), \mathit{cost}(o))$ is a *planning operator*, where $\mathit{name}(o) = \mathit{op\_name}(x_1, \ldots, x_k)$ (*op_name* is a unique operator name and $x_1, \ldots x_k$ are variable symbols, arguments, appearing in the operator) and $\mathit{pre}^+(o), \mathit{pre}^-(o), \mathit{eff}^-(o)$ and $\mathit{eff}^+(o)$ are sets of predicates with variables taken only from $x_1, \ldots x_k$ representing o's positive and negative preconditions, and negative and positive effects, and $\mathit{cost}(o)$ is a numerical value representing o's cost[1]. *Actions* are instances of planning operators. An action a is *applicable* in a planning state s if and only if $\mathit{pre}^+(a) \subseteq s \wedge \mathit{pre}^-(a) \cap s = \emptyset$. Application of a in s (if possible) results in a planning state $(s \setminus \mathit{eff}^-(a)) \cup \mathit{eff}^+(a)$.

A *planning domain model* is specified by a set of constants (domain-specific objects), a set of predicates and a set of planning operators accommodating these constants and predicates. A *planning problem* is specified via a (planning) domain model, a set of constants (problem-specific objects), an initial planning state, and a set of goal predicates. A goal predicate is an *open goal* if it has not yet been achieved in the current planning state. A *solution plan* of the planning problem is a sequence of actions such that their consecutive application starting from the initial planning state results in a planning state containing all the goal predicates.

[1] Implicitly, $\mathit{cost}(o) = 1$.

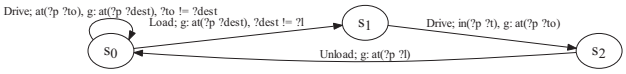


Figure 1: Transition-based DCK for our simple logistic domain.

Running Example

We consider a simple logistic domain, where packages have to be delivered from their initial locations to their goal locations by trucks that can carry at most one package each. No other constraints are defined, i.e., all locations are connected and a package can be carried by any of the trucks. We define three predicates: (at ?x ?l) – an object ?x (either truck or package) is at location ?l; (in ?p ?t) – a package ?p is in the truck ?t; (free ?t) – a truck ?t is empty (no package is in it). Then, we define three planning operators: Drive(?t ?from ?to) – a truck ?t moves from the location ?from to a location ?to; Load(?t ?p ?l) – a package ?p is loaded into the truck ?t at the location ?l; and Unload(?t ?p ?l) – a package ?p is unloaded from the truck ?t at the location ?l.

Transition-based Domain Control Knowledge

Generally speaking, Domain Control Knowledge (DCK) provides a guidance to planning engines and thus makes the planning process more efficient. Inspired by *Finite State Automata*, we designed a *transition-based DCK* that is easy to specify and can be encoded into planning domain models and problems (described later in the text). In principle, our transition-based DCK consists of a set of DCK states and transitions that refer to actions that can be applied under specified conditions in a given planning state. The formal definition of transition-based DCK follows.

Definition 1. *Transition-based DCK is a quadruple* $\mathcal{K} = (S, O, T, s_0)$*, where* S *is a set of DCK states,* $s_0 \in S$ *is the initial DCK state,* O *is a set of planning operators, and* T *is a set of transitions in the form* (s, o, C, s')*, where* $s, s' \in S$*,* $o \in O$ *and* C *is a set of constraints, where each constraint is in the form:*

- $p, \neg p$ *— a predicate* p *must or must not be present in the current planning state*
- $g{:}p$ *— a predicate* p *is an open goal in the current planning state*

In our running example, we assume that the packages are at their initial locations and the trucks are empty. We may observe that i) an empty truck has to be moved only to locations where some package is waiting for being delivered, and ii) if a package is loaded to the truck (in its initial location), then the truck has to move to package's goal location, where the package is then unloaded. Such an observation can be encoded as a transition-based DCK with s_0 as an initial DCK state as depicted in Figure 1.

A (classical) planning problem can be solved by a generic algorithm that starting in the initial planning state, iterates by non-deterministically selecting an action (an instance of a planning operator defined in the domain model of the problem) that is applicable in the current planning state and by applying the action (updating the current planning state) until the goal is reached (all the goal predicates are present in the current planning state). Transition-based DCK can be embedded into the generic algorithm as follows. Let s_Π be the current planning state and $s_\mathcal{K}$ be the current DCK state (we start in the initial planning state and in the initial DCK state). An action a can be selected for being applied if and only if a is applicable in s_Π and there is a transition $(s_\mathcal{K}, o, C, s'_\mathcal{K})$ such that a is an instance of o and for each element in C which is in form $p, \neg p$, or g:p it is the case that a corresponding instance of p is in s_Π, not in s_Π, or is an open goal respectively. After a is applied the current planning state is updated accordingly and $s'_\mathcal{K}$ becomes the current DCK state. The goal predicate is considered to be an open goal only before it is achieved for the first time. Reopening open goals requires more complex encoding and, moreover, we believe that it is not very useful to direct the search toward goals that need to be achieved more than once.

We can easily observe that the augmented generic planning algorithm is sound, i.e., a plan that is returned by the algorithm is a solution plan of the problem given on the input. This is because exploiting transition-based DCK only puts further restrictions on action selection. A *well-defined transition-based DCK* does not prune all solution plans of the problem, so it remains solvable. The DCK from our running example is well defined for problems where none of the packages that has to be delivered is initially loaded in any of the trucks and at least one truck is initially empty.

```
(:action Drive-empty
:parameters (?t - truck ?from ?to ?dest - location ?p - package)
:precondition (and (at ?t ?from)(at ?p ?to)(DCK-state s0)
        (open-goal-at ?p ?dest)(not (= ?to ?dest)))
:effect (and (not (at ?t ?from))(at ?t ?to))
)
(:action Drive-full
:parameters (?t - truck ?from ?to - location ?p - package)
:precondition (and (at ?t ?from)(DCK-state s1)
        (in ?p ?t)(open-goal-at ?p ?to))
:effect (and (not (at ?t ?from))(at ?t ?to)
        (not (DCK-state s1))(DCK-state s2))
)
(:action Load
:parameters (?t - truck ?p - package ?l ?dest - location)
:precondition (and (at ?t ?l)(at ?p ?l)(free ?t)(DCK-state s0)
        (open-goal-at ?p ?dest)(not (= ?to ?dest)))
:effect (and (not (at ?p ?l))(not (free ?t))(in ?p ?t)
        (not (DCK-state s0))(DCK-state s1))
)
(:action Unload
:parameters (?t - truck ?p - package ?l - location)
:precondition (and (at ?t ?l)(in ?p ?t)(DCK-state s2)
        (open-goal-at ?p ?l))
:effect (and (not (in ?p ?t))(free ?t)(at ?p ?l)
        (not (open-goal-at ?p ?l))
        (not (DCK-state s2))(DCK-state s0))
)
```

Figure 2: Modified planning operators (in PDDL) of our simple logistics domain with respect to the transition-based DCK as in Figure 1. Additional arguments and predicates introduced by the DCK are in italics.

Encoding Transition-based DCK into Planning Problems

In the case of our transition-based DCK, we have to encode DCK states, transitions, and constraints under which the transitions can be performed. The encoding will be de-

scribed in the following paragraphs.

DCK states can be encoded by adding a supplementary predicate with one argument, e.g., (DCK-state ?s), into a planning domain model. Concrete DCK states are encoded as domain-specific objects (e.g. s0, s1, s2). The initial DCK state is encoded into the initial planning state by a corresponding instance of the supplementary predicate. In our running example, the initial DCK state is s_0, therefore, (DCK-state s0) will be added to the initial planing state of each planning problem in our simple logistics domain.

Open goals can be encoded by adding supplementary "twin" predicates that have the same arguments as the goal predicates. Instances of these supplementary predicates that correspond to the goal predicates are added into the initial planning state. In our running example, if we have a problem with two goal predicates, (at pkg1 loc1) and (at pkg2 loc2), then the supplementary predicates, (open-goal-at pkg1 loc1) and (open-goal-at pkg2 loc2) are added into the initial planning state. Also, planning operators that achieve goal predicates (have them in the positive effects) are extended by adding the corresponding supplementary predicates into their negative effects. In our running example, the Unload operator achieves goal predicates, i.e., (at ?p ?l), so (open-goal-at ?p ?l) is added to its negative effects. Clearly, if no transition in the DCK considers open goals, there is no need to encode them.

Transitions in our transition-based DCK incorporate information about what planning operators and under which constraints they can be applied in given DCK states and how DCK states change. Such information can be encoded within the planning operators defined in the corresponding domain model. For each planning operator o we identify how many transitions (from T) refer to it, i.e., $t(o) = |\{t \mid t \in T, t = (s, o, C, s')\}|$. If for an operator o, $t(o) = 0$, then o can never be applied and thus will be removed from the domain model. If for an operator o, $t(o) > 1$, then we create $t(o)$ "clones" of o, i.e., we create $t(o)$ operators that are identical to o but having a unique operator name. In our running example, we have two transitions referring to the Drive operator. So, we create two operators, for instance, Drive-empty and Drive-full that have the identical structure to the original Drive operator (arguments, preconditions, and effects).

A transition (s, o, C, s') is encoded into o (or its corresponding "clone") as follows. Ensuring that instances of o can be applied only if s is the current DCK state is done by adding the corresponding supplementary predicate representing the DCK state (i.e., (DCK-state s)) into $pre^+(o)$. If $s' = s$, then the DCK state does not change, so effects of o remain intact. Otherwise, (DCK-state s) is added into $eff^-(o)$ and (DCK-state s') into $eff^+(o)$, so after applying an instance of o, the current DCK state changes to s'. The Load operator from our running example is modified by adding (DCK-state s0) into the positive preconditions as well as into the negative effects, and by adding (DCK-state s1) into the positive effects.

Each constraint $c \in C$ is encoded into o as follows depending on its form:

- p — add p into $pre^+(o)$
- $\neg p$ — add p into $pre^-(o)$
- g:p — add the "open goal twin" predicate of p into $pre^+(o)$

If some of the predicates that are added into the o's precondition contain arguments that are not defined in o, the list of o's arguments is updated accordingly.

The transition-based DCK as defined in Figure 1 is encoded within the planning operators of our simple logistic domain as depicted in Figure 2. Recall that Drive-empty and Drive-full are "clones" of the original Drive operator.

Solution plans of the DCK enhanced problems may not entirely correspond with solution plans of the original problems because of using different operator names for "cloned operators", and using extra arguments for accommodating additional preconditions. Hence, to get a valid solution plan for the original problem we have to rename all "clones" to the name of the corresponding original operators and remove all the extra arguments. For example, if the solution plan of the DCK enhanced simple logistic problem contains an action Drive-full(truck1 loc1 loc2 pkg1), then it has to be renamed to Drive and the extra argument pkg1 has to be removed, so we obtain Drive(truck1 loc1 loc2) which is a valid action for the corresponding original problem.

Experimental Evaluation

Our experiments aim to demonstrate how transition-based DCK influences the planning process in terms of planners' runtimes and quality of solution plans. We encoded the DCK into the domain models and problems of six domains – Barman, CaveDiving, ChildSnack, CityCar, Hiking, and Nomystery. For descriptions of these domains and related transition-based DCKs we specified, the reader is referred to our workshop paper (Chrpa and Barták 2015). The problem sets for these domains were taken from the agile track of the 8th International Planning Competition (IPC-8), except Nomystery, where the problem set was taken from the satisficing track of the IPC-7. All the problem sets consist of 20 problems. For comparing how the DCK influences the planning process, we have used six state-of-the-art planners that accommodate various planning techniques: LAMA (Richter and Westphal 2010), the winner of the IPC-7, and MpC, Probe, Mercury, Yahsp3, and Bfs-f that performed well in the IPC-8 (Vallati, Chrpa, and McCluskey 2014).

For analysis of planners' performance and quality of solution plans, we use IPC score as defined in the IPC-8 (Vallati et al. 2015). With regards to runtime, the score is calculated as follows. For an encoding e of a problem p, $IPC_t(p, e)$ is 0 if p is unsolved in e, and $1/(1+\log_{10}(T_{p,e}/T^*_p))$, where $T_{p,e}$ is the CPU-time needed to solve p in e and T^*_p is the smallest CPU-time needed to solve p in any considered encoding, otherwise. With regard to quality of solution plans (i.e., the sum of costs of all their actions), the score is calculated as follows. For an encoding e of a problem p, $IPC_q(p, e)$ is 0 if p is unsolved in e, and $(Q^*_p/Q_{p,e}))$, where $Q_{p,e}$ is the cost of the solution plan of p in e and Q^*_p is the smallest cost of the solution plan of p in any considered encoding, otherwise. Notice that the maximum IPC score for a particular p and e is 1. As in the agile track of IPC-8, we set the time limit to 5

Planner	Barman				CaveDiving				ChildSnack				CityCar				Hiking				Nomystery			
	Coverage		Δ IPC		Coverage		Δ IPC		Coverage		Δ IPC		Coverage		Δ IPC		Coverage		Δ IPC		Coverage		Δ IPC	
	O	E	T	Q	O	E	T	Q	O	E	T	Q	O	E	T	Q	O	E	T	Q	O	E	T	Q
Lama	16	20	+8.2	+6.9	6	7	+4.1	+1.0	0	19	+19.0	+19.0	5	20	+15.4	+13.9	5	19	+15.9	+13.9	14	14	+0.3	0.0
Mercury	7	20	+13.4	+15.3	2	3	+1.8	+1.0	5	20	+17.5	+15.0	2	20	+17.9	+18.8	8	17	+11.5	+7.0	13	15	+2.1	+2.0
MpC	0	20	+20.0	+20.0	4	4	+1.4	0.0	7	20	+11.4	+10.9	9	20	+12.8	+15.8	7	3	-3.0	-3.9	6	5	-1.2	-1.0
Probe	18	20	-2.7	+2.8	1	7	+6.4	+6.0	0	15	+15.0	+15.0	8	20	+12.8	+17.3	13	19	+10.9	+5.8	5	11	+7.0	+5.9
Yahsp	0	20	+20.0	+20.0	N/A	N/A	N/A	N/A	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	13	10	+0.6	+4.6	8	12	+7.2	+4.1
Bfs-f	20	20	+3.8	-1.5	7	7	+2.0	0.0	8	8	-0.8	-0.2	5	20	+17.5	+16.7	2	14	+13.4	+11.6	14	15	+4.7	+1.3

Table 1: Comparison between original (O) and enhanced (E) domain models. Δ IPC stands for a difference of the time (T) and quality (Q) IPC score of the original and the corresponding enhanced problem encodings (positive numbers refer to better performance/quality of the enhanced problems).

minutes per problem. All the experiments were run on Intel Xeon 2.53 Ghz with 2GB of RAM, CentOS 6.5.

Table 1 presents the results of comparison of planners' performance and quality of solution plans of the original problem encodings and the encodings enhanced by our transition-based DCK. Notice that Yahsp3 could not solve some problem sets (denoted as "N/A" in the table), since it does not fully support negative preconditions. Remarkable results have been achieved in Barman and CityCar domains, where using DCK allowed every planner (except Yashp3 in CityCar) to solve all the problems within the 5 minute limit. The reason is that in these domains the problems can be solved by using the "reach goals one-by-one" strategy that can be easily captured by transition-based DCK. Good results were achieved in the Childsnack domain, where our transition-based DCK can prevent "trapping" in dead-ends. In CaveDiving and Nomystery the results were modest. The reason is that transition-based DCK does not address the "combinatorial" part of these domains. In Hiking, the results of applying transition-based DCK were mixed, i.e., achieving good improvement for Lama, Mercury, Probe, and Bfs-f, while having rather detrimental effect on MpC and Yahsp3. MpC uses a structure, similar to Planning Graph (Blum and Furst 1997), that allows applying more actions in one step. This strategy is useful in Hiking. Transition-based DCK, however, interferes with such a strategy. In terms of quality of solution plans, there are no (or very marginal) differences in the Cave Diving and Nomystery domains (the quality score differs because the number of solved problems differs). This is because there is usually no other way how to solve these problems, so solution plans are very similar (some actions might be in a different order). In the other domains, the quality results are mixed (sometimes DCK enhanced problems lead to better solution plans, sometimes worse). Notice that in Hiking, Yahsp3 was able to provide considerably better plans (often more than 10x) when DCK was considered.

Conclusions

In this paper, we introduced transition-based DCK, inspired by Finite State Automata, that is planner-independent (can be encoded within planning problems), efficient and easy to specify due to its "schematical" representation.

In future, we plan to (semi)automatise the process of extracting transition-based DCK. We believe that taking inspiration from macro-operator generating techniques and/or tools for plan analysis will be a useful step towards such an achievement. Also, we plan to extend the concept of transition-based DCK to non-classical planning (e.g. temporal or continuous planning).

Acknowledgements The research was funded by the UK EPSRC Autonomous and Intelligent Systems Programme (grant no. EP/J011991/1) and by the Czech Science Foundation (project no. P103-15-19877S). The authors would like to acknowledge the use of the University of Huddersfield Queensgate Grid in carrying out this work.

References

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90(1-2):281–300.

Chrpa, L., and Barták, R. 2015. Enhancing domain-independent planning by transition-based domain control knowledge. In *The 33rd Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG)*.

Georgievski, I., and Aiello, M. 2015. HTN planning: Overview, comparison, and beyond. *Artificial Intelligence* 222:124–156.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning, theory and practice*. Morgan Kaufmann.

Korf, R. 1985. Macro-operators: A weak method for learning. *Artificial Intelligence* 26(1):35–77.

Kvarnström, J., and Doherty, P. 2000. TALplanner: a temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence* 30(1-4):119–169.

Minton, S., and Carbonell, J. G. 1987. Strategies for learning search control rules: An explanation-based approach. In *Proceedings of IJCAI*, 228–235.

Nau, D. S.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: an HTN planning system. *Journal of Artificial Intelligence Research (JAIR)* 20:379–404.

Richter, S., and Westphal, M. 2010. The LAMA planner: guiding cost-based anytime planning with landmarks. *Journal Artificial Intelligence Research (JAIR)* 39:127–177.

Vallati, M.; Chrpa, L.; Grzes, M.; McCluskey, T. L.; Roberts, M.; and Sanner, S. 2015. The 2014 international planning competition: Progress and trends. *AI Magazine* 36(3):90–98.

Vallati, M.; Chrpa, L.; and McCluskey, T. L. 2014. *The Eighth International Planning Competition. Description of Participant Planners of the Deterministic Track*.

A.7 Mixed-initiative planning, replanning and execution: From concept to field testing using AUV fleets

L. Chrpa, J. Pinto, T. S. Marques, M. A. Ribeiro, and J. B. de Sousa. Mixed-initiative planning, replanning and execution: From concept to field testing using AUV fleets. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*, pages 6825–6830, 2017.

2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 24–28, 2017, Vancouver, BC, Canada

Mixed-Initiative Planning, Replanning and Execution: From Concept to Field Testing using AUV Fleets

Lukáš Chrpa*†, José Pinto‡, Tiago Sá Marques‡, Manuel A. Ribeiro‡, João Sousa‡,
*Artificial Intelligence Center, Czech Technical University in Prague, Czech Republic
†Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic
‡Laboratório de Sistemas e Tecnologia Subaquática, University of Porto, Portugal
chrpaluk@fel.cvut.cz,{zepinto,tsmarques,maribeiro,jtasso}@fe.up.pt

***Abstract*— Mission planning and execution for autonomous vehicles is crucial for their effective and efficient operation during scientific exploration, or search and rescue missions, to mention a few. Automated Planning has shown to be a useful tool for "high level" mission planning, that is, allocating tasks to vehicles while following given constraints (e.g., energy, collision avoidance).**

In this paper, we focus on making mission planning flexible and robust. That is, a human mission coordinator can modify tasks during the mission execution, so the tasks have to be dynamically reallocated during the process. Moreover, we assume that communication might not be reliable when vehicles are "outside", i.e., performing the tasks, and thus we enforce vehicles to come back to their safe spots regularly. To address these requirements, we have developed two models, namely "all tasks" and "one round", and integrated them to the control software. We have evaluated our approach in a field experiment focused on a mine-hunting scenario.

I. Introduction

Autonomous Underwater Vehicles (AUVs) are becoming affordable and their use becoming ubiquitous for tasks such as ocean exploration, scientific observations, mine hunting, or search and rescue operations. AUVs have already shown their utility, for example, in benthic [1] or upper water column exploration [2], [3]. Automated Planning and Execution has been applied on a single AUV [4], [5] in various settings, most recently in monitoring and maintaining offshore subsea facilities [6]. Similarly, the single vehicle approach has been applied in the space domain [7]. Planning and executing complex missions where a fleet heterogeneous AUVs has to be effectively coordinated remains to a large extent a challenge for human mission operators who often have to perform "high level" control and monitoring manually.

The LSTS laboratory has developed and field-tested tools for the operation of multiple autonomous vehicles [8]. Currently, coordination and control of AUVs is maintained via advanced decision-support tools in the LSTS toolchain [9] that synthesizes onboard AUV control with interleaved "low level" planning and execution. AUVs can accept and execute "high level" commands (e.g., move to a location, sample an object with the camera, etc.) commanded by the on shore control system (NEPTUS) and then use their onboard intelligence to execute these commands autonomously.

In a nutshell, more complex mission planning requires allocation of tasks to vehicles that have needed capabilities, enough power and do not interfere with each other. When the tasks are allocated, the mission is executed in parallel by allocated vehicles. In practice, the human mission coordinator specifies tasks (in the NEPTUS control software, for instance) for a fleet of AUVs (or other types of autonomous vehicles) [7], and then (manually) dispatches these tasks to particular AUVs. In our recent work [10], we have applied Automated Planning techniques that automatically allocated tasks to the vehicles while considering the aforementioned constraints. We have demonstrated the feasibility of such an approach in a field experiment – the human coordinator could allocate tasks s/he has specified to vehicles automatically and these tasks were successfully completed. Although such work was a considerable step forward in use of mixed-initiative planning and control methods, there are some limitations that might hinder its use in real-world settings. In particular, missions are planned and executed in "one shot", that is, all tasks are allocated and the execution goes until all tasks are completed. In real missions, however, the operator should be able to do changes online (add, remove, or modify tasks) and the control system should reflect these changes. For example, the operator receives some interesting data from a surveillance task, s/he then adds several sampling tasks for further investigation. Also, the "one shot" approach is prone to failures (e.g. breakdown of a vehicle, failing to complete some task, etc.) as well as changes of conditions. Moreover, when the vehicles execute the mission they go underwater and communication between them and the on-shore control system might not be possible. With the "one shot" approach, therefore, communication might be impossible until the whole mission is finished. Hence, the system has very limited operationality with regards to addressing failures or changes of conditions.

In this paper, we address the aforementioned limitations. Similarly to the "one shot" model, the human mission coordinator specifies tasks in NEPTUS which calls a domain-independent planning engine that generates a plan that is dispatched to particular vehicles and then executed. Here, the coordinator can add/remove or modify tasks during the mission execution, then NEPTUS calls the planning engine again, a new plan is generated and dispatched to the vehicles. However, one of the assumptions we are dealing with is that communication might not be possible (or is not reliable) when the vehicle is executing the tasks. Hence, in our new model we enforce vehicles to regularly return to their "depots"

(safe spots near the on shore control center from which communication can be reliably established). In practice, new plans are dispatched to vehicles when they return to the "depot". In this paper, we introduce two models, *All Tasks* and *One Round*. The All Tasks model allocates all tasks to the vehicles, so typically they might do several "rounds" (i.e., they are forced to return to their depots several times). The One Round model is motivated by work of Martinez et al. [11] showing that planning for shorter time horizons is more efficient for more complex scenarios. The one round model, therefore, generates plans only for the next round (i.e., when a vehicle returns to its depot it has finished its turn) while maximizing the number of allocated tasks the vehicles will perform. Both models are evaluated on a field experiment in Porto harbor where 3 heterogeneous bottom-mapping AUVs were used in a mine-hunting application scenario where they were used simultaneously to detect potential threats (mines) using sidescan sonars and to verify detected objects as mines or false positives. The experiment demonstrated operationality and robustness of our approach in a multiple heterogeneous AUVs setting, that is, the system automatically allocates tasks to AUVs, monitors the mission execution, and re-plans if changes occur (e.g., the user adds a new task). Hence, we believe that our approach is a considerable step forward for automatizing mission planning and executing in more complex settings.

II. Background

A. Control Software

LSTS, has developed a software tool chain that is used to support the operation of heterogeneous networks of autonomous vehicles [12] composed by DUNE, IMC and NEPTUS. DUNE is embedded on-board the vehicles and manages "low level" control, i.e., localization, trajectory-planning, communication, data logging and command execution. IMC is the middle-ware that defines a common control message set that can be understood by all vehicles and computers in the network using transports such as Wi-Fi, Acoustic Modem or GSM. Finally, NEPTUS is a "high level" decision-support system providing situation awareness and interfaces for human operators to specify tasks for vehicles, simulate and supervise execution, and analyze collected data.

B. Automated Planning

In general, *Automated Planning* deals with a problem of finding a (partially) ordered sequence of actions that transform the environment from a given initial state to a desired goal state [13]. Temporal planning, which we focus on in this paper, considers a fully observable and deterministic environment and "durative actions" making the explicit use of the notion of time.

A *planning task* is composed from a *planning domain model*, which describes the environment and actions, and a *planning problem instance*, which describes the initial state of the environment, the goal to be achieved and the metric to be optimized.

We use PDDL 2.2 [14], an extension of the well known PDDL 2.1 [15] for representing planning tasks. The environment is specified via predicates and numeric fluents. In a state of the environment, predicates that are present in that state (e.g. `(completed t1)`) refer to properties that are true in that state (`t1` has been completed), while numeric fluents represent values of "resources" in that state (e.g. `(completed-tasks)` refers to the number of tasks that have been completed in that state). Note that if a predicate is not present in the state of the environment, we assume that the property it represents does not hold in that state.

Actions are specified via preconditions (i.e., what must hold prior their execution), effects (i.e., what will happen after their execution) and the duration of their execution. Preconditions are specified via sets of literals (e.g., `(at lauv1 scene2)` or `(not (completed t1))`) and binary relations between numeric fluents (e.g., `(>= (completed-tasks) 5)`. Each element can take place either prior to starting action execution, prior to finishing action execution, or over the whole time period when the action is executed. Effects are sets of literals and numeric fluent assignments or adjustments (increase or decrease of fluent's value), where each effect can take place just after starting, or just after finishing action execution.

Specifying problem instances in PDDL 2.2 is straightforward. We need to specify objects (e.g., concrete vehicles, locations, payloads etc.), an initial state, a goal and an optimization metric. An *initial state* consists a set of predicates referring to properties that initially hold in the environment and a set of numeric fluents' initial assignments (e.g., `(= (completed-tasks) 0)`. Notice that we can encode properties that will hold at some (non-initial) time. For this purpose, we use "timed-initial literals", a feature of PDDL 2.2, that allow us to encode such "delayed" properties (e.g., `(at 60 (at lauv1 depot1))` represents that `lauv1` will be at `depot1` at time 60). A *Goal* is specified in a similar way as action's preconditions, i.e., via sets of literals and binary relations between numeric fluents. An *optimization metric* is specified via a numeric fluent that has to be minimized (e.g. `(total-time)`) or maximized (e.g. `(completed-tasks)`.

A *plan* that solves a given planning task is a set of tuples ⟨timestamp,action,duration⟩ such that executing these actions in the corresponding timestamps for given durations (it must always be possible) transforms the environment from the initial state to some state where the goal is satisfied (i.e. a goal state). Notice that quality of plans is measured via the metric they are optimized for (e.g., a plan that completes more tasks has greater value when optimizing for number of completed tasks).

C. Embedding Planning into Control Software

Domain-independent planning engines accept a description of a domain model and a problem instance in some standard language such as PDDL and produces a plan. Noticeably, one domain model is usually used for a class of problem instances. Hence, planning engines and domain models are decoupled and can be understood as standalone components. In practice, this means that the domain model is not tied to a specific planning engine

as well as it allows replacing the planning engine by another without modifications to the model.

In order to exploit these "planning components", the control software (NEPTUS), has to generate a problem instance description (in PDDL 2.2) and has to interpret the resulting plan. Generating problem instances involves a translation of NEPTUS's internal representation of the environment into the PDDL one, which is straightforward given the predicate/numeric fluents representation required by PDDL 2.2. Since plans contain information about what has to be done and when, it is straightforward for NEPTUS to interpret them and distribute corresponding commands to particular vehicles.

When NEPTUS calculates the initial state to feed the planner, some of the vehicles may already be undergoing an autonomous behavior. In order to estimate when these vehicles will be available for the planner, NEPTUS uses a shore-side simulation engine that keeps track of what behavior each vehicle is doing and integrates incoming Wi-Fi and acoustic position reports to maintain an up-to-date estimation of future availability.

It is worth noting that such "planning components" (e.g., a domain model and a planning engine) can be exploited in any robotic system. In principle, the integration of the "planning components" into any robotic system is analogous to the integration we have done with NEPTUS. Environment specification and actions (i.e., a domain model) have to correspond with a particular application (e.g., robots' capabilities).

III. Specifying Domain Models

As demonstrated in our earlier work [10], Automated Planning can be effectively exploited for allocating tasks for heterogeneous AUVs. Chrpa et al.'s model assumes "one shot" planning, i.e., allocating all user specified tasks to AUVs, and then executing the plan until it finishes. Such a model works under the assumptions that i) tasks do not change, ii) communication with AUVs is always possible, and iii) AUVs will not fail during plan execution.

These assumptions limit operationality of the system and might introduce unnecessary overheads for the user. Hence, (i) the user should be able to add/remove/modify tasks during the plan execution (the user might get new data, or requests), (ii) the vehicles are not "error-proof" as they might fail, (iii) other vehicles can join during execution, and (iv) communication with vehicles might not be always possible when vehicles are "away" (e.g., the system cannot send plans to submerged vehicles). Addressing (i)-(iii) involves dynamic re-planning. In other words, when the user adds/removes/changes any task, or a new vehicle joins, the system should re-plan in order to re-allocate the tasks among the vehicles. It similarly applies to situations where the vehicle fails to fulfill a given task (the task is put back into the system), or the vehicle breaks down in which case the system must re-plan as well. Regarding (iv) we assume that communication is always possible when the vehicle is in its "depot". Since we have no guarantee that communication can be established when the vehicle is outside its "depot", we have to define a maximum duration for which the vehicle can be away.

In order to comply with the above requirements, we have developed two domain models, namely *All Tasks* and *One Round*[1]. All Tasks allocates all the tasks to available vehicles, while One Round plans for "one shift", i.e., just for tasks that can be fulfilled before vehicles have to come back to their "depots".

A. All Tasks

Similarly to Chrpa et al.'s model [10], we consider the following object types: *AUVs* (?v), *Locations* (?l), *Objects of Interest* (?o), *Areas of Interest* (?a), *Payloads* (?p) and *Tasks* (?t).

The static part of the environment is specified as follows. Each vehicle is specified by its capabilities, i.e., its speed, payloads it carries, by a location of its "depot", and by the maximum time it can operate before returning to its depot (denoted as (max-to-depot ?v)). By "depot" we mean a safe location where the vehicle can communicate with the on-shore control system. Tasks that are specified by user of the control system map objects or areas of interest with payloads that will be used to collect data. Each object of interest has its location. Areas of interest are specified via entry and exit points and length of the survey path. All locations are interconnected and distances between each two locations are specified.

The dynamic part of the environment specifies the current position of vehicles, whether vehicles collected data with regards to user specified tasks, and how long vehicles are operating since their last visit of their depots (denoted as (from-depot ?v)). We also consider collision avoidance constraints, that is, at most one vehicle can be at a location at time and at most one vehicle can perform the survey action in an area of interest at time. It should be noted that in contrast to Chrpa et al.'s model [10], we do not consider energy consumption. With the maximum time the vehicle can be "on mission" we can estimate how much energy a vehicle needs to make its "round" before returning to its depot and, moreover, when the vehicle has low energy it drops the remaining tasks and returns to the "depot". Hence, we believe it is not necessary to explicitly model energy consumption in our case.

The action types, namely *move, sample, survey, communicate*, are the same as in Chrpa et al.'s model [10], however, there are some differences in their implementation in order to comply with our requirements. In our model, we implement three types of *move* action that specifies vehicle's movement between two locations. These types, namely *move-to-object, move-to-area* and *move-to-depot*, "enforce" the destination location to be a location of object of interest, entry point to area of interest, and vehicle's depot respectively. After the vehicle performs *move-to-object* or *move-to-area* action it has to then perform sampling or surveillance, respectively. This constraint is used to optimize the planning process

[1]The models are available at https://github.com/LSTS/neptus/tree/develop/conf/pddl

by not allowing vehicles to go to locations where they have nothing to do. *Move-to-depot* resets the value of the `(from-depot ?v)` fluent to zero. In the *sample* and *survey* actions the vehicle collects data of a given task by sampling the specified object of interest or surveying the specified area of interest by the specified payload. Clearly, the vehicle must carry the specified payload and be in a corresponding location prior to executing of the sample/survey action. Noticeably, the survey action can consider up to three payloads (there is no vehicle with more than three payloads in our fleet) and thus the vehicle can collect data of up to three tasks referring to the same area of interest. The *communicate* action transmits the data from the vehicle to the control system. The vehicle must be in its depot for the whole time the data transfer takes place. Estimated duration of the *move* and *survey* actions is calculated from vehicle's speed and the distance the vehicle has to travel. Estimated duration of the *sample* and *communication* actions is constant.

In order to enforce the "out of depot" constraint, $(\leq$ `(from-depot ?v)(max-to-depot ?v))` must always hold for each vehicle `?v`. Each action, except *communicate*, increments `(from-depot ?v)` by its duration at the start of its execution. The *move-to-depot* action sets `(from-depot ?v)` to zero at the end of its execution.

Initial states are generated from the current states of the system (e.g, positions of vehicles, locations of objects/areas of interest, tasks description, vehicles' capabilities etc.). Usually, it is assumed that such an information (e.g., `(at lauv1 depot1))` holds at time zero – just before plan execution. However, in case of re-planning, some vehicles might not be immediately available (e.g., performing tasks, communication not established etc.). Then, according to the current plan we can estimate when the vehicle will be available (e.g. when it returns back to its depot). Such information can be encoded into the initial state by using "timed-initial literals" in form `(at 100 (at lauv1 depot1))`, `(at 100 (can-move lauv1))` (so, lauv1 will be available at depot1 in time 100).

Goals are to have all tasks completed and data transmitted to the control system. Plans are optimized for minimizing makespan, i.e., the duration of plans' execution, and the number of *move-to-depot* actions. The latter is used to minimize the number of "rounds" vehicles have to take as well as the number of vehicles necessary to fulfill the tasks.

B. One Round

The main difference in the *One Round* domain model from the *All Tasks* domain model is that the former only allocates tasks that can be completed in one round of each vehicle (i.e., before it returns back to its depot). In particular, the *communicate* action is not modeled and after executing the *move-to-depot* action the vehicle cannot perform any other action. Whereas the latter difference is straightforward – we plan just for one round the vehicle has to operate before returning to its depot – the former difference assumes that communicating collected data will be done implicitly after the vehicle comes back to its depot (since there are no other actions the vehicle can execute in that planning episode).

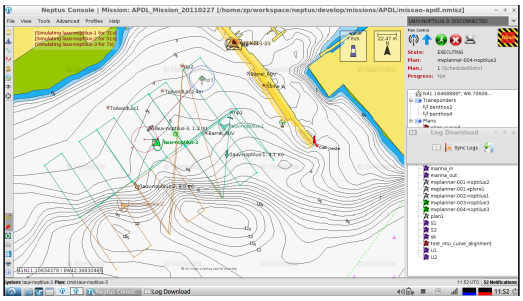


Fig. 1. `NEPTUS` snapshot taken during the experiment when all vehicles were underwater executing sidescan / camera surveys

Goals, in contrast to the *All Tasks* model, are to have all involved vehicles in their depots and a specified minimum number of tasks completed. Also, it is possible to explicitly specify which tasks have to be completed. Notice that it might not be possible to complete all the specified tasks in one round, so the user has to be "lenient" when specifying the tasks that have to be completed in a given round. Plans are optimized for maximizing the number of completed tasks.

IV. Execution

In order to interface fleets of autonomous vehicles, we have extended `NEPTUS` with a plug-in for mixed-initiative planning. When vehicles execute known behavior, `NEPTUS` uses a simulation engine to estimate the future state when the vehicle will be ready to receive more commands. Moreover, the same simulation engine is used to predict vehicle behavior in real-time when they are disconnected due to communication limitations, which is used to augment user awareness.

