Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Abstractions Exploiting Statistical Information

Vasily Levitskiy

Supervisor: Ing. Rostislav Horčík, Ph.D
Study Program: Open Informatics
Specialisation: Artificial Intelligence and Computer Science
May 2023

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Levitskiy Vasily**	Personal ID number:	**499042**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Specialisation:	**Artificial Intelligence and Computer Science**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Abstractions Exploiting Statistical Information

Bachelor’s thesis title in Czech:

Abstrakce využívající statistickou informaci

Guidelines:

The topic of the thesis investigates the exploitability of statistical information induced by abstractions. Abstractions in classical planning are used to construct a lower bound on the plan cost, usually quite a pessimistic one. Using statistical information induced by the construction of the abstract transition system, one could compute an expected plan cost instead of the lower bound. More precisely, the student is expected to achieve the following goals:
1. Acquaint yourself with the abstractions and abstract transition systems used in classical planning, particularly with projections of SAS+ planning tasks (Helmert et al. 2014; Seipp and Helmert 2018).
2. Implement a program that, for a projection of a given SAS+ planning task, constructs a Markov decision process (Russell and Norvig 1995), enriching the corresponding abstract transition system by the statistical information induced by the projection.
3. Compare the expected cost with the lower bound and the actual cost for the initial state. To test your implementation, use the planning tasks from the International Planning Competitions.

Bibliography / sources:

[1] Helmert, Malte, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. 2014. “Merge-and-Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces.” Journal of the ACM 61 (3): 1–63. https://doi.org/10.1145/2559951.
[2] Seipp, Jendrik, and Malte Helmert. 2018. “Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning.” Journal of Artificial Intelligence Research 62 (July): 535–77. https://doi.org/10.1613/jair.1.11217.
[3] Russell, Stuart Jonathan, and Peter Norvig. 1995. Artificial Intelligence: A Modern Approach. Prentice-Hall.

Name and workplace of bachelor’s thesis supervisor:

Ing. Rostislav Horčík, Ph.D.   Department of Computer Science FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **10.01.2023**   Deadline for bachelor thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

Ing. Rostislav Horčík, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I would like to thank my advisor, Rostislav Horčík, for providing immense help and support in this thesis. Without their assistance, this thesis wouldn't be possible. Also big thanks to my friends and family for supporting me during my studies at FEE CTU.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 25, 2023

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 25 května 2023

Abstract

This bachelor thesis presents a novel approach for enhancing the precision of planners that employ abstractions. In traditional abstraction processes, certain information is inevitably discarded, leading to a potential loss of accuracy. However, our proposed method utilizes a part of this discarded information by incorporating it into transition probabilities. By assigning probabilities to the transitions based on the discarded information, we construct a Markov Decision Process (MDP) that facilitates the application of value iteration for cost prediction in the given problem.

The primary objective of this research is to demonstrate the feasibility and effectiveness of leveraging statistical information within abstractions. Through a comprehensive examination of the abstraction process, we illustrate how our method optimizes the use of available data, ultimately resulting in more informative heuristics.

A series of experiments are conducted to evaluate our method's performance. The findings demonstrate the superiority of our approach in terms of precision. Additionally, we discuss potential avenues for further development and optimization.

Keywords: Planner, PDDL, Planning, MDP, Abstractions, Projections, Probability

Supervisor: Ing. Rostislav Horčík, Ph.D
E-322, Charles Square 13, Prague 2

Abstrakt

Tato bakalářská práce představuje nový přístup ke zvýšení přesnosti plánovačů využívajících abstrakce. V tradičních procesech abstrakce dochází k nevyhnutelné ztrátě určitých informací, což může vést ke ztrátě přesnosti. Nicméně náš navrhovaný přístup využívá část těchto ztracených informací tím, že je zahrnuje do pravděpodobností přechodů. Přiřazením pravděpodobností přechodům na základě těchto ztracených informací konstruujeme Markovský rozhodovací proces (MDP), který umožňuje aplikaci iterace hodnot pro predikci nákladů v daném problému.

Hlavním cílem této práce je demonstrovat proveditelnost a účinnost využívání statistických informací v rámci abstrakcí. Skrze podrobné zkoumání procesu abstrakce ukazujeme, jak náš přístup optimalizuje využití dostupných dat a nakonec vede k více informovaným heuristikám.

Pro vyhodnocení výkonnosti naší metody jsou provedeny řady experimentů. Výsledky těchto experimentů jednoznačně prokazují převahu našeho přístupu v oblasti přesnosti. Navíc diskutujeme možnosti dalšího rozvoje a optimalizace.

Klíčová slova: Plánovač, PDDL, Plánování, MDP, Abstrakce, Projekce, Pravděpodobnost

Překlad názvu: Abstrakce využívající statistickou informaci

Contents

Figures Tables

Chapter 1
Introduction

Planning is a field within artificial intelligence that dates back to the 1960s, with the goal of replicating human problem-solving abilities. There are many different approaches to planning, each with its own specific goals and scope, but all of them aim to design a tool that can automatically find a plan based on a high-level description of an initial state, a goal, and a set of available operators. A plan is a series of operators that transforms the initial state into one that satisfies the goal.

The ideal goal in planning is to create a single tool that can solve all tasks in a reasonable amount of time. One of the main problems of planning is the vast state space of planning tasks, making it impossible to solve in real-time. Different methods have been used to tackle this problem. What discerns our method from the others is that methods that use abstractions (which will be explained in more detail later) result in the loss of a lot of information, making the solution much more imprecise. The main difference between these methods and ours is that we utilize a part of the lost information instead of discarding it entirely.

1.1 Methods of solution

Planning tasks are commonly solved using search algorithms such as A* (A-star) or GBFS (Greedy Best-First Search) that are guided by a heuristic function denoted as h. The purpose of the heuristic function is to estimate the distance or cost from each state to the goal state. By providing this estimation, the heuristic guides the search algorithm to prioritize exploration in directions that are likely to lead to the goal state. It is important to note that heuristics are not guaranteed to find optimal solutions but rather aim to find reasonable solutions quickly. The effectiveness of a heuristic depends on the quality of the estimates it provides and how well they reflect the actual cost of the solution.

One approach to constructing a heuristic function is through the use of abstractions.

1.2 Abstractions

Abstractions[9] involves simplifying the planning problem by representing it at a higher level of granularity. This process reduces the complexity of the problem by omitting or aggregating specific details. The abstraction can then be used to derive a heuristic function that provides estimates of distance or cost based on the simplified representation. This enables more efficient search and decision-making by focusing on essential aspects of the problem while still aiming to reach the actual goal state. This process can be observed in Figure 1.1

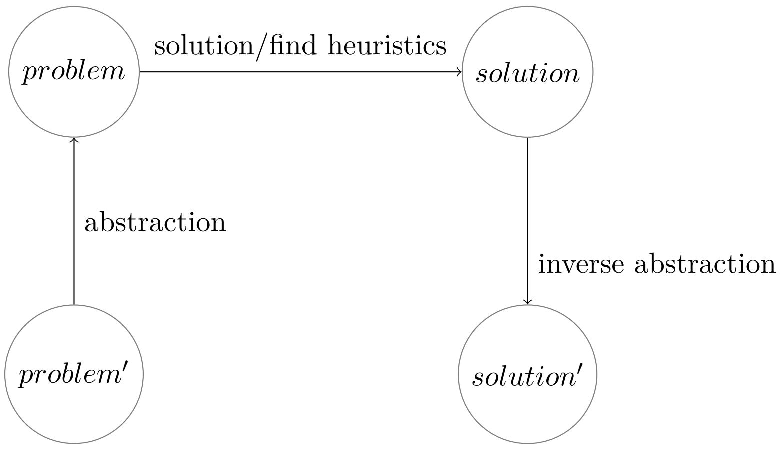

Figure 1.1: Correspondence between Abstract and Concrete Level

It is important that the abstraction accurately reflects the task, as the solution derived from the abstract representation should be directly applicable to the original task. In other words, the abstraction must be a faithful representation of the task in order to be useful. By stripping away unnecessary details and focusing on the key components, abstractions allow us to understand and solve complex problems more efficiently.

For example, if we have a planning problem with a delivery system, we can ignore a trucks fuel reserve or the capacity of the truck can be overlooked.

To demonstrate the role of abstraction, suppose that we have n operators and k states. A state space can then be represented as a $n * k * k$ table, where 1 will mean that a path exists from k to k' and 0 means that a path doesn't exist. Such a table of integers will take $4 * n * k * k$ bytes, quickly becoming impossible to fit into a regular computer.

This work focuses on a particular type of abstraction, namely projections. Projections are among the most common abstractions applied in classical planning. They became popular after their use in the pattern database heuristics [4].

1.3 MDP

To create the heuristics, we will turn the projection into an MDP. The Markov decision process (MDP) is a mathematical framework that has become a prominent tool for modeling decision-making under uncertainty; see, for instance, [8]. The MDP allows decision-makers to choose among a set of actions in order to maximize some long-term reward or utility, even when the outcomes of these actions are uncertain and depend on various factors. MDPs have found broad applicability in many different fields, including robotics, automatic control, economics, and manufacturing, where decision-making processes are often complex and uncertain. By utilizing the MDP framework, decision-makers can model and solve problems in which multiple decision points are present, and decisions must be made over an extended period of time with uncertain outcomes.

The mathematical foundation of MDPs traces back to Andrey Markov, a prominent Russian mathematician. The MDP represents a generalization of Markov chains, which are mathematical models that describe the probability of a system transitioning from one state to another over time. MDPs extend this model by adding decision-making processes.

MDPs are often represented as a tuple, consisting of a set of states, a set of actions, a transition function that describes the probability of transitioning between states given a particular action, a reward function that assigns a reward to each state-action pair, and a discount factor that reflects the preference for immediate rewards over future rewards. Given this representation, a decision-maker can use various algorithms, such as value iteration or policy iteration, to find the optimal policy for decision-making.

1.4 Summary

In this thesis, we develop a method using statistical information when creating abstractions to turn the abstract transition system into an MDP. Abstractions allow us to represent and reason about complex environments and tasks in a more simplified and tractable manner. However, the creation of abstractions also results in the loss of information. To address this issue, we propose using statistical information during the abstraction creation process, which can then be incorporated into a Markov Decision Process (MDP) to calculate an expected cost to the goal. That can then be used for countering the information loss.

Chapter 2
Background

2.1 Labeled Transition Systems and Abstractions

To define planning tasks formally, we must first introduce labeled transition systems (LTSs) because planning tasks are succinctly represented LTSs; for details, see [6].

Definition 2.1. A *labeled transition system* is a tuple $\Theta = \langle S, L, T, s_0, S_* \rangle$ where S is a finite set of states, L is a finite set of labels, $T \subseteq S \times L \times S$ is a set of labeled transitions, $s_0 \in S$ is the initial state, and $S_* \subseteq S$ is a set of goal states.

An LTS Θ can be viewed as a directed graph whose arcs are labeled by elements from L. A triple $\langle s, l, t \rangle \in T$ represents an arc leading from s to t and labeled by l. We denote such a triple as $s \xrightarrow{l} t$.

A path $\pi = s_0 \xrightarrow{l_1} s_1 \xrightarrow{l_2} s_2 \cdots s_{n-1} \xrightarrow{l_n} s_n$ from s_0 to a state $s_n \in S_*$ is a *solution* for Θ. The shortest path is called *optimal*. For any state $s \in S$, we define $h^*(s)$ as the length of the shortest path from s to a goal state.

The optimal solution for an LTS can be found using an algorithm that finds the shortest path in a directed graph, like Dijkstra's algorithm. However, planning often deals with huge transition systems that do not fit into memory. Thus algorithms like A* or Greedy Best First Search (GBFS) are usually applied to find a solution; see e.g. [8]. These algorithms are navigated by a heuristic function. A heuristic function for an LTS Θ is a function $h\colon S \to \mathbb{R}$ estimating $h^*(s)$ for each state $s \in S$.

There are various methods of constructing heuristic functions. One of them employs abstractions.

Definition 2.2. Let $\Theta = \langle S, L, T, s_0, S_* \rangle$ be an LTS and $\alpha\colon S \to S^\alpha$ a surjective function to a set of *abstract* states. The map α defines an LTS $\Theta^\alpha = \langle S^\alpha, L, T^\alpha, \alpha(s_0), \alpha(S_*) \rangle$ where $T^\alpha = \{\langle \alpha(s), l, \alpha(t) \rangle \mid \langle s, l, t \rangle \in T\}$.