The developed plug-in allows users to add high-level tasks in a visual intuitive way and periodically converts unallocated tasks, together with current vehicle states into `PDDL`. In order to translate the world state and tasks to `PDDL`, the plug-in (1) determines the capabilities of each available vehicle; (2) estimates the future state where each vehicle will become ready to execute commands; (3) splits tasks that are too large into smaller tasks and (4) calculates task constraints such as distance to other locations, required execution time, its start and end locations. After calculating all this information, the plug-in uses a `PDDL` translator to generate problem specification (in `PDDL`). The generated problem description together with specified domain model (in `PDDL`) are passed to the planner that produces a plan file. `NEPTUS` then converts the plan file into a sequence of vehicle behaviors and monitors its execution.

If during the execution a new planning request arrives while a vehicle is executing the current plan, the vehicle continues executing the current plan until its earliest arrival to its depot. Then, the new plan is assigned to the vehicle (notice that the planning request considers the estimated time when the vehicle will be available). Whenever a vehicle finishes executing its assigned behaviors, `NEPTUS` verifies if the behaviors were executed

correctly (success result) or not. If the vehicle failed to execute its assigned behaviors, the associated tasks become unallocated again. If the vehicle succeeded in the execution, the tasks are signaled as finished and are removed from the interface.

In our previous work, the commands dispatched to the vehicles did not include any timing requirements which resulted in some lag between planned and actual action durations due to variable environment conditions [10]. To address this problem, in this work, we translate move actions into timed waypoint behaviors which specify target locations and absolute times of arrival. As a result, when vehicles move between locations they adapt their speed according to remaining time to arrive at the target location. Moreover, the vehicle clocks are synchronized by GPS time when they surface.

V. Experimental Results

A. Scenario

For validating our concept we opted to apply it to a mine-hunting scenario. Typically, mine-hunting operations are divided into two phases: at phase 1, one or more AUVs carrying a sidescan sonar payload are used to search contacts in the sea floor by mapping the area. At phase 2, a set of AUVs carrying video camera and/or magnetometer revisit the contacts in order to identify the objects as either mines or false positives. Typically, some AUVs carries both sidescan sonar and camera and thus can participate in both phases. In contrast to Chrpa et al. [10], where both phases were planned and executed separately, our approach allow simultaneous execution of both phases.

Figure 3 shows the tasks that were used for these tests. In the first phase (left side of the picture) six sidescan sonar survey tasks were configured each taking around 375 seconds to complete and making up an area of 38400 m^2. On the second phase (right side of the picture), 4 camera sampling tasks were used to revisit and identify the contacts from phase 1. Moreover, an additional multibeam sonar sampling and a multibeam sonar survey were added in two steep areas which appear as shadows in the sidescan sonar surveys[2].

B. Experiment overview

In this work, we have used **LPG-TD** [16] as the planning engine. **LPG-TD** supports `PDDL` 2.2 and scales well for both models, i.e., it usually found a plan in a few seconds. **LPG-TD** is an anytime planner, so it keeps running until the given time elapses (in our case 10 seconds), while keeping improving plans quality (the best quality plan is returned).

For the field test, we used three LAUV autonomous submarines, two of them being equipped with a sidescan sonar and one equipped with a video camera (see Figure 4). These vehicles were deployed to the water and their depots have been selected to be close to the base station for improved communication bandwidth. For monitoring tasks execution while underwater, the vehicles periodically sent (every minute) a state message including

[2]See the video at https://www.youtube.com/watch?v=DYADNTokaXc

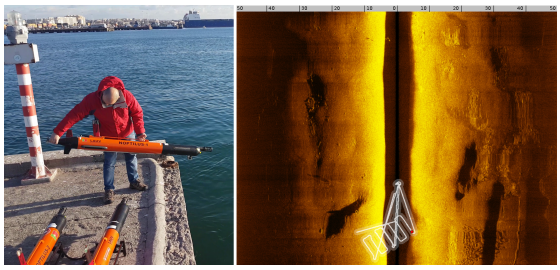


Fig. 2. LAUV vehicles being deployed for the experiment (left) and sidescan data acquired (right)

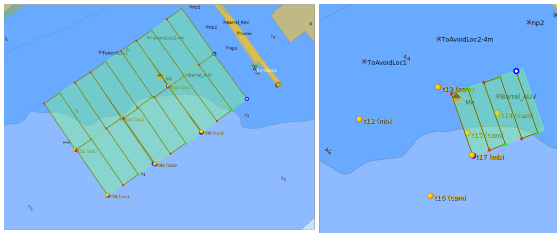


Fig. 3. Tasks to be executed by the vehicles

location, behavior execution progress and remaining fuel. An acoustic modem deployed at the base station is used to receive these data and also to send short commands such as interrupting current behavior and moving to a pre-determined location.

Nominal speed of these vehicles is 1 m/s which is the optimal speed for sonar surveys. While traveling between tasks all vehicle used different depths to prevent collisions and for each task the payload type defined the traveling depth (or altitude from bottom). For instance, while performing sidescan sonar surveys, the vehicles travel at 3 meters from the bottom and when executing multibeam sonar surveys they travel at 2 meters of depth.

C. Replanning and Onboard Adaptation

Regardless of the planning model in use, all the vehicles deployed have the capability to adapt a task's execution according to how late or early they in comparison to what was defined by the planner. If, for instance, a vehicle finishes a survey later than what was expected,

Fig. 4. Sidescan data acquired by the vehicles during phase 1 (One Round model)

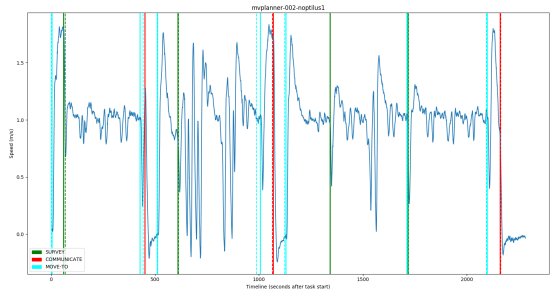


Fig. 5. Task's execution and plan times, and vehicle speed throughout

while moving to another objective, the vehicle will adapt its speed in order to reach it on time.

Figure 5 shows the vehicle's speed throughout a plan execution (time series in blue) and the times at which each task has actually started (vertical colored lines) and what was defined by the plan (vertical dashed line). Most time differences are quite small (less than 3 seconds). Around time 1000, however, there was a noticeable difference between planned and actual end time of a survey task that had started on time. This happened because, during the execution, the ocean floor was too steep and the vehicle had to stop the propeller to prevent collisions and ascend for a while, and then continue the survey. The drops in speed during the survey are visible in Figure 5 between times 600 and 1000. Even though the survey finished a bit later, the vehicle adapted the plan execution and accelerated to reach the next objective on time. To note that the vehicle only adapts its speed while en-route to a new objective, and not during execution of surveys, i.e., if the vehicle is already late while completing, for instance, a survey it won't speed up, rather, it will complete the current objective at the predefined speed and then, when finished, accelerate towards the next one. This is because most sonars require a specific constant speed for the best results.

During the test execution, LAUV-Noptilus-3 reported a temporary depth sensor fault and interrupted during one of the assigned plans (All Tasks model, phase 2). As a result, `NEPTUS` received the error and marked all related tasks as unallocated. Despite the (unintentional) errors, all tasks were again reassigned and executed to completion by the system.

D. All Tasks vs. One Round

In both models, plans for our field experiments were generated within the 10 seconds threshold. We have noticed that the All Tasks model allocates more tasks to vehicles for the first round than the One Round model. For smaller exercises, the All Tasks model seems to be more efficient, however, we believe that for larger exercises (dozens of tasks), the All Tasks model might struggle to generate plans in required time (plans will be longer) while the One Round model should be still able to generate plans in the 10 seconds time limit.

VI. Conclusions and Future Work

In this paper, we have addressed limitations of Chrpa et al.'s model [10] for "high level" task planning in order to allow human coordinators to specify and modify tasks online (even during the mission execution) and to consider communication limitation (the control system might not be able to send commands to vehicles that are executing the tasks). To do so, we have developed two domain models, *All Tasks* and *One Round*. Our concepts have been verified in field experiments, where we demonstrated utility of the introduced approach.

In future, we would like to evaluate our concepts on large exercises ($\sim$100 tasks, $\sim$15 vehicles). Furthermore, we will investigate how the models can be adapted in order to support collaborative tasks, which two or more vehicles have to perform simultaneously.

Acknowledgements: Experiments at sea were funded by MARINFO project (NORTE-01-0145-FEDER-000031). Research was partially funded by the Czech Science Foundation (project no. 17-17125Y).

References

[1] D. Yoerger, M. Jakuba, A. Bradley, and B. Bingham, "Techniques for Deep Sea Near Bottom Survey Using an Autonomous Underwater Vehicle," *The International Journal of Robotics Research*, vol. 26, no. 1, pp. 41–54, 2007.

[2] J. P. Ryan, S. Johnson, A. Sherman, K. Rajan, F. Py, H. Thomas, J. Harvey, L. Bird, J. Paduan, and R. Vrijenhoek, "Mobile autonomous process sampling within coastal ocean observing systems," *Limnology & Oceanograhy: Methods*, vol. 8, pp. 394–402, 2010.

[3] J. Gottlieb, R. Graham, T. Maughan, F. Py, G. Elkaim, and K. Rajan, "An Experimental Momentum-based Front Detection for Autonomous Underwater Vehicles," in *ICRA*, 2012.

[4] K. Rajan and F. Py, "T-REX: Partitioned Inference for AUV Mission Control," in *Further Advances in Unmanned Marine Vehicles*, G. N. Roberts and R. Sutton, Eds. The Institution of Engineering and Technology (IET), August 2012.

[5] M. Cashmore, M. Fox, T. Larkworthy, D. Long, and D. Magazzeni, "Auv mission control via temporal planning," in *ICRA*, 2014, pp. 6535–6541.

[6] N. Palomeras, A. Carrera, N. Hurtós, G. C. Karras, C. P. Bechlioulis, M. Cashmore, D. Magazzeni, D. Long, M. Fox, K. J. Kyriakopoulos, P. Kormushev, J. Salvi, and M. Carreras, "Toward persistent autonomous intervention in a subsea panel," *Auton. Robots*, vol. 40, no. 7, pp. 1279–1306, 2016.

[7] J. Bresina, A. Jonsson, P. Morris, and K. Rajan, "Activity Planning for the Mars Exploration Rovers," in *ICAPS*, 2005.

[8] M. Faria, J. Pinto, F. Py, J. Fortuna, H. Dias, R. M. F. Leira, T. A. Johansen, J. B. Sousa, and K. Rajan, "Coordinating UAVs and AUVs for Oceanographic Field Experiments: Challenges and Lessons Learned," in *ICRA*, 2014.

[9] J. Pinto, P. Calado, J. Braga, P. Dias, R. Martins, E. Marques, and J. Sousa, "Implementation of a control architecture for networked vehicle systems," in *Proc. IFAC Workshop on Navigation, Guidance and Control of Underwater Vehicles*, 2012.

[10] L. Chrpa, J. Pinto, M. A. Ribeiro, F. Py, J. Sousa, and K. Rajan, "On mixed-initiative planning and control for autonomous underwater vehicles," in *IROS*, 2015, pp. 1685–1690.

[11] M. Martínez, F. Fernández, and D. Borrajo, "Planning and execution through variable resolution planning," *Robotics and Autonomous Systems*, vol. 83, pp. 214–230, 2016.

[12] J. Pinto, P. S. Dias, R. Martins, J. Fortuna, E. Marques, and J. Sousa, "The LSTS toolchain for networked vehicle systems," in *OCEANS-Bergen, 2013 MTS/IEEE*. IEEE, 2013, pp. 1–9.

[13] M. Ghallab and D. Nau and P. Traverso, *Automated Planning Theory and Practice*. Elsevier Science, 2004.

[14] S. Edelkamp and J. Hoffmann, "PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition," Albert-Ludwigs-Universitat Freiburg, Institut fur Informatik, Tech. Rep. 195, 2004.

[15] M. Fox and D. Long, "PDDL2.1: an extension to PDDL for expressing temporal planning domains," *J. Artif. Intell. Res. (JAIR)*, vol. 20, pp. 61–124, 2003.

[16] A. Gerevini, A. Saetti, and I. Serina, "Planning through stochastic local search and temporal action graphs in LPG," *J. Artif. Intell. Res. (JAIR)*, vol. 20, pp. 239–290, 2003.

A.8 Outer entanglements: a general heuristic technique for improving the efficiency of planning algorithms

L. Chrpa, M. Vallati, and T. L. McCluskey. Outer entanglements: a general heuristic technique for improving the efficiency of planning algorithms. *J. Exp. Theor. Artif. Intell.*, 30(6):831–856, 2018.

JOURNAL OF EXPERIMENTAL & THEORETICAL ARTIFICIAL INTELLIGENCE
2018, VOL. 30, NO. 6, 831–856
https://doi.org/10.1080/0952813X.2018.1509377

Outer entanglements: a general heuristic technique for improving the efficiency of planning algorithms

Lukáš Chrpa[a,b], Mauro Vallati[c] and Thomas Leo McCluskey[c]

[a]Department of Computer Science, Czech Technical University in Prague, Prague, Czech Republic; [b]Department of Theoretical Computer Science and Mathematical Logic, Charles University in Prague, Prague, Czech Republic; [c]Department of Informatics, University of Huddersfield, Queensgate, UK

ABSTRACT

Domain independent planning engines accept a planning task description in a language such as PDDL and return a solution plan. Performance of planning engines can be improved by gathering additional knowledge about a class of planning tasks. In this paper we present *Outer Entanglements*, relations between planning operators and predicates, that are used to restrict the number of operator instances. Outer Entanglements can be encoded within a planning task description, effectively reformulating it. We provide an in depth analysis and evaluation of outer entanglements illustrating the effectiveness of using them as generic heuristics for improving the efficiency of planning engines.

ARTICLE HISTORY
Received 15 June 2017
Accepted 7 July 2018

KEYWORDS
Classical planning; outer entanglements; domain reformulation; state space pruning

1. Introduction

Automated planning is an important research area of artificial intelligence (AI) where an autonomous entity (e.g. a robot) reasons about the way it can act in order to achieve its goals. AI planning has therefore a great potential for applications where a certain level of autonomy is required such as in the Deep Space 1 (Bernard et al., 2000).

Automated planning involves combinatorial search that, roughly speaking, examines numerous combinations of action sequences in order to find a solution plan (an action sequence transforming the environment from the initial state to some goal state). In the last few decades, there has been a great deal of activity in the research community designing planning techniques and planning engines. The International Planning Competition (IPC)[1] has been staged regularly since 1998 and is increasingly attracting the attention of the AI planning community (Vallati et al., 2015). Thanks to the IPC we have many advanced planning engines, and PDDL (Ghallab et al., 1998) that is a standardised language family for describing planning tasks and standard benchmarks for measuring planners' performance. Along with those planning engines, many novel planning techniques have been proposed, such as heuristic search (Bonet & Geffner, 1999), translating planning tasks into SAT (Kautz & Selman, 1992) just to mention a few.

The performance of planning engines can be improved by extracting and exploiting Domain Control Knowledge (DCK), i.e. additional knowledge about planning tasks, for instance, in the form of Control Rules (Minton & Carbonell, 1987) or Decision Trees (De La Rosa, Celorrio, Fuentetaja, & Borrajo, 2011). DCK, roughly speaking, provides a guidance for planning engines, so they can find solution plans more quickly. The usefulness of learning and exploiting DCK in planning has been demonstrated by Yoon, Fern, and Givan (2008)

CONTACT Lukáš Chrpa chrpaluk@fel.cvut.cz

whose approach that learns DCK from relaxed plans (obtained by solving planning tasks while omitting negative effects of actions) won the best learner award at IPC 2008. However, these types of knowledge are often tied to specific planning engines using a planner-specific language (e.g. as with TALPlanner (Kvarnström & Doherty, 2000)) or a convention on extending the input language to take advantage of them, so that planning engines can maintain some level of domain-independence. On the contrary, knowledge which can be directly encoded into the standard definition language (such as PDDL) is planner independent, so a standard planning engine can straightforwardly exploit it. For example, action-centric DCK can be compiled into PDDL (Baier, Fritz, & McIlraith, 2007). The best known specific type of DCK, macro-operators ('macros'), which encapsulate sequences of operators, can be encoded as normal planning operators, so they can be exploited in a planner-independent way (Chrpa, 2010; Korf, 1985; McCluskey & Porteous, 1997; Newton, Levine, Fox, & Long, 2007). Abstracting planning tasks by their reformulation in order to reveal their hierarchical structures can mitigate 'accidental complexity' of their domain models[2] (Haslum, 2007).

Beside macros, another type of domain-independent DCK are *Entanglements* (Chrpa & Barták, 2009; Chrpa & McCluskey, 2012), which represent relations between planning operators and predicates, aimed at eliminating unpromising alternatives in a planning engine's search space. *Inner Entanglements* (Chrpa & McCluskey, 2012) are relations between pairs of operators and predicates which capture exclusivity of predicate achievement or requirement between the given operators. The vast majority of planning engines such as FF (Hoffmann & Nebel, 2001) or those built on top of the Fast Downward planner (Helmert, 2006) perform a pre-processing step – called grounding – in which they compute all atoms and actions that can be reachable from a given initial state, mutual exclusivity of pair of atoms (those that cannot be both present in any reachable state), and structures such as Causal Graph (Knoblock, 1994). Large grounded representation, i.e. a large number of atoms describing the environment and actions one can perform, poses high CPU time and memory requirements for traditional domain-independent planning engines. Noteworthy, use of some types of DCK such as macros or inner entanglements often exacerbate the problem of large representation, since, for example, macros have more arguments than ordinary operators and thus have much more instances.

Outer Entanglements (Chrpa & Barták, 2009; Chrpa & McCluskey, 2012), we focus on in this paper, aim at reducing the size of problem representation by eliminating possibly unpromising grounded actions. Outer entanglements are relations between planning operators and predicates whose instances are present in the initial state or the goal. These relations capture a useful knowledge that some planning operators are needed only to modify initial situations (e.g. picking up a package at its initial location) or achieve goal situations (e.g. delivering a package to its goal location). Hence, only limited numbers of instances of 'entangled' operators have to be considered in the planning tasks reducing the size and complexity of the state space planning engines have to search through. In particular, eliminating some actions (operators' instances) often makes some atoms unreachable (e.g. a package cannot be in other than initial or goal location). Smaller problem representation increases efficiency of pre-processing planning engines perform. Consequently, state space is smaller too, as some unpromising alternatives in it are eliminated. Hence, planning engines have to make less effort to search through such a reduced state space in order to find solution plans.

Outer entanglements can be encoded in planning tasks, effectively re-formulating them, and thus they are planner independent. Deciding whether a given outer entanglement holds in a planning task is PSPACE complete, the same complexity class as the problem of solving the planning task, hence finding outer entanglements for a planning task is generally as hard as solving the task. On the other hand, outer entanglements are often domain specific rather than task specific. Therefore, we have developed an approximate, heuristic method

that is used to learn outer entanglements from training plans, solution plans of simpler tasks in a given domain. One of the advantages of the learning approach is that we do not have to know *why* a set of outer entanglements holds in a given domain. Arguably, the nature of outer entanglements differs per different domain models as discussed in Section 4.4. On the other hand, the heuristic method follows the premise that outer entanglements generalise well, i.e. if a set of outer entanglements holds for a set of (simpler) planning tasks, then it also holds for the whole class of the planning tasks sharing the same domain model. Being a *heuristic* method, it is possible that the reformulation results in incorrect choices, and there may even be instances where is preferable to use the original search space: we investigate this issue in our experimental evaluation.

Initial work on outer entanglements has been reported in a series of shorter papers detailing the discovery, use and effectiveness of outer entanglements. In this paper, we integrate and extend previous work, with:

- encodings of outer entanglements (Chrpa & Barták, 2009) including formal proofs of their correctness;
- a collected summary of the known complexity results (Chrpa, McCluskey, & Osborne, 2012), and trivial cases where entanglements hold (Chrpa & Barták, 2009);
- case studies in which we investigate what outer entanglements hold, and under what conditions;
- an analysis of the potential impact of outer entanglements on the planning process;
- an approximation method for extracting outer entanglements (Chrpa & Barták, 2009);
- an extensive empirical study of the impact of outer entanglements in the planning process using all the domains from the 6th and 7th IPC's learning track,[3] and 7 state-of-the-art planning engines based on very different principles.

The main empirical findings from this paper are that the use of outer entanglements improves the planning process, often remarkably, through the planner and domain model combinations we experimented with. The results demonstrate that the learning method, despite being heuristically based, often learns a useful set of outer entanglements that generalise well for non-training planning tasks. Also, the thorough experimental analysis provides invaluable lessons from which domain engineers can learn how to extract effective and efficient sets of outer entanglements for their domain models.

The paper is organised as follows. After discussing related work, classical planning is introduced, the required terminology is defined and a BlocksWorld domain running example is introduced. Then, outer entanglements are defined, complexity of their identification and case studies referring to easy and possibly hard instances of outer entanglement identification are discussed. After that a reformulation approach enforcing outer entanglements in a planner-independent way (i.e. any standard planner can make use of outer entanglements) is presented, and an approximation algorithm for learning outer entanglements from training plans is introduced. Empirical analysis of impact of outer entanglements in the planning process is provided after that and then, finally, we conclude and present some future avenues of research.

2. Related work

Generating DCK which can be exploited by planning engines dates back into 1970s when systems such as REFLECT (Dawson & Siklóssy, 1977) were developed. Macros, which encapsulate sequences of ordinary planning operators, are one of the best known types of DCK in classical planning,

because they can be encoded as normal planning operators and thus easily added into planning domain models. Macro-FF CA-ED version (Botea, Enzenberger, Müller, & Schaeffer, 2005), which learns macros through analysis of relations between static predicates, and Wizard (Coles, Fox, & Smith, 2007), which learns macros by genetic algorithms, are good examples of planner-independent macro-learning systems.

While the main disadvantage of using macros is the risk of a significant increase of the branching factor during searching, there are several techniques which are used for reducing the branching factor. One way is to combine macros with another learning technique, specifically aimed at pruning unpromising instances of macros, such as in McCluskey's early work (McCluskey, 1987). The complementary technique here created 'chunks' – learnt relations between initial states and operator preconditions, similar to entanglements by init (a subtype of outer entanglements). The method required a specially extended planner, however, and was aimed at plan space search rather than state space search. Recently, it has been shown that there is a synergy between macros and outer entanglements, in other words, outer entanglements can prune unpromising instances of macros as well as provide heuristics for their generation, which has been demonstrated by MUM (Chrpa, Vallati, & McCluskey, 2014) and OMA (Chrpa, Vallati, & McCluskey, 2015b), where the former learns macros from training plans while the latter generates macros online (without training plans).

Determining action relevance is an important branch of research, which reduces the number of instances of planning operators planning engines have to deal with. The FF planner (Hoffmann & Nebel, 2001) instantiates only actions appearing at some level of relaxed Planning Graph. FastDownward (Helmert, 2006) uses the 'reach-one-goal' idea, i.e. achieves the goals of the planning task consecutively, where the solver focuses only on such actions that may be relevant for a particular goal. Other work focusing on cost-optimal SAS+ planning (Coles & Coles, 2010) prunes irrelevant actions (e.g. actions changing a value of a variable having no dependants from a goal value) or exploits 'tunnel macro-actions' (i.e. if a certain action is executed then there is no other choice than to execute specific actions forming the 'tunnel'). Haslum and Jonsson (2000) proposed a technique for pruning actions whose effects can be acquired by executing a sequence of different actions. Scholz (2004) proposed a method for determining action relevance on problems with acyclic causal graphs. Scholz's method has recently been extended to cover problems with non-acyclic causal graphs (Haslum, Helmert, & Jonsson, 2013). *Expansion Core* is a method which in a node expansion phase (in A* search) restricts on relevant Domain Transition Graphs rather than all of them (Chen & Yao, 2009). The idea of 'expansion cores' is extended into *strong stubborn sets* that guarantee stronger pruning than expansion cores (Wehrle, Helmert, Alkhazraji, & Mattmüller, 2013). Recently, *factored planning* (Brafman & Domshlak, 2013) has been exploited for extending the strong stubborn sets approach and introducing *decoupled strong stubborn sets* that in some cases provide exponentially stronger reductions of the problem (Gnad, Wehrle, & Hoffmann, 2016). In contrast to aforementioned techniques, outer entanglements focus on pruning operator instances that either do not require initial atoms, or do not achieve goal atoms and thus are complementary to aforementioned techniques.

3. Preliminaries

This section is devoted to introducing the terminology that will be used throughout the paper.

3.1. *Classical planning*

Classical planning is concerned with finding a (partially or totally ordered) sequence of actions transforming the static, deterministic and fully observable environment from the given initial state to a desired goal state (Fox & Long, 2003; Ghallab, Nau, & Traverso, 2004).

In the classical representation, a *planning task* consists of a *planning domain model* and a *planning problem*, where the planning domain model describes the environment and defines planning operators while the planning problem defines concrete objects, an initial state and a set of goals. The environment is described by *predicates* that are specified via a unique identifier and terms (variable symbols or constants). For example, a predicate at(?t ?p), where at is a unique identifier, and ?t and ?p are variable symbols, denotes that a truck ?t is at location ?p. Predicates thus capture general relations between objects.

Definition 1: A **planning task** is a pair $\Pi = (Dom_\Pi, Prob_\Pi)$ where a **planning domain model** $Dom_\Pi = (Preds_\Pi, Ops_\Pi)$ is a pair consisting of a finite set of predicates $Preds_\Pi$ and planning operators Ops_Π, and a **planning problem** $Prob_\Pi = (Objs_\Pi, I_\Pi, G_\Pi)$ is a triple consisting of a finite set of objects $Objs_\Pi$, initial state I_Π and goal G_Π.

Let ats_Π be the set of all **atoms** that are formed from the predicates $Preds_\Pi$ by substituting the objects $Objs_\Pi$ for the predicates' arguments. In other words, an atom is an **instance** of a predicate (in the rest of the paper when we use the term instance, we mean an instance that is fully *grounded*). A **State** is a subset of ats_Π, and the **initial state** I_Π is a distinguished state. The **goal** G_Π is a non-empty subset of ats_Π, and a **goal state** is any state that contains the goal G_Π.

Notice that the semantics of state reflects the full observability of the environment. That is, that for a state s, atoms present in s are assumed to be true in s, while atoms not present in s are assumed to be false in s.

Planning operators are 'modifiers' of the environment. They consist of *preconditions*, i.e. what must hold prior operators' application, and *effects*, i.e. what is changed after operators' application. *Actions* are instances of planning operators, i.e. operators' arguments as well as corresponding variable symbols in operators' preconditions and effects are substituted by objects (constants). Planning operators capture general types of activities that can be performed. Similarly to predicates that can be instantiated to atoms to capture given relations between concrete objects, planning operators can be instantiated to actions to capture given activities between concrete objects.

Definition 2: **A planning operator** is a tuple $o = (\ name(o),\ pre(o),\ eff^-(o),\ eff^+(o))$ is specified such that $name(o) =\ op\_name(x_1, \ldots, x_k)$, where op_name is a unique identifier and $x_1, \ldots x_k$ are all the variable symbols (arguments) appearing in the operator, $pre(o)$ is a set of predicates representing o's precondition, $eff^-(o)$ and $eff^+(o)$ are sets of predicates representing o's negative and positive effects. Notice that all the predicates in o's definition must be defined in $Preds_\Pi$ of the corresponding domain model. **Actions** are instances of planning operators that are formed by substituting objects, which are defined in a planning problem, for operators' arguments as well as for corresponding variable symbols in operators' preconditions and effects. An action $a = (\ pre(a),\ eff^-(a),\ eff^+(a))$ is **applicable** in a state s if and only if $pre(a) \subseteq s$. If possible, application of a in s, denoted as $\gamma(s, a)$, results in a state $(s \setminus\ eff^-(a)) \cup\ eff^+(a)$.

A solution of a planning task is a sequence of actions transforming the environment from the given initial state to a goal state.

Definition 3: A **solution plan**, or shortly **plan**, of a planning task $\Pi = (Dom_\Pi, Prob_\Pi)$, where $Prob_\Pi = (Objs_\Pi, I_\Pi, G_\Pi)$, is a sequence of actions $a_1, \ldots, a_n$ (all actions are instances of planning operators defined in Dom_Π) such that $G_\Pi \subseteq \gamma(\ldots \gamma(I_\Pi, a_1), \ldots, a_n)$.

3.2. *Blocksworld domain*

We briefly introduce a model representing the *BlocksWorld* domain (Slaney & Thiébaux, 2001), which is one of the best known planning domains, that will be used as a running example in the paper.

The BlocksWorld domain describes an environment where we have a finite number of blocks, one table with unlimited space, and one robotic hand. A block can be either stacked on another block, placed on the table or held by the robotic hand. No block can be stacked on more than one block at the same time as well as no more than one block can be stacked on a block at the same time. The robotic hand can hold at most one block. The BlocksWorld domain model consists of four operators: pickup(?x) refers to a situation when the robotic hand picks up a block ?x from the table, putdown(?x) refers to a situation when the robotic hand puts down the block ?x it is holding to the table, unstack(?x ?y) refers to a situation when the robotic hand unstacks a 'clear' block ?x from a block ?y, and stack(?x ?y) refers to a situation when the robotic hand stacks the block ?x it is holding to a 'clear' block ?y. As mentioned before, planning operators are instantiated by substituting constants (objects) for variable symbols that appear in operators' definition. For example, putdown(?x) can be instantiated by substituting a, which refers to a concrete block 'a', for ?x. We then obtain an action putdown(a) that requires the robotic hand to hold the block a, and the effect is that the block a is placed on the table, the block a is clear (no other block is stacked on it), and the hand no longer holds it.

4. Outer entanglements

This section formally introduces outer entanglements and provides theoretical analysis of them.

4.1. *Introduction*

Outer Entanglements are relations between planning operators and predicates whose instances are present in the initial state or the goal of some solution plan. In a BlocksWorld planning task, there exists a solution plan where the unstack actions may only unstack blocks from their initial positions (e.g. if on(a b) holds in the initial state, then unstack(a,b) ca be present in the plan) and the stack actions may only stack blocks to their goal position. Notice that blocks can be temporarily put on the table (there are no space limits on the table). Formally speaking, an *entanglement by init* will capture that if an atom on(a b) is to be achieved for a corresponding instance of the operator unstack(?x ?y) (unstack(a b)), then the atom is present in the initial state. Similarly, an *entanglement by goal* will capture that an atom on(b a) achieved by a corresponding instance of the operator stack(?x ?y) (stack(b a)) is present in the goal. For illustration, see Figure 1. The outer entanglements relation, i.e. the entanglements by init and goal, are defined as follows.

Definition 4: Let Π be a planning task, where I_Π is the initial state and G_Π is the goal. Let o be a planning operator and p be a predicate defined in the domain model of Π. We say that operator o is **entangled by init (resp. goal)** with a predicate p in Π if and only if $p \in pre(o)$ (resp. $p \in eff^+(o)$) and there exists π, a solution plan of Π, such that for every action $a \in \pi$ being an instance of o and for every atom p_{gnd} being an instance of p, it holds: $p_{gnd} \in pre(a) \Rightarrow p_{gnd} \in I_\Pi$ (resp. $p_{gnd} \in eff^+(a) \Rightarrow p_{gnd} \in G_\Pi$). We also say that π **satisfies** the entanglement (by init or goal) conditions.

Henceforth, entanglements by init and goal are denoted as **outer entanglements**.

Notice that the definition allows for initial atoms to be deleted/re-achieved during the plan, without falsifying the entanglement relationship with an operator instance requiring such an atom later in the plan.

Also, a single outer entanglement requires only the existence of one solution plan of the given planning task where the entanglement conditions are met. However, different entanglements might hold in different solution plans. It is practical to consider a set of outer entanglements for a planning task rather than a single one, which means that there must exist a solution plan in which all

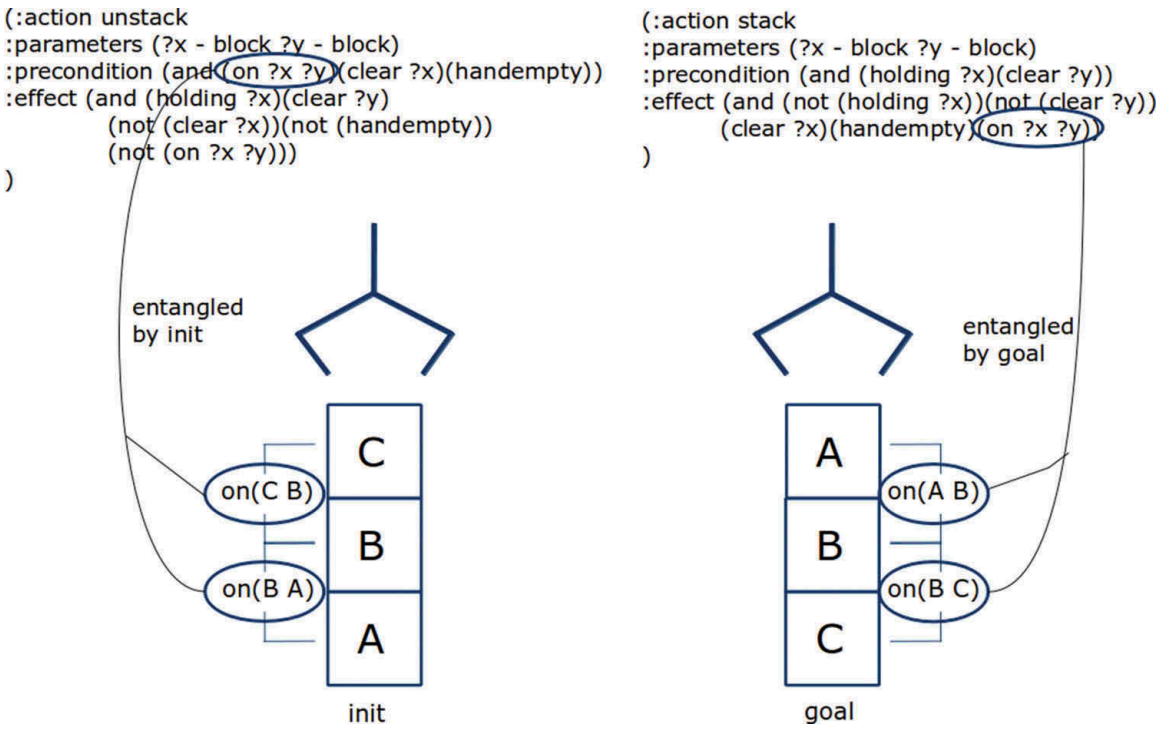


Figure 1. An illustrative example of outer entanglements. On the left hand side, unstack is entangled by init with on, and, on the right hand side, stack is entangled by goal with on.

entanglements from the set hold. Also, in practice, outer entanglements are domain – or class of tasks specific rather than task specific. The above definition can be extended to reflect these aspects.

Definition 5: Let Π be a planning task. We say that a set of outer entanglements ENT_Π holds for Π if and only if there exists a solution plan of Π which satisfies all the entanglements from ENT_Π.

Similarly, $ENT_\mathcal{P}$ holds for a set of planning tasks $\mathcal{P}$ if and only if $ENT_\mathcal{P} = \cap_{\Pi \in \mathcal{P}} ENT_\Pi$.

Both the aforementioned BlocksWorld-related outer entanglements hold for every BlocksWorld planning task with unlimited table space.

4.2. *Intractability of deciding on entanglements*

Landmark theory (Hoffmann, Porteous, & Sebastia, 2004) is a useful framework for studying structures of planning tasks. We will use a fragment of the landmark theory to prove intractability (PSPACE completeness) of deciding whether a given outer entanglement holds in a given task. *Landmarks* are atoms which must be achieved at some point in every solution plan of a given planning task. *Action landmarks* are actions which must be applied at some point in every solution plan of a given planning task. Deciding whether atoms are landmarks as well as whether actions are action landmarks is PSPACE complete (Hoffmann et al., 2004)

The intractability (PSPACE completeness) of deciding whether a given outer entanglement holds is proved by the following theorem.

Theorem 1: *Let Π be a planning task, o be a planning operator and p a predicate defined in the domain model of Π. The problem of deciding whether o is entangled by init (resp. goal) with p in Π is PSPACE complete.*

Proof. First, we show that the problem of deciding whether *o* is entangled by init (resp. goal) with *p* in Π belongs to the PSPACE class. To do this, we reformulate Π by encoding the given entanglement as described in Section 5.2. Hence, the decision problem of whether the given outer entanglement holds can be encoded as a planning task, i.e. the entanglement holds if

and only if the reformulated planning task is solvable. We know that we can solve planning tasks in polynomial space, hence this decision problem belongs to PSPACE. Next, we reduce (in polynomial time) the problem of deciding whether an action a is an action landmark in a planning task Π', which is PSPACE complete, to the problem of deciding whether o is entangled by init (resp. goal) with p in Π. Let o' be a planning operator defined in the domain model of Π' such that a is its instance. Let p be a predicate which has the same variable symbols (arguments) as operator o' and without loss of generality we assume that p is not defined in the domain model of Π'.