Using an abstraction, we reduce the size of the original LTS Θ and replace it with a smaller one Θ^α so that it fits into memory. Consequently, we can solve the abstract LTS Θ^α by searching the shortest path. Even though we lose some information on Θ, we can at least get a lower bound for $h^*(s)$ in Θ.

Note that if

$$\pi = s_1 \xrightarrow{l_1} s_2 \xrightarrow{l_2} s_3 \cdots s_{n-1} \xrightarrow{l_n} s_n \in S_*$$

is a path in Θ ending in a goal state, then

$$\alpha(\pi) = \alpha(s_1) \xrightarrow{l_1} \alpha(s_2) \xrightarrow{l_2} \alpha(s_3) \cdots \alpha(s_{n-1}) \xrightarrow{l_n} \alpha(s_n) \in \alpha(S_*)$$

is a path in Θ^α ending in a goal state. Consequently, if π is the shortest path from s_1 to a goal state of length n in Θ, $\alpha(\pi)$ is a path in Θ^α of the same length. Thus we have $h^*(\alpha(s_1)) \leq h^*(s_1)$.

Each abstraction $\alpha\colon S \to S^\alpha$ induces an equivalence relation on the set of states S as follows:

$$s \sim^\alpha t \quad \text{if, and only if,} \quad \alpha(s) = \alpha(t).$$

In other words, s and t are equivalent if they are mapped to the same abstract state. For a state $s \in S$, the equivalence class containing s is denoted $[s]_\alpha = \{t \in S \mid \alpha(s) = \alpha(t)\}$. Given an abstract state $s' \in S^\alpha$, one can also define the equivalence class of all states that are mapped to s' as $\alpha^{-1}(s') = \{s \in S \mid \alpha(s) = s'\}$.

2.2 Planning

In this thesis, we will consider the SAS+ planning tasks [1].

Definition 2.3. A planning task is a 4-tuple $\mathcal{P} = \langle V, O, s_0, s_g\rangle$, where:

- $V = \{v_1, v_2, ..., v_N\}$ is a set of variables such that each variable $v \in V$ has its finite domain $\mathsf{Dom}(v)$. Using the variables, we define (partial) states. A *partial state* is a partial map s from V to $\cup_{v\in V}\mathsf{Dom}(v)$ such that $s(v) \in \mathsf{Dom}(v)$ for each $v \in V$ where s is defined.

  We can view s as a set of pairs in the form $\langle v, d\rangle$, where $v \in V$ and $d \in \mathsf{Dom}(v)$. Given a partial state s, the set of its variables is $\mathsf{Var}(s) = \{v \in V \mid \langle v, d\rangle \in s \text{ for some } d \in \mathsf{Dom}(v)\}$.

  A partial state s is called a *state* if s is defined for all $v \in V$, i.e., for each variable $v \in V$ there is $d \in \mathsf{Dom}(v)$ such that $\langle v, d\rangle \in s$. The set of all states over V is denoted S_V.

- $O = \{o_1, o_2, ...o_N\}$ is a set of operators in the form $o = \langle \mathsf{Pre}_o, \mathsf{Eff}_o\rangle$, where Pre_o, Eff_o are partial states. To simplify the construction of MDP, we assume that all operators have a unit cost.

- s_0 is a state called *initial*.

- s_g is a partial state called *goal*.

Now let's define transitions. An operator $o \in O$ is applicable in a state s if $\mathsf{Pre}_o \subseteq s$. If we apply o in s, we obtain a state $t = o[s]$ defined as follows:

$$t(v) = \begin{cases} \mathsf{Eff}_o(v) & \text{if } \mathsf{Eff}_o(v) \text{ is defined,} \\ s(v) & \text{otherwise.} \end{cases} \tag{2.1}$$

Given a state s, an *s-plan* is a sequence of operators $\pi = o_1, ..., o_n$, such that there exist states $s = s_0, s_1, ..., s_n$ where $o_i \in O$ is applicable in s_{i-1} for all $i \in 1, ..., n$ and s_n is a goal state. A state s_n is a goal state if $s_g \subseteq s_n$. As we consider only unit costs, the cost of π is its length, i.e., n. The shortest plan is said to be *optimal*.

Note that any planning task $\mathcal{P} = \langle V, O, s_0, s_g \rangle$ induces an LTS $\Theta_{\mathcal{P}} = \langle S_V, O, T, s_0, S_* \rangle$ where $S_* = \{s \in S_V \mid s_g \subseteq s\}$ and

$$T = \{\langle s, o, t \rangle \mid o \text{ is applicable in } s \text{ and } t = o[s]\}.$$

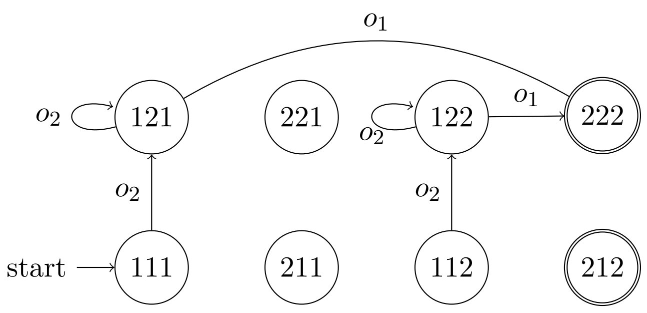

Figure 2.1: Example LTS

Example 2.4. Suppose $V = \{v_1, v_2, v_3\}$ such that $\mathsf{Dom}(v_i) = \{1, 2\}$ for all $i \in \{1, 2, 3\}$. The number of all states is $|S_V| = 2^3 = 8$. Consider a planning task $\mathcal{P} = \langle V, O, s_0, s_g \rangle$, where $s_0 = \{\langle v_1, 1 \rangle, \langle v_2, 1 \rangle, \langle v_3, 1 \rangle\}$, $s_g = \{\langle v_1, 2 \rangle, \langle v_3, 2 \rangle\}$, and $O = \{o_1, o_2\}$ are operators defined as follows:

$$\begin{aligned} \mathsf{Pre}_{o_1} &= \{\langle v_1, 1 \rangle, \langle v_2, 2 \rangle\} \\ \mathsf{Eff}_{o_1} &= \{\langle v_1, 2 \rangle, \langle v_3, 2 \rangle\} \\ \mathsf{Pre}_{o_2} &= \{\langle v_1, 1 \rangle\} \\ \mathsf{Eff}_{o_2} &= \{\langle v_2, 2 \rangle\} \end{aligned}$$

The corresponding LTS is depicted in Figure 2.1. The states are represented by sequences of respective variable values. For instance, the sequence 211 denotes the state $\{\langle v_1, 2 \rangle, \langle v_2, 1 \rangle, \langle v_3, 1 \rangle\}$. The optimal plan o_2, o_1 corresponds to the shortest path from 111 to any of the ending states 222 and 212. It has a length of 2.

2.3 Projections

Let $\mathcal{P} = \langle V, O, s_0, s_g \rangle$ be a planning task. One of the standard abstractions applied for planning tasks is projections. Projections have several advantages. In particular, they are easily computed, and the resulting abstract LTS can also be represented as a planning task. Let $U \subseteq V$ be a subset of variables V. Given a partial state s, we define the restriction of s to U by $s|_U = \{\langle v, d \rangle \in s \mid v \in U\}$.

Definition 2.5. Let $\mathcal{P} = \langle V, O, s_0, s_g \rangle$ be a planning task, $\Theta_{\mathcal{P}}$ its LTS, and $U \subseteq V$. The projection is the map $\pi_U \colon S_V \to S_U$ mapping a state $s \in S_V$ to its restriction $s|_U \in S_U$. The corresponding abstract LTS $\Theta_{\mathcal{P}}^{\pi_U}$ can be represented as the planning task $\mathcal{P}_U = \langle U, O_U, s_0|_U, s_g|_U \rangle$ where the set operators of $O_U = \{\langle \mathsf{Pre}_o|_U, \mathsf{Eff}_o|_U \rangle \mid o \in O\}$.

We will abuse the notation and denote the operators in $\mathcal{P}_U$ by the same symbols as operators in $\mathcal{P}$. The projection $\mathcal{P}_U$ determines an equivalence relation $\sim_U$ on the set of states S_V. We have $s \sim_U s'$ if the states $s, s' \in S_V$ agree on U, i.e., $s|_U = s'|_U$. For any state s and its equivalence class $[s]_U = \{s' \in S_V \mid s \sim_U s'\}$, the size of $[s]_U$ equals $\prod_{v \in V \setminus U} |\mathsf{Dom}(v)|$.

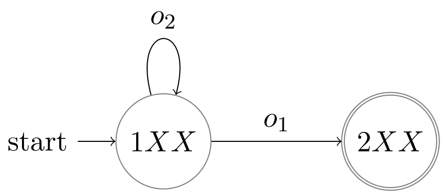

Figure 2.2: The projection of $\mathcal{P}$ from Example 2.4 into the variable v_1.

Example 2.6. Consider the planning task $\mathcal{P}$ from Example 2.4. Let $U = \{v_1\}$. The projection $\mathcal{P}_U$ consists of 2 states, namely $S_U = \{\{\langle v_1, 1\rangle\}, \{\langle v_1, 2\rangle\}\}$. The initial state is $s_0|_U = \{\langle v_1, 1\rangle\}$ and the goal is $s_g|_U = \{\langle v_1, 2\rangle\}$.

Its transition system corresponding to $\mathcal{P}_U$ is depicted in Figure 2.2. The states are denoted by the value of variable v_1 followed by XX as the second and the third variable were discarded, e.g., $\{\langle v_1, 1\rangle\}$ is denoted $1XX$. The state $2XX$ is the only goal state. The length of the optimal plan for $\mathcal{P}_U$ is 1, particularly we have $1 = h^*(1XX) = h^*(\pi_U(111)) \leq h^*(111) = 2$.

For each state $s \in S_V$, the equivalence class $[s]_U$ is of size $2^2 = 4$. For example, $\pi_U^{-1}(1XX) = \{111, 112, 121, 122\}$.

2.4 MDP

Definition 2.7. A *Markov Decision Process* (MDP) is $\mathcal{M} = \langle S, A, T, R, \gamma \rangle$ where

- S is a set of states,
- A is a map assigning to each state s a set $A(s)$ of applicable actions in s,
- T is a transition map assigning to each state $s \in S$, an action $a \in A(s)$, and a state t a probability $T(t|s, a) \in [0, 1]$ that the MDP gets into t if a is applied in s,
- R is a reward function assigning to each transition, i.e., a triple $\langle s, a, t \rangle$ for states s, t and an action $a \in A(s)$, a real number $R(s, a, t) \in \mathbb{R}$.
- and $\gamma \in [0, 1)$ is a discount factor.

2.4.1 Optimal policy

A *policy* is a map σ assigning to each state s a probability distribution over $A(s)$, i.e., $\sigma(a \mid s)$ expresses the probability that we apply an action $a \in A(s)$ provided we are in the state s. The policy σ is called *determinitic* if $\sigma(a \mid s) \in \{0,1\}$, i.e., we always apply a single action for a fixed state s.

Let σ be a policy and s a state. If we start in s and follow the policy σ, we obtain a sequence $\langle R_t \mid t = 0, \ldots \rangle$ of random rewards. A *value function* for the policy σ is the function $V^\sigma \colon S \to \mathbb{R}$ assigning to each state $s \in S$ the expected value of discounted cummulative reward:

$$V^\sigma(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R_t\right].$$

The discounted factor γ ensures the converge of the above infinite sum for $\gamma < 1$. It reflects the trade-off between the immediate reward and the long-term rewards.

To solve an MDP, we look for a policy σ^* that maximizes its value function denoted V^*, i.e., $V^*(s) \geq V^\sigma(s)$ for any policy σ and any state $s \in S$. It is known that such policy σ^* exists and we may even assume that σ^* is deterministic.

2.4.2 Bellman Equation

To construct an algorithm computing the optimal value function V^*, one can start from the Bellman Equation [2]. The Bellman equation combines two essential concepts: the immediate expected reward obtained from taking an action in a state and the expected value of the resulting state. It considers the probabilistic nature of transitions between states and the discounted future rewards.

$$V^*(s) = \max_{a \in A(s)} \left(\sum_{s' \in S} T(s'|s,a) R(s,a,s') + \gamma \sum_{s' \in S} T(s'|s,a) V^*(s') \right) \tag{2.2}$$

where $\sum_{s' \in S} T(s'|s,a) R(s,a,s')$ is the expected reward when we apply the action $a \in A(s)$ in s and $\sum_{s' \in S} T(s'|s,a) V^*(s')$ is the expected value of the next state over all possible next states s'.