To decide that an action a is an action landmark in a planning task Π' by exploiting entanglements by init, we modify Π' in a following way. We create a planning operator o as a modification of o', that is, $o = (\ name(o'),\ pre(o') \cup \{p\},\ eff^-(o'),\ eff^+(o'))$. Then, we create a planning operator o'' such that o'' has the same variable symbols (arguments) as o', $pre(o'') = eff^-(o'') = \emptyset$ and $eff^+(o'') = \{p\}$. Finally, we create a planning task Π by modifying Π' in such a way that o' is removed, and o and o'' are added into the set of operators, p is added into the set of predicates, and all the instances of p but one that has the same arguments as the action a are added into the initial state. We can see that o is entangled by init with p in Π if and only if a is not an action landmark in Π'. This is, because the entanglement tells us that there exists a solution plan of Π' where a is not present. The missing instance of p can be achieved by a corresponding instance of o'', so Π remains solvable even if a is an action landmark, however, the entanglement does not hold.

To decide that an action a is an action landmark in a planning task Π' by exploiting entanglements by goal, we modify Π' in a following way. We create a planning operator o as a modification of o', that is, $o = (\ name(o'),\ pre(o'),\ eff^-(o'),\ eff^+(o') \cup \{p\})$. Then, we create a planning task Π by modifying Π' in such a way that o' is removed, and o is added into the set of operators, p is added into the set of predicates, and all the instances of p but one that has the same arguments as the action a are added into the initial state and into the goal. We can see that o is entangled by goal with p in Π if and only if a is not an action landmark in Π'. This is as in the previous case, because the entanglement tells us that there exists a solution plan of Π' where a is not present. If a must be applied in all solution plans of Π', then the instance of p which is missing in the goal of Π is always achieved and thus the entanglement does not hold.

Clearly, modification of Π' in both cases is done in polynomial time. Hence, since the problem of deciding whether a is a landmark action in Π' is PSPACE complete, the problem deciding whether modified o is entangled by init (resp. goal) with p in Π, which belongs to PSPACE, is PSPACE complete as well.

Intractability of deciding whether a single outer entanglement holds for a given planning task implies intractability of deciding whether a set of outer entanglements holds for that task.

Corollary 1: *Let e_1 and e_2 be outer entanglements that hold in a planning task Π. The problem of deciding whether a set $\{e_1, e_2\}$ holds in Π is PSPACE complete.*

Proof. Without loss of generality, let Π_{e_1} be a planning task obtained by reformulating Π considering e_1 (see Section 5.2). Then, the problem of deciding whether $\{e_1, e_2\}$ holds in Π is equivalent to the problem of deciding whether e_2 holds in Π_{e_1} which is PSPACE complete.

4.3. *Special cases*

Despite the intractability, there are some cases where we can trivially identify outer entanglements (hereinafter referred as trivial outer entanglements). The following situations refer to special cases where there is no way to violate outer entanglements in the planning process. However, trivial outer entanglements do not provide any new domain-specific information and hence we do not have to consider them in the reformulation.

For instance, if a predicate p is not achieved by any operator defined in the domain model (e.g. p is a static predicate), then any operator having p in its precondition is entangled by init with p.

Lemma 1: *Let* Π *be a planning task, p be a predicate and Ops be the set of planning operators defined in the domain model of* Π. *If* $p \notin eff^+(o)$ *for every* $o \in Ops$, *then for every* $o \in Ops$ *it is the case that o is entangled by init with p in* Π *if and only if* $p \in pre(o)$.

Another trivial example where outer entanglements can be easily determined is when all instances of a predicate are present in the initial state or the goal.

Lemma 2: *Let* Π *be a planning task, I its initial state and G its goal. Let Ops be the set of planning operators and p be a predicate defined in the domain model of* Π *such that all possible instances of p are present in I (resp. G). Then, an operator* $o \in Ops$ *is entangled by init (resp. goal) with p in* Π *if and only if* $p \in pre(o)$ *(resp.* $p \in eff^+(o)$*).*

4.4. *Case studies*

In the BlocksWorld domain, which we also used as a running example, we can identify two non-trivial outer entanglements. The operator unstack is entangled by init with the predicate on and the operator stack is entangled by goal with on. A typical task (re-stacking the blocks from initial stacks to goal stacks) can be solved as follows. Blocks are unstacked from their initial positions and put down on the table until all the blocks are on the table. Then, we pick the blocks up and stack them on their goal positions (in the right order). We can see that both the entanglements hold for such solution plans. However, limiting the space on the table (in some modification of the domain) might invalidate these entanglements since due to lack of table space we might be forced to temporarily stack blocks on other blocks. Clearly, if the table space is greater or equal the number of blocks, or if goal stacks of blocks are reverted initial stacks of blocks, the entanglements still hold. However, in a general case deciding whether one or both entanglements hold for a given planning task having the modified domain model can be as hard as solving the task.

In the well-known ZenoTravel domain, which addresses the problem of transporting passengers by planes between cities, we can observe that the operator board is entangled by init with the predicate at and the operator debark is entangled by goal with at. A typical problem can be solved as follows. Each passenger can board an aircraft at the location of origin (if no aircraft is there, then it will arrive from a different location), then the aircraft flies to passenger's destination location where the passenger debarks. The entanglements hold in such solution plans. However, modifying the domain by constraining the locations where a particular aircraft can fly might invalidate the entanglements which will be the case when some passenger would have to change the aircraft at some (non-initial) location. Deciding whether the entanglements (or one of them) hold is easy in the modified domain, since for each passenger we can check whether there is a direct flight or not. Similarly, we can identify outer entanglements in the similar logistic-based domains.

Although we identified some domain-specific cases where identifying (non-trivial) outer entanglements is easy, we are still missing a more general domain-independent approach for identifying a subclass of non-trivial outer entanglements in polynomial time. We believe that analysing structure of planning tasks (e.g. relations between planning operators, mutexes) can be useful for identifying some non-trivial outer entanglements.

5. The use of outer entanglements to speed-up the planning process

This section is devoted to practical use of outer entanglements.

5.1. *Motivation*

The reason for introducing the outer entanglement relation was to form the basis of a tool for eliminating potentially unnecessary instances of planning operators and thus reduce 'overheads' for planning engines (Chrpa & Barták, 2009). This follows the observation that in many domains some

operators are needed only to modify the initial state of the object, or achieve the goal state of the object. Given the example of the BlocksWorld domain (see Figure 1 in Section 4), we can see that since the unstack operator is entangled by init with the on predicate only the instances unstack(b a) and unstack(c b) are necessary, so we can prune the rest of unstack's instances because they are not necessary to find a solution plan. Similarly, we can see that since the stack operator is entangled by goal with the on predicate only the instances stack(a b) and stack(b c) are necessary, so we can prune the rest of stack's instances. Usefulness of such pruning can be demonstrated in the following way. Given n blocks, we can have at most $n \cdot (n-1)$ instances of stack or unstack (we do not consider instances when a block is unstacked from or stacked on itself – e.g. stack(a a)). Considering both the entanglements, we can have at most $n-1$ instances of the stack or unstack operators. In summary, while in the original setting, the number of operators' instances grows quadratically with the number of blocks, considering outer entanglements reduces the growth of the number of operators' instances to linear. Consequently, the state space (i.e. the number of reachable states) can be also reduced. In the BlocksWorld case, a block cannot be stacked on another block unless it is its initial or goal configuration. Hence, when the given outer entanglements are applied there are only at most two (other) blocks a block can be stacked on at any point of the planning process. Otherwise (in the original encoding), a block can be stacked on $n-1$ other blocks (excluding itself) at any point of the planning process.

Outer entanglements are encoded by supplementary static predicates that are added into preconditions of operators involved in the outer entanglement relation. Most of existing planning engines generate operators' instances in pre-processing, i.e. they perform grounding. Static predicates are only useful at this stage; they are useful in filtering unreachable operators' instances,[4] however, static predicates do not provide any valuable information in search, so planners are compiling them away after grounding. Hence, introducing supplementary static predicates does not increase the number of atoms planners have to deal with during the search. Reducing the number of actions planners have to consider during the search reduces the branching factor and thus 'narrows' the search space. Moreover, memory requirements for planners can be often considerably lowered.

However, outer entanglements might cause some actions to become irreversible. For example, if we allow unstacking blocks only from their initial positions (captured by the entanglement by init mentioned before), then we cannot recover from a situation where a block becomes eventually stacked on an 'incorrect' block. Also by having a 'symmetrical' entanglement by goal between the operator stack and the predicate on which allows stacking blocks only on their goal position, we might not recover from a situation where the goal tower of blocks is being built from 'the middle'. Hence, outer entanglements may introduce dead-ends which might be detrimental for some planning techniques, especially those based on local search.

5.2. *Reformulating planning tasks*

To exploit outer entanglements during the planning process we have to develop a specific planner, modify an existing one, or we have to reformulate planning tasks in such a way that outer entanglements are enforced during the search. The last option is planner independent because, as we will show later, reformulation does not require any features which are not provided within classical (STRIPS) planning (see Section 3).

Encoding outer entanglements is done by introducing static predicates that eliminate instances of operators that do not 'comply' with these entanglements (for more background details, see (Chrpa & Barták, 2009)). Let Π be a planning task, I be its initial state and G its goal. Let an operator o be entangled by init (resp. goal) with a predicate p (o and p are defined in the domain model of Π) in Π. Then the task Π is reformulated as follows:

(1) Create a predicate p' (not defined in the domain model of Π) having the same arguments as p and add p' into the domain model of Π.

(2) Modify the operator o by adding p' into its precondition. p' has the same arguments as p which is in precondition (resp. positive effects) of o.
(3) Create all possible instances of p' which correspond to instances of p listed in I (resp. G) and add the instances of p' to I.

Adding p', which is in fact a static predicate, into precondition of o causes that instances of o that are 'prohibited' by the entanglement become unreachable. Figure 2 depicts the encoding of an entanglement by init between the unstack operator and the predicate on. In our terminology, unstack(?x ?y) refers to o, on(?x ?y) to p and stai_on(?x ?y) to p'. Correctness of the reformulation is formally proved as follows.

Proposition 1: *Let Π be a planning task, o be a planning operator and p be a predicate (o and p are defined in the domain model of Π) such that o is entangled by init (resp. goal) with p in Π. Let Π' be a planning task obtained by reformulating Π using the previous approach. π' is a solution plan of Π' if and only if π' is a solution plan of Π that satisfies the entanglement conditions (see Definition 4).*

Proof. Hereinafter, we will refer to modified o as o'. Adding predicates only into a precondition of an operator does not affect the result of application of its instances. For each ground substitution ξ (mapping variable symbols to constants) it holds that applying $\xi(o')$ in some state s (if possible) results in the same state as applying $\xi(o)$ in s (o and o' have the same variable symbols). From this, we can observe that if π' is a solution plan of Π', then π' is a solution plan of Π. For each ground substitution ξ (mapping variable symbols to constants) it holds that $\xi(p') \in I' \leftrightarrow \xi(p) \in I$ or $\xi(p') \in I' \leftrightarrow \xi(p) \in G$, respectively ($I'$ is the initial state of Π'). No instance of p' can be achieved or deleted during the planning process, since no operator defined in the domain model of Π' has p' in its positive or negative effects. Hence, for every o's instance $a \in \pi'$ and p's corresponding instance $p_{gnd} \in pre(a)$ it is the case that $p_{gnd} \in pre(a) \rightarrow p_{gnd} \in I$ (resp. $p_{gnd} \in pre(a) \rightarrow p_{gnd} \in G$). From this, π' also satisfies the entanglement conditions in Π (see Definition 4). Also, given that each instance of p present in I (resp. G) has its 'twin', i.e. a corresponding instance of p' present in I', only instances of o that violate the entanglement are pruned. Therefore, if π' is a solution plan of Π that satisfies the entanglement conditions, then π' is a solution plan of Π'.

5.3. *Extracting entanglements from training plans*

Deciding whether a given outer entanglement holds is generally PSPACE complete as well as deciding whether the set of outer entanglements hold (as discussed in Section 4.2). Trivial entanglements, which can be identified easily (see Section 4.3), are not informative and thus not considered for task reformulation. Therefore, we have to devise an effective approximation technique for extracting sets of outer entanglements. We assume that tasks having the same domain model have a similar structure, so the same set of outer entanglements holds in all of them. Hence, we can select a representative set of simple tasks for each domain model as training tasks, so those can be solved easily by standard planning engines. Generated training plans, that are the solutions of these training tasks, are then explored in order to find what entanglements hold in them.

The above approach can be formalised as follows. Let $\mathcal{P}$ be a class of planning tasks that has the same domain model. Let $\mathcal{P}_T \subset \mathcal{P}$ be a set of training tasks. In our approximation method, we assume that $ENT_{\mathcal{P}_T} = ENT_{\mathcal{P}}$, in other words, a set of outer entanglements valid on training planning tasks is also

```
(:action unstack
:parameters (?x - block ?y - block)
:precondition (and (on ?x ?y)(clear ?x)(handempty)(stai_on ?x ?y))
:effect (and (holding ?x)(clear ?y)
             (not (clear ?x))(not (handempty))(not (on ?x ?y)))
)
```

Figure 2. An example of the encoding of an entanglement by init between the unstack operator and the on predicate.

valid on the whole class of planning tasks. This assumption is a potential source of incompleteness, since using a set of outer entanglements that does not hold for some planning tasks may make a task unsolvable within that reformulation. On the other hand, planning tasks having the same domain model are of similar structure (e.g. they differ only by number of objects), which is the case of the most of IPC benchmarks. This provides evidence that selecting a small set of these tasks such that selected tasks are easy but not trivial, can mitigate the heuristic nature of the method, and thus support the assumption. A thorough empirical study that also explores these issues is provided in Section 6.

Determining whether a set of outer entanglements holds in all the training plans is often not a very efficient way to determine a useful set of outer entanglements, as previously discussed in the literature (Chrpa & Barták, 2009). There are two main reasons. First, training plans might contain redundant actions or very sub-optimal sub-plans which can prevent detecting some useful entanglements. Second, there might be several strategies how a task can be solved, where only some of these lead into discovery of some useful entanglements. For example, in BlocksWorld, we might 'put aside' blocks in two different ways: put them on the table, or stack them on other blocks. Only the former way leads to the discovery of two useful outer entanglements (i.e. unstack is entangled by init with on and stack is entangled by goal with on). Using optimal planners might alleviate the issue related to sub-optimal plans but it might be computationally very expensive even for training tasks. Moreover, using optimal planners might not handle the 'strategy issue'.

Algorithm 1 Extracting a set of outer entanglements from training plans.

Require: a set of training tasks with corresponding solution plans (training plans), flaw ratio η
Ensure: a set of outer entanglements *ENT*

initialize_ent_arrays(); {create empty arrays $entI, entG$ of size [Ops, Preds]}
initialize_op_counter(); {create an empty array *counter* of size [Ops]}
for each training plan $\pi = \langle a_1, \ldots a_n \rangle$ **do**
  for $i := 1$ to n **do**
    for each $p \in pre(a_i)$ **do**
      if $p \in I$ **then**
        $entI[is\_inst(a_i), is\_inst(p)] + +;$
      end if
    end for
    for each$p \in eff^+(a_i)$ **do**
      if $p \in G$ **then**
        $entG[is\_inst(a_i), is\_inst(p)] + +;$
      end if
    end for
    $counter[is\_inst(a_i)] + +;$
  end for
end for
$ENT = \emptyset$
for each$(o, p) \in$ [Ops,Preds] such that $counter(o) > 0$ **do**
  if not trivial $e_I(o, p)$ **and** entI$[o, p]/counter(o) \geq 1 - \eta$ **then**
    $ENT = ENT \cup \{e_I(o, p)\}$
  end if
  if not trivial $e_G(o, p)$ **and** entG$[o, p]/counter(o) \geq 1 - \eta$ **then**
    $ENT = ENT \cup \{e_G(o, p)\}$
  end if
end for

Introducing a *flaw ratio* $\eta \in [0;1]$ which is a parameter referring to an allowed percentage of 'flaws' in training plans can identify outer entanglements that can be discovered in plans that are somehow 'close' to the training plans. Let η be a flaw ratio, then the outer entanglements are extracted as follows:

$$e_I(o,p) \Leftrightarrow \frac{entI[o,p]}{counter[o]} \geq 1-\eta \tag{1}$$

$$e_G(o,p) \Leftrightarrow \frac{entG[o,p]}{counter[o]} \geq 1-\eta \tag{2}$$

The method for extracting sets of outer entanglements in training plans is presented in our previous work (Chrpa & Barták, 2009). For every action we check how many times instances of predicates in its precondition and positive effects, respectively, correspond with atoms in the initial state and the goal, respectively, of the given training task. This information is then used for extracting a set of outer entanglements. This idea is elaborated in Algorithm 1. We define an array *counter*, which stores information about how many instances of given operators occur in the training plans, arrays *entI*, *entG*, which count how many times the conditions of entanglement by init or goal are satisfied for pairs of operators and predicates (Lines 3–17). Function $is\_inst(arg)$ returns either an operator if *arg* (action) is an instance of it or a predicate if *arg* (atom) is an instance of it. Then, a set of outer entanglements is extracted according to a given flaw ratio η (equations (1) and (2)) while ignoring trivial outer entanglements (Lines 18–26).

Algorithm 1 requires linear time with respect to the lengths of given training plans if the number of atoms in actions' preconditions and effects is much lower than lengths of training plans, so it can be bounded by a constant.

Algorithm 2 Extraction of outer entanglements with the flaw ratio.

Require: *init-fr* (the initial value of flaw ratio), *step* (a decrement of flaw ratio)
Ensure: reformulated planning tasks
1: generate training plans
2: $\eta = init\text{-}fr + step$
3: **repeat**
4: $\eta = \max(0, \eta - step)$
5: extract entanglements by Alg. 1 considering η
6: generate reformulated training tasks
7: **until** $\eta = 0$ **or** all the reformulated training tasks are solvable
8: generate reformulated (testing) tasks

Introducing the flaw ratio (η) might invalidate the assumption that the extracted set of outer entanglements (by Algorithm 1) holds for the training tasks. Hence, the assumption must be verified after the set of outer entanglements is extracted. This idea is elaborated in Algorithm 2. A value of flaw ratio η is initially set to *init-fr+step* (Line 2). The main loop (Lines 3–7) iteratively decreases η by the decrement *step* (Line 4), extracts a set of outer entanglements by Algorithm 2 (Line 5), reformulates the training tasks according to the approach described in Section 5.2 (Line 6) and tries to solve these reformulated training tasks. Failing to solve any of the reformulated training tasks indicates that the extracted set of entanglements does not hold for all the training tasks (so, we have to continue by going back to Line 3). Clearly, if $\eta = 0$ then, the training plans are also solution plans of the reformulated training tasks.

6. Experimental evaluation

This section is devoted to the empirical evaluation of the impact of outer entanglements in the plan generation process. The aims of the experiments are: (i) to analyse the impact of outer entanglements on state-of-the-art planning engines; (ii) to assess how different training plans influence extraction of outer entanglements; and (iii) to measure the influence of outer entanglements on grounding, i.e. how reduced is the size of the search space.

6.1. *Experimental setup*

In order to perform our analysis, we selected a number of planners according to (i) their performance in the IPCs, and (ii) the variety of techniques they exploit. Selected planners are: *Metric-FF* (Hoffmann, 2003), *LPG-td* (Gerevini, Saetti, & Serina, 2003), *LAMA* (Richter & Westphal, 2010; Richter, Westphal, & Helmert, 2011), *Probe* (Lipovetzky & Geffner, 2011; Lipovetzky, Ramirez, Muise, & Geffner, 2014), *MpC* (Rintanen, 2012, 2014), *Yahsp3* (Vidal, 2014), and *Mercury* (Domshlak, Hoffmann, & Katz, 2015).

For the empirical evaluation purposes we selected all the domains used in the learning tracks of IPC-6 and IPC-7; since outer entanglements are automatically extracted domain-specific knowledge, the learning track benchmarks seem to be the most appropriate. This test set is thus independent, open, and gives a relatively wide coverage.

In each domain, the planning tasks have the same domain model and thus differ only by planning problem specifications. Henceforth, *training problems* denote tasks that are used for learning entanglements, and *testing problems* denote tasks that are used as benchmarks. In the learning track of IPC-7 (Coles et al., 2012), a set of training problems is not explicitly provided and thus the training problems have to be generated by provided problem generators.

In Machine Learning, it is important to have a good quality training set in order to maximise the outcome of the learning process. From the planning perspective, training plans should capture the important structural aspects that are generalisable to the whole class of planning tasks. According to the observation made by Chrpa, Vallati, and Osborne (2013) sets of extracted entanglements often do not change with increasing number of training problems. Similar observations have been made when configuring portfolios of planners (Núñez, Borrajo, & Linares López, 2012). On the other hand, using very few training problems increases the risk of extracting outer entanglements that do not generally hold (we might be 'lucky' to have a very atypical problem as a training one). Following these observations, 5 training problems per domain were used. Regarding complexity of training problems, there are some aspects that should be taken into account. If training plans are too short, it indicates that their structure might be over-constrained and thus not typical for tasks in a given class. Consequently, we might extract some outer entanglements that do not hold for such 'typical' tasks. On the other hand, obtaining long training plans might be too time consuming or even impossible, since planning is computationally very expensive. Hence, we have experimentally observed – by conducting preliminary investigations on a disjoint set of benchmarks, and by considering results from literature (Chrpa & Barták, 2009; Chrpa & McCluskey, 2012) – that a reasonable size for training problems is when the length of their solution plans is at least 20 in average. With larger number of defined operators in the domain model the length of training plans should be higher (more operators yield longer solution plans).

The benchmark planners were used to generate training plans. The flaw ratio (η) was initially set to 0.2, and, in case of any of the training problems became unsolvable after incorporating outer entanglements,[5] the flaw ratio was iteratively reduced by the decrement of 0.05 until the set of extracted outer entanglement held for all training problems, or the flaw ratio dropped to 0.0 (for details, see Algorithm 2). Although in the literature (Chrpa & Barták, 2009; Chrpa & McCluskey, 2012) the flaw ratio is set to 0.1, we observed on some preliminary experiments, performed on a small set of benchmarks (not included in the rest of this experimental analysis) that such a value is too conservative. On the other hand, setting the value above 0.2 led to extraction of outer entanglements that often did not hold in the training problems.

A CPU-time cutoff of 900 s (15 min, as in learning tracks of IPC) was used for both learning and testing runs. All the experiments were run on 3.0 Ghz CPU machine with 4GB of RAM. In this experimental analysis, IPC scores as defined in IPC-7 are used. For a planner $\mathcal{C}$ and a problem p, $Time(\mathcal{C}, p)$ is 0 if p is unsolved, and $1/(1 + \log_{10}(T_p(\mathcal{C})/T_p^*))$, where $T_p(\mathcal{C})$ is the CPU time needed by planner $\mathcal{C}$ to solve problem p (if the actual CPU time is less than 1 s, then $T_p(\mathcal{C}) = 1$, i.e. 1 s is considered as a minimum CPU time needed to solve any problem) and T_p^* is the CPU time needed by the best considered planner, otherwise. Similarly, $Qual(\mathcal{C}, p)$ is 0 if p is unsolved, and $N_p^*/N_p(\mathcal{C})$, where $N_p(\mathcal{C})$ is the cost of the plan, solution of p, obtained by $\mathcal{C}$ and N_p^* is the minimal cost of the solution plan of p among all the considered planners, otherwise. The IPC score on a set of problems is given by the sum of the scores achieved on each considered problem.

6.2. *Experimental results: the learning phase*

Table 1 shows the different sets of outer entanglements that were extracted by using considered planners for generating training plans. The planners used for generating training plans are arranged into sets, where each set contains all the planners that generate the same set of outer entanglements. Where applicable, the sets are ordered according to the $\supset$ relation, which denotes the superset relation between corresponding sets of outer entanglements. For example, in Bw, we have extracted two different sets of outer entanglements. The first set was extracted from training plans generated by Probe, the second was extracted from training plans generated by either LAMA or Mercury or Yahsp. The first set is a superset of the second one. Notice that using training plans generated by either FF or LPG or MpC has led to an empty set of outer entanglements (i.e. no non-trivial outer entanglements have been extracted). Hereinafter, a *planner*-plan set of entanglements will denote a set of outer entanglements obtained by using training plans extracted by a given *planner*, i.e. an FF-plan set of entanglements is obtained by using plans extracted by FF. We have made the following observations:

- no outer entanglements were detected at all in the nPuzzle domain, hence we have omitted the results from this domain in the rest of the analysis;
- in five domains, namely Barman, Rovers, Satellite, Sokoban and Spanner, the sets of outer entanglements were the same for all the planners;
- in Matching-bw and Thoughtful, there were considerable differences among the planners;
- in Parking and Thoughtful, there is not a set that contains all the outer entanglements (i.e. there is not a superset to all the other sets)

Table 1. Sets of extracted outer entanglements according to planners whose training plans were used. $\forall$ denotes the set outer entanglements consisting of all the planners. The $\supset$ relation denotes the superset relation between corresponding sets of outer entanglements.

Domain	Outer Entanglements
Barman	$\forall$
Bw	{Probe} $\supset$ {Lama,Mercury,Yahsp}
Depots	{FF,Lama,LPG,MpC,Probe}
Gold-m	$\forall \backslash$ {Yahsp} $\supset$ {Yahsp}
Gripper	$\forall \backslash$ {Yahsp} $\supset$ {Yahsp}
Matching-Bw	{Lama} $\supset$ {FF,Mercury} $\supset$ $\begin{matrix}\{\text{LPG}\}\\ \{\text{MpC, Yahsp}\}\end{matrix}$ $\supset$ {Probe}
Parking	{FF}, {Lama}
Rovers	$\forall$
Satellite	$\forall$
Sokoban	$\forall$
Spanner	$\forall$
Thoughtful	{Probe} $\supset$ {FF}, {MpC} $\supset$ {LPG}, {LAMA}, {Mercury}, {Yahsp}
TPP	{FF,LPG,Mercury} $\supset$ {Lama,MpC,Probe,Yahsp}

Pruning power of outer entanglements along with numbers of extracted outer entanglements is shown in Table 2. Ratios of instantiated atoms and actions by the Probe planner in reformulated vs. original problems are presented (e.g. a value of 0.7 means that 70% of atoms/actions are considered in the reformulated tasks). Notice that Mercury- and Probe-sets in Thoughtful do not hold for 6 and 4 testing problems, respectively. In these problems, the reachability check did not reach the goal, so these (reformulated) problems are unsolvable. Table 2 therefore refers to reduction of the size of problem representation when outer entanglements are applied.

Remarkably, different outer entanglements have different pruning power. For example, in Rovers five entanglements prune only about 11% of actions and 5% of atoms, while one entanglement in Bw (set no. II) prunes about 97% of actions and 94% of atoms.

Table 2. Numbers of entanglements by init (EI) and by goal (EG) per set. Ratios of instantiated atoms and actions by the Probe planner in reformulated vs. original testing problems (in ascending order per domain). ∀ denotes the set consisting of all the planners.

No.	Set	EI	EG	Atoms	Actions
Barman					
I	∀	1	1	0.54	0.53
Bw					
I	{Probe}	1	1	0.06	0.02
II	{Lama,Mercury,Yahsp}	0	1	0.06	0.03
Depots					
I	{FF,Lama,LPG,MpC,Probe}	2	1	0.28	0.07
Gold-miner					
I	∀\{Yahsp}	3	0	1.00	0.90
II	{Yahsp}	2	0	1.00	1.00
Gripper					
I	∀\{Yahsp}	2	1	0.71	0.07
II	{Yahsp}	2	0	1.00	0.54
Matching-bw					
I	{Lama}	1	4	0.25	0.08
II	{FF,Mercury}	0	4	0.25	0.10
III	{LPG}	0	3	0.63	0.44
IV	{MpC,Yahsp}	0	3	0.63	0.44
V	{Probe}	0	2	0.63	0.56
Parking					
I	{Lama}	1	0	1.00	0.79
II	{FF}	1	0	1.00	0.79
Rovers					
I	∀	2	3	0.95	0.89
Satellite					
I	∀	0	1	0.52	0.98
Sokoban					
I	∀	2	0	0.97	0.90
Spanner					
I	∀	1	0	1.00	1.00
Thoughtful					
I	{Yahsp}	6	0	1.00	0.70
II	{FF}	8	0	1.00	0.79
III	{Lama}	5	1	1.00	0.81
IV	{MpC}	10	0	1.00	0.81
V	{LPG}	6	0	1.00	0.92
VI	{Mercury}	7	1	N/A	N/A
VII	{Probe}	15	0	N/A	N/A
TPP					
I	{FF,LPG,Mercury}	3	0	0.40	0.03
II	{Lama,MpC,Probe,Yahsp}	2	0	0.40	0.09

6.3. *Experimental results: the testing phase*

Table 3 gives us an overview of performance of planners on the original testing problems and problems reformulated by different sets of outer entanglements we extracted during the learning phase (see Tables 1 and 2). In particular, the results simulate a 'competition' between different encodings for each domain and planner. Coverage denotes how many tasks (out of 30) were solved in the given time-limit (15 min) for each configuration. Remarkably, in Bw, Gripper, and Matching-Bw, the use of outer entanglements allowed some planners to solve all 30 problems in the given time-limit in spite of the fact that they did not solve any task using the original encoding. The IPC score gives a relative evaluation that ranges per task from 0, i.e. the task has not been solved, to 1, i.e. the task has been solved in the smallest CPU-time (or in 1 s) or the solution plan is of the best quality. Hence, if the IPC score is equal (or very close to) the coverage, then the tasks were solved in (nearly) the smallest time or (nearly) the best quality among the encodings. For example, in Depots, the I set provides the best results against the original encoding among almost all the planners.

In summary, in the vast majority of cases, using outer entanglements has positive impact on the planners' performance. For example, in Bw, Depots, Matching-bw, Gripper and TPP the impact is remarkable. Table 4 summarises the results across the planners for each set of outer entanglements as well as original encodings and ranks them according to achieved score in three categories – coverage, speed and quality. As it can be seen from Table 2, the sets of outer entanglements are ordered according to their pruning power, i.e. the set I is the most pruning set of outer entanglements. Also, as Table 1 shows in all the cases, except Parking and Thoughtful, set I contains all the outer entanglements extracted in a particular domain, i.e. I is a superset of all other sets of outer entanglements.

In three domains out of 13, namely Barman, Parking and Sokoban, the original encoding achieved the best overall results, although in Sokoban, the original encoding yielded to worse coverage. In Thoughtful, the set III yielded to the best performance, while the set I unperformed even the original encoding. In TPP, the set II was the best in the coverage and quality metrics, while the set I was the best in the speed metric. In the rest of domains, the set I shows the best performance according to all the criteria.

Figure 3 shows the coverage performance of the considered planners when exploiting the original encodings, and the encodings enhanced with the I sets of outer entanglements. Results are cumulative across all the testing benchmarks and demonstrate that coverage increased overall for each planner when the I sets of outer entanglements are used. The largest impact can be observed on the performance of the MpC planner, where the coverage is increased by 125 instances (32.1%). We believe that such an improvement was achieved because outer entanglements reduced, often considerably, memory requirements. On the contrary, Yahsp coverage performance is less affected by the exploitation of the I sets of outer entanglements: 39 more instances are solved (10.0%).

6.4. *Analysis of the results*

The aim of outer entanglements is to (i) eliminate unpromising instances of planning operators, which, consequently, reduces the branching factor, and (ii) reduce the size of task representation, which, consequently, can also reduce the size of the state space. Given the planner-independent nature of outer entanglements, i.e. they can be encoded directly in the planning task, any standard planning engine can benefit from them. The most significant impact on planners' performance is given by outer entanglements in Bw, Depots, Gripper, Matching-bw and TPP. In these domains, the I sets of outer entanglements had the strongest pruning power, in particular, they eliminated more than 90% of actions (see Table 2). Also, the number of atoms was apart of the Gripper domain reduced by at least 60%. The results (see Tables 3 and 4) demonstrate that the performance gain of

Table 3. Comparing planners' performance in terms of (C)overage, (S)peed IPC score and (Q)uality IPC score on the (O)riginal and reformulated tasks by different sets of outer entanglements. The best results per planner and domain are highlighted.

	FF			LPG			Lama			Probe			MpC			Mercury			Yahsp		
Set	C	S	Q	C	S	Q	C	S	Q	C	S	Q	C	S	Q	C	S	Q	C	S	Q
Barman																					
O	0	0.0	0.0	0	0.0	0.0	0	0.0	0.0	4	3.3	4.0	0	0.0	0.0	**24**	**24.0**	**23.9**	0	0.0	0.0
I	0	0.0	0.0	0	0.0	0.0	0	0.0	0.0	**24**	**24.0**	4.**23.1**	0	0.0	0.0	3	2.4	3.0	0	0.0	0.0
Bw																					
O	0	0.0	0.0	24	8.0	16.1	25	10.3	16.0	25	8.0	21.4	0	0.0	0.0	19	9.6	7.7	28	17.4	5.1
I	**30**	**30.0**	**30.0**	**30**	29.3	**30.0**	28	**28.0**	**28.0**	**30**	**29.8**	**29.9**	**30**	**29.7**	**30.0**	**30**	**26.6**	**30.0**	**30**	**28.9**	**30.0**
II	21	12.6	11.0	**30**	**29.6**	16.1	**29**	19.9	15.2	**30**	28.4	29.2	**30**	24.3	22.6	29	25.5	11.1	**30**	27.5	5.2
Depots																					
O	1	0.3	0.7	11	4.0	9.5	0	0.0	0.0	**30**	12.8	27.4	18	6.1	15.1	0	0.0	0.0	21	8.9	4.4
I	**30**	**30.0**	**30.0**	**30**	**30.0**	**29.8**	**27**	**27.0**	**27.0**	**30**	**29.5**	**29.7**	**30**	**30.0**	**30.0**	**27**	**27.0**	**27.0**	**30**	**30.0**	**30.0**
Gold-miner																					
O	**30**	24.8	**30.0**	**30**	**30.0**	**29.5**	**30**	**30.0**	14.2	**30**	**30.0**	29.4	**30**	**30.0**	24.0	**30**	28.1	19.5	25	24.0	24.1
I	**30**	**30.0**	27.6	**30**	**30.0**	29.4	**30**	**30.0**	**29.6**	**30**	**30.0**	**29.5**	**30**	**30.0**	**27.3**	**30**	**30.0**	**30.0**	**30**	**30.0**	**27.3**
II	**30**	24.8	**30.0**	**30**	**30.0**	**29.5**	**30**	**30.0**	28.2	**30**	**30.0**	29.4	**30**	**30.0**	27.3	**30**	27.9	19.5	28	25.1	27.0
Gripper																					
O	0	0.0	0.0	**30**	16.1	27.1	0	0.0	0.0	0	0.0	0.0	0	0.0	0.0	0	0.0	0.0	0	0.0	0.0
I	**3**	**3.0**	**3.0**	**30**	**30.0**	29.9	0	0.0	0.0	**30**	**30.0**	**30.0**	**30**	**30.0**	**30.0**	0	0.0	0.0	0	0.0	0.0
II	0	0.0	0.0	**30**	21.0	**29.9**	0	0.0	0.0	0	0.0	0.0	15	7.6	14.9	0	0.0	0.0	0	0.0	0.0
Matching-bw																					
O	12	4.7	10.2	21	10.4	15.0	22	17.4	15.7	15	6.6	9.2	0	0.0	0.0	9	4.7	6.4	14	9.4	6.0
I	**30**	**30.0**	**29.8**	**30**	**29.6**	**29.4**	**30**	**29.8**	**29.6**	**30**	**29.7**	**30.0**	**30**	**30.0**	**29.8**	**30**	**29.9**	**30.0**	**30**	**30.0**	**30.0**
II	**30**	25.9	24.0	**30**	26.9	25.3	**30**	26.0	21.5	**30**	25.2	19.5	**30**	**30.0**	22.7	**30**	27.5	19.5	**30**	28.1	14.4
III	25	13.6	19.5	27	22.3	19.0	28	21.1	20.8	21	14.0	13.5	22	9.4	9.6	25	15.0	14.1	26	19.9	11.9
IV	27	17.8	21.4	**30**	12.3	23.6	29	20.5	20.7	25	14.3	16.0	25	14.8	11.7	28	17.5	17.3	28	24.1	12.8
V	21	10.1	16.7	27	16.7	20.2	28	19.4	20.3	16	9.9	10.3	12	6.0	5.1	24	14.0	12.9	25	18.6	11.3
Parking																					
O	**7**	5.2	**6.6**	0	0.0	0.0	**9**	**8.5**	**9.0**	3	2.8	2.8	**5**	**5.0**	**5.0**	6	5.5	6.0	0	0.0	0.0
I	5	4.6	4.8	0	0.0	0.0	2	1.5	1.4	1	1.0	1.0	1	0.5	0.8	**8**	**7.7**	**6.3**	0	0.0	0.0
II	**7**	**6.5**	6.4	0	0.0	0.0	2	2.0	1.5	**5**	**4.8**	**4.6**	4	3.4	3.5	1	0.6	0.7	**5**	**5.0**	**5.0**
Rovers																					
O	0	0.0	0.0	**28**	**26.7**	**27.9**	28	25.4	27.7	28	26.5	27.6	6	4.2	5.9	**24**	23.6	**24.0**	**30**	21.4	**30.0**
I	0	0.0	0.0	26	25.8	25.9	**29**	**29.0**	**28.9**	**29**	**29.0**	**28.7**	**23**	**23.0**	**22.9**	**24**	**23.9**	**24.0**	**30**	**30.0**	**30.0**
Satellite																					

(*Continued*)

Table 3. (Continued).