2.4.3 Value Iteration

The value iteration[8] algorithm is an iterative process used to find the optimal value function V^*. The value iteration leverages the Bellman equation (2.2). By iteratively improving the value estimates, value iteration converges to the actual optimal values, allowing the agent to make optimal decisions based on the computed values.

The algorithm goes as follows:

1. Initialize the value function for each state to zero.

2. For each state, update the value function based on the Bellman equation using the previous value function and the transition probabilities and rewards.
3. Repeat Step 2 until convergence, where convergence is defined as a small change in the value function or a maximum number of iterations is reached.
4. Once the value function has converged, compute the optimal policy by selecting the action that maximizes the expected value using the updated value function.
5. Return the optimal policy and the value function.

We need only the value function V^* since our heuristic introduced in the next chapter utilizes only V^*.

Chapter 3

Abstractions with Statistical Information

Moving on from acquiring foundational knowledge regarding the methods utilized in this thesis, we now delve into the discussion of our own methodology. In Section 3.1, we will tackle the issue of lost information by using probabilities in a projection. Then, in Section 3.2, we will explain the necessary changes to operators in the projection. Finally, in Section 3.3, we will combine it to create the MDP.

Before going into the exact details, let us briefly explain the method. Consider that we have a task where after creating an abstraction, we have three states left only; the task is depicted in Figure 3.1.

State 2 is divided into two halves, such that operator o_2 is applicable only in the bottom half. If we use the abstraction, without our method, the cost from *State 1* to *State 3* would be 2, which will most likely be wrong since in the original problem, such a path need not exist, but due to the "merge" *State 2*, the states inside become indistinguishable, thus, making a valid path in the abstraction.

In the abstract LTS, we do not know if the resulting original state after applying o_1 belongs to the applicability region of o_2. However, we can at least quantify the chances of it by computing the probability that o_2 is applicable for a randomly chosen original state in *State 2*. Consequently, it allows us to build an MDP from the abstract LTS.

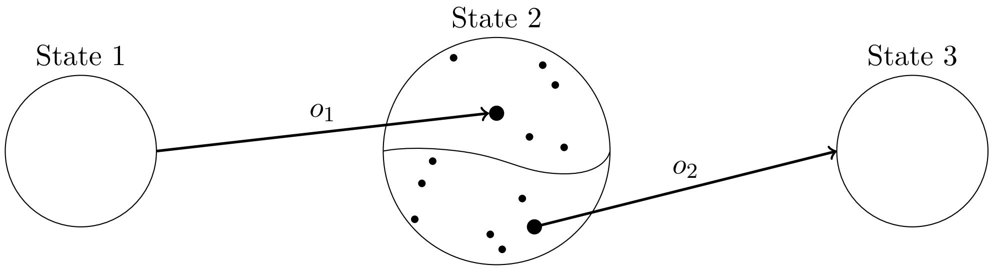

Figure 3.1: An example showing that an abstract plan need not be a plan in the original LTS.

3.1 Operator Probability in Projections

Whenever we apply a non-trivial abstraction to an LTS, we lose some information on the transitions. As a result, a plan found in the LTS need not be a plan in the original planning task. Consider the planning task $\mathcal{P} = \langle V, O, s_o, s_g \rangle$ from Example 2.4 and its projection $\mathcal{P}_U$ from Example 2.6. The plan for the projection consists of a single operator o_1. However, o_1 is not applicable in the initial state s_0 of $\mathcal{P}$. Generally, whenever we apply an operator o in an abstract state s', it need not be applicable in all preimages from $\pi_U^{-1}(s') = \{s \in S_V \mid \pi_U(s) = s'\}$. Even though we lose the information where o is applicable, we can at least collect some statistics, particularly for how many states from $\pi^{-1}(s')$ is o applicable. This allows us to define a probability that application o will be successful for a randomly chosen state in the equivalence class $\pi^{-1}(s')$.

Let $\mathcal{P} = \langle V, O, s_0, s_g \rangle$ be a planning task and $U \subseteq V$ defining the projection. Let $s \in S_V$ be a state. Suppose the an operator o is applicable in the abstract state $s|_U$, i.e., $\mathsf{Pre}_o|_U \subseteq s|_U$. Now let us consider a random state $s' \in [s]_U$. What is the probability that o is applicable in s'? First, note that the number of states in $[s]_U$ is $\prod_{v \in V \setminus U} |\mathsf{Dom}(v)|$. As we have $\mathsf{Pre}_o|_U \subseteq s|_U$, the operator o is applicable in s' if the preconditions are also satisfied on the remaining variables $V \setminus U$, i.e., $\mathsf{Pre}_o|_{V \setminus U} \subseteq s'$. The number of states $s' \in [s]_U$ satisfying this condition is

$$\prod_{v \in V \setminus (U \cup \mathsf{Var}(\mathsf{Pre}_o))} |\mathsf{Dom}(v)|,$$

so transforming this, we can get the probability:

$$p(o, s|_U) = \frac{\prod_{v \in V \setminus (U \cup \mathsf{Var}(\mathsf{Pre}_o))} |\mathsf{Dom}(v)|}{\prod_{v \in V \setminus U} |\mathsf{Dom}(v)|} \tag{3.1}$$

$$= \frac{1}{\prod_{v \in \mathsf{Var}(\mathsf{Pre}_o) \setminus U} |\mathsf{Dom}(v)|} \tag{3.2}$$

To better understand the concept, let us demonstrate the planning task $\mathcal{P}$ from Example 2.4 and its projection $\mathcal{P}_U$ from Example 2.6. Two operators o_1 and o_2 are applicable in the initial state 1. The corresponding equivalence class of preimages $\pi^{-1}(1) = \{111, 112, 121, 122\}$ has four elements. The operator o_1 is applicable in two states from $\pi^{-1}(1)$, namely 121 and 122. Thus the probability of successful application of o_1 on randomly chosen state in $\pi^{-1}(1)$ is 1/2. This agrees with Equation (3.2) as we have

$$p(o_1, 1) = \frac{1}{|\mathsf{Dom}(v_2)|} = 1/2.$$

Similarly, o_2 is applicable in any state from $\pi^{-1}(1)$ because $\mathsf{Pre}_{o_2} = \{\langle v_1, 1 \rangle\}$. Thus $p(o_2, 1) = 1$ as there is no variable in $\mathsf{Var}(\mathsf{Pre}_{o_2}) \setminus U = \emptyset$.