	FF			LPG			Lama			Probe			MpC			Mercury			Yahsp		
Set	C	S	Q	C	S	Q	C	S	Q	C	S	Q	C	S	Q	C	S	Q	C	S	Q
O	0	0.0	0.0	**30**	27.4	**30.0**	3	3.0	2.9	0	0.0	0.0	**1**	**1.0**	**1.0**	**20**	**19.9**	**20.0**	16	15.4	16.0
I	0	0.0	0.0	**30**	30.0	**30.0**	**4**	**4.0**	**4.0**	0	0.0	0.0	**1**	0.9	**1.0**	20	**20.0**	20.0	**16**	**16.0**	**16.0**
Sokoban																					
O	18	16.0	17.5	**28**	**25.9**	23.9	**19**	**18.1**	**18.0**	**24**	20.6	**22.2**	**30**	**28.6**	**28.3**	**19**	**18.1**	**18.0**	25	20.2	23.8
I	**22**	**19.9**	**20.5**	26	21.4	**24.4**	**19**	14.6	15.3	**24**	**22.4**	21.7	**30**	27.7	27.9	**19**	14.6	15.3	**28**	**26.2**	**24.5**
Spanner																					
O	0	0.0	0.0	**30**	29.9	**30.0**	0	0.0	0.0	0	0.0	0.0	**30**	29.9	**30.0**	0	0.0	0.0	0	0.0	0.0
O	0	0.0	0.0	**30**	29.9	**30.0**	0	0.0	0.0	0	0.0	0.0	**30**	29.9	**30.0**	0	0.0	0.0	0	0.0	0.0
Thoughtful																					
O	17	12.9	16.0	0	0.0	0.0	25	19.4	22.3	20	16.7	18.1	0	0.0	0.0	21	17.3	19.7	8	6.3	6.0
I	17	11.6	15.3	1	0.6	0.9	23	20.0	18.8	18	15.1	13.7	0	0.0	0.0	21	18.7	19.0	2	2.0	1.7
II	18	16.7	17.2	1	0.6	0.9	25	19.4	22.3	21	15.6	18.5	**2**	**2.0**	**2.0**	22	20.5	21.3	17	14.6	16.2
III	**27**	**25.3**	**26.2**	0	0.0	0.0	**28**	**24.5**	**27.4**	**25**	**19.5**	**23.7**	0	0.0	0.0	**27**	**24.2**	**26.1**	**23**	**18.9**	**21.8**
IV	17	14.0	15.9	1	1.0	1.0	20	17.7	17.8	21	19.2	18.5	0	0.0	0.0	19	16.9	17.7	6	4.5	3.6
V	15	11.2	14.1	0	0.0	0.0	25	18.9	22.5	17	14.1	15.8	0	0.0	0.0	20	16.3	18.8	12	9.8	9.8
VI	16	13.5	13.8	**17**	**15.3**	**17.0**	15	14.0	13.2	15	12.1	12.9	1	1.0	1.0	15	13.6	12.9	13	9.4	11.8
VII	11	7.4	8.7	11	11.0	9.5	10	7.2	8.1	8	5.9	6.2	1	1.0	1.0	10	7.3	7.6	10	9.2	9.0
TPP																					
O	0	0.0	0.0	1	0.3	0.6	16	6.1	15.3	14	5.1	13.8	11	4.7	10.6	19	8.3	19.0	20	12.7	19.9
I	0	0.0	0.0	27	**26.8**	26.3	**30**	**30.0**	28.9	**30**	**30.0**	27.0	**21**	**19.7**	**20.8**	**30**	**30.0**	26.0	**30**	**30.0**	29.4
II	**3**	**3.0**	**3.0**	**29**	20.7	**27.7**	**30**	20.0	**29.2**	**30**	18.7	**29.7**	18	14.5	17.4	**30**	22.2	**28.9**	**30**	26.2	**29.8**

Table 4. Ranking the cumulative results for (O)riginal tasks and reformulated tasks by different sets of outer entanglements in coverage, and speed and quality IPC score.

Coverage		Speed		Quality		Coverage		Speed		Quality	
Set	Count	Set	Score	Set	Score	Set	Count	Set	Score	Set	Score
Barman						**Rovers**					
O	28	O	27.3	O	27.9	I	161	I	160.7	I	160.4
I	27	I	26.4	I	26.1	O	144	O	127.7	O	143.1
Bw						**Satellite**					
I	208	I	202.3	I	207.9	I	71	I	70.9	I	70.9
II	199	II	167.8	II	119.2	O	70	O	66.6	O	69.9
O	121	O	53.4	O	66.2	**Sokoban**					
Depots						I	168	O	147.3	O	151.7
I	204	I	203.5	I	203.4	O	163	I	146.9	I	149.6
O	81	O	32.0	O	57.1	**Spanner**					
Gold-miner						I/O	60	I	60.0	I/O	60.0
I	210	I	210.0	I	200.7	I/O	60	O	59.9	I/O	60.0
II	208	II	197.9	II	190.9	**Thoughtful**					
O	205	O	196.9	O	170.8	III	130	III	112.4	III	125.1
Gripper						II	102	II	88.5	II	95.8
I	93	I	93.0	I	92.8	VI	92	VI	78.9	VI	82.4
II	45	II	28.6	II	44.8	O	91	IV	73.3	O	82.1
O	30	O	16.1	O	27.1	V	89	O	72.6	V	81.0
Matching-bw						IV	84	V	70.2	IV	74.5
I/II	210	I	209.0	I	208.7	I	82	I	60.8	I	69.3
I/II	210	II	189.6	II	146.9	VII	61	VII	49.0	VII	50.2
IV	192	IV	123.6	IV	123.4	**TPP**					
III	174	III	115.4	III	108.4	II	169	I	166.5	II	165.7
V	153	V	94.6	V	96.7	I	167	II	125.4	I	158.5
O	93	O	53.2	O	62.4	O	80	O	37.3	O	79.2
Parking											
O	30	O	27.0	O	29.3						
II	24	II	22.3	II	21.7						
I	17	I	15.3	I	17.2						

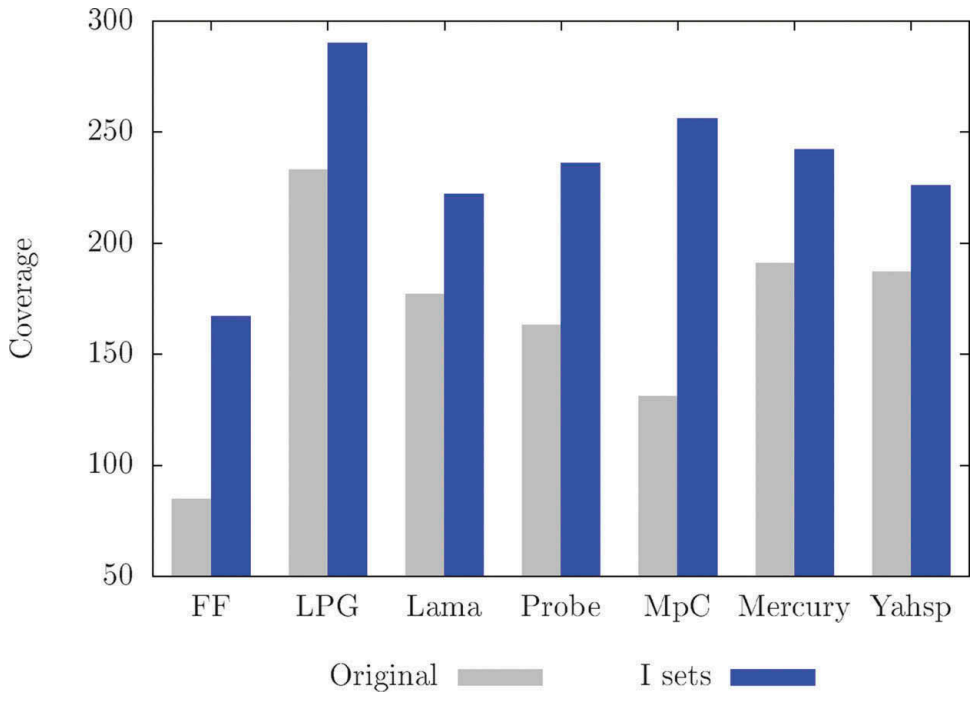


Figure 3. Cumulative coverage performance of the considered planners exploiting the original encodings and the encodings enhanced with the I sets of outer entanglements, across all the testing benchmarks.

the planning engines is often considerable. Therefore, outer entanglements learnt in these domains do efficiently both – eliminating unpromising operators' instances and (considerably) reducing the state space.

By analysing the impact of outer entanglements in particular domains, we can see the following. In Bw and its variant Matching-bw, there are two possibilities how blocks can be temporarily put aside – putting them on the table, or stacking them on other blocks. Outer entanglements enforce the former option. This drastically reduces the size of the state space because blocks can be only in their initial or goal positions, on the table, or being held by the robotic hand. Moreover, temporarily stacking blocks on other (non-goal) blocks introduces further constraints, i.e. a block on which we temporarily stack another block cannot be moved without taking that other block out. A possible drawback is in introducing dead-ends. If a stack of blocks is incorrectly built from 'the middle', the planner cannot repair it (it is impossible to unstack blocks from other than initial configurations) and has to backtrack. However, the results clearly indicate that introducing dead-ends in these domains does not negatively affect planners' performance. The Gripper domain describes the problem of moving balls between rooms by robots with two grippers. Outer entanglements in this domain prevent to pickup a ball in other than its initial location as well as to drop the ball in other than its goal location. The planners thus do not have to consider to temporarily leave balls in non-goal locations, which as the results indicate is beneficial for some planners. Similar observations can be made in Depots and TPP. Thoughtful is a variant of the well known freecell card game (Bjarnason, Tadepalli, & Fern, 2007). There are a number of strategies that can be exploited in order to achieve a goal configuration of cards. Interestingly, each planner exploited a different strategy while solving training problems which led to extraction of different sets of outer entanglements most of which are incomparable to each other (see Table 1). As mentioned before, the sets VI and VII do not hold for 4 and 6 testing problems, respectively. This indicates that the training problems were too constrained and that some strategies feasible for solving them might not generalise well for wider range of (testing) problems. On the bright side, the set III brought a considerable performance improvement among the planners (except LPG that benefited from the VI set). A closer analysis of the IPC speed scores indicates that testing problems do not unanimously benefit from a single encoding (i.e. there is a gap between coverage and the IPC speed score). Also, given the discrepancies between learnt sets of entanglements for each planner (in spite of the fact that training problems were same for all the planners), planners' 'sensitivity' might very vary for particular sets of outer entanglements. In such cases determining the most promising set of entanglements can be done by cross-validating their performances on several tasks which are more complex than the training ones.

In Barman and Parking, outer entanglements underperformed the original encodings. Table 3 shows that the results are more mixed, i.e. some planners benefit from outer entanglements, some do not. In Barman, outer entanglements enforce cocktails to be poured only into the 'goal' shots. Probe considerably benefits from such a restriction, while it has a very detrimental impact for Mercury. The reason for the latter seems to be in Mercury's inefficient handling of situations where the 'goal' shot is not clean while the cocktail is being prepared. Parking, which deals with a problem of rearranging cars on a parking lot, is a combinatorial domain. The sets of outer entanglements seem to be beneficial only for some planning techniques and a limited number of testing problems. Such results point to the fact that despite their reasonable pruning power (around 50% of actions and atoms were pruned in the Barman domain) outer entanglements can have detrimental effects on some planning techniques (such as Red-Black heuristics accommodated in the Mercury planner (Domshlak et al., 2015)).

In Gold-miner, Rovers, Satellite, Sokoban and Spanner, outer entanglements slightly outperformed the original encodings. In these domains, however, outer entanglements have limited pruning power and hence their impact on planners' performance is limited.

In spite of a few cases where outer entanglements have rather detrimental effects on planning engines, the results demonstrated, as summarised in Figure 3, that the use of outer entanglements

improves performance of planning engines regardless planning techniques they exploit across a number of different domains. Hence, outer entanglements can be considered a fruitful technique to be exploited both in domain-independent and planner-independent fashion.

7. Discussion

This section is devoted to discussion of benefits and drawbacks of outer entanglements and provide general recommendation for their extraction and use.

7.1. *Extracting 'good' sets of outer entanglements*

The outcome of the outer entanglement learning process depends on training problems and solution plans of these problems (i.e. training plans). According to the study of Chrpa et al. (2013) if the number and complexity of training problems, which is determined by length of solution plans, increase above certain thresholds, the impact on the outcome of the outer entanglement learning process is negligible. On the other hand, using different planners to generate training plans might lead to considerably different results.

Our experimental results, where the number of training problems was set to 5 and whose solution (or training) plans consisted of at least 20 actions in average per domain, have shown that these thresholds are sufficient for generating sets of outer entanglements that hold also for testing problems and improve planners' performance. The only exception has been observed in the Thoughtful domain, where Mercury- and Probe-sets did not hold for some of the testing problems. Since the Thoughtful domain is complex (containing more than 20 planning operators), the used thresholds might be too low. Hence, when setting up the training problems, it is important to consider complexity of the domain model and adjust the thresholds accordingly, i.e. the number and complexity of training problems should be higher for more complex domain models.

For generating training plans, we have used 7 different planners. In 5 domains, the extracted sets of outer entanglements were identical regardless of the used planner. On the other hand, in Matching-bw and Thoughtful, the extracted sets of outer entanglements considerably differed (for details, see Tables 1 and 2). Also, the impact of different sets of outer entanglements on planners' performance often varied considerably (see Tables 3 and 4). Except Parking and Thoughtful, we were able to identify a set of outer entanglements that subsumes the other sets. Since such a set has always the strongest pruning power, in our experiments the set was denoted as I. With a few notable exceptions (e.g. Barman), the I sets outperformed, often considerably, the other sets as well as the original encodings. In Thoughtful, the Lama-set (set III) was the best performing set, while in Parking, the Lama-set (set I) was the worst performing set. Interestingly, in both cases Lama generated the best quality training plans. Moreover, the Lama-set in Thoughtful does not have the strongest pruning power, it is the Yahsp-set (set I) that underperformed even the original encoding. It should be, however, noted that training plans in Thoughtful generated by Yahsp were of a low quality (the worst among the planners).

Lessons learnt from analysing the experimental results indicate that (i) poor quality training plans lead to empty or 'poor' sets of outer entanglements, (ii) the best quality training plans do not necessarily lead to 'good' sets of outer entanglements, (iii) sets containing all extracted outer entanglements tend to be the best performing ones, and (iv) incomparable sets of outer entanglements indicate lack of training data (i.e. a small number and low complexity of training problems). Notice that the best quality training plans are not necessarily optimal (unless an optimal planner is used for their extraction). As poor quality training plans we consider those that are at least 50% longer than the best quality ones. We believe that taking into account these lessons provides a general guidance for performing effective and efficient outer entanglement learning process.

7.2. *Heuristic nature of the learning method*

Because deciding whether an outer entanglement holds in a given planning task is PSPACE complete, we have developed an approximation method that learns outer entanglements from training plans, solutions of simple planning tasks. Our method, therefore, follows an assumption that a learnt set of outer entanglements holds for every task in a given class (or domain). There is, however, no theoretical guarantee that the assumption will hold for every (non-training) planning task. Although the experiments have demonstrated a strong support for the assumption, in a few cases we have observed that the assumption did not hold, and to solve a problem the system would have to revert to its original formulation. Theoretically, after a reformulated task is proved to be unsolvable, the original task has to be solved (or proven unsolvable too). Such an approach might be practically reasonable only in cases in which the unsolvability of the reformulated task is proved quickly (e.g. goals are not reachable). Another possibility to alleviate the heuristic issue is to integrate outer entanglement reformulated tasks within planning portfolios. Also, an engineer who has developed a domain model might manually decide whether a learnt set of outer entanglements holds for planning tasks using that domain model. In cases such as BlocksWorld, it might be easy.

7.3. *Optimality*

The quality of plans has improved in many cases when outer entanglements were used. Intuitively, pruning the search space may force planners to find better solution plans. On the other hand, outer entanglements do not guarantee optimality in general. Strengthening definitions of outer entanglements to guarantee plans optimality is, of course, theoretically possible. Given the complexity results of 'normal' entanglements, we can expect the same for 'optimal' entanglements. Using the approximation algorithm for learning outer entanglements on optimal training plans with zero flaw ratio might extract some useful 'optimal' outer entanglements. However, we believe that there is a high risk of extracting 'sub-optimal' outer entanglements that prune optimal solution plans. For example, in Logistic-like domain, it is often an optimal strategy to pick up packages from their initial locations and deliver them to their goal locations. This can be captured by outer entanglements. However, if driving between some locations is very expensive, it might be better to move some packages from other trucks to one truck which will perform the 'expensive' journey. Here, the outer entanglements will prevent to do so and thus will prune optimal solutions (although the task will remain solvable).

8. Conclusions and future work

In this paper we presented outer entanglements, relations between planning operators and predicates whose instances are in the initial state or the goal. Outer entanglements are used to eliminate unpromising instances of planning operators and thus reduce branching factor in the state space as well as the size of problem representation (and, consequently, the size of the search space). To deal with the intractability of deciding whether a given outer entanglement holds for a given planning task (see Section 4.2), we used a learning method for extracting 'domain-specific' sets of outer entanglements from training plans, solution plans of simple tasks. Outer entanglements can be encoded into domain models without extending the input language of a planner (see Section 5.2) and, therefore, they can be understood and exploited as planner-independent knowledge.

The extensive experimental analysis in this paper demonstrates that outer entanglements improve the planning process considerably within a wide range of competition domains and state-of-the-art planning engine combinations. Our experiments using 7 state-of-the-art planning engines, and 14 benchmark domain models, have given a good indication of the planner and domain independence, and the effectiveness of the method: in the overwhelming majority of the cases, outer entanglements caused a substantial improvement in plan generation speed and solution plan quality.

We identified several avenues for future research. First, we plan to incorporate outer entanglements into well known planning frameworks such as Fast Downward (Helmert, 2006) or LAPKT (Ramirez, Lipovetzky, & Muise, 2015) so they can be exploited, for instance, for computing heuristics. Second, given the encouraging spread of results among sets of planners and domains, we intend to work towards including an entanglements generating facility as part of a knowledge engineering workbench. Finally, we plan to extend the concept of outer entanglements for non-classical planning which will potentially improve performance of real-world planning applications (preliminary work has been started on this in numerical planning (Chrpa, Scala, & Vallati, 2015a)).

Notes

1. http://ipc.icaps-conference.org.
2. 'Accidental complexity of domain models' means their inefficient encodings decreasing performance of planning engines.
3. Learning track benchmarks are more natural, since entanglements extraction phase can be understood as a learning process.
4. By unreachable operator instances we mean those not applicable at any point of the planning process.
5. By 'unsolvable' we mean those problems where the planner did not find a solution within the given time limit of 900 CPU-time seconds.

Acknowledgements

This Research was funded by the Czech Science Foundation (project no. 17-17125Y) and by the UK EPSRC Autonomous and Intelligent Systems Programme (Grant no. EP/J011991/1).

The authors would like to acknowledge the use of the University of Huddersfield Queensgate Grid in carrying out this work.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work was supported by the Engineering and Physical Sciences Research Council [Grant number EP/J011991/1]; Czech Science Foundation (Grantová Agentura České Republiky) [Grant number 17-17125Y].

References

Baier, J. A., Fritz, C., & McIlraith, S. A. (2007). Exploiting procedural domain control knowledge in state-of-the-art planners. *Proceedings of the seventeenth international conference on automated planning and scheduling, ICAPS* (pp. 26–33).

Bernard, D., Gamble, E., Rouquette, N., Smith, B., Tung, Y., Muscettola, N., Taylor, W. (2000). Remote agent experiment ds1 technology validation report. Technical report, Ames Research Center and JPL.

Bjarnason, R., Tadepalli, P., & Fern, A. (2007). Searching solitaire in real time. *International Computer Games Association Journal*, *30*(3), 131–142.

Bonet, B., & Geffner, H. (1999). Planning as heuristic search: New results. *European conference on planning, ECP* (pp. 360–372).

Botea, A., Enzenberger, M., Müller, M., & Schaeffer, J. (2005). Macro-ff: Improving ai planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research (JAIR)*, *24*, 581–621.

Brafman, R. I., & Domshlak, C. (2013). On the complexity of planning for agent teams and its implications for single agent planning. *Artificial Intelligence*, *198*, 52–71.

Chen, Y., & Yao, G. (2009). Completeness and optimality preserving reduction for planning. *International joint conference on artificial intelligence, IJCAI* (pp. 1659–1664).

Chrpa, L. (2010). Generation of macro-operators via investigation of action dependencies in plans. *Knowledge Engineering Review*, *25*(3), 281–297.

Chrpa, L., & Barták, R. (2009). Reformulating planning problems by eliminating unpromising actions. *Symposium on abstraction, reformulation, and approximation, SARA* (pp. 50–57).
Chrpa, L., & McCluskey, T. L. (2012). On exploiting structures of classical planning problems: Generalizing entanglements. *European conference on artificial intelligence, ECAI* (pp. 240–245).
Chrpa, L., McCluskey, T. L., & Osborne, H. (2012). Reformulating planning problems: A theoretical point of view. *International Florida artificial intelligence research society conference, FLAIRS* (pp. 14–19).
Chrpa, L., Scala, E., & Vallati, M. (2015a, 11–13 June). Towards a reformulation based approach for efficient numeric planning: numeric outer entanglements. *Proceedings of the eighth annual symposium on combinatorial search, SOCS 2015* (pp. 166–170). Ein Gedi, the Dead Sea, Israel.
Chrpa, L., Vallati, M., & McCluskey, T. (2015b). On the online generation of effective macro-operators. *International joint conference on artificial intelligence, IJCAI 2015* (pp. 1704–1711).
Chrpa, L., Vallati, M., & McCluskey, T. L. (2014). MUM: A technique for maximising the utility of macro-operators by constrained generation and use. *The International conference on automated planning and scheduling, ICAPS* (pp. 65–73).
Chrpa, L., Vallati, M., & Osborne, H. (2013). Learnability of specific structural patterns of planning problems. *International conference on tools with artificial intelligence, ICTAI* (pp. 18–23).
Coles, A., Coles, A., Garca Olaya, A., Jiménez, S., Linares López, C., Sanner, S., & Yoon, S. (2012). A survey of the seventh international planning competition. *AI Magazine*, *33*(1), 83–88.
Coles, A., Fox, M., & Smith, A. (2007). Online identification of useful macro-actions for planning. *The international conference on automated planning and scheduling, ICAPS* (pp. 97–104).
Coles, A. J., & Coles, A. I. (2010). Completeness-preserving pruning for optimal planning. *European conference on artificial intelligence, ECAI* (pp. 965–966).
Dawson, C., & Siklóssy, L. (1977). The role of preprocessing in problem solving systems. *International joint conference on artificial intelligence, IJCAI* (pp. 465–471).
De La Rosa, T., Celorrio, S. J., Fuentetaja, R., & Borrajo, D. (2011). Scaling up heuristic planning with relational decision trees. *Journal of Artificial Intelligence Research (JAIR)*, *40*, 767–813.
Domshlak, C., Hoffmann, J., & Katz, M. (2015). Red-black planning: A new systematic approach to partial delete relaxation. *Artificial Intelligence*, *221*, 73–114.
Fox, M., & Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)*, *20*, 61–124.
Gerevini, A., Saetti, A., & Serina, I. (2003). Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research (JAIR)*, *20*, 239–290.
Ghallab, M., Isi, C. K., Penberthy, S., Smith, D. E., Sun, Y., & Weld, D. (1998). Pddl - the planning domain definition language. Technical report.
Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated planning, theory and practice*. San Francisco, USA: Morgan Kaufmann Publishers.
Gnad, D., Wehrle, M., & Hoffmann, J. (2016, 9–15 July). Decoupled strong stubborn sets. *Proceedings of the twenty-fifth international joint conference on artificial intelligence, IJCAI 2016* (pp. 3110–3116). New York, NY.
Haslum, P. (2007). Reducing accidental complexity in planning problems. *International joint conference on artificial intelligence, IJCAI* (pp. 1898–1903).
Haslum, P., Helmert, M., & Jonsson, A. (2013). Safe, strong, and tractable relevance analysis for planning. *Proceedings of the international conference on automated planning and scheduling, ICAPS* (pp. 317–321).
Haslum, P., & Jonsson, P. (2000). Planning with reduced operator sets. *International conference on artificial intelligence planning systems, AIPS* (pp. 150–158).
Helmert, M. (2006). The fast downward planning system. *Journal of Artificial Intelligence Research*, *26*, 191–246.
Hoffmann, J. (2003). The metric-ff planning system: translating "ignoring delete lists" to numeric state variables. *Journal Artificial Intelligence Research (JAIR)*, *20*, 291–341.
Hoffmann, J., & Nebel, B. (2001). The FF planning system: fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*, 253–302.
Hoffmann, J., Porteous, J., & Sebastia, L. (2004). Ordered landmarks in planning. *Journal of Artificial Intelligence Research (JAIR)*, *22*, 215–278.
Kautz, H., & Selman, B. (1992). Planning as satisfiability. *European conference on artificial intelligence, ECAI* (pp. 359–363).
Knoblock, C. A. (1994). Automatically generating abstractions for planning. *Artificial Intelligence*, *68*(2), 243–302.
Korf, R. (1985). Macro-operators: A weak method for learning. *Artificial Intelligence*, *26*(1), 35–77.
Kvarnström, J., & Doherty, P. (2000). Talplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence*, *30*(1–4), 119–169.
Lipovetzky, N., & Geffner, H. (2011). Searching for plans with carefully designed probes. *The 21st international conference on automated planning and scheduling (ICAPS-11)*. AAAI press.
Lipovetzky, N., Ramirez, M., Muise, C., & Geffner, H. (2014).Width and inference based planners: siw, bfs(f), and probe. *The eighth international planning competition. Description of participant planners of the deterministic track* (pp. 6–7).

McCluskey, T. L. (1987). Combining weak learning heuristics in general problem solvers. *International joint conference on artificial intelligence, IJCAI* (pp. 331–333).

McCluskey, T. L., & Porteous, J. M. (1997). Engineering and compiling planning domain models to promote validity and efficiency. *Artificial Intelligence*, *95*(1), 1–65.

Minton, S., & Carbonell, J. G. (1987). Strategies for learning search control rules: an explanation-based approach. *International joint conference on artificial intelligence, IJCAI* (pp. 228–235).

Newton, M. A. H., Levine, J., Fox, M., & Long, D. (2007). Learning macro-actions for arbitrary planners and domains. *The international conference on automated planning and scheduling, ICAPS* (pp. 256–263).

Núñez, S., Borrajo, D., & Linares López, C. (2012). Performance analysis of planning portfolios. *Proceedings of the fifth annual symposium on combinatorial search, SOCS*.

Ramirez, M., Lipovetzky, N., & Muise, C. (2015). Lightweight automated planning toolkit. Retrieved from http://lapkt.org/. Accessed 2016, 11 1.

Richter, S., & Westphal, M. (2010). The lama planner: Guiding cost-based anytime planning with landmarks. *Journal Artificial Intelligence Research (JAIR)*, *39*, 127–177.

Richter, S., Westphal, M., & Helmert, M. (2011). Lama 2008 and 2011. *Booklet of the 7th international planning competition*.

Rintanen, J. (2012). Engineering efficient planners with sat. *European conference on artificial intelligence, ECAI* (pp. 684–689).

Rintanen, J. (2014).Madagascar: Scalable planning with sat. *The eighth international planning competition*. Description of Participant Planners of the Deterministic Track (pp. 66–70).

Scholz, U. (2004). *Reducing planning problems by path reduction* (PhD thesis). Darmstadt University of Technology.

Slaney, J., & Thiébaux, S. (2001). Blocks world revisited. *Artificial Intelligence*, *125*(1–2), 119–153.

Vallati, M., Chrpa, L., Grzes, M., McCluskey, T. L., Roberts, M., & Sanner, S. (2015). The 2014 international planning competition: progress and trends. *AI Magazine*, *36*(3), 90–98.

Vidal, V. (2014).Yahsp3 and yahsp3-mt in the 8th international planning competition. *The eighth international planning competition. Description of participant planners of the deterministic track* (pp. 64–65).

Wehrle, M., Helmert, M., Alkhazraji, Y., & Mattmüller, R. (2013). The relative pruning power of strong stubborn sets and expansion core. *The international conference on automated planning and scheduling, ICAPS*.

Yoon, S. W., Fern, A., & Givan, R. (2008). Learning control knowledge for forward search planning. *Journal of Machine Learning Research*, *9*, 683–718.

A.9 Improving domain-independent planning via critical section macro-operators

L. Chrpa and M. Vallati. Improving domain-independent planning via critical section macro-operators. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, pages 7546–7553, 2019.

The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)

Improving Domain-Independent Planning via Critical Section Macro-Operators

Lukáš Chrpa
Faculty of Electrical Engineering
Czech Technical University in Prague &
Faculty of Mathematics and Physics
Charles University in Prague

Mauro Vallati
School of Computing and Engineering
University of Huddersfield

Abstract

Macro-operators, macros for short, are a well-known technique for enhancing performance of planning engines by providing "short-cuts" in the state space. Existing macro learning systems usually generate macros from most frequent sequences of actions in training plans. Such approach priorities frequently used sequences of actions over meaningful activities to be performed for solving planning tasks.

This paper presents a technique that, inspired by resource locking in critical sections in parallel computing, learns macros capturing activities in which a limited resource (e.g., a robotic hand) is used. In particular, such macros capture the whole activity in which the resource is "locked" (e.g., the robotic hand is holding an object) and thus "bridge" states in which the resource is locked and cannot be used. We also introduce an "aggressive" variant of our technique that removes original operators superseded by macros from the domain model. Usefulness of macros is evaluated on several state-of-the-art planners, and a wide range of benchmarks from the learning tracks of the 2008 and 2011 editions of the International Planning Competition.

Introduction

Automated Planning, in a nutshell, is about finding a sequence of actions whose application in an initial state of the environment leads to a desired goal state (Ghallab, Nau, and Traverso 2004). Whereas a lot of effort has been traditionally given to developing efficient planning engines, usually based on heuristic search (Bonet and Geffner 2001), another line of research focuses on increasing efficiency of the planning process by reformulating the domain knowledge, to obtain models that are more amenable for automated reasoners.

A very well-known reformulation approach is the generation of macro-operators, macros for short, that encapsulate sequences of (original) planning operators. Macros are encoded as ordinary planning operators and, hence, they can be added into domain models such that standard planning engines can straightforwardly take advantage of them. Macros, informally speaking, provide "short-cuts" in the state space and, consequently, planning engines can generate plans in less number of steps. This comes with the cost of increased branching factor since macros often have much more instances than ordinary operators and thus their use might introduce additional overheads as well as larger memory requirements. Although in theory the use of macros might reduce complexity of planning (Korf 1985), in practice macros are considered if their use is frequent (Hofmann, Niemueller, and Lakemeyer 2017), their number of instances is small (Chrpa, Vallati, and McCluskey 2014), or they address weaknesses of a specific planner (Coles, Fox, and Smith 2007).

In this paper, we introduce *Critical Section Macros*. Being inspired by resource locking in critical section in parallel computing, these macros capture whole activities in which a resource is used. For example, a robotic hand manipulates with objects (e.g. blocks in BlocksWorld, or shaker in Barman). When the robotic hand grasps an object it becomes "locked", i.e., no other object can be grasped by that hand, until the hand releases the object it holds. Hence, Critical Section Macros aim to capture sequences of operators such that the first operator locks a resource (e.g. a robotic hand grasps an object), the last operator releases the resource (e.g. the robotic hand drops the object), and the intermediate operators, if any, uses the resource (e.g. shaking a cocktail). From the technical perspective, Critical Section Macros "bridge" states in which the resource is locked and cannot be used. This is thought to be particularly beneficial for techniques that exploit delete-relaxation (Hoffmann and Nebel 2001) as they tend to incorrectly assume that a resource can be used by multiple activities (e.g. a robotic hand holding multiple objects) and thus provide largely incorrect heuristic estimations. As additional contributions, we illustrate how the proposed reformulation approach can be exploited in an "aggressive" variant, that removes the elementary operators that are included in the macros, and can be combined with more traditional techniques for generating macros. The usefulness of Critical Section Macros, in all the variants depicted above, is evaluated on several state-of-the-art planners, and a wide range of benchmarks from learning tracks of the International Planning Competition.

Related Work

Using macros dates back to 1970s and 1980s. REFLECT (Dawson and Siklóssy 1977) builds macro-operators from pairs of primitive operators that can be successively applied and share at least one argument. MOR-

RIS (Minton 1988) learns macro-operators from parts of plans appearing frequently (S-macros) or being potentially useful despite having low priority (T-macros). Macro Problem Solver (Korf 1985) learns macros for particular non-serializable sub-goals (e.g. in Rubik's cube).

Recent planner-independent techniques aim at improving performance of any standard planner. MacroFF (Botea et al. 2005) generates macros according to several pre-defined rules (e.g., the "locality rule") that apply on adjacent actions in training plans. Wizard (Newton et al. 2007) learns macros from training plans by exploiting genetic programming. Alhossaini and Beck (2013) selects problem-specific macros from a given pool of macros (hand-coded or generated by another technique). Dulac et al. (2013) exploits n-gram algorithm to analyze training plans to learn macros. DBMP/S (Hofmann, Niemueller, and Lakemeyer 2017) applies Map Reduce for learning macros from a larger set of training plans. CAP (Asai and Fukunaga 2015) exploits component abstraction (introduced by MacroFF) for generating sub-goal specific macros.

MUM (Chrpa, Vallati, and McCluskey 2014) exploits "outer entanglements" (Chrpa and McCluskey 2012) as a heuristics for generating macros with limited number of instances. BloMa (Chrpa and Siddiqui 2015) exploits block deordering (Siddiqui and Haslum 2012) for generating possibly longer macros. Our "critical section" macros share some characteristics with "block" macros generated by BloMa. BloMa, however, initially generates a large pool of macros that is later reduced by applying (strict) frequency requirements.

Classical Planning

The classical (STRIPS) representation considers static and fully observable environment, and deterministic and instantaneous action effects. The environment is described by first-order logic *predicates* defined as $p = pred\_name(x_1, \ldots, x_n)$, where *pred_name* is a unique predicate name and $x_1, \ldots x_n$ are variable symbols. *States* are defined as sets of *atoms* (grounded predicates whose variable symbols are substituted with constants - problem-specific objects). We say that $o = (name(o), pre(o), del(o), add(o))$ is a *planning operator*, where $name(o) = op\_name(x_1, \ldots, x_k)$ (*op_name* is an unique operator name and $x_1, \ldots x_k$ are variable symbols (arguments) appearing in the operator) and $pre(o), del(o)$ and $add(o)$ are sets of (ungrounded) predicates with variables taken only from $x_1, \ldots x_k$ representing o's precondition, delete, and add effects respectively. *Actions* are grounded instances of planning operators. An action a is *applicable* in a state s if and only if $pre(a) \subseteq s$. Application of a in s (if possible) results in a state $(s \setminus del(a)) \cup add(a)$.

A *planning domain model* $D = (P, O)$ is specified by a set of predicates (P) and a set of planning operators(O). A *planning task* $\Pi = (D, I, G)$ is specified via a domain model (D), initial state (I) and set of goal atoms (G). Given a planning problem, a *plan* is a sequence of actions such that their consecutive application starting in the initial state results in a state containing all the goal atoms.

Given a planning task Π, we say that a state s' is *reachable* from a state s if and only if there exists a sequence of actions such that their consecutive application starting in s results in s'. We say that an action a_i is an *achiever* for an action a_j if an only if $add(a_i) \cap pre(a_j) \neq \emptyset$. We also say that actions a_i and a_j are *independent* if and only if $del(a_i) \cap (pre(a_j) \cup add(a_j)) = \emptyset$ and $del(a_j) \cap (pre(a_i) \cup add(a_i)) = \emptyset$.

Macro-operators

Macros represent sequences of (ordinary) planning operators. Advantageously, macros can be encoded in the same form as planning operators (i.e., having a precondition, add and delete effects). Hence, macros can be added into a domain model and thus can be exploited in a *planner independent* way (e.g. encoded in PDDL).

Formally, a macro $o_{i,j}$ is constructed by assembling planning operators o_i and o_j (in that order) as follows. Let Φ and Ψ be mappings between variable symbols (we need to appropriately rename variable symbols of o_i and o_j to construct $o_{i,j}$).

- $pre(o_{i,j}) = pre(\Phi(o_i)) \cup (pre(\Psi(o_j)) \setminus add(\Phi(o_i)))$
- $del(o_{i,j}) = (del(\Phi(o_i)) \setminus add(\Psi(o_j))) \cup del(\Psi(o_j))$
- $add(o_{i,j}) = (add(\Phi(o_i)) \setminus del(\Psi(o_j))) \cup add(\Psi(o_j))$

Longer macros, i.e., those encapsulating longer sequences of original planning operators can be constructed iteratively by the above approach.

For a macro to be *sound*, no instance of $\Phi(o_i)$ can delete an atom required by a corresponding instance of $\Psi(o_j)$, otherwise they cannot be applied consecutively. Whereas it is obvious that if a predicate deleted by $\Phi(o_i)$ (and not added back) is the same (both name and variable symbols) as a predicate in the precondition of $\Psi(o_j)$ then the macro $o_{i,j}$ is unsound, another source of macro unsoundness is often not being even considered in literature. For example, in the Blocks-World domain, a macro pickup-stack(?x ?y) that has (clear ?x)(ontable ?x)(clear ?y)(handempty) in its precondition can be instantiated into pickup-stack(A A) that is applicable if (clear A)(ontable A)(handempty) is true in some state. However, actions (pickup A) and stack(A A) cannot be applied consecutively because (pickup A) deletes (clear A) which is required by stack(A A). By generalizing this observation we can see that if some (different) variable symbols are substituted by the same constants, a macro might become unsound. To avoid such cases, a constraint requiring different instantiation of affected variable symbols is added into macro's precondition (e.g. (not (= ?x ?y)) is added into pickup-stack(?x ?y)'s precondition).

Critical Section Macros

In parallel computing, critical sections are used to regulate the access to resources, to guarantee the integrity of the overall system by avoiding situations where many different processes are concurrently modifying a resource. To prevent other processes (or threads) to access the resource while it is in use, the resource is locked at the beginning of the critical section, then the required operations are made with the resource by the process that is locking it, and then –at the

end of critical section– the resource is released and is therefore available for other processes.

In planning, we can observe that some subsequences of actions in plans replicate the underlying structure of critical section, i.e., locking a resource, using it, and releasing it. In Blocks-World, the robotic hand can be seen as a resource. When the robotic hand picks-up or unstacks a block it becomes "locked", that is, no other block can be carried by the robotic hand at that time. When the robotic hand stacks or puts-down the block it is holding, then the hand is "released", that is, it can be used to manipulate other blocks. A more complicated example can be found in the Barman domain, in which a robotic barman prepares cocktails. Here, a critical section activity, for instance, involves grabbing a shaker (locking robot's hand), shaking a cocktail, putting the cocktail to a shot, cleaning the shaker before putting it back on the table (releasing robot's hand).

Technically speaking, a free resource, as well as a locked resource, is represented by corresponding predicates. For example, (handfree) represents a free resource (robotic hand) while (holding ?x) represents a locked resource (robotic hand carrying a block). Also, corresponding instances of these predicates must be mutually exclusive (mutex for short), i.e., no resource can be both free and locked at the same time.

Definition 1. *Let Π be a planning task and p and q be predicates defined in the domain model of Π. Let Φ, Ψ be substitutions mapping variable symbols to variable symbols. We say that p and q with respect to Φ and Ψ are **mutex** in Π if and only if for all ground instances of $\Phi(p)$ and $\Psi(q)$ it is the case that they are not both true in all reachable states from the initial state of Π.*

Having mutex predicates p and q that represent a free and locked resource respectively, is a necessary condition for recognising "critical section" activities. Also, arguments of q must be a superset of arguments of p (they can have the same set of arguments). Arguably, since q represents a locked resource, it might contain additional arguments referring to why the resource is locked (e.g. a hand holding an object), however, it cannot contain less arguments (in order to recover a corresponding instance of p after releasing the resource).

Locking and releasing resources is done by specific planning operators. A planning operator that deletes p and adds a corresponding variant of q, a *locker*, locks a given resource. Analogously, a planning operator that deletes q and adds a corresponding variant of p, a *releaser*, releases (unlocks) the given resource. This idea is formalised as follows.

Definition 2. *Let $\Pi = (D, I, G)$ be a problem instance and $D = (P, O)$ be a domain model. Let $p, q \in P$ be predicates such that p and q with respect to substitutions Φ, Ψ are mutex in Π and $args(\Phi(p)) \subseteq args(\Psi(q))$. We say that an operator $o \in O$ is a p,q-**locker** if $\Phi(p) \in del(\Theta(o))$ and $\Psi(q) \in add(\Theta(o))$ (Θ is a renaming substitution). We also say that $o' \in O$ is a p,q-**releaser** if $\Phi(q) \in del(\Theta'(o'))$ and $\Psi(p) \in add(\Theta'(o'))$ (Θ' is a renaming substitution). We also say that $o'' \in O$ is a p,q-**user** if $\Phi(q) \in pre(\Theta''(o'')) \setminus (del(\Theta''(o'')) \cup add(\Theta''(o'')))$ (Θ'' is a renaming substitution).*

Algorithm 1 Learning Critical Section Macros from training plans

```
cs ← {(p, q, O_l, O_r) | o_l ∈ O_l is a p,q-locker; o_r ∈ O_r is a p,q-releaser}
for each ⟨a_1, ..., a_n⟩ in Training_Plans do
    lr_pairs ← {(a_l, a_r, p_g, q_g) | (p, q, O_l, O_r) ∈ cs; r > l; a_l, a_r, p_g, q_g are instances of o_l ∈ O_l, o_r ∈ O_r, p, q respectively; p_g ∈ del(a_l) ∩ add(a_r); q_g ∈ add(a_l) ∩ del(a_r)}
    for each (a_l, a_r, p_g, q_g) ∈ lr_pairs do
        in_ma ← {a_k | l < k < r; q_g ∈ pre(a_k)} ∪ {a_l, a_r}
        out_ma ← {a_k | l < k < r; a_k ∉ in_ma}
        ConsiderDependent(in_ma,out_ma)
        if no "gluing" action added extra argument then
            o^m ←CreateMacro(in_ma)
            ConsiderGoalAchieving(o^m)
            AddMacro(mcr_db,o^m)
        end if
    end for
end for
FilterUnderrepresentedMacros(mcr_db)

function CONSIDERDEPENDENT(in_ma,out_ma)
    while ∃k : {i | a_i ∈ out_ma; i < k} = ∅ and {a_i | a_i ∈ in_ma; i < k; a_i is an achiever for a_k or a_i is not independent with a_k} = ∅ do
        out_ma ← out_ma \ {a_k}
    end while
    while ∃k : {j | a_j ∈ out_ma; j > k} = ∅ and {a_j | a_j ∈ in_ma; j > k; a_k is an achiever for a_j or a_j is not independent with a_k} = ∅ do
        out_ma ← out_ma \ {a_k}
    end while
    in_ma ← in_ma ∪ out_ma
end function
```

The above definition considers one phase single locks. That is, that resources are locked/released by one operator (rather than their sequence) and only one lock (i.e., instance of q) is acquired. The definition can be extended to consider multiple-phase and multiple locks, however, for practical reasons (i.e., such situations are very uncommon if any in standard benchmarks used in the international planning competition) and for the sake of clarity we resort to the cases covered by Definition 2.

Constructing Critical Section Macros

The rationale behind the Critical Section Macros is to *bridge* resource use with a single macro. Delete-relaxation (Hoffmann and Nebel 2001) is a popular approach for many state-of-the-art planning engines. Delete-relaxation, roughly speaking, ignores delete effects of planning operators. Consequently, mutex relations between grounded predicates are also ignored. In situations of resource locking, the difference between delete-relaxed approximation and reality can

be considerably large, hence undermining the usefulness of the heuristic evaluation. For example, in delete-relaxation, the robotic hand can hold all the blocks at the same time. As it is apparent, such discrepancies can easily cause local minima on landscape of heuristic functions based on delete-relaxation (Hoffmann 2011).

Critical Section Macros encapsulate sequences of operators such that they start with p, q-lockers and end with p, q-releasers. For longer sequences, p, q-users and "gluing" operators are present in between. For example, a macro pickup-stack is a Critical Section Macro consisting of only a locker (pickup) and a releaser (stack). In Barman, for example, a macro grasp-fillshot-leave is a Critical Section Macro consisting of a locker (grasp), a releaser (leave) and a user (fillshot). In Gripper, a macro pick-move-drop is a Critical Section Macro consisting of a locker (pick), a releaser (drop) and a gluing operator (move).

Algorithm 1 describes the method for learning Critical Section Macros from training plans. Quadruples (p, q, O_l, O_r) (Line 1) are determined by considering whether each operator deleting p (or q) adding a corresponding variant of q (or p), in other words, each operator having p or q in its effects is either a p, q-locker or p, q releaser. Also, if corresponding instances of p and q are not simultaneously present in initial states of training tasks (and testing tasks), then p and q are mutex (with respect to corresponding substitutions). For each training plan, we determine all possible locker/releaser pairs (a_l, a_r) with corresponding instances of the involved predicates (p_g, q_g) (Line 3). Then, we iterate through the locker/releaser pairs (Lines 4–13). Besides a_l (p_g, q_g-locker) and a_r (p_g, q_g-releaser) we consider p_g, q_g-users into a possible macro (Line 5). Other actions placed in between a_l and a_r are checked whether they can be moved away (either before a_l or after a_r). This is done by the ConsiderDependent function. The idea of how intermediate actions can be moved away is based on the observation that: if for two adjacent actions a, a' in a plan (in this order), it is the case that a and a' are independent and a is not an achiever for a', then a, a' can be swapped without compromising the correctness of the plan (similar approach has been used by MUM (Chrpa, Vallati, and McCluskey 2014)). Those actions that cannot be moved away are *gluing* actions. If none of the gluing actions introduces an extra argument, in other words, does not have to reason with additional objects than the locker, releaser and the user actions (Line 8), then the action sequence in consideration can be considered as a macro (Line 9). After the macro is created, it is checked whether it is *goal achieving*, i.e., whether some of its add effects are goal atoms. Goal achieving macros, achieving instances of p that are present in the goal, extend the ordinary macros by introducing a supplementary static predicate p^G, added into macro's precondition, that has the same variable symbols as p in the macro's add effects but a different name. Also, a problem-instance is modified such that for each instance of $p \in G$, a corresponding instance of p^G is added to I. Noteworthy, such a concept is analogous to the use of outer entanglements (Chrpa and McCluskey 2012).

Macros that are "underrepresented", i.e., their number in the "macro database" is below a specified threshold are filtered out. Underrepresented macros, in a Machine Learning terminology, are noise in training data. That are, for example, problem-specific macros that do not generalize for a class of planning problems, or macros capturing peculiarities in training plans. Such macros are very unlikely to be beneficial. Other macro learning techniques such as MacroFF (Botea et al. 2005) or MUM (Chrpa, Vallati, and McCluskey 2014) also eliminate underrepresented macros from the same reason.

For macros that are assembled from the same operators but differ by being goal achieving, the most restrictive macro (achieving most goals) is only considered.

Aggressive Approach

Adding (sound) macros into a domain model does not compromise completeness. On the other hand, the size of the (grounded) representation can considerably grow as macros often have more instances than ordinary operators, due to the larger number of arguments. Consequently, planners might suffer with increased memory requirements and with extra burden in pre-processing.

To mitigate such an issue, original operators that are effectively replaced by macros can be removed from the domain model. Critical Section Macros have a good potential to replace original operators that operate with particular resources because the activities these macros represent have to be usually performed either as whole or not at all. For example, in Gripper, a Critical Section Macro pick-move-drop replaces original operators pick and drop (unless some robot initially holds some ball or it is required that some robot holds some ball in a goal state).

The aggressive version of our Critical Section Macro approach consists of the following steps:

1. Generate Critical Section Macros (Algorithm 1) and add them into the domain model.
2. Remove lockers and releasers of each macro from the domain model.
3. Generate plans for the training tasks with the modified domain model.
4. If some task cannot be solved, then fail (removed original operators are necessary).
5. Otherwise analyse the plans and eventually remove also those operators whose instances are never used in these plans.

The aggressive approach can compromise completeness as it can remove original operators that might be necessary to solve some (non-training) tasks. On the other hand, the aggressive approach by removing original operators can (sometimes considerably) reduce the size of the representation. In the Gripper example, removing the pick and drop operators prunes out states in which a ball is held by a robotic gripper and thus considerably reduces the size of (grounded) representation. The risk of making a task unsolvable can be alleviated by incorporating aggressive approaches into portfolios containing conservative components (e.g., the original model, a model with macros but containing all original operators).

Combination with other Approaches

Critical Section Macros focus on capturing activities in which a resource is locked. Other approaches consider different criteria for macro generation such as frequency of operators' "consecutivity" or the possible number of macros' instances. In particular, "chaining" approaches such as MacroFF (Botea et al. 2005) or MUM (Chrpa, Vallati, and McCluskey 2014), which construct macros iteratively, have a good potential to complement our approach as they can possibly chain the activities into longer and more useful macros.

Combining Critical Section Macros with other approaches (e.g., MUM) can be done straightforwardly. Critical Section Macros can be generated, as described in the previous sections, and then added to the domain model. Such enhanced domain model is then used for the training of a different macro generation approach. In fact, for the other macro learning approach, Critical Section Macros can be considered as original operators, and will be treated as original operators. On the other hand, as some macro learning approaches perform filtering of unpromising macros it might be useful to consider Critical Section Macros as macros in certain occasions. In particular, MUM filters out macros that are replaced by longer macros (e.g. a move-drop macro can be replaced by the pick-move-drop macro). Also, MUM uses "entanglements" to eliminate possibly unpromising instances of macros. For such occasions, we consider Critical Section Macros as macros. In all the remaining cases, Critical Section Macros are considered as ordinary operators, so they cannot be filtered out from different reasons.

Experimental Results

The purpose of this experimental analysis is to i) evaluate planners' performance on Critical Section macros as well as their combination with MUM (both conservative and aggressive versions), ii) compare them against related state-of-the-art techniques, MUM (Chrpa, Vallati, and McCluskey 2014) and BloMa (Chrpa and Siddiqui 2015) and iii) analyse impact of quality of training plans on generated macros utility . We considered domains from the learning track of IPC 2008 and 2011.

We have selected 6 state-of-the-art planning engines, according to their results in recent IPCs, and to the exploited planning techniques, namely: *LAMA* (Richter and Westphal 2010), *Probe* (Lipovetzky et al. 2014), *MpC* (Rintanen 2014), *Yahsp3* (Vidal 2014), *FDSS 2018* (Seipp and Röger 2018) and *Dual BFWS* (Lipovetzky et al. 2018).

Evaluation Metrics. Three metrics were used to evaluate planners' performance, namely *coverage* (number of solved problems), *PAR10 score* and *IPC quality score*. For each testing task time limit of 900 seconds and memory limit of 4 GB is applied (as in the learning tracks of IPCs). All the experiments were conducted on Intel Xeon E5 2.0 Ghz, Debian 9.

Penalised Average Runtime (PAR10) score is a metric usually exploited in machine learning and algorithm configuration techniques. This metric trades off coverage and runtime for solved problems: if a planner p solves a problem instance Π in time $t \leq T$ ($T = 900$s in our case), then $\mathit{PAR10}(p, \Pi) = t$, otherwise $\mathit{PAR10}(p, \Pi) = 10T$ (i.e., 9000s in our case).

IPC quality score is defined as in the learning track of IPC-7 (Coles et al. 2012) as follows. For an encoding e of a problem instance Π, $\mathit{IPC}(\Pi, e)$ is 0 if Π is unsolved in e, and $(m^*_{\Pi,e}/m_{\Pi}))$, where $m_{\Pi,e}$ is the cost of the plan of Π in e and m^*_{Π} is the smallest cost of the plan of Π in any considered encodings, otherwise.

Learning. We have considered two methodologies, one that have been used by MUM (Chrpa, Vallati, and McCluskey 2014) and one that have been used by BloMa (Chrpa and Siddiqui 2015). For both methodologies, we considered 6 training tasks per each domain such that their plan length was mostly within 40-80 actions[1]. Both methodologies consider one training plan per a training task.

The MUM methodology uses the same planner for generating training plans as for solving testing tasks. In other words, planners learn macros for themselves (and not for other planners). This methodology follows an intuition that most promising knowledge for a given planner can be extracted by analysing its outputs (plans).

The BloMa methodology, in contrast, selects a planner which generates, for a given domain, best quality training plans (e.g. the shortest plans). This methodology follows an intuition that good quality training plans yield to most promising knowledge for all planners.

The threshold for "underrepresented" macros was set to 6 (as the number of training tasks). The learning process took at most several seconds.

Results. The results for domains in which Critical Section Macros were generated are summarised in Table 1 and 2 for the MUM learning methodology (i.e., a planner learns for itself) and the BloMa learning methodology (i.e., considering best quality training plans), respectively. We can observe that macros learnt by the BloMa methodology perform better across the considered domains and planners. For example, in Parking, Critical Sequence Macros generated from poor quality training plans capture meaningless activities (e.g. moveCarToCar-moveCarToCar) while no macros were generated from the best quality training plans. Intuitively, better quality training plans carry better information that can be exploited by a range of planning engines.

The conservative variant of Critical Section Macros outperforms BloMa (in PAR10) in about 72% of cases considering the MUM learning methodology while in about 63% of cases considering the BloMa learning methodology. With regards to MUM, the results are mixed, in about 50% of cases the conservative variant of Critical Section Macros outperforms MUM (in both methodologies). Combination of Critical Section Macros and MUM (the conservative version) outperforms both MUM and Critical Section Macros in about 60% of cases (overall). The aggressive versions outperform the corresponding conservative versions in about 90% of cases in which the aggressive versions generated

[1]For the IPC 2011 domains, we used provided problem generators while for the IPC 2008 domains, we selected the tasks from the provided sets of "bootstrap" tasks.

Planner	Coverage							PAR10							IPC Quality						
	O	M	B	C	CM	AC	ACM	O	M	B	C	CM	AC	ACM	O	M	B	C	CM	AC	ACM
barman																					
lama	2	-	19	30	30	30	30	8428	-	3653	385	8.6	11	1.8	1.7	-	18.9	26.4	29.9	29.9	29.9
probe	2	-	7	2	3	24	23	8427	-	7022	8455	8136	2104	2379	1.4	-	4.8	1.7	2.6	23.9	22.8
MpC	0	-	0	0	0	0	0	9000	-	9000	9000	9000	9000	9000	0.0	-	0.0	0.0	0.0	0.0	0.0
yahsp	0	-	-	0	0	0	0	9000	-	-	9000	9000	9000	9000	0.0	-	-	0.0	0.0	0.0	0.0
BFWS	0	-	0	3	0	12	30	9000	-	9000	8148	9000	5606	1.2	0.0	-	0.0	2.8	0.0	12.0	29.2
FDSS	20	-	24	30	30	30	30	3234	-	2164	330	166	16	2.1	17.5	-	23.9	29.3	29.3	29.3	29.3
bw																					
lama	28	29	0	28	29	28	28	681	411	9000	711	379	617	616	24.5	22.2	0.0	22.1	23.4	23.0	23.0
probe	25	-	30	30	30	30	30	1679	-	156	215	195	0.4	0.3	21.4	-	28.4	27.2	27.1	27.5	29.9
MpC	0	-	0	30	30	11	18	9000	-	9000	177	173	5700	3600	0.0	-	0.0	12.3	28.7	3.7	16.7
yahsp	27	24	22	26	26	30	30	948	1833	2437	1239	1227	0.1	0.1	5.9	19.6	16.4	22.4	20.3	27.1	27.2
BFWS	3	-	0	0	0	0	0	8106	-	9000	9000	9000	9000	9000	3.0	-	0.0	0.0	0.0	0.0	0.0
FDSS	22	20	21	19	22	30	30	2506	3104	2780	3387	2492	11	11	18.0	14.9	14.7	14.7	15.1	23.4	23.4
depots																					
lama	0	-	0	2	0	30	30	9000	-	9000	8417	9000	0.4	0.3	0.0	-	0.0	1.4	0.0	29.5	29.9
probe	30	30	30	30	30	30	30	39	40	21	88	42	0.3	0.1	26.1	26.4	27.0	25.8	27.4	28.0	29.4
MpC	17	25	0	15	24	30	30	3985	1635	9000	4589	1944	0.4	0.1	11.7	18.6	0.0	11.2	17.7	28.7	29.8
yahsp	21	-	12	5	5	30	30	2809	-	5558	7545	7555	1.4	0.1	2.3	-	1.8	0.7	0.7	27.4	30.0
BFWS	9	15	3	15	14	30	30	6383	4577	8108	4555	4892	0.1	0.2	5.9	7.6	1.3	9.1	6.4	28.1	28.1
FDSS	19	-	0	14	12	30	30	3838	-	9000	5024	5602	0.6	0.5	18.9	-	0.0	12.7	11.1	27.5	28.4
gripper																					
lama	6	30	17	30	30	30	30	7342	101	4242	160	105	4.8	4.2	5.6	29.9	16.9	29.9	29.9	25.7	25.7
probe	0	0	0	0	-	30	30	9000	9000	9000	9000	-	16	17	0.0	0.0	0.0	0.0	-	29.9	29.9
MpC	0	0	0	0	0	30	30	9000	9000	9000	9000	9000	48	1.3	0.0	0.0	0.0	0.0	0.0	29.7	29.8
yahsp	0	-	0	0	0	0	30	9000	-	9000	9000	9000	9000	0.2	0.0	-	0.0	0.0	0.0	0.0	30.0
BFWS	0	5	5	0	5	27	27	9000	7527	7539	9000	7527	1372	1382	0.0	4.9	4.9	0.0	4.9	26.9	26.9
FDSS	0	14	0	9	13	30	-	9000	5082	9000	6492	5361	7.3	-	0.0	13.9	0.0	8.9	12.9	29.9	-
matching-bw																					
lama	26	30	-	30	29	30	30	1202	2.8	-	2.5	301	0.1	0.1	18.3	21.9	-	20.4	20.9	23.4	29.5
probe	13	23	0	22	23	-	-	5108	2123	9000	2434	2115	-	-	9.7	19.2	0.0	20.3	20.5	-	-
MpC	0	0	-	26	24	-	-	9000	9000	-	1294	1885	-	-	0.1	0.1	-	22.7	20.8	-	-
yahsp	0	25	-	26	26	30	30	9000	1523	-	1201	1204	0.1	0.1	0.1	14.0	-	17.3	17.2	22.4	29.8
BFWS	11	15	0	27	29	30	30	5748	4604	9000	987	444	0.1	0.1	8.8	12.3	0.0	21.5	25.2	27.4	27.5
FDSS	30	30	30	30	30	30	30	2.2	2.0	1.8	2.3	1.7	0.2	0.2	22.4	21.4	21.8	20.4	21.0	24.6	29.4
parking																					
lama	21	-	0	-	-	-	-	2896	-	9000	-	-	-	-	20.0	-	0.0	-	-	-	-
probe	2	-	0	0	0	0	0	8436	-	9000	9000	9000	9000	9000	2.0	-	0.0	0.0	0.0	0.0	0.0
MpC	5	-	0	0	-	-	-	7539	-	9000	9000	-	-	-	5.0	-	0.0	0.0	-	-	-
yahsp	0	-	0	0	-	0	0	9000	-	9000	9000	-	9000	9000	0.0	-	0.0	0.0	-	0.0	0.0
BFWS	20	-	0	-	-	-	-	3087	-	9000	-	-	-	-	19.9	-	0.0	-	-	-	-
FDSS	10	-	0	-	-	-	-	6155	-	9000	-	-	-	-	9.4	-	0.0	-	-	-	-
rovers																					
lama	30	30	30	30	30	30	30	153	134	129	136	116	101	81	27.0	26.7	26.7	28.4	28.7	29.3	29.5
probe	29	26	29	30	28	29	29	699	1541	640	400	954	624	620	28.1	25.4	28.3	28.9	27.2	28.0	28.3
MpC	9	3	5	7	4	8	4	6512	8181	7602	7060	7898	6783	7913	8.7	2.1	2.6	6.9	2.9	6.6	2.6
yahsp	30	30	30	30	30	30	30	19	19	21	18	18	18	19	26.7	26.6	26.5	29.6	28.1	29.2	29.5
BFWS	17	12	1	19	16	-	-	4197	5596	8718	3590	4449	-	-	16.7	11.7	0.9	18.2	15.4	-	-
FDSS	30	30	24	30	30	30	30	372	306	2207	354	280	301	274	27.4	27.4	21.9	29.1	29.0	28.9	28.7
sokoban																					
lama	20	-	21	22	19	-	-	3017	-	2731	2431	3326	-	-	17.9	-	17.7	18.3	14.2	-	-
probe	25	-	27	29	28	-	-	1539	-	924	329	631	-	-	21.0	-	23.2	23.5	23.0	-	-
MpC	30	-	30	30	29	-	-	1.3	-	1.0	2.7	303	-	-	27.6	-	28.3	24.5	24.7	-	-
yahsp	25	-	25	28	27	-	-	1527	-	1577	724	966	-	-	17.2	-	13.3	25.6	24.5	-	-
BFWS	30	-	30	30	30	-	-	1.2	-	3.0	1.9	2.3	-	-	28.5	-	26.3	27.8	28.3	-	-
FDSS	30	-	30	30	30	-	-	22	-	13	37	30	-	-	27.3	-	20.2	24.0	24.5	-	-

Table 1: Coverage, average PAR10 score (in seconds), and IPC quality score of the (O)riginal, (M)UM, (B)LoMa, (C)ritical Section Marcos, Aggressive Critical Section Macros (AC) and their combination with MUM (CM, ACM respectively) encodings, using the MUM learning methodology. ”-” denotes that no macros have been generated. Gray indicates cases where C or CM model allow the planner to outperform the Original, MUM, and BloMa models in terms of PAR10. Light gray is used for cases where the improvement is achieved only by AC or ACM.

macros and successfully removed the replaced original operators.

Interestingly, in 6 domains (Barman, BlocksWorld, Depots, Gripper, Matching-Bw, Rovers), the aggressive variant (the BloMA learning methodology) generates Critical Section Macros that can replace all primitive operators operating with a resource. Only in Sokoban, original operators cannot be removed as it renders training tasks unsolvable (a macro push-push cannot replace the push operator). Combination of aggressive Critical Section Macros with MUM, especially when using the BloMa learning methodology, leads to further performance boost. In Bar-

Planner	Coverage							PAR10							IPC Quality						
	O	M	B	C	CM	AC	ACM	O	M	B	C	CM	AC	ACM	O	M	B	C	CM	AC	ACM
barman																					
lama	2	-	14	30	30	30	30	8428	-	5022	396	8.5	11	1.9	1.7	-	13.9	26.4	30.0	30.0	30.0
probe	2	-	24	0	30	30	30	8427	-	2044	9000	43	231	0.5	1.4	-	22.5	0.0	29.9	29.9	29.9
MpC	0	-	0	0	30	1	30	9000	-	9000	9000	11	8700	0.6	0.0	-	0.0	0.0	30.0	1.0	30.0
yahsp	0	-	29	30	30	30	30	9000	-	442	253	0.3	10	0.1	0.0	-	21.5	29.1	30.0	30.0	30.0
BFWS	0	-	30	20	30	30	30	9000	-	4.9	3007	2.2	1.8	0.1	0.0	-	30.0	18.7	28.2	28.2	28.2
FDSS	20	-	25	30	30	30	30	3234	-	1820	310	152	15	1.9	17.6	-	24.9	29.3	29.4	29.4	29.4
bw																					
lama	28	-	29	28	29	28	30	681	-	380	711	404	616	0.4	15.3	-	12.1	12.7	13.0	12.8	30.0
probe	25	-	29	30	30	30	30	1679	-	558	214	172	0.4	0.3	21.4	-	27.3	27.2	27.0	27.3	29.9
MpC	0	-	0	30	30	11	20	9000	-	9000	172	168	5700	3012	0.0	-	0.0	9.5	22.0	2.2	20.0
yahsp	27	-	24	26	27	30	30	948	-	1848	1235	950	0.1	0.2	4.4	-	13.7	18.1	18.5	20.5	30.0
BFWS	3	-	1	1	3	30	30	8106	-	8700	8709	8114	58	3.2	0.6	-	0.1	0.4	1.3	28.2	29.9
FDSS	22	-	22	21	21	30	30	2506	-	2493	2806	2794	11	0.6	14.2	-	10.4	12.0	11.2	16.3	30.0
depots																					
lama	0	0	0	2	0	30	30	9000	9000	9000	8417	9000	0.5	0.3	0.0	0.0	0.0	1.4	0.0	29.5	29.9
probe	30	30	30	30	30	30	30	39	38	43	87	37	0.3	0.1	26.2	26.8	26.7	25.8	27.6	28.0	29.4
MpC	17	25	23	14	24	30	30	3985	1649	2221	4883	1947	0.4	0.1	11.7	18.6	17.1	10.6	17.9	28.7	29.8
yahsp	21	20	20	5	20	30	30	2809	3050	3061	7550	3060	1.4	0.1	2.3	3.2	3.2	0.7	3.2	27.4	30.0
BFWS	9	13	14	15	15	30	30	6383	5180	4873	4550	4620	0.1	0.1	5.3	9.3	9.0	7.8	10.8	20.4	29.7
FDSS	19	12	13	12	12	30	30	3838	5597	5325	5592	5601	0.6	0.4	18.9	11.2	12.2	11.0	11.2	27.5	28.4
gripper																					
lama	6	30	30	24	27	30	30	7342	101	103	1926	985	4.8	4.1	5.6	30.0	30.0	24.0	27.0	25.8	25.8
probe	0	0	0	0	0	30	30	9000	9000	9000	9000	9000	17	17	0.0	0.0	0.0	0.0	0.0	29.9	29.9
MpC	0	0	0	0	0	30	30	9000	9000	9000	9000	9000	48	1.3	0.0	0.0	0.0	0.0	0.0	29.7	29.8
yahsp	0	0	0	0	0	0	30	9000	9000	9000	9000	9000	9000	0.2	0.0	0.0	0.0	0.0	0.0	0.0	30.0
BFWS	0	5	5	0	5	27	27	9000	7529	7526	9000	7525	1371	1381	0.0	4.9	4.9	0.0	4.9	26.9	26.9
FDSS	0	15	14	9	10	30	30	9000	4802	5081	6493	6202	7.2	6.4	0.0	14.9	13.9	8.9	9.9	29.9	29.9
matching-bw																					
lama	26	26	0	30	30	30	30	1202	1202	9000	2.6	2.2	0.1	0.1	21.2	20.7	0.0	25.3	25.8	27.4	27.4
probe	13	18	0	30	30	30	30	5108	3610	9000	0.7	1.5	0.1	0.1	7.1	11.2	0.0	29.0	28.3	28.8	28.9
MpC	0	0	0	25	17	15	20	9000	9000	9000	1582	3982	4500	3000	0.0	0.0	0.0	17.3	13.7	9.0	19.9
yahsp	0	28	0	25	25	30	30	9000	639	9000	1501	1512	0.1	0.1	0.0	20.9	0.0	21.9	20.3	27.7	27.7
BFWS	11	15	0	27	29	30	30	5748	4604	9000	985	442	0.1	0.1	8.8	12.3	0.0	21.5	25.2	27.4	27.5
FDSS	30	30	15	30	30	30	30	2.2	1.8	4505	2.4	1.9	0.1	0.1	24.2	24.6	10.0	22.5	23.4	25.9	25.9
rovers																					
lama	30	30	30	30	30	30	30	153	123	163	118	91	110	81	26.6	26.2	26.3	29.5	29.6	28.8	29.0
probe	29	23	22	29	30	28	29	699	2392	2789	675	361	907	634	28.1	22.5	20.8	28.0	29.3	27.0	28.1
MpC	9	6	8	7	7	8	8	6512	7333	6786	7060	7058	6787	6788	8.3	5.6	7.5	6.6	6.9	6.3	6.7
yahsp	30	30	30	30	30	30	30	19	18	17	18	18	18	18	26.2	26.6	26.1	29.1	29.8	28.6	29.3
BFWS	17	11	2	18	15	21	19	4197	5881	8441	3854	4734	3016	3615	16.7	10.8	1.9	17.3	14.4	19.9	17.9
FDSS	30	30	23	30	30	30	30	372	302	2477	354	258	348	248	27.4	27.3	20.8	29.1	28.9	28.6	28.7
sokoban																					
lama	20	-	12	22	19	-	-	3017	-	5414	2431	3326	-	-	18.5	-	9.8	18.8	14.6	-	-
probe	25	-	27	29	28	-	-	1539	-	918	329	631	-	-	20.9	-	22.1	24.1	23.1	-	-
MpC	30	-	30	30	30	-	-	1.3	-	1.1	2.6	4.7	-	-	27.6	-	28.3	24.5	25.3	-	-
yahsp	25	-	26	28	27	-	-	1527	-	1297	722	965	-	-	17.2	-	13.8	25.6	24.5	-	-
BFWS	30	-	30	30	30	-	-	1.2	-	2.5	1.8	2.2	-	-	28.5	-	26.3	27.8	28.3	-	-
FDSS	30	-	30	26	29	-	-	22	-	12	1220	329	-	-	27.4	-	20.3	20.8	23.6	-	-

Table 2: Coverage, average PAR10 score (in seconds), and IPC quality score of the (O)riginal, (M)UM, (B)LoMa, (C)ritical Section Marcos, Aggressive Critical Section Macros (AC) and their combination with MUM (CM, ACM respectively) encodings, using the BloMa learning methodology. "-" denotes that no macros have been generated. Gray indicates cases where C or CM model allow the planner to outperform the Original, MUM, and BloMa models in terms of PAR10. Light gray is used for cases where the improvement is achieved only by AC or ACM.

man, BlocksWorld, Depots, Gripper, Matching-Bw, there exists at least one planner that solves each "enhanced" testing task within 1 second in average (that is considerably better than for the original tasks). Critical Section Macros are particularly useful in Barman, where we could learn two "long" macros, one which puts two ingredients into the shaker and cleans the shot used to fill the shaker, and the other which shakes the cocktail, pours it into the required shot and cleans the shaker afterwards. When combined with MUM, a "supermacro" assembling these two macros together was generated (hence, a cocktail can be made in one step). In Rovers, Critical Section Macros capture more marginal activities (sampling and dropping rock or soil) and thus the macros are improving performance marginally.

The IPC quality score indicates that the expectable plan quality degradation by macro use is marginal (the score is often close to the coverage). In Bw and Depots, plan quality even increases when macros are used.

We can observe that if Critical Section Macros capture more complex activities such as in Barman their impact on performance tend to be larger. In contrast, if Critical Section Macros capture non-frequent activities (i.e., macros are used a few times in plans) their impact is marginal such as in Rovers. Replacing all original operators dealing with a re-

source in the aggressive variant usually results in a considerable performance improvement. In the conservative variant, reasoning with such a resource is not completely eliminated as the original operators remain in the domain model and hence planning engines might not fully benefit from Critical Section Macros. Finally, using better quality training plans leads to more representative macros.

Conclusion

In this paper, we introduced Critical Section Macros that are inspired by resource locking in critical sections in parallel computing. Roughly speaking, these macros capture whole activities in which a resource is locked (e.g., shaking a cocktail and cleaning the shaker afterwards). The results have shown the usefulness of Critical Section Macros, especially in domains where non-trivial activities can be captured (e.g. Barman) or where original operators dealing with resources can be removed (e.g. BlocksWorld, Gripper).

In future, we would like to extend the methods for temporal planning, since we believe that the nature Critical Section Macros can capture useful activities also in partially ordered action sequences common in temporal plans.

Acknowledgements

This research was funded by the Czech Science Foundation (project no. 18-07252S). Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

References

Alhossaini, M. A., and Beck, J. C. 2013. Instance-specific remodelling of planning domains by adding macros and removing operators. In *Proceedings of SARA*, 16–24.

Asai, M., and Fukunaga, A. 2015. Solving large-scale planning problems by decomposition and macro generation. In *ICAPS*, 16–24.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1-2):5–33.

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-FF: improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research (JAIR)* 24:581–621.

Chrpa, L., and McCluskey, T. L. 2012. On exploiting structures of classical planning problems: Generalizing entanglements. In *Proceedings of ECAI*, 240–245.

Chrpa, L., and Siddiqui, F. H. 2015. Exploiting block deordering for improving planners efficiency. In *IJCAI*, 1537–1543.

Chrpa, L.; Vallati, M.; and McCluskey, T. L. 2014. MUM: a technique for maximising the utility of macro-operators by constrained generation and use. In *ICAPS*, 65–73.

Coles, A.; Coles, A.; Olaya, A. G.; Jiménez, S.; Lòpez, C. L.; Sanner, S.; and Yoon, S. 2012. A survey of the seventh international planning competition. *AI Magazine* 33:83–88.