3.2 Operator reduction

Using the definition from the previous section, we can define the probability of every operator in the projection. Quite often, several operators represent the same transition in the projection. For efficiency reasons, we want to keep only a single transition between any pair of abstract states. The question is how to determine its probability.

Assume that $\mathcal{P} = \langle V, O, s_o, s_g \rangle$ is a planning task, $U \subseteq V$, and $\mathcal{P}_U$ is the corresponding projection. Let $s', t' \in S_U$ be abstract states and $O' = \{o \in O \mid s' \xrightarrow{o} t'\}$, i.e., the set of all operators such that o represents a transition from s' to t'. We replace the set O' with a single transition with a probability p. The probability p expresses our chances to get from a randomly chosen state $s \in \pi_U^{-1}(s')$ by applying a suitable operator from O' depending on the chosen s.

There are two obvious bounds on p. The lower bound is $\max\{p(o, s') \mid o \in O'\}$ as we can always try to apply the operator having the maximum probability. On the other hand, a trivial upper is 1. The upper bound corresponds to the situation when we do not consider the statistical information at all. Another observation regarding the probability p is that it is sufficient to consider only operators $o \in O'$ with incomparable preconditions. More precisely, let $o, o' \in O'$. If $\mathsf{Pre}_o \subseteq \mathsf{Pre}_{o'}$, then o is applicable everywhere where o' is. Moreover, $p(o, s') \geq p(o', s')$ by Equation 3.2. Consequently, o' can be safely removed from O'.

Now we assume that O' contains only operators with pairwise incomparable preconditions with respect to the inclusion. To compute p precisely, we would have to calculate the size of the union $\bigcup\{s \in \pi_U^{-1}(s') \mid \mathsf{Pre}_o \subseteq s, o \in O'\}$. As the sets in the union need not be disjoint, the size can be computed by the inclusion-exclusion principle. However, this involves an exponential computation in the number of operators in O'. To simplify the calculation, we propose to estimate the probability p by the following formula:

$$p = \max\{1, \sum_{o \in O'} p(o, s')\}. \tag{3.3}$$

Hence p might be larger than the actual probability, but it is bounded by the upper bound used in the projection without the statistical information.

To summarize, the reduction process consists of the following steps:

1. Find the clusters O' of operators representing the same operator in the projections.

2. Prune each cluster O' by removing the operators $o' \in O'$ such that there is another operator $o \in O'$ with $\mathsf{Pre}_o \subseteq \mathsf{Pre}_{o'}$.

3. Finally, apply Equation 3.3 to compute the probability of the single abstract operator in the projection representing the whole cluster O'.

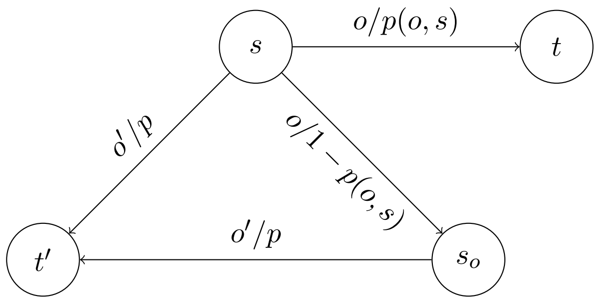
$$\begin{tikzcd} & s \arrow[rr, "o/p(o,s)"] \arrow[dl, "o'/p"'] \arrow[dr, "o/1-p(o,s)"] & & t \\ t' & & s_o \arrow[ll, "o'/p"'] & \end{tikzcd}$$

Figure 3.2: The shadow state s_o behaving like s without the operator o, i.e., whenever there is a transition from s to a state t' labelled by an operator $o' \neq o$ with a probability p, the same transition leads from s_o.

3.3 MDP from Projection

An MDP's transitions should be a stochastic matrix, meaning that the rows/column in sum should add to 1. We currently have a probability $p(o, s)$, which means that the operator o was successfully applied in an abstract state s. But what if the action can't be applied in the state s? For this reason, we introduce so-called "shadow" states, which represent an unsuccessful application of an operator o.

Let s be an abstract state and o an operator applicable in s. The probability that the application of o is unsuccessful is:

$$1 - p(o, s) \tag{3.4}$$

So we need to expand our abstract LTS by a new transition labeled by o leading from s with the probability $1 - p(o, s)$. It remains to decide where this transition leads. One possibility is to model the transition as a loop on s. However, if we are in a state represented by the abstract state s where o is not applicable, we cannot fix it by repetitive applications of o. We must apply another operator(s) first. Therefore, we introduce a shadow state s_o having the same transitions as s except that o is not applicable in s_o; see Figure 3.2. Thus, if o was unsuccessfully applied in s, we get into s_o where another operator has to be applied.

Now, let's define an MDP $\mathcal{M} = \langle S, A, T, R, \gamma \rangle$ on a projection $\mathcal{P}_U$. The set of states S will now be $S_{\{v_0\} \cup U}$ where $\mathsf{Dom}(v_0) = O \cup \{\emptyset\}$.

A normal abstract state $s \in S_U$ corresponds to $\{\langle v_0, \emptyset \rangle\} \cup s$, while a shadow state of state s corresponding to an unsuccessful application of an operator o is represented as $\{\langle v_0, o \rangle\} \cup s$.

All applicable actions for any state $s = \{\langle v_0, d \rangle\} \cup s'$ for some $d \in \mathsf{Dom}(v_0)$ and $s' \in S_U$ are defined as follows:

$$A(s) = \begin{cases} \emptyset & \text{if } s' \text{ is a goal state in } \mathcal{P}_U, \\ \{o \in O \mid \mathsf{Pre}_o|_U \subseteq s'\} & \text{if } s' \text{ not a goal and } d = \emptyset \text{ (normal state)}, \\ \{o \in O \mid o \neq o', \mathsf{Pre}_o|_U \subseteq s'\} & \text{if } d = o' \in O \text{ (shadow state)} \end{cases} \tag{3.5}$$

The shadow state has the same actions as the original state except for the action that created the shadow state.