Coles, A.; Fox, M.; and Smith, A. 2007. Online identification of useful macro-actions for planning. In *Proceedings of ICAPS*, 97–104.

Dawson, C., and Siklóssy, L. 1977. The role of preprocessing in problem solving systems. In *Proceedings of IJCAI*, 465–471.

Dulac, A.; Pellier, D.; Fiorino, H.; and Janiszek, D. 2013. Learning useful macro-actions for planning with n-grams. In *ICTAI*, 803–810.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning, theory and practice*. Morgan Kaufmann.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research (JAIR)* 14:253–302.

Hoffmann, J. 2011. Analyzing search topology without running any search: On the connection between causal graphs and h+. *Journal Artificial Intelligence Research (JAIR)* 41:155–229.

Hofmann, T.; Niemueller, T.; and Lakemeyer, G. 2017. Initial results on generating macro actions from a plan database for planning on autonomous mobile robots. In *ICAPS*, 498–503.

Korf, R. 1985. Macro-operators: A weak method for learning. *Artificial Intelligence* 26(1):35–77.

Lipovetzky, N.; Ramirez, M.; Muise, C.; and Geffner, H. 2014. Width and inference based planners: Siw, bfs(f), and probe. In *The Eighth IPC. Description of Participant Planners of the Deterministic Track*, 6–7.

Lipovetzky, N.; Ramirez, M.; Frances, G.; and Geffner, H. 2018. Best-first width search in the ipc2018: Complete, simulated, and polynomial variants. In *The Ninth International Planning Competition. Description of Participant Planners of the Deterministic Track*.

Minton, S. 1988. Quantitative results concerning the utility of explanation-based learning. In *Proceedings of AAAI*, 564–569.

Newton, M. A. H.; Levine, J.; Fox, M.; and Long, D. 2007. Learning macro-actions for arbitrary planners and domains. In *Proceedings of ICAPS*, 256–263.

Richter, S., and Westphal, M. 2010. The LAMA planner: guiding cost-based anytime planning with landmarks. *Journal Artificial Intelligence Research (JAIR)* 39:127–177.

Rintanen, J. 2014. Madagascar: Scalable planning with sat. In *The Eighth IPC. Description of Participant Planners of the Deterministic Track*, 66–70.

Seipp, J., and Röger, G. 2018. Fast downward stone soup 2018. In *The Ninth International Planning Competition. Description of Participant Planners of the Deterministic Track*.

Siddiqui, F., and Haslum, P. 2012. Block-structured plan deordering. In *25th Australasian Joint Conference*, volume 7691 of *LNAI*, 803–814.

Vidal, V. 2014. Yahsp3 and yahsp3-mt in the 8th international planning competition. In *The Eighth IPC. Description of Participant Planners of the Deterministic Track*, 64–65.

A.10 Inner entanglements: Narrowing the search in classical planning by problem reformulation

L. Chrpa, M. Vallati, and T. L. McCluskey. Inner entanglements: Narrowing the search in classical planning by problem reformulation. *Computational Intelligence*, 35(2):395–429, 2019.

Received: 24 June 2016 | Revised: 6 July 2017 | Accepted: 25 February 2019

DOI: 10.1111/coin.12203

ORIGINAL ARTICLE

WILEY Computational Intelligence

Inner entanglements: Narrowing the search in classical planning by problem reformulation

Lukáš Chrpa[1,2] | Mauro Vallati[3] | Thomas Leo McCluskey[3]

[1]Faculty of Electrical Engineering, Czech Technical University in Prague, Prague, Czech Republic

[2]Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

[3]School of Computing and Engineering, University of Huddersfield, Huddersfield, UK

Correspondence
Lukáš Chrpa, Faculty of Electrical Engineering, Czech Technical University in Prague, 160 00 Prague 6, Czech Republic; or Faculty of Mathematics and Physics, Charles University, 116 36 Prague 1, Czech Republic.
Email: chrpaluk@fel.cvut.cz

Funding information
UK Engineering and Physical Sciences Research Council Autonomous and Intelligent Systems Programme, Grant/Award Number: EP/J011991/1; Czech Science Foundation, Grant/Award Number: 17-17125Y; OP VVV-funded project, Grant/Award Number: CZ.02.1.01/0.0/0.0/16_019/0000765

Abstract

In the field of automated planning, the central research focus is on domain-independent planning engines that accept planning tasks (domain models and problem descriptions) in a description language, such as Planning Domain Definition Language, and return solution plans. The performance of planning engines can be improved by gathering additional knowledge about specific planning domain models/tasks (such as control rules) that can narrow the search for a solution plan. Such knowledge is often learned from training plans and solutions of simple tasks. Using techniques to reformulate the given planning task to incorporate additional knowledge, while keeping to the same input language, allows to exploit off-the-shelf planning engines. In this paper, we present *inner entanglements* that are relations between pairs of operators and predicates that represent the exclusivity of predicate achievement or requirement between the given operators. Inner entanglements can be encoded into a planner's input language by transforming the original planning task; hence, planning engines can exploit them. The contribution of this paper is to provide an in-depth analysis and evaluation of inner entanglements, covering theoretical aspects such as complexity results, and an extensive empirical study

using International Planning Competition benchmarks and state-of-the-art planning engines.

KEYWORDS

classical planning, knowledge representation, machine learning, problem reformulation

1 | INTRODUCTION

Automated planning is an important research area of artificial intelligence (AI), where an autonomous entity (eg, a robot) reasons about the way it can act in order to achieve its goals. AI planning has therefore a great potential for applications where a certain level of autonomy is required, such as in the Deep Space 1 mission.[1] Classical planning is a subarea of AI planning that deals with a static and fully observable environment and where actions have deterministic and instantaneous effects. Classical planning is, however, intractable (PSPACE-complete).[2]

In the last few decades, there has been a great deal of activity in the research community designing planning techniques and planning engines. In 1998, the International Planning Competition (IPC)* was organized and has since been increasingly attracting the attention of the AI planning community. Due to the IPC, we have the Planning Domain Definition Language (PDDL),[3] which is a widely used language for describing planning tasks, and a wide range of benchmarks that can be used for measuring planners' performance. Currently, PDDL is supported by a large number of advanced planning engines. Along with those planning engines, many novel planning techniques have been proposed, such as heuristic search,[4] translating planning tasks into SAT,[5] just to mention a few.

The performance of planning engines can be improved by restricting the search space, ie, by introducing pruning techniques that "cut off" branches that are unnecessary or redundant. *Commutativity pruning* eliminates all but one permutation of commutative actions (can be applied in any order).[6] *Symmetry breaking* reuses information about one object to its symmetric "twin" in such a way that "bad" states of one object can be avoided for its symmetric "twin."[7] *Reachability analysis* can determine whether the goal is unreachable from a current state.[4]

Another way how performance of planning engines can be improved is by gathering domain control knowledge (DCK), ie, additional knowledge about planning tasks indicating how solution plans would look like. DCK can be expressed, for instance, in the form of control rules,[8] temporal logic formulas,[9] or decision trees.[10] With growing interest in extracting DCK automatically, emphasis was given on exploiting machine learning techniques that can acquire useful DCK, usually, by analyzing "training plans," which are solutions of simple planning tasks. This motivated the foundation of the learning track in the IPC, which has been organized since 2008. It should be noted that an approach that learns DCK from relaxed plans (obtained by solving planning tasks while omitting negative effects of actions)[11] won the best learner award at IPC 2008. However, such types of knowledge often require specific planning engines such as TALplanner[12] in the case of control rules. Alternatively, DCK can be directly encoded into the domain and problem descriptions (usually in PDDL). Such an approach is planner independent; hence, a standard planning engine can straightforwardly exploit it.

The best-known planning task reformulation technique, macro-operators ("macros"), which encapsulate sequences of PDDL operators, can be encoded as normal planning operators; hence,

*http://ipc.icaps-conference.org

they can be easily added into domain models.[13-16] Abstracting planning tasks by their reformulation in order to reveal their hierarchical structures can mitigate the "accidental complexity" of their domain models.[†17,18]

Apart from macros, another type of domain-independent DCK is *Entanglements*,[19,20] which represent relations between planning operators and predicates, aiming at eliminating unpromising alternatives in a planning engine's search space. Technically speaking, entanglements are task specific, ie, relations described by entanglements hold in at least one solution plan of a given task. Entanglements usually generalize well, that is, a set of entanglements holds for a class of planning tasks with the same domain model.

Outer entanglements[19] are relations between planning operators and predicates whose instances are present in the initial state or the goal. *Inner entanglements*,[20,21] on which we focus in this paper, are relations of the exclusivity of predicate achievement or requirement between pairs of operators. Inner entanglements can be encoded in planning tasks, effectively reformulating them, and thus, they are planner independent. Deciding whether a given inner entanglement holds in a given planning task is generally intractable (PSPACE-complete) and, thus, as hard as solving a planning task. Such a theoretical result indicates practical infeasibility of enumerating entanglements for a given task prior to solving it. Since inner entanglements generalize well as reported in the literature,[20,21] ie, they are rather domain specific than task specific, for extracting them, we have exploited the "learning for planning" paradigm, which identifies DCK from a set of "training" planning tasks. Therefore, inner entanglements can be learned on simple training tasks, which are easy to solve, and then used for more complex tasks (in the same class) for speeding up the plan generation process. Our initial work on inner entanglements has been reported in a couple of shorter papers detailing their discovery, use, and effectiveness.[20,21] In this paper, we integrate and extend our previous work, with

- a detailed description of the encodings of inner entanglements, including formal proofs of their correctness;
- a collected summary of the known complexity results and trivial cases where inner entanglements hold;
- case studies in which we investigate the knowledge engineering aspects of (re)using inner entanglements;
- an analysis of the potential impact of inner entanglements on the planning process;
- an approximation method for extracting entanglements enriched by filtering unpromising inner entanglements; and
- an extensive empirical study of the impact of inner entanglements in the planning process using all the domains from the 7th IPC's learning track[‡] and seven state-of-the-art planning engines based on very different principles.

Although our approximation method for learning inner entanglements does not theoretically guarantee that the reformulated tasks remain solvable, the main empirical findings from this paper are that the use of inner entanglements improves the planning process generally through the considered planner and domain model combinations. In addition, in the experimental scenarios we used, the potential for identification of incorrect inner entanglements stemming from our

[†] "Accidental complexity of domain models" means that their inefficient encodings decrease the performance of planning engines.

[‡] Learning track benchmarks are more natural, since the inner-entanglement extraction phase can be understood as a learning process.

approximation method for their extraction did not explicitly manifest itself in the results. Using the "learning for planning" paradigm, ie, learning domain-specific knowledge on a small set of training tasks, has demonstrated its usefulness in the inner-entanglement case. Noticeably, the issue of making some reformulated tasks unsolvable can be alleviated (i) by running the planner on the original task if the reformulated task was unsolved, (ii) by domain engineers who can verify the correctness of learned inner entanglements, or (iii) by incorporating reformulated tasks along with the original ones into portfolios such as PbP.[22]

This paper is organized as follows. After discussing related work, basic terminology is provided. Then, inner entanglements are introduced. After that, the reformulation of planning tasks in order to enforce inner entanglements is presented. Then, a theoretical analysis of inner entanglements is provided, and an approximation algorithm for extracting inner entanglements is presented (including the filtering technique for unpromising inner entanglements). After that, an empirical analysis of the impact of inner entanglements in the planning process is provided. Finally, we give conclusions and present future avenues of research.

2 | RELATED WORK

Generating DCK, which can be exploited by planning engines, dates back to the 1970s, when systems such as REFLECT[23] were developed. Macros are one of the best-known type of DCK in classical planning, because they can be encoded as normal planning operators and, thus, easily added into planning domain models.[14] Macro-FF CA-ED version,[24] which learns macros through an analysis of relations between static predicates; Wizard,[25] which learns macros by genetic algorithms; and BLOMA,[26] which exploits a block decomposition technique[27] to learn "long" macros, are good examples of planner-independent macro learning systems. Although macros and inner entanglements are based on a similar idea, ie, enforcing (primitive) operators to be applied in certain order, inner entanglements do not require the affected operators to be applied strictly consecutively, and inner entanglements can be represented in such a way that the number of operators' instances (after grounding) is not higher than when the original models are considered. The relation between inner entanglements and macros and how inner entanglements can be exploited for macro learning has been studied in the work of Chrpa et al.[28]

A general technique, called *commutativity pruning*, is used to discard all but one permutation of commutative (or independent) actions, which do not influence each other and, thus, can be executed in any order.[6] Graphplan,[29] which is one of the best-known planning algorithms, allows the execution of commutative actions in parallel (in one step). *Symmetry breaking* is a well-known technique for pruning unneeded alternatives in the search space. In planning, some objects might be symmetric, which can be exploited for avoiding alternatives concerning one object that has been already tried with the object's symmetric "twin."[7]

In the spirit of the works of Emerson and Sistla[30] and Rintanen,[31] Pochter et al[32] present a pruning technique that identifies symmetries by exploring automorphisms in state-transition systems. This approach has been recently extended for cost-optimal planning.[33] Motivated by the idea of partial order–based reduction used in model checking,[34] Chen and Yao[35] introduce an *Expansion Core* method, focusing on cost-optimal SAS+ planning,[36] which, in a node expansion phase (in the A* search), restricts on relevant domain transition graphs rather than all of them. The idea of "expansion cores" is extended into *strong stubborn sets* that guarantee stronger pruning than "expansion cores."[37] In contrast, inner entanglements prune asymmetrical alternatives. Outer entanglements[19] are relations between operators and the initial or goal atoms that aim to prune

unpromising instances of these operators. Outer and inner entanglements are complementary as has already been demonstrated in the literature.[38]

A recent work, which is, to some extent, similar to inner entanglements, proposes a method to learn "bad" causal links in order to generate plans of better quality.[39] In contrast to this work, inner entanglements aim to capture possibly "good" causal links that are enforced in the planning process. In addition, "bad" causal links are learned by exploring the differences between (different) plans solving a single planning task, whereas entanglements are learned by exploring similarities in structures of solution plans of several planning tasks.

3 | PRELIMINARIES

This section is devoted to introducing the terminology that will be used throughout this paper.

3.1 | Classical planning

Classical planning is concerned with finding a (partially or totally ordered) sequence of actions transforming the static, deterministic, and fully observable environment from the given initial state to a desired goal state.[40,41]

In the classical representation, a *planning task* consists of a *planning domain model* and a *planning problem*, where the planning domain model describes the environment and defines planning operators, whereas the planning problem defines concrete objects, an initial state, and a set of goals. The environment is described by *predicates* that are specified via a unique identifier and terms (variable symbols or constants). For example, a predicate at(?t ?p), where at is a unique identifier and ?t and ?p are variable symbols, denotes that a truck ?t is in a location ?p. Predicates thus capture general relations between objects.

Definition 1. A **planning task** is a pair $\Pi = (Dom_\Pi, Prob_\Pi)$, where a **planning domain model** $Dom_\Pi = (P_\Pi, Ops_\Pi)$ is a pair consisting of a finite set of predicates P_Π and planning operators Ops_Π, and a **planning problem** $Prob_\Pi = (Objs_\Pi, I_\Pi, G_\Pi)$ is a triple consisting of a finite set of objects $Objs_\Pi$, initial state I_Π, and goal G_Π.

Let ats_Π be the set of all **atoms** that are formed from the predicates P_Π by substituting the objects $Objs_\Pi$ for the predicates' arguments. In other words, an atom is an **instance** of a predicate (in the rest of this paper, when we use the term instance, we mean an instance that is fully *ground*). A **state** is a subset of ats_Π, and the **initial state** I_Π is a distinguished state. The **goal** G_Π is a nonempty subset of ats_Π, and a **goal state** is any state that contains the goal G_Π.

Notice that the semantics of a state reflects the full observability of the environment. That is, that for a state s, atoms present in s are assumed to be true in s, whereas atoms not present in s are assumed to be false in s.

Planning operators are "modifiers" of the environment. They consist of *preconditions*, ie, what must hold prior operators' application, and *effects*, ie, what is changed after operators' application. Specifically, we distinguish between *negative effects*, ie, what becomes false, and *positive effects*, ie, what becomes true after operators' application. *Actions* are instances of planning operators, ie, operators' arguments as well as the corresponding variable symbols in operators' preconditions and effects are substituted by objects (constants). Planning operators capture general types of activities that can be performed. Planning operators can be instantiated to actions in order to capture given activities between concrete objects.

Definition 2. A **planning operator** $o = (name(o), pre(o), eff^-(o), eff^+(o))$ is specified such that $name(o) = op\_name(x_1, \ldots, x_k)$, where op_name is a unique identifier and $x_1, \ldots, x_k$ are all the variable symbols (arguments) appearing in the operator, $pre(o)$ is a set of predicates representing an operator's precondition, and $eff^-(o)$ and $eff^+(o)$ are sets of predicates representing an operator's negative and positive effects. **Actions** are instances of planning operators that are formed by substituting objects, which are defined in a planning problem, for operators' arguments as well as for the corresponding variable symbols in operators' preconditions and effects. An action $a = (pre(a), eff^-(a), eff^+(a))$ is **applicable** in a state s if and only if $pre(a) \subseteq s$. The application of a in s, if possible, results in a state $(s \backslash eff^-(a)) \cup eff^+(a)$.

A solution of a planning task is a sequence of actions transforming the environment from the given initial state into a goal state.

Definition 3. A **plan** is a sequence of actions. A plan is a **solution** of a planning task Π, a **solution plan** of Π in other words, if and only if a consecutive application of the actions from the plan starting in the initial state of Π results in the goal state of Π.

Determining equality of predicates (needed for set operations such as intersection) is done such that predicates are *equal* if they have the same name and their arguments (including their order) are identical. Hence, an expression $p \in X \cap Y$, where X and Y are sets of predicates, means that p has the same name and arguments (in the same order) in both X and Y. A predicate p is a *variant* of a predicate q[§] if, by renaming p's variable symbols (arguments), we get a predicate equal to q.

3.2 | Relations between actions and operators

By analyzing the preconditions and effects of actions or operators, we can identify how these influence each other. As discussed in Chapman's earlier work,[42] an action having some atom in its positive effects is a possible achiever of that atom for some other action having that atom in its precondition. The opposite for being a possible achiever is being a possible clobberer (below referred to simply as clobberer), which means that action a_i deletes atom(s) that a_j has in its precondition. Note that being a clobberer refers to the notion of "threat" in plan-space planning.[43]

Definition 4. Let a_i and a_j be actions. We say that a_i **possibly achieves** an atom p for a_j if and only if $p \in eff^+(a_i) \cap pre(a_j)$. We say that a_i is a **possible clobberer** for a_j if and only if $eff^-(a_i) \cap pre(a_j) \neq \emptyset$.

Notions of a possible achiever and clobberer can be easily extended for planning operators.

Definition 5. Let o_i and o_j be planning operators and p be a predicate. We say that o_i **possibly achieves** a predicate p for o_j if and only if there exist a_i, a_j, and p_g, instances of o_i, o_j, and p, respectively, such that a_i possibly achieves p_g for a_j, ie, $p_g \in eff^+(a_i) \cap pre(a_j)$. Similarly, we say that o_i is a **possible clobberer** for o_j if and only if there exist a_i and a_j, instances of o_i and o_j, respectively, such that $eff^-(a_i) \cap pre(a_j) \neq \emptyset$.

In every solution plan, every atom in a precondition of an action a_j is (necessarily) achieved in the sense that there exists a possible achiever action a_i for the atom before a_j and that there is no action in between a_i and a_j, which deletes the atom (here, the initial state can be viewed as

[§]We can also say that p is unifiable with q.

the initial action that only adds atoms, and the goal can be viewed as the final action that only has precondition atoms). Notice that being an achiever relates to the notion of "causal link" in plan-space planning.

Definition 6. Let $\langle a_1, a_2, \ldots, a_n \rangle$ be a solution plan of some planning task. We say that an action a_i **achieves** an atom p for an action a_j if and only if $i < j, p \in eff^+(a_i) \cap pre(a_j)$ and $p \notin eff^-(a_k)$ for every $k \in \{i+1, \ldots, j-1\}$.

Of course, an action can achieve an atom that then appears in preconditions of several following actions. Likewise, several actions can achieve an atom for one action. For the purpose of defining inner entanglements, we have to introduce special cases of the achiever relation. If a_i achieves an atom required by a_j and no action in between them also achieves the atom, then a_i is the primary achiever of the atom. In another case, where an action a_i achieves an atom for another action a_j and no other action in between has that atom in its precondition or its positive effects, we say that a_i first achieves the atom required by a_j. If a_i first achieves an atom, it follows that it is also the primary achiever of it.

Definition 7. Let $\langle a_1, a_2, \ldots, a_n \rangle$ be a solution plan of some planning task. We say that an action a_i **is the primary achiever** of an atom p for an action a_j if and only if a_i achieves p for a_j, and $p \notin eff^+(a_k) \cup eff^-(a_k)$ for every $k \in \{i+1, \ldots, j-1\}$.

We also say that an action a_i **first achieves** an atom p required by an action a_j if and only if a_i achieves p for a_j, and $p \notin eff^+(a_k) \cup pre(a_k) \cup eff^-(a_k)$ for every $k \in \{i+1, \ldots, j-1\}$.

3.3 | BlocksWorld domain

We briefly introduce the *BlocksWorld* domain,[44,45] which is one of the best-known planning domains, that will be used as a running example in this paper.

The BlocksWorld domain describes an environment where we have a finite number of blocks, one table with unlimited space, and one robotic hand. A block can be either stacked on another block, placed on the table, or held by the robotic hand. No block can be stacked on more than one block at the same time as well as no more than one block can be stacked on a block at the same time. The robotic hand can hold, at most, one block. The BlocksWorld domain consists of four operators: pickup(?x) refers to a situation when the robotic hand picks up a block ?x from the table, putdown(?x) refers to a situation when the robotic hand puts down the block ?x it is holding on the table, unstack(?x ?y) refers to a situation when the robotic hand unstacks a "clear" block ?x from a block ?y, and stack(?x ?y) refers to a situation when the robotic hand stacks the block ?x it is holding to a "clear" block ?y. As mentioned before, planning operators are instantiated by substituting constants (objects) for variable symbols that appear in operators' definition. For example, putdown(?x) can be instantiated by substituting a, which refers to a concrete block "a," for ?x. We then obtain an action putdown(a) that requires the robotic hand to hold the block a, and the effect is that the block a is placed on the table, the block a is clear (no other block is stacked on it), and the hand no longer holds it.

4 | INNER ENTANGLEMENTS

Inner entanglements are relations between pairs of planning operators and predicates. Inner entanglements, informally speaking, represent the exclusivity of "achieving" or "requiring"

predicates between operators. That is, that for a given planning task, there exists at least one solution plan where a given inner entanglement holds. In other words, considering that inner entanglement while solving the task will not prune all possible solution plans. Typically, a predicate can be achieved by more than one operator, as well as more than one operator might require the same predicate. However, it is often the case that some combinations "achiever-requirer" are not useful.

Specifically, we have two types of inner entanglements, *entanglements by succeeding* and *entanglements by preceding*. An entanglement by succeeding represents the exclusivity of achievement of a predicate p by an operator o_i for an operator o_j. For a planning task, where such an entanglement holds, there exists a solution plan such that instances of o_i first achieve instances of p exclusively only for instances of o_j. An entanglement by preceding, on the other hand, represents the exclusivity of requirement of a predicate p by an operator o_j from an operator o_i. For a planning task, where such an entanglement holds, there exists a solution plan such that only instances of o_i are exclusive primary achievers of instances of p for instances of o_j.

For example, in the BlocksWorld domain, it may be observed that operator pickup(?x) possibly achieves predicate holding(?x) for operators stack(?x ?y) and putdown(?x). Similarly, it may be observed that predicate holding(?x) is possibly achieved for operator putdown(?x) by operators unstack(?x ?y) and pickup(?x). We may require that every instance of pickup(?x) first achieves an instance of holding(?x) exclusively for a corresponding instance of stack(?x ?y) since putdown(?x) would just reverse the effects of pickup(?x) (see Figure 1, right). In other words, pickup(?x) is entangled by succeeding stack(?x ?y) with holding(?x). Analogously, we may require that for every instance of putdown(?x), a corresponding instance of unstack(?x ?y) is the exclusive primary achiever of an instance of holding(?x) because, again, putdown(?x) would just reverse the effects of pickup(?x) (see Figure 1, left). In other words, putdown(?x) is entangled by preceding unstack(?x ?y) with holding(?x).

Roughly speaking, inner entanglements provide restrictions to the plan generation process since they allow only some combinations of action sequences while not affecting the solvability of considered planning tasks. Whereas the BlocksWorld example (see Figure 1) indicates one

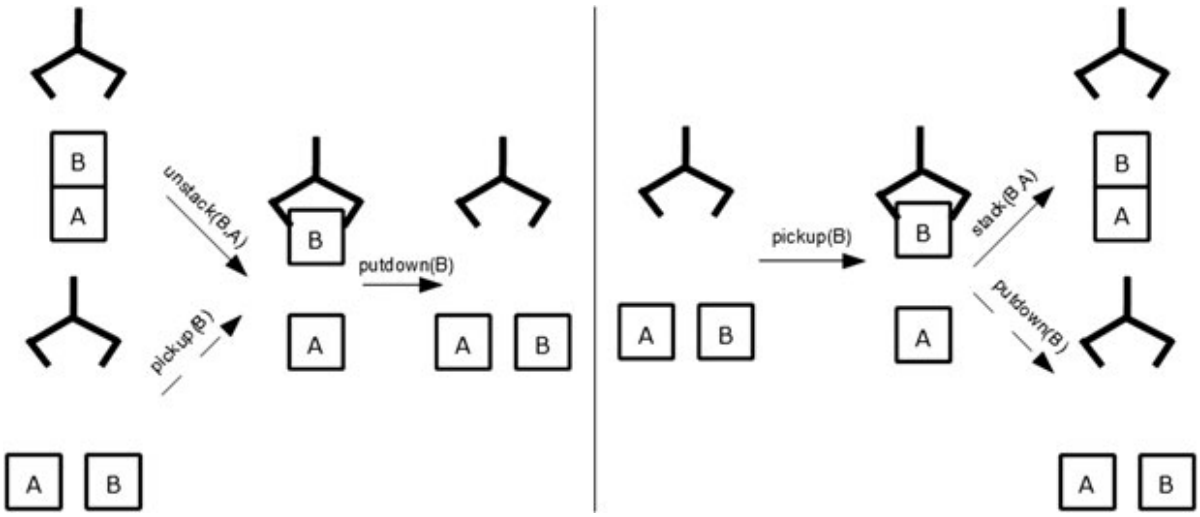


FIGURE 1 Motivating example for inner entanglements, concretely entanglements by preceding (left) and by succeeding (right). Whereas holding(B) can be achieved by either pickup(B) or unstack(B A), for putdown(B) requiring holding(B), only unstack(B A) is useful. Similarly, whereas holding(B) is required by either putdown(B) or stack(B A), only stack(B A) is useful if holding(B) is achieved by pickup(B)

possible nature of inner entanglements, in the general case, the reason why given inner entanglements hold in a given domain model might vary. Hence, our definition of inner entanglements does not explicitly capture their nature and "maintains" only solvability of considered planning tasks.

We distinguish two variants of them, namely, *strict* and *nonstrict*. The strict variant captures the exclusivity of predicate achievement strictly between involved operators, whereas the nonstrict variant allows situations where some instances of the predicates are present in the initial state or can be present in the goal state. For example, if the initial state of some planning task contains an atom holding(a), then the strict version of the above entanglement by preceding prevents applying putdown(a) in the initial state, whereas the nonstrict variant of the entanglement allows to apply putdown(a) in the initial state. Both strict and nonstrict variants of inner entanglements are defined as follows. Notice that we assume that operators o_1 and o_2 share arguments that are relevant to p. For example, pickup(?x) and stack(?x ?y) share the argument ?x, since it is relevant for holding(?x).

Definition 8. Let Π be a planning task. Let o_1 and o_2 be planning operators and p be a predicate (o_1, o_2, and p are defined in the domain model of Π) such that $p \in \mathit{eff}^+(o_1) \cap \mathit{pre}(o_2)$. We say that o_1 is **strictly entangled by succeeding** o_2 with p in Π if and only if there exists a solution plan π of Π, and for each $a_1 \in \pi$ being an instance of o_1, there exists $a_2 \in \pi$ being an instance of o_2 such that a_1 first achieves an atom p_{gnd}, where p_{gnd} is an instance of p, required by a_2.

We also say that o_2 is **strictly entangled by preceding** o_1 with p in Π if and only if there exists a solution plan π of Π, and for each $a_2 \in \pi$ being an instance of o_2, there exists $a_1 \in \pi$ being an instance of o_1 such that a_1 is the primary achiever of an atom p_{gnd}, where p_{gnd} is an instance of p, for a_2.

Henceforth, strict entanglements by preceding and succeeding are denoted as **strict inner entanglements**.

Definition 9. Let Π be a planning task. Let o_1 and o_2 be planning operators and p be a predicate (o_1, o_2, and p are defined in the planning domain model of Π) such that $p \in \mathit{eff}^+(o_1) \cap \mathit{pre}(o_2)$. We say that o_1 is **nonstrictly entangled by succeeding** o_2 with p in Π if and only if there exists a solution plan π of Π, and for every $a_1, a_2 \in \pi$ such that a_1 first achieves an atom p_{gnd}, where p_{gnd} is an instance of p, required by a_2; it holds that if a_1 is an instance of o_1, then a_2 is an instance of o_2.

We also say that o_2 is **nonstrictly entangled by preceding** o_1 with p in Π if and only if there exists a solution plan π of Π, and for every $a_1, a_2 \in \pi$ such that a_1 is the primary achiever of an atom p_{gnd}, where p_{gnd} is an instance of p, for a_2; it holds that if a_2 is an instance of o_2, then a_1 is an instance of o_1.

Henceforth, nonstrict entanglements by preceding and succeeding are denoted as **nonstrict inner entanglements**.

Inner entanglements (both strict and nonstrict) can be used for pruning some unpromising alternatives in the search space, in other words, reducing the branching factor. Notice that a predicate involved in some inner entanglement relation might be true for some time after it is achieved; in other words, the predicate does not have to be "used" immediately after being achieved. Since the previous example of BlocksWorld might be confusing in this sense (the predicate holding(?x) is immediately "used" after being achieved), we provide another example in a modification of the BlocksWorld domain that considers more than one robotic hand. Let

pickup(?h ?x) be strictly entangled by succeeding stack(?h ?x ?y) with holding(?h ?x) in some planning task. If action pickup(h1 a) is applied at step i, then action stack(h1 a ?y) (any other block than a can be substituted for ?y) must be applied at step j such that $j > i$. The entanglement prohibits applying action putdown(h1 a) at step k such that $i < k < j$. On the other hand, other actions that utilize different robotic hands than h1 can be applied in between the ith and the jth step.

A single inner entanglement requires only the existence of one solution plan of the given planning task where the entanglement conditions are met. However, different entanglements might hold in different solution plans. To consider multiple (different) inner entanglements rather than a single one, there must exist a solution plan in which all considered entanglements hold. Moreover, in practice, inner entanglements are domain specific or class of problems specific rather than problem specific. The above definition can be extended to reflect these aspects.

Definition 10. Let Π be a planning task. Let ENT_Π be a set of inner entanglements, where each element of ENT_Π is specified by a type of the inner-entanglement relation and involved a pair of planning operators and predicate. We say that a set of inner entanglements ENT_Π holds for Π if and only if there exists a solution plan of Π in which all the entanglements from ENT_Π hold.

Similarly, $ENT_\mathcal{P}$ holds for a set of planning tasks $\mathcal{P}$ sharing the same planning domain model if and only if $ENT_\mathcal{P} = \bigcap_{\Pi \in \mathcal{P}} ENT_\Pi$.

Both the BlocksWorld related entanglements hold for every BlocksWorld planning task. By adding two more inner entanglements, namely, unstack(?x ?y) to be (strictly) entangled by succeeding putdown(?X) and stack(?x ?y) to be (strictly) entangled by preceding pickup(?X), we restrict to solution plans where blocks are always put down on the table after being unstacked from other blocks and, eventually, picked up from the table and stacked on some other blocks. This might be useful since it introduces more restrictions on decisions the planner has to take during the search. With unlimited table space, these inner entanglements hold for every task.

5 | REFORMULATING PLANNING TASKS

To exploit inner entanglements during the planning process, we have to develop a specific planner, modify an existing one, or reformulate a planning task in such a way that the entanglements hold in every solution plan retrieved by a planner. The last option is planner independent: in fact, it involves the reformulation of domain and problem models using features of the PDDL (actually, STRIPS) language (see Section 3).

Hence, after inner entanglements are identified, we encode them directly into the planning task. The reformulated planning task is passed to a generic planning engine in order to generate a solution plan, which is also a solution plan of the original planning task. Encoding of inner entanglements as we show in this section prevents planning engines from exploring branches of the search space that violate these entanglements. In other words, reformulated tasks "narrow" the search space for planning engines for improving their performance.

Encoding inner entanglements is done by introducing supplementary predicates, "locks," that ensure that we cannot apply certain instances of operators in some stage of the planning process in order to enforce inner entanglements. Let Π be a planning task and Ops be the set of operators defined in the domain model of Π. Let an operator $o_1 \in Ops$ be (strictly or nonstrictly) entangled by a succeeding operator $o_2 \in Ops$ with a predicate p (defined in the domain model of Π) in Π.

```
(:action pick-up
:parameters (?x - block)
:precondition (and (clear ?x)(ontable ?x)(handempty))
:effect (and (not (ontable ?x))(not (clear ?x))(not (handempty))
             (holding ?x)(not (pick-up_stack_succ_holding ?x)))
)
(:action put-down
:parameters (?x - block)
:precondition (and (holding ?x)(pick-up_stack_succ_holding ?x))
:effect (and (not (holding ?x))(clear ?x)(handempty)(ontable ?x))
)
(:action stack
:parameters (?x - block ?y - block)
:precondition (and (holding ?x)(clear ?y))
:effect (and (not (holding ?x))(not (clear ?y))(clear ?x)
             (handempty)(on ?x ?y)(pick-up_stack_succ_holding ?x))
)
(:action unstack
:parameters (?x - block ?y - block)
:precondition (and (on ?x ?y)(clear ?x)(handempty))
:effect (and (holding ?x)(clear ?y)
             (not (clear ?x))(not (handempty))
             (not (on ?x ?y))(pick-up_stack_succ_holding ?x))
)
```

FIGURE 2 An example of the BlocksWorld operators reformulated by an entanglement by succeeding (between pickup, stack, and holding)

Then, Π is reformulated as follows.

(1) Create a predicate p' (not defined in the domain model of Π) having the same arguments as p and add p' to the domain model of Π.
(2) Modify the operator o_1 by adding p' into its negative effects. p' has the same arguments as p, which is in the positive effects of o_1.
(3) Modify the operator o_2 by adding p' into its positive effects. p' has the same arguments as p, which is in the precondition of o_2.
(4) Modify all operators $o \in Ops$ such that $o \neq o_2$ and having a variant of p in $pre(o)$ by adding p' into its precondition. p' has the same arguments as the variant of p.
(5) Modify all operators $o \in Ops$ such that $o \neq o_1$ and having a variant of p in $eff^+(o)$ by adding p' into its positive effects. p' has the same arguments as the variant of p.
(6) Add all possible instances of p' into the initial state of Π, and if the entanglement is strict, then also to the goal of Π.

Figure 2 depicts the BlocksWorld operators encoding an entanglement by succeeding, concretely that pick-up(?x) is (strictly) entangled by succeeding stack(?x ?y) with holding(?x). In our terminology, pick-up(?x) refers to o_1, stack(?x ?y) refers to o_2, holding(?x) refers to p, and pick-up_stack_succ_holding(?x) refers to p'. The correctness of the reformulation is proved as follows.

Proposition 1. *Let Π be a planning task and Ops be the set of planning operators defined in the domain model of Π. Let $o_1, o_2 \in$ Ops be planning operators and p be a predicate (p is defined in the domain model of Π) such that o_1 is strictly (respectively, nonstrictly) entangled by succeeding o_2 with p in Π. Let Π' be a planning task obtained by reformulating Π using the previous approach. π' is a solution plan of Π' if and only if π' is a solution plan of Π that satisfies the entanglement conditions (see Definitions 8 and 9).*

Proof. Hereinafter, the modified operators o_1 and o_2 will be denoted as o'_1 and o'_2. The strict entanglement by succeeding (see Definition 8) says that if an instance of o_1 that achieves an atom p_{gnd} that is an instance of p is applied at step i and a corresponding instance of o_2 that requires p_{gnd} is applied at step j, or never in the case of the nonstrict entanglement, so $j = \infty$, then no corresponding instance of any operator other than o_2 having p_{gnd} in its precondition can be applied at step k unless p_{gnd} is re-achieved by any operator different than o_1 at step l. Formally speaking, $j > i$ and, if $i < k < j$, then $i < l < k < j$.

Applying an instance of o'_1 results in removing an atom p'_{gnd} that is an instance of p' having the same arguments as p_{gnd} (notice that all the possible instances of p' are present in the initial state of Π'). From step 4 of the reformulation, p' is put into the precondition of any operator that has p in its precondition (both p and p' have the same arguments) except o_2. Hence, only instances of o'_2 having p_{gnd} in its precondition can be applied, since actions having p_{gnd} in their precondition that are not instances of o_2 have p'_{gnd} in their preconditions as well. If o'_2 is applied or p is re-achieved by any other (modified) operator than o'_1 (see step 5 of the reformulation), then a corresponding instance of p' is re-achieved as well.

For the strict version of the entanglement, all the instances of p' must be present in the goal state; hence, o'_2 must be applied at some point after o'_1. For the nonstrict version of the entanglement, there is no need to re-achieve all the instances of p'; hence, o'_2 does not have to be applied at some point after o'_1 in order to "use" the corresponding instance of p achieved by o'_1; however, no other operator can "use" it.

Straightforwardly, if π' is a solution plan of Π', then π' is a solution plan of Π that satisfies the entanglement conditions. The provided reformulation prevents only the application of operators in $Ops\backslash\{o_2\}$ having p in their precondition after o_1 achieved p. Therefore, if π' is a solution plan of Π that satisfies the entanglement conditions, then π' is a solution plan of Π'. □

Similarly, we use supplementary predicates, "locks," to enforce entanglements by preceding. Let Π be a planning task and *Ops* be the set of operators defined in the domain model of Π. Let an operator $o_2 \in Ops$ be (strictly or nonstrictly) entangled by a preceding operator $o_1 \in Ops$ with a predicate p (defined in the domain model of Π) in Π. Then, Π is reformulated as follows.

(1) Create a predicate p' (not defined in the domain model of Π) having the same arguments as p and add p' to the domain model of Π.
(2) Modify the operator o_1 by adding p' into its positive effects. p' has the same arguments as p, which is in the positive effects of o_1.
(3) Modify the operator o_2 by adding p' into its precondition. p' has the same arguments as p, which is in the precondition of o_2.
(4) Modify operators $o \in Ops$ such that $o \neq o_1$ and having a variant of p in $eff^+(o)$ by adding p' into its negative effects (p' has the same arguments as the variant of p).
(5) If the entanglement is nonstrict, then add all possible instances of p' to the initial state of Π.

Figure 3 depicts the BlocksWorld operators encoding an entanglement by preceding, concretely that put-down(?x) is (strictly) entangled by preceding unstack(?x ?y) with holding(?x). In our terminology, put-down(?x) refers to o_2, unstack(?x ?y) refers to o_1, holding(?x) refers to p, and put-down_unstack_prec_holding(?x) refers to p'. The correctness of the reformulation is proved as follows.

Proposition 2. *Let Π be a planning task and Ops be the set of planning operators defined in the domain model of Π. Let $o_1, o_2 \in Ops$ be planning operators and p be a predicate (p is defined in the domain model of Π) such that o_2 is strictly (respectively, nonstrictly) entangled by preceding o_1*

```
(:action pick-up
:parameters ( ?x - block)
:precondition (and (clear ?x)(ontable ?x)(handempty))
:effect (and (not (ontable ?x))(not (clear ?x))(not (handempty))
             (holding ?x)(not (put-down_unstack_prec_holding ?x)))
)
(:action put-down
:parameters ( ?x - block)
:precondition (and (holding ?x)(put-down_unstack_prec_holding ?x))
:effect (and (not (holding ?x))(clear ?x)(handempty)(ontable ?x))
)
(:action stack
:parameters ( ?x - block ?y - block)
:precondition (and (holding ?x)(clear ?y))
:effect (and (not (holding ?x))(not (clear ?y))(clear ?x)
             (handempty)(on ?x ?y))
)
(:action unstack
:parameters ( ?x - block ?y - block)
:precondition (and (on ?x ?y)(clear ?x)(handempty))
:effect (and (holding ?x)(clear ?y)(not (clear ?x))(not (handempty))
             (not (on ?x ?y))(put-down_unstack_prec_holding ?x))
)
```

FIGURE 3 An example of the BlocksWorld operators reformulated by an entanglement by preceding (between unstack, put-down, and holding)

with p in Π. Let Π' be a planning task obtained by reformulating Π using the previous approach. π' is a solution plan of Π' if and only if π' is a solution plan of Π that satisfies the entanglement conditions (see Definitions 8 and 9).

Proof. Hereinafter, the modified operators o_1 and o_2 will be denoted as o_1' and o_2'. The strict version of entanglement by preceding (see Definition 8) says that if an instance of o_2 requiring an atom p_{gnd} that is an instance of p is applied at step j and a corresponding instance of o_1 is applied at step i achieving p_{gnd} ($i < j$), then no corresponding instance of any operator other than o_2 having p_{gnd} in its positive effects can be applied at step k such that $i < k < j$.

Adding p' into o_2's precondition results in the situation that any instance of o_2' can be applied only after the corresponding instance of o_1' since p' is in o_1''s positive effects. In particular, an instance of o_1' that achieves an atom p_{gnd} (an instance of p) achieves also p'_{gnd} that is an instance of p' having the same arguments as p_{gnd}. The instance of o_2' that requires p_{gnd} requires p'_{gnd} as well. If p_{gnd} is re-achieved by an instance of other (modified) operators than o_1', then p'_{gnd} is removed (step 4 of the reformulation). Then, o_2' requiring p_{gnd} cannot be applied, since p'_{gnd} will not be true.

For the strict version of the entanglement, no instance of p' is present in the initial state; hence, o_1' must be applied at some point before o_2'. For the nonstrict version of the entanglement, all the instances of p' are present in the initial state; hence, o_2' does not have to be applied after o_1'; however, no other (modified) operator can re-achieve an instance p in between, since, otherwise, the corresponding instance of p' is removed.

Straightforwardly, if π' is a solution plan of Π', then π' is a solution plan of Π that satisfies the entanglement conditions. The provided reformulation prevents only the application of o_2 having p in their precondition unless o_1 achieved p. Therefore, if π' is a solution plan of Π that satisfies the entanglement conditions, then π' is a solution plan of Π'. □

```
(:action pick-up
:parameters ( ?x - block)
:precondition (and (clear ?x)(ontable ?x)(handempty))
:effect (and (not (ontable ?x))(not (clear ?x))(not (handempty))
             (stack_pick-up_both_holding ?x)(not (holding ?x)))
)
(:action put-down
:parameters ( ?x - block)
:precondition (and (holding ?x))
:effect (and (not (holding ?x))(clear ?x)(handempty)(ontable ?x))
)
(:action stack
:parameters ( ?x - block ?y - block)
:precondition (and (stack_pick-up_both_holding ?x)(clear ?y))
:effect (and (not (stack_pick-up_both_holding ?x))(not (clear ?y))
             (clear ?x)(handempty)(on ?x ?y))
)
(:action unstack
:parameters ( ?x - block ?y - block)
:precondition (and (on ?x ?y)(clear ?x)(handempty))
:effect (and (holding ?x)(clear ?y)(not (clear ?x))(not (handempty))
             (not (on ?x ?y))(not (stack_pick-up_both_holding ?x)))
)
```

FIGURE 4 An example of the BlocksWorld operators reformulated by both entanglements by preceding and succeeding between pick-up, stack, and holding

There are also situations where both the (strict) entanglements by preceding and succeeding hold for operators o_1 and o_2 and a predicate p. Of course, we can reformulate the problem according to previous reformulation approaches. On the other hand, it requires two supplementary predicates, and thus, the process might not be very efficient. Given the fact that the exclusivity of achievement and requirement of p is mutual between o_1 and o_2, we can replace p by its "twin" in the positive effects of o_1 and the precondition of o_2. Therefore, we introduce a more compact reformulation that exploits such a property.

Formally, let Π be a planning task and *Ops* be the set of operators defined in the domain model of Π. Let $o_1 \in Ops$ be nonstrictly entangled by succeeding $o_2 \in Ops$ with p (p is defined in the domain model of Π) in Π and o_2 be strictly entangled by preceding o_1 with p in Π.[¶] Then, Π is reformulated as follows.

(1) Create a predicate p' (not defined in the domain model of Π) having the same arguments as p and add p' to the domain model of Π.
(2) Modify the operator o_1 by replacing p by p' in o_1's positive effects and adding p into o_1's negative effects.
(3) Modify the operator o_2 by replacing p by p' in o_2's precondition and (possibly) negative effects.
(4) Modify all operators $o \in Ops$ such that $o \neq o_1$ and having a variant of p in $eff^+(o)$ by adding p' into its negative effects (p' has the same arguments as the variant of p).

Figure 4 depicts the BlocksWorld operators encoding both entanglements by preceding and succeeding, concretely that pick-up(?x) is nonstrictly entangled by succeeding stack(?x ?y) with

[¶]The nonstrict entanglement by succeeding, a weaker form of the strict entanglement by succeeding, is required for correct capture of the entanglements by encoding. Enforcing the strict entanglement by succeeding would require more complex encoding–hence mitigating benefits of the introduced compact encoding.

holding(?x) and stack(?x ?y) is strictly entangled by preceding pick-up(?x) with holding(?x). In our terminology, pick-up(?x) refers to o_1, stack(?x ?y) refers to o_2, holding(?x) refers to p, and stack_pick-up_both_holding(?x) refers to p'. The correctness of the reformulation is proved as follows.

Proposition 3. *Let* Π *be a planning task and Ops be the set of planning operators defined in the domain model of* Π. *Let* $o_1, o_2 \in$ *Ops be planning operators and* p *be a predicate (*p *is defined in the domain model of* Π*) such that* o_2 *is strictly entangled by preceding* o_1 *with* p *in* Π *and* o_1 *is nonstrictly entangled by succeeding* o_2 *with* p *in* Π *such that both entanglements are compatible. Let* Π' *be a planning task obtained by reformulating* Π *using the previous approach.* π' *is a solution plan of* Π' *if and only if* π' *is a solution plan of* Π *that satisfies the conditions of both entanglements (see Definitions 8 and 9).*

Proof. Hereinafter, the modified operators o_1 and o_2 will be denoted as o_1' and o_2'. If both the entanglements are strict (see Definition 8), then it says that if an instance of o_1 achieving an atom p_{gnd} (an instance of p) is applied at step i and a corresponding instance of o_2 requiring p_{gnd} is applied at step j ($j > i$), then no corresponding instance of any operator other than o_1 or o_2 having p_{gnd} in its precondition or positive effects can be applied at step k such that $i < k < j$.

We can observe that o_1' is the only operator achieving p' but no longer achieving p. Similarly, o_2' is the only operator requiring p' (having it in the precondition) but no longer requiring p. The entanglement by preceding cannot be affected by applying any (modified) operator o achieving p, since it removes p' as well (see step 4 of the reformulation), thus making o_2' inapplicable. Similarly, if p is true before applying o_1', it is removed after o_1' is applied, and hence, any operator requiring p becomes inapplicable. The strict entanglement by preceding is met since no instance of p' is in the initial state of P'. There is no restriction that prevents occurrences of p' in any of the goal states. Therefore, the entanglement by succeeding is nonstrict.

Hence, if π' is a solution plan of Π', then π' is a solution plan of Π that satisfies the conditions of both entanglements. The provided reformulation prevents only the application of o_2 having p in their precondition unless o_1 achieved p as well as the application of any operator other than o_2 having p in its precondition after o_1 achieved p. Therefore, if π' is a solution plan of Π that satisfies the entanglement conditions, then π' is a solution plan of Π'. □

6 | THEORETICAL FOUNDATIONS OF INNER ENTANGLEMENTS

This section is devoted to the theoretical properties of inner entanglements such as complexity results as well as their expected impact on planners.

6.1 | Landmark theory

Landmark theory[46] is a useful framework for studying structures of planning tasks. We will use a fragment of the landmark theory to prove intractability (PSPACE-completeness) of deciding whether a given inner entanglement holds. The notions we will use are briefly introduced in the following lines (for more details, see the work of Hoffmann et al[46]).

Landmarks are atoms that must be achieved at some point in every solution plan of a given planning task. Deciding whether atoms are landmarks is PSPACE-complete.[46]

Ordering landmarks is useful for computing heuristics.[47] Landmarks p and q are *greedily necessarily ordered* (we denote it as $p \rightarrow_g q$) if, for every solution plan of a given planning task, p is achieved before q is achieved for the first time. Deciding greedy necessary ordering of landmarks is also PSPACE-complete.[46]

6.2 | Intractability of entanglements

The intractability (PSPACE-completeness) of deciding whether a given inner entanglement holds in a given task is proved by the following theorem.

Theorem 1. *Let Π' be a planning task, $o_{p'}$ and $o_{q'}$ be planning operators, and p'' be a predicate defined in the domain model of Π'. The problem of deciding whether $o_{p'}$ is strictly entangled by succeeding $o_{q'}$ with p'' in Π' is PSPACE-complete. The problem of deciding whether $o_{q'}$ is strictly entangled by preceding $o_{p'}$ with p'' in Π' is also PSPACE-complete.*

Proof. First, we show that the problem of deciding whether $o_{p'}$ is strictly entangled by succeeding $o_{q'}$ with p in Π' as well as the problem of deciding whether $o_{q'}$ is strictly entangled by preceding $o_{q'}$ with p in Π' belongs to the PSPACE class. To do this, we reformulate Π' by encoding the given inner entanglement, as described in Section 5. Hence, the decision problem of whether the given inner entanglement holds can be encoded as a planning task, ie, the entanglement holds if and only if the reformulated task is solvable. We know we can solve planning tasks in polynomial space; hence, this decision problem belongs to PSPACE.

We reduce, in polynomial time, the problem of deciding whether landmarks p and q are greedily necessarily ordered, ie, $p \rightarrow_g q$, in some planning task Π, which is PSPACE-complete, to the problem of deciding strict entanglements by succeeding or preceding between $o_{p'}$, $o_{q'}$, and p in Π'. Without loss of generality, we assume that p and q are nullary predicates (atoms) defined in the domain model of Π.

We create a planning task Π' by modifying Π as follows. Let Ops be the set of planning operators defined in the domain model of Π. Let $Ops_p = \{o \mid o \in Ops, p \in eff^+(o)\}$ be the set of operators achieving p and $Ops_q = \{o \mid o \in Ops, q \in eff^+(o)\}$ be the set of operators achieving q. We extend the domain model of Π by adding atoms (nullary predicates) r, p', p'', q', and q'' (without loss of generality, we assume that none of these are defined in the domain model of Π). Then, we add r into preconditions of every operator from Ops. Then, we modify operators in Ops_p and Ops_q as follows. For every $o \in Ops_p$: replace p by p' in $eff^+(o)$ and add r into $eff^-(o)$. For every $o \in Ops_q$: add q' into $eff^+(o)$ and add q'' into $eff^-(o)$. The initial state I of Π is modified as follows. If $p \in I$, then replace p by p'. If $p \notin I$, then add r. If $q \in I$, then add q'; otherwise, add q'' (if $q \notin I$). Notice that q' becomes and remains true when q has been achieved and that q'' is true only before q is achieved (if q is true in the initial state, q'' is never true).

For the strict entanglements by succeeding case, we introduce the following operators (without loss of generality, we assume that none of the operators are defined in the domain model of Π), ie, $o_{p'} = (name(o_{p'}), \{p'\}, \{p'\}, \{p''\})$, $o_{q'} = (name(o_{q'}), \{p'', q'\}, \{p''\}, \{p, r\})$, and $o_{q''} = (name(o_{q''}), \{p'', q''\}, \{p''\}, \{p, r\})$, and add them into the domain model of Π. Notice that $name(o_{p'})$, $name(o_{q'})$, and $name(o_{q''})$ contain only unique operator identifiers (and no variable symbols). We can observe that if $o_{p'}$ is strictly entangled by succeeding $o_{q'}$ with p''

in Π' (the modification of Π), then q must be true before or at the same time p is achieved. This is because q'' becomes true after q is achieved (as mentioned before), and according to the entanglement, there is a solution plan π' of Π' such that $o_{p'}$ always achieves p'' for $o_{q'}$. Removing instances of $o_{p'}$ and $o_{q'}$ from π' gives us a plan π, which is a solution plan of Π. Given the modification of all operators from Ops_p, p becomes true in π in the same time as p' becomes true and r becomes false in π'. Then, only $o_{p'}$ and $o_{q'}$ can be applied (in this order) in π', because other operators have r in their preconditions, and r can be re-achieved by $o_{q'}$. From this, we can get that q' must be achieved before $o_{p'}$ is applied in π'. Therefore, q is achieved before or in the same time as p in π, which is a solution plan of Π, and thus, $p \rightarrow_g q$ does not hold in Π. Hence, $o_{p'}$ is strictly entangled by succeeding $o_{q'}$ with p'' in Π' (the modification of Π) if and only if $p \rightarrow_g q$ does not hold in Π.

For the strict entanglement by preceding case, we introduce the following operators (without loss of generality, we assume that none of the operators are defined in the domain model of Π), ie, $o_{p'} = (name(o_{p'}), \{p', q'\}, \{p'\}, \{p''\})$, $o_{p''} = (name(o_{p''}), \{p', q''\}, \{p''\}, \{p''\})$, and $o_{q'} = (name(o_{q'}), \{p''\}, \{p''\}, \{p, r\})$, and add them into the domain model of Π. Notice that $name(o_{p'})$, $name(o_{p''})$, and $name(o_{q'})$ contain only unique operator identifiers (and no variable symbols). Analogously to the previous case, we can observe that if $o_{q'}$ is strictly entangled by preceding $o_{p'}$ with p'' in Π' (the modification of Π), then q must be true before or at the same time p is achieved. Therefore, there exists π', a solution plan of Π' where the entanglement holds. Again, removing instances of $o_{p'}$ and $o_{q'}$ from π' gives us a plan π, which is a solution plan of Π. Analogously to the previous case, after a modified operator from Ops_p is applied in π', then only $o_{p'}$ and $o_{q'}$ (in this order) can be applied before any other operator can be applied. Therefore, q' must be achieved before $o_{p'}$ is applied in π', and thus, q is achieved before or in the same time as p in π; hence, $p \rightarrow_g q$ does not hold in Π. Hence, $o_{q'}$ is strictly entangled by preceding $o_{p'}$ with p'' in Π' (the modification of Π) if and only if $p \rightarrow_g q$ does not hold in Π.

Clearly, the modification of Π in both cases is done in polynomial time. Hence, since the problem of deciding whether landmarks p and q are greedily necessarily ordered in Π is PSPACE-complete, the problem deciding whether $o_{p'}$ is strictly entangled by succeeding $o_{q'}$ with p in Π' as well as the problem deciding whether $o_{q'}$ is strictly entangled by preceding $o_{p'}$ with p in Π', where both problems belong to PSPACE, is PSPACE-complete as well. □

Corollary 1. *Let Π' be a planning task, $o_{p'}$ and $o_{q'}$ be planning operators, and p'' be a predicate defined in the domain model of Π'. The problem of deciding whether $o_{p'}$ is nonstrictly entangled by succeeding $o_{q'}$ with p'' in Π' is PSPACE-complete. The problem of deciding whether $o_{q'}$ is nonstrictly entangled by preceding $o_{p'}$ with p'' in Π' is also PSPACE-complete.*

Proof. The problem of deciding on either of the nonstrict inner entanglements can be encoded as a planning task (Π' is reformulated as described in Section 5); hence, it belongs to PSPACE. Since the strict version of inner entanglements is a special case of the nonstrict version, the problem is PSPACE-complete. □

Intractability of deciding whether a single entanglement holds for a given planning task implies intractability of deciding whether a set of inner entanglements holds for that task.

Corollary 2. *Let e_1 and e_2 be inner entanglements that hold in a planning task Π. The problem of deciding whether a set $\{e_1, e_2\}$ holds in Π is PSPACE-complete.*

Proof. Without loss of generality, let Π_{e_1} be a planning task obtained by reformulating Π considering e_1. Then, the problem of deciding whether $\{e_1, e_2\}$ holds in Π is equivalent to the problem of deciding whether e_2 holds in Π_{e_1} which is PSPACE-complete. □

The presented theoretical results say that deciding whether a set of inner entanglements holds in a planning task is (theoretically) as hard as solving the task. Hence, in order to benefit from inner entanglements, we have to spend (much) less time on their generation than how much time we can save by their use. Learning them from simple planning tasks is a viable option, since such tasks can usually be solved and analyzed very quickly.

6.3 | Trivial entanglements

Despite the complexity results, there are some cases where we can trivially identify inner entanglements (hereinafter referred to as trivial inner entanglements). The following situations refer to special cases where there is no way to violate inner entanglements in the planning process. However, trivial inner entanglements do not provide any new domain-specific information, and hence, we do not have to consider them in the reformulation.

We can observe that having only one achiever or "requirer" of some predicate trivially satisfies the conditions of exclusivity. In other words, if only one operator achieves a certain predicate, then it is its exclusive achiever for all the operators that require this predicate. Similarly, if only one operator requires a certain predicate, then it is its exclusive "requirer" from all the operators that achieve this predicate.

Lemma 1. *Let* Π *be a planning task, Ops be the set of planning operators, and p be a predicate defined in the domain model of* Π. *If there exists exactly one* $o_i \in Ops$ *such that* $p \in eff^+(o_i)$, *then, for every* $o_k \in Ops$ *such that* $p \in pre(o_k)$, *it holds that* o_k *is nonstrictly entangled by preceding* o_i *with p in* Π.

Lemma 2. *Let* Π *be a planning task, Ops be the set of planning operators, and p be a predicate defined in the domain model of* Π. *If there exists exactly one* $o_i \in Ops$ *such that* $p \in pre(o_i)$, *then, for every* $o_k \in Ops$ *such that* $p \in eff^+(o_k)$, *it holds that* o_k *is nonstrictly entangled by succeeding* o_i *with p in* Π.

6.4 | Identifying inner entanglements: case studies

This section is devoted to investigating, identifying, and (re)using inner entanglements from a knowledge engineering perspective. Whereas it is usually feasible to consider inner entanglements as domain specific rather than task specific, even small modifications in domain models can invalidate some of the entanglements and, possibly, introduce some other entanglements.

An illustrative example we used earlier in the text identified two inner entanglements in the BlocksWorld domain, ie, the operator putdown is entangled by the preceding operator unstack with the predicate holding, and the operator pickup is entangled by the succeeding operator stack with the predicate holding. Whether the entanglements are strict or nonstrict depends on whether a block is initially held by the robotic hand or whether the same is required in the goal state. The entanglements, in fact, prevent applying the operators pickup and putdown consecutively since they just reverse each other's effects; thus, doing so is clearly meaningless. Extending the BlocksWorld domain by introducing an operator paint, which paints the block while it is held

by the robotic hand, might invalidate the entanglements in some cases. pickup can then achieve holding for both stack and paint; thus, the exclusivity required by the entanglement is not met. putdown can meaningfully use the predicate holding achieved by pickup since we can paint the block (apply the paint operator) in between; thus, the entanglement by preceding might not be met.

The Depots domain is a combination of the BlocksWorld domain and the Logistics domain such that crates are arranged in stacks and operated by hoists in the same way as blocks in BlocksWorld but can be also transported by trucks between different locations. The lift and drop operators correspond with the BlocksWorld's unstack and stack operators, respectively. The load and unload operators are variants of the BlocksWorld's putdown and pickup operators such that instead of putting on and picking up crates from the table, they load crates on or unload crates from trucks, respectively. In Depots, we may observe, for instance, that the operator lift is entangled by the succeeding operator load with the predicate lifting, load is entangled by preceding lift with lifting, the operator drop is entangled by the preceding operator unload with lifting, and unload is entangled by succeeding drop with lifting. If no instance of lifting is present in the initial state or the goal, then the entanglements are strict. If there is no truck defined in the problem, then we cannot apply load or unload; hence, the entanglements do not hold (otherwise, it will not be possible to apply lift and drop consecutively). Modifying the domain model in such a way that particular trucks can move only between some locations might introduce the necessity of reloading crates from one truck to another. This will certainly affect two of the entanglements; in particular, load will no longer be entangled by preceding lift with lifting, and unload will no longer be entangled by succeeding drop with lifting. However, tasks in which some crate(s) have to be reloaded can be easily identified.

It should be noted that the aforementioned examples indicate that the nature of inner entanglements varies per domain model. Therefore, it seems, in our opinion, that refining not very restrictive general rules for identifying inner entanglements might not be a feasible option. Domain model engineers can either identify inner entanglements by hand or exploit our method based on the "learning in planning" paradigm that is presented in Section 7.

6.5 | Expected impact of inner entanglements on the planning process

Inner entanglements eliminate unpromising alternatives in the search space, which reduces the branching factor in search. Introducing supplementary predicates required for encoding inner entanglements, however, introduces additional facts (atoms) planners have to deal with during search, and moreover, memory requirements might therefore be higher. Hence, the impact of inner entanglements is determined by considering whether the potential benefits of reducing the branching factor outweigh overheads caused by handling supplementary predicates. An analogy can be seen in determining whether a macro-operator is useful, which is also referred to as a *utility problem* in the literature.[48]

Taking a closer look on how inner entanglements are encoded provides insights into how they may influence delete-relaxed heuristics, which is a common technique used in planning engines. Having an operator o_2 strictly entangled by a preceding operator o_1 with a predicate p captures a situation where an instance of o_2 can be applied only if a corresponding instance of p is achieved by an instance of o_1. This is enforced by putting a supplementary predicate p' into o_1's positive effects and into o_2's precondition. In delete-relaxed plans, o_1 must be also applied at some point before o_2. However, an operator $o \neq o_2$ achieving p (and thus removing p') can be placed in

between o_1 and o_2 in delete-relaxed plans, which does not correspond with the entanglement conditions. Entanglements by preceding are therefore only partially taken into account while computing delete-relaxed heuristics. Having an operator o_1 strictly entangled by a succeeding operator o_2 with a predicate p captures a situation where an instance of o_1 achieves a corresponding instance of p for an instance of o_2. This is enforced by putting a supplementary predicate p' into o_1's negative effects and into preconditions of operators other than o_2 that have p in their preconditions. However, in delete-relaxed plans, applying o_1 does not prevent applying any other operator having p in its precondition. Therefore, entanglements by succeeding are not taken into account while computing delete-relaxed heuristics. Intuitively, only entanglements by preceding might be beneficial on planners based on delete-relaxed heuristics (eg, FF).

However, recent empirical results do not confirm this intuition by showing that, in some cases, entanglements by succeeding can be very beneficial even for planners based on delete-relaxed heuristics.[21] To understand the potential benefits of entanglements by succeeding, we have to take a different view. A heuristic may suggest applying an operator $o \neq o_2$ requiring p from o_1. However, after actual application of o_1, it will become impossible to apply o (due to the entanglement conditions), since o_2 will be enforced. Although it might cause planners to be "trapped" in a local minimum of the heuristics, it might also prevent planners to get into "deeper" local minima, which might eventually happen if o is applied instead of o_2.

If both (strict) entanglements by preceding and succeeding hold between o_1, o_2, and p, the compact encoding involves replacing p with p' in o_1's positive effects and o_2's precondition. In delete-relaxed plans, o_2 cannot be applied unless o_1 is, which is similar to the entanglements by preceding case, and moreover, o_1 cannot achieve p for any other operator than o_2 (because p is replaced by p'). Although, as in the entanglements by preceding case, an operator achieving p can be placed in between o_1 and o_2 in delete-relaxed plans, which does not correspond to the entanglement conditions, both the entanglements are taken into account to a reasonable extent while computing delete-relaxed heuristics.

Compact encoding (when both entanglements by preceding and succeeding hold between a pair of operators and a predicate) is intuitively beneficial for planners. The potential impact of inner entanglements seems to be correlated with the shape of search space, in other words, whether inner entanglements can prevent planners to end up in undesirable states (eg, dead ends, "deep" local minima). We believe that maximizing sets of compatible inner entanglements does not imply maximizing planners' performance, because some of the entanglements might, in fact, have a negative impact, for instance, by introducing supplementary predicates planners might deal with or introducing local minima in the heuristics landscape. Possible examples of "bad" inner entanglements are those that consist of operators whose instances appear sporadically in plans, because such inner entanglements bring only little information for possibly high overheads. Moreover, if an inner entanglement prunes only a few alternatives, then overheads introduced with it might be higher than its possible benefit. For example, after picking up a block, we might either put it down or stack it on some other clear block. Clearly, the number of clear blocks might be up to $n - 1$, where n is the number of all blocks. If pickup(?x) is (strictly) entangled by succeeding stack(?x ?y) with holding(?x), then we cannot apply putdown(?x) after pickup(?x). Hence, we prune one alternative, keeping $n - 1$ alternatives in the worst case. Similarly, after unstacking a block from another block, we can either put it down or stack it on some clear block. If putdown(?x) is (strictly) entangled by preceding unstack(?x ?y), then we cannot apply stack(?x ?z) after unstack(?x ?y). Hence, we keep only one alternative, pruning $n - 1$ alternatives in the best case. Given this observation, the latter entanglement is much more informative than the former one. Intuitively, the former entanglement is not helpful and very

likely will worsen the planning process. The latter entanglement, on the other hand, seems to be helpful and should improve the planning process.

7 | EXTRACTING INNER ENTANGLEMENTS

Deciding whether a given set of inner entanglements holds in a given task is generally PSPACE-complete (as discussed in Section 6). Moreover, trivial entanglements (see Section 6.3) are not informative and, thus, not considered for task reformulation. Therefore, we have to devise an effective approximation technique for extracting sets of inner entanglements. We assume that tasks having the same domain model have a similar structure; thus, the same set of inner entanglements holds in all of them. We can then select a representative set of simple tasks for each domain model as training tasks; thus, those can be solved easily by standard planning engines. Generated training plans, which are the solutions of these training tasks, are then explored in order to find what inner entanglements hold in them.

The above approach can be formalized as follows. Let $\mathcal{P}$ be a class of planning tasks that has the same domain model. Let $\mathcal{P}_T \subset \mathcal{P}$ be a set of training tasks. In our approximation method, we assume that $ENT_{\mathcal{P}_T} = ENT_{\mathcal{P}}$; in other words, a set of inner entanglements holding on training planning tasks also holds on the whole class of planning tasks. This assumption is, of course, a source of incompleteness, since enforcing incorrect entanglements may cause some tasks becoming unsolvable. On the other hand, planning tasks having the same domain model are of similar structure (eg, they differ only by number of objects), which is the case for most of the IPC benchmarks. Hence, we believe that selecting a small set of these tasks such that selected tasks are easy but not trivial can alleviate the incompleteness issue and, thus, support the assumption. Our empirical study that also explores these issues is provided in Section 8.

The method for extracting inner entanglements from (training) plans works as follows. For every action, we check which actions achieved atoms for it, or vice versa. This information is used to determine the cases where the exclusivity of the predicate's achievement or the requirement between a pair of operators applies. This concept is elaborated in Algorithm 1. For this purpose, we define an array *counter*, which stores information on how many instances of given operators occur in the training plans, 3D arrays *entP*, *entS*, which count how many times a given operator achieves/requires a predicate to/from another operator. Function *is_inst*(*arg*) returns either an operator if *arg* (action) is an instance of it or a predicate if *arg* (atom) is an instance of it. Function *last_achiever*(p, $\langle a_1, \ldots, a_k \rangle$) returns the last action in the sequence ($\langle a_1, \ldots, a_k \rangle$) that has p in its positive effects, or NULL if no such action exists (ie, p is an initial atom).

Algorithm 1 requires linear time with respect to the lengths of given training plans if the number of atoms in actions' preconditions and effects is much lower than lengths of training plans; thus, it can be bounded by a constant. Notice that information retrieved by the *last_achiever* function can be stored in a hash table; hence, constant time is needed.

7.1 | Flaw ratio

From Algorithm 1, it is easy to determine whether a given set of inner entanglements holds in all the training plans. However, it is often not a very efficient way to determine a useful set of inner entanglements. There are two main reasons. First, training plans might contain redundant actions or very suboptimal subplans, which can prevent detecting some useful entanglements. Second, there might be several strategies on how a task can be solved, where only some of these lead into

the discovery of some useful entanglements. For example, in BlocksWorld, we might "put aside" blocks in two different ways: put them on the table or stack them on other blocks. Only the former way leads to the discovery of two useful inner entanglements, ie, unstack is (strictly) entangled by succeeding putdown with holding, and stack is (strictly) entangled by preceding pickup with holding.

Algorithm 1 Checking how many times a given operator achieves (requires) a predicate to (from) another operator in the training plans

```
1: initialize_ent_arrays();                          ▷ create empty arrays entP, entS of size [Ops,Ops,Preds]
2: initialize_op_counter();                          ▷ create an empty array counter of size [Ops]
3: for each training plan π = ⟨a_1, ..., a_n⟩ do
4:     for i := 1 to n do
5:         for each p ∈ pre(a_i) do
6:             a := last_achiever(p, ⟨a_1, ..., a_{i-1}⟩);
7:             if a ≠ NULL then
8:                 entP[is_inst(a_i), is_inst(a), is_inst(p)] + +;
9:                 entS[is_inst(a), is_inst(a_i), is_inst(p)] + +;
10:            end if
11:        end for
12:        counter[is_inst(a_i)] + +;
13:    end for
14: end for
```

Introducing a *flaw ratio* $\eta \in [0;1]$, which is a parameter referring to an allowed percentage of "flaws" in training plans, can identify inner entanglements that can be discovered in plans that are "close" to the training plans. In other words, the exclusivity of predicate achievement or requirement between a pair of operators might only be satisfied to some extent in the training plans, whereas in some other solution plans, the exclusivity can be fully satisfied. For example, in BlocksWorld, the blocks might occasionally be "put aside" to other blocks in the training plans and, thus, cause that the useful inner entanglements (as above) are not detected. By considering flaw ratio, these inner entanglements can be found.

Let η be the flaw ratio, then the following equations determine when a given inner entanglement can be considered (*sprec* and *ssucc* stand for the strict version of entanglements by preceding and succeeding, respectively):

$$(prec, o_1, o_2, p) \Leftrightarrow entP[o_1, o_2, p] > 0 \wedge \forall o \neq o_2 : \frac{entP[o_1, o, p]}{counter[o_1]} \leqslant \eta \tag{1}$$

$$(succ, o_1, o_2, p) \Leftrightarrow entS[o_1, o_2, p] > 0 \wedge \forall o \neq o_2 : \frac{entS[o_1, o, p]}{counter[o_1]} \leqslant \eta \tag{2}$$

$$(sprec, o_1, o_2, p) \Leftrightarrow \frac{entP[o_1, o_2, p]}{counter[o_1]} \geqslant 1 - \eta \wedge \forall o \neq o_2 : \frac{entP[o_1, o, p]}{counter[o_1]} \leqslant \eta \tag{3}$$

$$(ssucc, o_1, o_2, p) \Leftrightarrow \frac{entS[o_1, o_2, p]}{counter[o_1]} \geqslant 1 - \eta \wedge \forall o \neq o_2 : \frac{entS[o_1, o, p]}{counter[o_1]} \leqslant \eta. \tag{4}$$

7.2 | Filtering unpromising inner entanglements

Following the discussion from Section 6.5, we can derive that the pruning power of inner entanglements is crucial for having a positive impact on the planning process. In other words, inner entanglements are more likely to be beneficial if they can prune a relatively large number of search alternatives. Otherwise, inner entanglements might have a detrimental effect on the performance of planning engines because of the overhead caused by their representation.

We identified two main cases in which inner entanglements do not have a strong pruning power. The first case is where inner entanglements with operators that are rarely applied in plans are involved. Training plans can provide a good indication of "rare" operators. Therefore, we can assume that if an operator appears rarely in training plans, then it will be used rarely also for other planning problems in a given domain. Hence, we define a threshold ϵ and filter out such inner entanglements where any of the involved operators (o_1 and o_2) have less instances in the training plans, ie,

$$counter[o_1] < \epsilon \vee counter[o_2] < \epsilon.$$

The second case is comparing the number of arguments that "entangled" and "prohibited" operators have. Recall the example from Section 6.5, where pickup(?x) is (strictly) entangled by succeeding stack(?x ?y) with holding(?x). The entanglements prohibit applying putdown(?x) after pickup(?x). In our words, stack(?x ?y) is the "entangled" operator, and putdown(?x) is the "prohibited" operator. Clearly, only one alternative is pruned (only one instance of putdown(?x) can be applied after pickup(?x)), whereas up to $n-1$ alternatives are allowed (up to $n-1$ instances of stack(?x ?y) can be applied after pickup(?x)); hence, the pruning power of the entanglement is poor. The number of operators' arguments is thus a good indicator for estimating the numbers of pruned search alternatives. Hence, if the number of arguments of the "entangled" operator is higher than that of all the "prohibited" operators, then the entanglement is unpromising. Formally, let $arg(o)$ denote the number of arguments of an operator o. Let an operator o_1 be (strictly) entangled by a succeeding operator o_2 with a predicate p, then the entanglement is considered as unpromising if

$$\forall o \neq o_2, p \in pre(o) : arg(o) < arg(o_2).$$

Analogously, let an operator o_2 be (strictly) entangled by a preceding operator o_1 with a predicate p, then the entanglement is considered as unpromising if

$$\forall o \neq o_1, p \in eff^{+}(o) : arg(o) < arg(o_1).$$

Unpromising inner entanglements are filtered out except cases where both types of inner entanglements hold for the operators o_1 and o_2 and the predicate p, and only one of the entanglements is unpromising. Such an exception follows the observation discussed in Section 6.5 that the compact encoding of such entanglements does not introduce more overheads than the encoding of a single (inner) entanglement.