Now let us define the transition probabilities.
Let $s = \{\langle v_0, d\rangle\} \cup s'$ for some $d \in \mathsf{Dom}(v_0)$ and $s' \in S_U$ and $o \in A(s)$, for some $t \in S$,

$$T(t|s,o) = \begin{cases} p(o,s') & \text{if } t = \{\langle v_0, \emptyset\rangle\} \cup o[s'], \\ 1 - p(o,s') & \text{if } t = \{\langle v_0, o\rangle\} \cup s', \\ 0 & \text{otherwise.} \end{cases} \tag{3.6}$$

As we assume that the operators have the unit cost, the rewards are defined as follows:

$$R(s,o,s') = \begin{cases} -1 & \text{if } s' \text{ is not a shadow state,} \\ 0 & \text{otherwise.} \end{cases} \tag{3.7}$$

Continuing from our projection example (Example 2.6), Figure 3.3 shows an MDP created from the projection depicted in Figure 2.2, where node $1'$ denotes the state $\{\langle v_0, o_1\rangle, \langle v_1, 1\rangle\}$. The letter S will be added to the node names to denote shadow states.

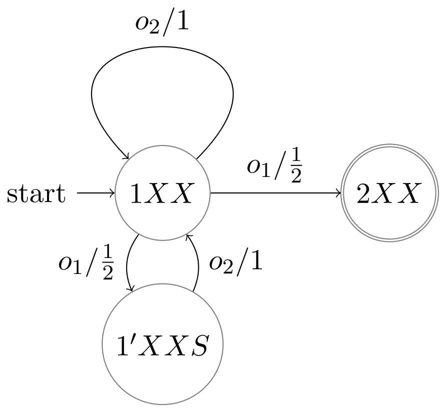

Figure 3.3: The MDP created from the projection presented in Example 2.6.

3.4 Heuristic values

After creating the MDP, we can start the Value Iteration algorithm on our MDP to compute the optimal value function V^*. Let s be a state in the MDP. Recall that $V^*(s)$ is the expected discounted cumulative reward for s. As our rewards are either -1 or 0, the opposite number $-V^*(s)$ equals the expected cost of getting from the state s to a goal state. Thus $-V^*$ can then be used as a heuristic for a search algorithm, such as GBFS. Since the original states have been "merged" into one state, the heuristic for the original state will be the value function for the state that it belongs to in the abstraction.

Chapter 4
Example

Having provided a detailed exposition of the underlying concepts and modifications to traditional approaches, it is now possible to showcase the practical application of our proposed methodology in a real-world scenario.

Our example will be a simplified version of a task called "Transport" from the IPC 2008 competition. In our problem, we have trucks, packages, and the capacities of the trucks. Our goal is to get different packages from some cities to others while minimizing the number of times the trucks move. For simplicity reasons, we omit the truck capacity in our example below.

Our planning task will be described with SAS+ [1], which is a variation on the propositional STRIPS [5].

4.1 SAS+ example

Let us consider the transport problem as a planning task $\mathcal{P} = \langle V, O, s_0, s_g \rangle$. We have 2 variables, so $V = \{v_1, v_2\}$ and each variable domain is defined as follows:

$$\mathsf{Dom}(v_1) = \{1, 2\}$$
$$\mathsf{Dom}(v_2) = \{1, 2, T\}$$

The domain values for v_1 are interpreted as at(truck-1, city-loc-1), at(truck-1, city-loc-2), respectively. Analogously, the interpretation of the domain values for v_2 is at(package-1, city-loc-1), at(package-1, city-loc-2), in(package-1, truck-1), respectively.

The initial and goal states are defined as follows:

$$s_0 = \{\langle v_1, 1 \rangle, \langle v_2, 2 \rangle\}$$
$$s_g = \{\langle v_2, 1 \rangle\}$$

Since most of the operators are the same, just the variables in the preconditions are different, we list only some of them. The symbol o_1 denotes the operator **drive truck-1 city-loc-1 city-loc-2** and o_2 denotes **drop truck-1 city-loc-1 package-1**:

$$\begin{aligned}\mathsf{Pre}_{o_1} &= \{\langle v_1, 1\rangle\}\\ \mathsf{Eff}_{o_1} &= \{\langle v_1, 2\rangle\}\end{aligned}$$

$$\begin{aligned}\mathsf{Pre}_{o_2} &= \{\langle v_1, 1\rangle, \langle v_2, T\rangle\}\\ \mathsf{Eff}_{o_2} &= \{\langle v_2, 1\rangle\}\end{aligned}$$

For simplicities sake, we will use the operators' short name (e.g., drive1 instead of drive truck-1 city-loc-1 city-loc-2). Figure 4.1 shows the full state space of the task, where the node naming follows the same rules as in Example 2.4, so the sequence $2T$ denotes the state $\{\langle v_1, 2\rangle, \langle v_2, T\rangle\}$.

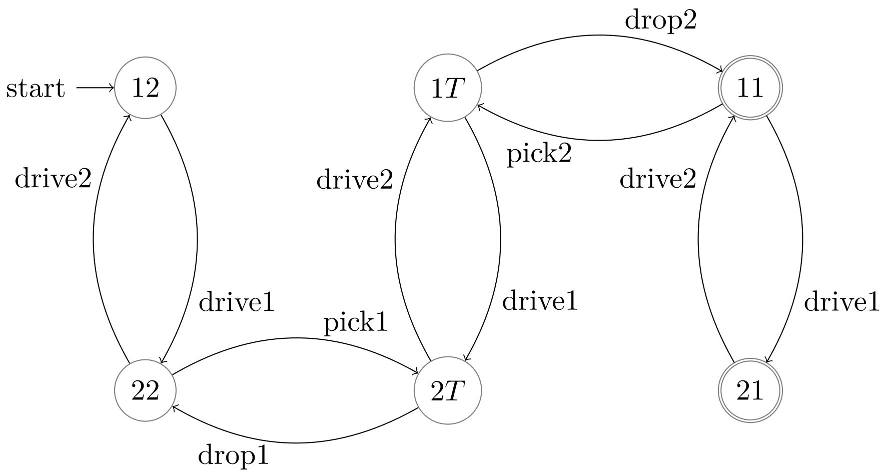

Figure 4.1: Simplified transport LTS.

4.2 Projection

For our projection, we keep the package only, i.e., the variable v_3. Thus, the truck and capacity do not exist anymore. So for example, the state $\{\langle v_1, 1\rangle, \langle v_2, 2\rangle\}$ will now be $\{\langle v_2, 2\rangle\}$. As for operators, drive1 have no preconditions or effects, so we can apply it anywhere, and it is just a self-loop. Analogously drive2 is a self-loop on any abstract state.