7.3 | Inner-entanglement extraction

Algorithm 2 wraps up the method for extracting inner entanglements. Given generated training plans, we can fill the arrays *entP*, *entS*, and *counter* by running Algorithm 1. An initial value *init-fr* of the flaw ratio η is assigned. The main loop (Lines 4-12) iteratively validates whether using the given flaw ratio does not lead to the extraction of entanglements that do not hold in the training tasks. The validation is done by extracting the nontrivial inner entanglements

using the current flaw ratio η (Line 5), filtering unpromising inner entanglements (Line 6), generating reformulated training problems considering the extracted entanglements (Line 7), and running a planner on these reformulated problems (Line 8). Introducing the flaw ratio may cause that the set of extracted inner entanglements does not even hold for the training problems. If such a situation occurs, the flaw ratio is decreased by *step* (Line 11), and the process (for Line 4) is repeated. Clearly, if $\eta = 0$, then the set of extracted inner entanglements holds for the training tasks.

Algorithm 2 Extraction of entanglements with the flaw ratio

```
1: generate training plans
2: fill arrays (Algorithm 1)
3: η = init-fr
4: while η > 0 do
5:     extract inner entanglements considering η (see Equations (1)-(4))
6:     filter unpromising inner entanglements (see Section 7.2)
7:     generate reformulated training problems
8:     if reformulated training problems are solvable then
9:         break
10:    end if
11:    η = max(0, η − step)
12: end while
13: generate reformulated (testing) problems
```

8 | EXPERIMENTAL EVALUATION

This section is devoted to the empirical evaluation of the impact of entanglements in the plan generation process. The aims of the experiments are to analyze the impact of inner entanglements on state-of-the-art planning engines and how quality of training plans influences the detection and extraction of inner entanglements. For empirical evaluation purposes, we used all the domains from the learning track of IPC-7; since inner entanglements are automatically extracted domain-specific knowledge, the learning track benchmarks seem to be appropriate. This test set is thus independent, is open, and gives a relatively wide coverage.

In each domain, the planning tasks have the same domain model and, thus, differ only by planning problem specifications. Henceforth, *training problems* denote tasks that are used for learning entanglements, and *testing problems* denote tasks that are used as benchmarks.

8.1 | Benchmark planners

In order to perform our analysis, we selected a number of planners according to (i) their performance in the IPCs and (ii) the variety of techniques they exploit. The selected planners are *Metric-FF*,[49] *LPG-td*,[50] *LAMA*,[47,51] *Probe*,[52,53] *MpC*,[54,55] *Yahsp3*,[56] and *Mercury*.[57]

Metric-FF[49] is an extension of the well-known FF planner[58] that won the 2nd IPC. The FF's search strategy is a variation of hill-climbing over the space of the world states, and in FF, the

goal distance is estimated by solving a relaxed task for each successor world state. Compared to the first version of FF, Metric-FF is enhanced with the goal ordering pruning technique and with the ordering knowledge provided by a goal agenda.

LPG-td won the 3rd IPC. It uses stochastic local search in a space of partial plans represented through *linear action graphs*, which are variants of the very well-known planning graph.[29] The search steps are graph modifications, transforming an action graph into a different one.

LAMA[47,51] won the 6th and 7th IPC (sequential satisficing track). LAMA translates the PDDL problem specification into a multivalued state variable representation ("SAS+") and searches for a plan in the space of the world states using a heuristic derived from the causal graph, which is a particular graph representing the causal dependencies of SAS+ variables. Its core feature is the use of a pseudo-heuristic derived from landmarks.

Probe[52,53] was successful in IPC-7 and IPC-8. It implements a dual-search architecture for planning, which is based on the idea of *probes*: single-action sequences computed without search from a given state that can quickly go deep into the state space, terminating either in the goal or in failure.

MpC[54,55] was a runner-up in the agile track of IPC-8. MpC is a SAT-based planner that exploits an extremely compact SAT representation of planning tasks and an integrated SAT solver.

Yahsp3[56] won the agile track of IPC-8. Yahsp is a heuristic search–based planner that exploits information obtained from the computation of the heuristics, which is similar to the heuristic used in FF. Such information is used to find "lookahead states" that are reachable but "far" from the current state.

Mercury[57] was a runner-up in the satisficing track of IPC-8. Similarly to LAMA, Mercury translates the PDDL representation into a SAS+ multivalued state variable representation. It then exploits the Red-Black heuristics, which uses only partial delete-relaxation.

8.2 | Experimental setup

In machine learning, it is important to have a good-quality training set in order to maximize the outcome of the learning process. From the planning perspective, training plans should well capture the important structural aspects that are generalizable to the whole class of planning tasks. If training plans are too short, their structure might be over-constrained, and thus, we might extract some inner entanglements that do not hold for many typical tasks of a given class. On the other hand, planning is computationally very expensive, and thus, obtaining long training plans might be too time consuming or even impossible. Hence, we have observed that a reasonable size for a training problem is when the length of its solution plan is between 20 and 100 actions, depending on the number of defined operators in the domain models (having more operators yields longer solution plans). Moreover, the number of training problems does not have to be high. This follows the observation made by Chrpa et al[59] that the set of extracted entanglements often does not change, or changes are very small, with increasing number of training problems. Similar observations have been made when configuring portfolios of planners.[60] On the other hand, using very few training problems increases the risk of extracting inner entanglements that do not hold (we might be "lucky" to have a very atypical problem as a training one). Following these observations, five training problems per domain were used. Notice that in the learning track of IPC-7,[61] a set of training problems is not explicitly provided, and thus, the training problems were generated by existing problem generators.

Strict versions of inner entanglements were learned.[#] The benchmark planners were used to generate training plans. The flaw ratio (η) was initially set to 0.2, and, in case where any of the training problems became unsolvable after incorporating entanglements,[‖] the flaw ratio was iteratively reduced by 0.05 until all the training problems became solvable while entanglements were considered, or the flaw ratio dropped to 0.0 (for details, see Algorithm 2). Although, in the previous work,[20] the flaw ratio is set to 0.1, we observed on some preliminary experiments, performed on a small set of benchmarks (not included in the rest of this experimental analysis), that such a value is too conservative. On the other hand, setting the value above 0.2 led to the extraction of inner entanglements that often did not hold in the training problems. The threshold ϵ (see Section 7.2) is set to 20, which means that the operator must be used at least four times, on average, in each training plan.

A CPU-time cutoff of 900 seconds (15 minutes, as in learning tracks of IPC) was used for both learning and testing runs. All the experiments were run on a quad-core 2.8-GHz CPU machine with 4 GB of RAM. In this experimental analysis, IPC scores as defined in IPC-7 are used. For a planner C and a problem p, $Time(C, p)$ is 0 if p is unsolved, and $1/(1 + \log_{10}(T_p(C)/T_p^*))$, where $T_p(C)$ is the CPU time needed by planner C to solve problem p and T_p^* is the CPU time needed by the best considered planner, otherwise. Similarly, $Qual(C, p)$ is 0 if p is unsolved, and $N_p^*/N_p(C)$, where $N_p(C)$ is the cost of the plan, solution of p, obtained by C and N_p^* is the minimal cost of the solution plan of p among all the considered planners, otherwise. The IPC score on a set of problems is given by the sum of the scores achieved on each considered problem.

8.3 | Experimental results: the learning phase

As discussed in the literature,[59] the structure of solution plans might differ according to a planner that generated them, and hence, the set of inner entanglements extracted from such plans can differ as well. In order to improve sets of extracted inner entanglements (ie, maximize the number of useful entanglements and minimize the number of "peculiar" entanglements), we selected, for each training problem, the best-quality (shortest) plan from those produced by all the considered planners. These best-quality training plans were then used in the entanglement extraction method (see Section 7). Hereinafter, a set of inner entanglements extracted by exploiting this approach will be denoted as the *"best-plan set"* of inner entanglements.

Intuitively, using good-quality training plans leads to extracting good-quality DCK (inner entanglements in this case). To test this intuition, we also considered the worst-quality plans from those produced by all the considered planners (hereinafter denoted as *"worst-plan set"* of inner entanglements).

The results of the learning phase are as follows.

- In Gripper, Rovers, Satellite, and Spanner, no inner entanglements have been extracted, ie, both the best-plan and worst-plan sets are empty.
- In Depots, Parking, and TPP, the best-plan and worst-plan sets are the same.
- In BlocksWorld (Bw), the worst-plan set is empty, whereas the best-plan set is not empty.
- In Barman, both the best-plan and worst-plan sets are not empty but different.

[#] Although the compact encoding for situations where both types of inner entanglements are involved requires the nonstrict version of entanglements by succeeding, correctness is not compromised, since the strict versions of inner entanglements are special cases of the nonstrict versions.

[‖] By "unsolvable," we mean those problems where the planner did not find a solution in the time limit of 600 seconds.

In the first case, the structure of the domain models prevents capturing any nontrivial inner entanglements. In the second case, the quality of training plans does not make any difference. This is due to the fact that the "important" part of the training plan structure does not change that much than the quality of these training plans, and by using the flaw ratio, small structural changes of training plans are "absorbed." The third case refers to the situation where good-quality plans usually follow the strategy of putting blocks to the table whereas bad-quality plans usually temporarily stack blocks on other blocks. In the last case, the worst-plan set is a superset of the best-plan set. Although such a result is counterintuitive, we observed that in the Barman domain, drinks can be prepared by using clean shots or by reusing "dirty" shots if the same ingredient is put into them. Using always clean shots provides a "narrower" structure of solution plans; however, these plans are of worse quality since shots have to be always cleaned. It should be noted that best-plan and worst-plan sets were different only in two out of nine domains. Although the work of Chrpa et al[59] indicates that the differences should be larger, incorporating the filtering technique for unpromising inner entanglements (see Section 7.2) into the learning method "absorbs" some of these differences.

8.4 | Experimental results: the testing phase

The results shown in Table 1 demonstrate the positive impact of inner entanglements on the planning process. It can be seen that only Probe solved all original testing problems in Bw and Depots. While inner entanglements (best-plan sets) were considered, in Bw, Depots, and TPP, some planners were able to solve all the testing problems. In Parking and Barman, the results are mixed. In Parking, the overall results are rather negative; in Barman, Probe (best-plan set) and Lama (worst-plan set) benefit from inner entanglements, whereas Mercury, on the other hand, has much worse performance on inner entanglement–enhanced problems. Assuming that we can run all planners with original and inner entanglement–enhanced domain models in parallel, then, by using inner entanglements, we can solve two more problems in Parking and three more problems in Barman. In addition, nine problems in Barman can be solved faster when inner entanglements are considered.

Whereas the results generally support the claim that inner entanglements can effectively prune search space by eliminating unpromising alternatives, some results, however, require more attention. Lama does not perform well for the best-plan set in Bw, whereas it performs considerably well in the worst-plan set in Barman. This might lead to an observation that Lama performs well in the worst-plan sets rather than in the best-plan sets. We, however, believe that this observation is of domain- and planner-specific nature and, thus, might not be generalized. The reason for Lama's good performance in the worst-plan set in Barman is in the fact that enforcing the planner to use only clean shots makes the landmark-based heuristics more informative. On the other hand, the best-plan set in Bw enforces the planner to put blocks on the table before stacking them in goal positions. Lama, however, has already a good performance on the original setting—its heuristics is well informed. Inner entanglements, in this case, might introduce some suboptimalities (as the quality results indicate) and, thus, slow down the planning process of Lama. In Mercury's case, we can observe that it already performs well on the original Barman problems. Inner entanglements, however, seem to introduce overheads and possibly make Mercury's heuristics less informative.

We have also observed that using good-quality training plans is useful for the learning process since the structure of the plans has less noise (eg, redundant actions). Despite some results of

TABLE 1 Comparing planners' performance on (O)riginal and (B)est- and (W)orst-plan sets of inner-entanglement encodings. Δ IPC Score refers to a difference in the International Planning Competition (IPC) score between the reformulated and original encodings (positive values—higher score for the reformulated encoding). "–" means no inner entanglements were produced

	Coverage			Δ IPC Score—Speed		Δ IPC Score—Quality	
Planner	O	B	W	B	W	B	W
Barman							
FF	0	0	0	0.0	0.0	0.0	0.0
LPG	0	0	0	0.0	0.0	0.0	0.0
Lama	1	1	14	0.0	+13.4	0.0	+12.6
Probe	5	12	2	+7.5	−3.0	+7.1	−3.0
MpC	0	0	0	0.0	0.0	0.0	0.0
Yahsp	0	0	0	0.0	0.0	0.0	0.0
Mercury	25	21	2	−8.5	−24.1	−3.9	−23.5
Bw							
FF	0	0	–	0.0	–	0.0	–
LPG	25	30	–	+14.5	–	+7.9	–
Lama	28	28	–	−1.4	–	−2.3	–
Probe	30	30	–	+4.1	–	+2.8	–
MpC	0	14	–	+14.0	–	+14.0	–
Yahsp	29	30	–	+12.9	–	−7.1	–
Mercury	19	29	–	+11.0	–	+7.0	–
Depots							
FF	1	3	3	+2.3	+2.3	+1.8	+1.8
LPG	10	24	24	+16.7	+16.7	+14.1	+14.1
Lama	0	2	2	+2.0	+2.0	+2.0	+2.0
Probe	30	30	30	−3.2	−3.2	+2.0	+2.0
MpC	19	30	30	+17.6	+17.6	+11.3	+11.3
Yahsp	22	30	30	+18.7	+18.7	+20.8	+20.8
Mercury	0	0	0	0.0	0.0	0.0	0.0
Parking							
FF	11	9	9	−3.0	−3.0	−2.2	−2.2
LPG	0	0	0	0.0	0.0	0.0	0.0
Lama	8	7	7	−2.4	−2.4	−0.7	−0.7
Probe	7	6	6	−0.9	−0.9	−0.8	−0.8
MpC	5	5	5	+0.4	+0.4	−0.1	−0.1
Yahsp	0	0	0	0.0	0.0	0.0	0.0
Mercury	8	7	7	−1.7	−1.7	−0.9	−0.9
TPP							
FF	0	3	3	+3.0	+3.0	+3.0	+3.0
LPG	0	29	29	+29.0	+29.0	+29.0	+29.0
Lama	20	30	30	+20.3	+20.3	+10.3	+10.3
Probe	15	30	30	+23.6	+23.6	+14.7	+14.7
MpC	15	21	21	+14.1	+14.1	+6.0	+6.0
Yahsp	20	20	20	+1.7	+1.7	+0.0	+0.0
Mercury	26	30	30	+15.4	+15.4	+2.4	+2.4

TABLE 2 Comparing planners' performance on (O)riginal, (N)o flaw ratio nor filtering, Flaw (R)atio only, (F)iltering only, and (A)ll (flaw ratio and filtering) on the best-plan sets of inner-entanglement encodings. Δ IPC Score refers to a difference in the International Planning Competition (IPC) score between the reformulated and original encodings (positive values—higher score for the reformulated encoding). "–" means no inner entanglements were produced

	Coverage					Δ IPC Score—Speed				Δ IPC Score—Quality			
Planner	O	N	R	F	A	N	R	F	A	N	R	F	A
Barman													
Lama	1	0	0	–	1	−1.0	−1.0	–	0.0	−1.0	−1.0	–	0.0
Probe	5	0	0	–	12	−5.0	−5.0	–	+7.5	−5.0	−5.0	–	+7.1
Mercury	25	0	0	–	21	−25.0	−25.0	–	−8.5	−25.0	−25.0	–	−3.9
Bw													
LPG	25	–	1	–	30	–	−24.4	–	+14.5	–	−24.1	–	+7.9
Lama	28	–	0	–	28	–	−28.0	–	−1.4	–	−28.0	–	−2.3
Probe	30	–	0	–	30	–	−30.0	–	+4.1	–	−30.0	–	+2.8
MpC	0	–	0	–	14	–	0.0	–	+14.0	–	0.0	–	+14.0
Yahsp	29	–	21	–	30	–	−14.0	–	+12.9	–	−27.2	–	−7.1
Mercury	19	–	0	–	29	–	−19.0	–	+11.0	–	−19.0	–	+7.0
Depots													
FF	1	1	3	1	3	+0.4	+2.3	+0.4	+2.3	−0.1	+1.8	−0.1	+1.8
LPG	10	18	24	18	24	+8.0	+16.7	+8.0	+16.7	+6.5	+14.1	+6.5	+14.1
Lama	0	0	2	0	2	0.0	+2.0	0.0	+2.0	0.0	+2.0	0.0	+2.0
Probe	30	27	30	27	30	−9.3	−3.2	−9.3	−3.2	−6.7	+2.0	−6.7	+2.0
MpC	19	30	30	30	30	+16.2	+17.6	+16.2	+17.6	+13.2	+11.3	+13.2	+11.3
Yahsp	22	30	30	30	30	+18.7	+18.7	+18.7	+18.7	+23.0	+20.8	+23.0	+20.8

Lama that contradicts the observation, we believe that the "best plan" strategy will be useful also in other learning-based techniques (eg, generating macros).

8.5 | Impact of flaw ratio and filtering

Table 2 provides a comparison of the impact of the heuristics (flaw ratio, filtering) on the "quality" of the learned set of inner entanglements. Only the best-plan sets were considered for this comparison. Noticeably, in Parking and TPP, the sets are the same regardless of which heuristics is used or not, and thus, these domains are not listed in Table 2. Moreover, planners that did not solve any task in any of the encodings in a given domain are not listed in Table 2. The results provide clear evidence, mainly in Barman and Bw, that both heuristics—flaw ratio and filtering—are useful when applied together.

Technically speaking, when only flaw ratio is considered, the set of inner entanglements is the superset or equal to the set of inner entanglements without considering flaw ratio. Filtering, on the other hand, removes possibly unpromising inner entanglements from the learned set. In other words, flaw ratio and filtering heuristics provide a useful synergy for maximizing the potential of inner entanglements.

8.6 | Discussion of results

This subsection is devoted to discussing the interesting aspects of the experimental analysis results.

8.6.1 | Summary of performance improvement

Inner entanglements eliminate unpromising alternatives in the search space. As already discussed in the paper, the pruning power of inner entanglements is a key factor for their usefulness. Therefore, we proposed a method for filtering inner entanglements whose pruning power is small (see Section 7.2). Our experiments confirmed that the filtering method often manages to filter out unpromising inner entanglements while keeping the promising ones. Inner entanglements are efficient if the exclusivity of both predicate achievement and requirement between a pair of operators holds. The reason mainly lies in the compact and informative encoding (see Section 5). Such inner entanglements were extracted in Bw (ie, putting the block on the table always after it is unstacked), in Depots (ie, loading a crate always after it is lifted), and in TPP (ie, loading goods always after buying it). Our experiments showed a performance improvement among the planners in these domains. Such results indicate that inner entanglements have a good potential for improvement. We have also identified a few cases where inner entanglements have a detrimental effect on planners (eg, Mercury in Barman). As discussed in Section 6.5, the representation of inner entanglements has an impact on heuristics computation. Generally speaking, despite pruning the search space, the representation of inner entanglements might introduce local minima of heuristic functions that, in consequence, might have a detrimental effect on planning engines since they need to search more nodes to escape such minima.

8.6.2 | Completeness issues

As discussed earlier, our method for extracting inner entanglements follows an assumption that a set of inner entanglements that holds for a set of training planning tasks also holds for the whole class of planning tasks (ie, the testing ones). If this assumption does not hold for some tasks in the class, they become unsolvable if inner entanglements are enforced. We observed in our experiments that the majority of reformulated tasks (by encoding inner entanglements) was solved by at least one of the planners. In Barman and Parking, 5 and 16 reformulated tasks, respectively, have not been solved by any of the planners. However, no evidence was obtained whether this was caused by their unsolvability or whether these tasks were too hard for the planners. In other words, the planners on these tasks run out of time or memory.

To alleviate the incompleteness issue, we can try to solve the original task after the reformulated one failed. Specifically, we run the planner on the reformulated task, and if the task is considered unsolvable before the time limit is reached, then we run the planner on the original task. Theoretically, the unsolvability of a planning task can be identified in finite time if a complete planning engine is considered. In practice, we can identify some unsolvable tasks in little time if the reachability analysis reveals that the goal cannot be reached.[4] Since we have not identified any unsolvable reformulated task in the given time limit, the same results as for the best-plan or the worst-plan set of entanglements would have applied for the aforementioned approach. Alternatively, we can alleviate the incompleteness issue by manually verifying the correctness of extracted inner entanglements or by incorporating reformulated tasks along with original tasks into planning portfolios such as PbP.[22]

8.6.3 | Improvement to the quality of plans generated

In general, inner entanglements do not guarantee the optimality of solution plans. Strengthening definitions of inner entanglements to guarantee plan optimality is, of course, theoretically possible. Given the complexity results of "normal" inner entanglements, we can expect the same for

"optimal" inner entanglements. Using the approximation algorithm for extracting inner entanglements on optimal training plans with zero flaw ratio might extract some useful "optimal" inner entanglements. However, we believe that there is a high risk of extracting incorrect "optimal" inner entanglements. For example, the recently mentioned inner entanglements in the Depots domain are "optimal" for problems where each crate must be delivered to a different location. If, in some problem, a crate must be stacked on a different pallet but within the same location, such inner entanglements will force the planner (even the optimal one) to extract suboptimal plans. Speaking about satisficing planning, these entanglements will prevent planners from finding a plan only if no truck is available. Such a problem is very atypical. Hence, there is a very low risk of extracting incorrect "normal" inner entanglements, and similar observations can be made in other domains. Our experimental results have not clearly indicated any case in which the extracted set of inner entanglements did not hold.

8.6.4 | Relationship to other pruning or problem reformulation techniques

Although there are several techniques based on pruning or problem reformulation techniques (discussed in the Related Work section), inner entanglements are complementary to these techniques. Pruning techniques are often an inseparable part of advanced planning engines. We used several of such planning engines, which were successful in the past IPCs, for our experiments. We demonstrated that inner entanglements can often significantly improve their performance. Outer entanglements[19,20] prune unpromising instances of the planning operator according to their relations with initial or goal atoms. Inner entanglements are complementary to outer entanglements as has already been demonstrated in the previous work.[20] Another well-known technique for reformulating domain models is learning macros. A recent work introducing ASAP, which is a planner based on the algorithm selection approach that selects the best couple (planner,encoding) for a given domain, has shown that inner entanglements and the combination of outer and inner entanglements often outperformed macros.[38] Exploiting a natural property of inner entanglements, ie, exclusivity of predicate achievement or requirement, has been also used for generating macros.[28] Such macros can be, in some cases, beneficial; however, such an approach cannot be used in cases where operators in an inner-entanglement relation cannot be applied consecutively. Inner entanglements can support also other learning techniques that are used in planning. Roller[10] is a system that learns decision trees that are then used to guide depth-first search. Combining Roller with entanglements (both inner and outer), such a system is called Rollent, brought promising results as well.[62]

9 | CONCLUSIONS AND FUTURE WORK

In this paper, we have presented inner entanglements, which are relations between pairs of planning operators and predicates such that an operator exclusively achieves a predicate for another operator or an operator exclusively requires a predicate from another operator. To deal with the intractability of deciding whether a given inner entanglement holds for a given planning task (see Section 6), we used an approximation method for extracting "domain-specific" sets of inner entanglements from training plans, solution plans of simple tasks. Inner entanglements can be encoded into domain models without extending the input language of a planner (see Section 5), and therefore, they can be understood and exploited as planner-independent knowledge.

Inner entanglements are able to considerably improve the planning process as our experiments demonstrated. In Bw, Depots, and TPP, the considerable performance improvement was observed among almost all the planners. As discussed before, inner entanglements are especially powerful if the exclusivity of both predicate achievement and requirement between a given pair of operators holds, which is the case of Bw, Depots, and TPP. Generally, inner entanglements have a good potential for a performance improvement if they are not "clashing" with a given planning technique, as demonstrated in Barman (Mercury) and Bw (Lama).

The pruning power of inner entanglements is a crucial aspect for their success. In particular, we need to avoid creating them with rarely used operators and when the argument count of an entangled operator is higher than certain other operators in the domain model (as explained in Section 7.2). Incorporating the aforementioned filtering technique into the inner-entanglement learning method alleviated most of the performance concerns raised in the previous works.[20,21]

We identified several avenues for future research. First, we believe that inner entanglements can be considered directly in heuristics—rather than being encoded in PDDL—by, for instance, penalizing possibilities that violate these entanglements. Second, we believe that inner entanglements can be encoded, for instance, in decision trees or control rules. This might improve the performance of related planners, ie, Roller[10] or TALplanner.[12] Third, we will investigate in which cases deciding nontrivial inner entanglements is tractable. Given the insights in this paper (see Section 6.4), we believe that by analyzing the domain structure, we can identify some useful inner entanglements in polynomial time. Finally, given the encouraging spread of results among sets of planners and domains, we intend to work toward including an inner entanglement–generating facility as part of a knowledge engineering workbench.

ACKNOWLEDGMENTS

The authors would like to acknowledge the use of the University of Huddersfield Queensgate Grid in carrying out this work. This research was partly funded by the UK Engineering and Physical Sciences Research Council Autonomous and Intelligent Systems Programme (grant EP/J011991/1), by the Czech Science Foundation (project 17-17125Y), and by the OP VVV–funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics."

ORCID

Lukáš Chrpa https://orcid.org/0000-0001-9713-7748
Mauro Vallati https://orcid.org/0000-0002-8429-3570

REFERENCES

1. Bernard DE, Gamble EB, Rouquette NF, et al. *Remote Agent Experiment DS1 Technology Validation Report*. Technical Report. Mountain View, CA: Ames Research Center and JPL; 2000.
2. Bylander T. The computational complexity of propositional STRIPS planning. *Artif Intell*. 1994;69(1-2):165-204.
3. Ghallab M, Howe A, Knoblock C, et al. *PDDL - The Planning Domain Definition Language*. Technical Report. New Haven, CT: Yale Center for Computational Vision and Control; 1998.
4. Bonet B, Geffner H. Planning as heuristic search: new results. In: *Recent Advances in AI Planning: 5th European Conference on Planning, ECP'99, Durham, UK, September 8-10, 1999. Proceedings*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 1999.

5. Kautz HA, Selman B. Planning as satisfiability. In: Proceedings of European Conference on Artificial Intelligence; 1992; Vienna, Austria.
6. Godefroid P, Kabanza F. An efficient reactive planner for synthesizing reactive plans. In: Proceedings of the 9th National Conference on Artificial Intelligence; 1991; Anaheim, CA. http://www.aaai.org/Library/AAAI/1991/aaai91-100.php
7. Fox M, Long D. The detection and exploitation of symmetry in planning problems. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI); 1999; Stockholm, Sweden. http://ijcai.org/Proceedings/99-2/Papers/041.pdf
8. Minton S, Carbonell JG. Strategies for learning search control rules: an explanation-based approach. In: Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI); 1987; Milan, Italy.
9. Bacchus F, Kabanza F. Using temporal logics to express search control knowledge for planning. *Artif Intell*. 2000;116(1-2):123-191. https://doi.org/10.1016/S0004-3702(99)00071-5
10. de la Rosa T, Celorrio SJ, Fuentetaja R, Borrajo D. Scaling up heuristic planning with relational decision trees. *J Artif Intell Res*. 2011;40:767-813.
11. Yoon S, Fern A, Givan R. Learning control knowledge for forward search planning. *J Mach Learn Res*. 2008;9:683-718.
12. Kvarnström J, Doherty P. TALplanner: a temporal logic based forward chaining planner. *Ann Math Artif Intell*. 2000;30(1-4):119-169.
13. Chrpa L. Generation of macro-operators via investigation of action dependencies in plans. *Knowl Eng Rev*. 2010;25(3):281-297.
14. Korf RE. Macro-operators: a weak method for learning. *Artif Intell*. 1985;26(1):35-77.
15. McCluskey TL, Porteous JM. Engineering and compiling planning domain models to promote validity and efficiency. *Artif Intell*. 1997;95(1):1-65.
16. Newton MAH, Levine J, Fox M, Long D. Learning macro-actions for arbitrary planners and domains. In: Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS); 2007; Providence, RI.
17. Haslum P. Reducing accidental complexity in planning problems. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI); 2007; Hyderabad, India.
18. Tozicka J, Jakubuv J, Svatos M, Komenda A. Recursive polynomial reductions for classical planning. In: Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS); 2016; London, UK. http://www.aaai.org/ocs/index.php/ICAPS/ICAPS16/paper/view/13088
19. Chrpa L, Barták R. Reformulating planning problems by eliminating unpromising actions. In: Proceedings of the Eight Symposium on Abstraction, Reformulation, and Approximation (SARA); 2009; Lake Arrowhead, CA.
20. Chrpa L, McCluskey TL. On exploiting structures of classical planning problems: generalizing entanglements. In: Proceedings of the 20th European Conference on Artificial Intelligence (ECAI); 2012; Montpellier, France.
21. Chrpa L, Vallati M. Revisiting inner entanglements in classical planning. In: Proceedings of Scandinavian Conference on Artificial Intelligence; 2013; Aalborg, Denmark.
22. Gerevini A, Saetti A, Vallati M. Planning through automatic portfolio configuration: the pbp approach. *J Artif Intell Res*. 2014;50:639-696. https://doi.org/10.1613/jair.4359
23. Dawson C, Siklóssy L. The role of preprocessing in problem solving systems: an ounce of reflection is worth a pound of backtracking. In: Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI); 1977; Cambridge, MA.
24. Botea A, Enzenberger M, Müller M, Schaeffer J. Macro-FF: improving AI planning with automatically learned macro-operators. *J Artif Intell Res*. 2005;24:581-621.
25. Coles A, Fox M, Smith A. Online identification of useful macro-actions for planning. In: Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS); 2007; Providence, RI.
26. Chrpa L, Siddiqui FH. Exploiting block deordering for improving planners efficiency. In: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI); 2015; Buenos Aires, Argentina. http://ijcai.org/Abstract/15/220
27. Siddiqui FH, Haslum P. Block-structured plan deordering. In: *AI 2012: Advances in Artificial Intelligence: 25th Australasian Joint Conference, Sydney, Australia, December 4-7, 2012. Proceedings*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2012:803-814.

28. Chrpa L, Vallati M, McCluskey TL, Kitchin D. Generating macro-operators by exploiting inner entanglements. In: Proceedings of the Tenth Symposium on Abstraction, Reformulation, and Approximation (SARA); 2013; Leavenworth, WA.
29. Blum AL, Furst ML. Fast planning through planning graph analysis. *Artif Intell*. 1997;90(1-2):281-300.
30. Emerson EA, Sistla AP. Symmetry and model checking. *Formal Methods Syst Des*. 1996;9(1-2):105-131. https://doi.org/10.1007/BF00625970
31. Rintanen J. Symmetry reduction for SAT representations of transition systems. In: Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS); 2003; Trento, Italy. http://www.aaai.org/Library/ICAPS/2003/icaps03-004.php
32. Pochter N, Zohar A, Rosenschein JS. Exploiting problem symmetries in state-based planners. In: Proceedings of the 25th AAAI Conference on Artificial Intelligence; 2011; San Francisco, CA.
33. Domshlak C, Katz M, Shleyfman A. Enhanced symmetry breaking in cost-optimal planning as forward search. In: Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS); 2012; Atibaia, Brazil.
34. Valmari A. The state explosion problem. In: *Lectures on Petri Nets I: Basic Models: Advances in Petri Nets*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 1996:429-528. https://doi.org/10.1007/3-540-65306-6_21
35. Chen Y, Yao G. Completeness and optimality preserving reduction for planning. In: Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI); 2009; Pasadena, CA.
36. Bäckström C, Nebel B. Complexity results for SAS+ planning. *Comput Intell*. 1995;11(4):625-655.
37. Wehrle M, Helmert M, Alkhazraji Y, Mattmüller R. The relative pruning power of strong stubborn sets and expansion core. In: Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS); 2013; Rome, Italy.
38. Vallati M, Chrpa L, Kitchin D. ASAP: an automatic algorithm selection approach for planning. *Int J Artif Intell Tools*. 2014;23(6):1-25.
39. Celorrio SJ, Haslum P, Thiebaux S. Pruning bad quality causal links in sequential satisfying planning. Paper presented at: ICAPS 2013 Workshop on Planning and Learning; 2013; Rome, Italy.
40. Fox M, Long D. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J Artif Intell Res*. 2003;20:61-124.
41. Ghallab M, Nau D, Traverso P. *Automated Planning: Theory and Practice*. San Francisco, CA: Morgan Kaufmann Publishers; 2004.
42. Chapman D. Planning for conjunctive goals. *Artif Intell*. 1987;32(3):333-377.
43. Sacerdoti ED. The nonlinear nature of plans. In: Proceedings of the Fourth International Joint Conference on Artificial Intelligence; 1975; Tbilisi, USSR. http://ijcai.org/Proceedings/75/Papers/028.pdf
44. Gupta N, Nau DS. On the complexity of blocks-world planning. *Artif Intell*. 1992;56(2-3):223-254. https://doi.org/10.1016/0004-3702(92)90028-V
45. Slaney J, Thiébaux S. Blocks world revisited. *Artif Intell*. 2001;125(1-2):119-153.
46. Hoffmann J, Porteous J, Sebastia L. Ordered landmarks in planning. *J Artif Intell Res*. 2004;22:215-278.
47. Richter S, Westphal M. The LAMA planner: guiding cost-based anytime planning with landmarks. *J Artif Intell Res*. 2010;39:127-177.
48. Minton S. Quantitative results concerning the utility of explanation-based learning. In: Proceedings of the Seventh AAAI National Conference on Artificial Intelligence; 1988; Saint Paul, MN.
49. Hoffmann J. The metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *J Artif Intell Res*. 2003;20:291-341.
50. Gerevini A, Saetti A, Serina I. Planning through stochastic local search and temporal action graphs in LPG. *J Artif Intell Res*. 2003;20:239-290.
51. Richter S, Westphal M, Helmert M. LAMA 2008 and 2011. In: Booklet of the 7th International Planning Competition. 2011.
52. Lipovetzky N, Geffner H. Searching for plans with carefully designed probes. In: Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS); 2011; Freiburg, Germany.
53. Lipovetzky N, Ramirez M, Muise C, Geffner H. Width and inference based planners: SIW, BFS(f), and PROBE. In: The Eighth International Planning Competition. Description of Participant Planners of the Deterministic Track. 2014:6-7.

54. Rintanen J. Engineering efficient planners with SAT. In: Proceedings of the 20th European Conference on Artificial Intelligence (ECAI); 2012; Montpellier, France.
55. Rintanen J. Madagascar: scalable planning with SAT. In: The Eighth International Planning Competition. Description of Participant Planners of the Deterministic Track. 2014:66-70.
56. Vidal V. YAHSP3 and YAHSP3-MT in the 8th international planning competition. In: The Eighth International Planning Competition. Description of Participant Planners of the Deterministic Track. 2014:64-65.
57. Domshlak C, Hoffmann J, Katz M. Red-black planning: a new systematic approach to partial delete relaxation. *Artif Intell*. 2015;221:73-114. https://doi.org/10.1016/j.artint.2014.12.008
58. Hoffmann J, Nebel B. The FF planning system: fast plan generation through heuristic search. *J Artif Intell Res*. 2001;14:253-302.
59. Chrpa L, Vallati M, Osborne H. Learnability of specific structural patterns of planning problems. In: Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence (ICTAI); 2013; Herndon, VA.
60. Núñez S, Borrajo D, López CL. Performance analysis of planning portfolios. In: Proceedings of the Fifth Annual Symposium on Combinatorial Search (SOCS); 2012; Niagara Falls, Canada.
61. Coles A, Coles A, García Olaya A, et al. A survey of the seventh international planning competition (review). *AI Magazine*. 2012;33(1):83-88.
62. Fuentetaja R, Chrpa L, McCluskey TL, Vallati M. Exploring the synergy between two modular learning techniques for automated planning. In: Proceedings of the Eighth Annual Symposium on Combinatorial Search (SOCS); 2015; Ein Gedi, Israel.

How to cite this article: Chrpa L, Vallati M, McCluskey TL. Inner entanglements: Narrowing the search in classical planning by problem reformulation. *Computational Intelligence*. 2019;35:395–429. https://doi.org/10.1111/coin.12203