The probability of the drive1 operator will be 1/2 since it is applicable in the half of all states where v_1 equals 1. Similarly, the probability of derive2 is 1/2. Since drive1 and drive2 represent the same abstract transitions, we merge them into a single operator drive with probability 1 because the preconditions of drive1 and drive2 are incomparable; see Section 3.2.

The probability of pick1, drop1, pick2, and drop2 are 1/2. To see that consider, for example, the operator drop1. It is applicable in the abstract state XT. Its corresponding equivalence class is $\pi_U^{-1}(XT) = \{1T, 2T\}$. The original operator drop1 is applicable only in $2T$. Thus the probability is 1/2. The other probabilities can be computed analogously.

The resulting abstract LTS together with the operator probabilities is depicted in Figure 4.2.

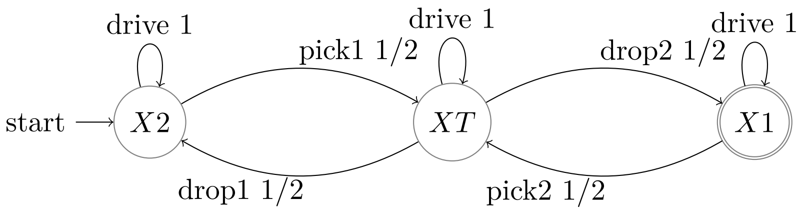

Figure 4.2: Projection of the transport graph

4.3 MDP

Before creating the MDP, we must create the shadow states. For each operator whose probability is not 1, we should make a shadow state that the unsuccessful operator goes to.[1] The shadow state have the same operators as the original state, except for the original operator. To improve clarity and make the graph more easily understandable, we abbreviate the names of the operators. So drop1 is denoted d1, pick1 is denoted p1, and analogously for pick2 and drop2. The names of the shadow states for an operator o are expanded by the name of o. For example, the shadow state expressing the unsuccessful application of p1 at $X2$ is denoted p1$X2$. The resulting MDP is shown in Figure 4.3.

[1]For the operators with probility 1, it makes no sense to introduce a shadow state because the probability of the transition into it would be 0.

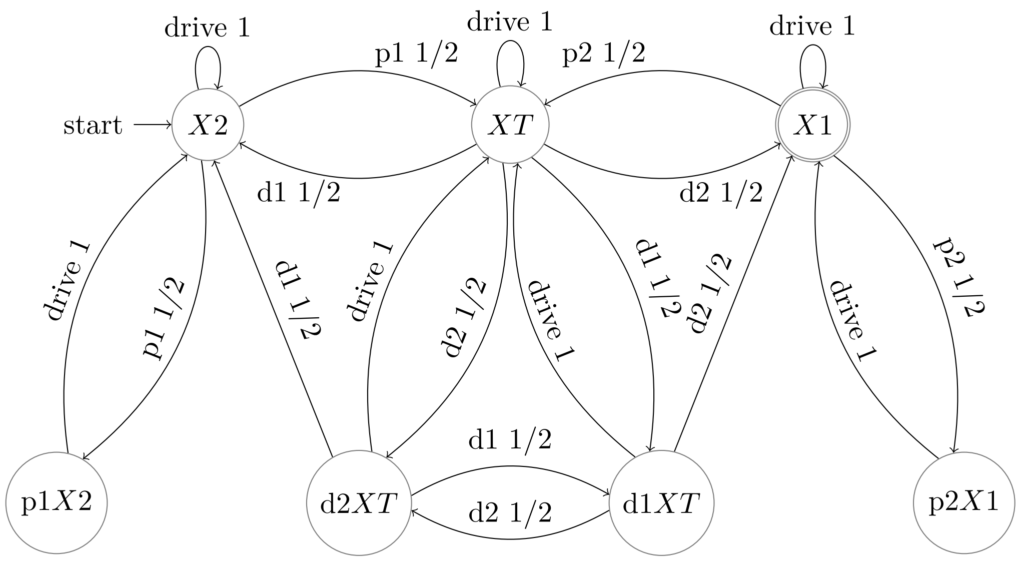

Figure 4.3: MDP of the transport task

4.4 Value Iteration

Now let us perform some steps of the value iteration algorithm to find the cost from the start to the goal state. Let us consider that $\gamma = 0.9$ and $\epsilon = 0.01$. Since our task contains seven states, we will demonstrate only 1 step on the starting state. First, we should initialize the value of each state to 0.

$$V(s) = 0, \forall s \in S \tag{4.1}$$

$$V(X2) = \max\left\{ \overbrace{\left(\frac{1}{2} \times -1 + \frac{1}{2} \times 0 + 0.9 \times \frac{1}{2} \times 0 + 0.9 \times \frac{1}{2} \times 0\right)}^{\text{p1}}, \right.$$

$$\left. \overbrace{-1 \times 1 + 0.9 \times 1 \times 0}^{\text{drive}} \right\} = -\frac{1}{2}$$

$$V(\text{p1}X2) = \max\left\{ \overbrace{-1 \times 1 + 0.9 \times 1 \times 0}^{\text{drive}} \right\} = -1$$

After the first iteration, the value function for our states will be $V(X2) = -\frac{1}{2}$ and $V(\text{p1}X2) = -1$.

Here is the second iteration for the previous states:

$$V(X2) = \max \left\{ \overbrace{\frac{1}{2} \times -1 + \frac{1}{2} \times 0 + 0.9 \times \frac{1}{2} \times -\frac{1}{2} + 0.9 \times \frac{1}{2} \times -1}^{\text{p1}} \right.$$

$$\left. \overbrace{-1 \times 1 + 0.9 \times 1 \times -\frac{1}{2}}^{\text{drive}} \right\} = -1.175$$

$$V(\text{p1}X2) = \max \left\{ \overbrace{-1 \times 1 + 0.9 \times 1 \times -\frac{1}{2}}^{\text{drive}} \right\} = -1.45$$

As we can see, the costs are gradually increasing, while the cost of the shadow state is higher because it is further away from the goal. The value iteration will continue until the difference between the previous values and the current is smaller than our defined ϵ or until the maximum amount of iterations is completed.

Here is how the value function will look like after 23 iterations:

$$\begin{aligned}
V(X2) &= -2.8026446283589297,\\
V(XT) &= -1.5964107708929134,\\
V(X1) &= 0.0,\\
V(\text{p1}X2) &= -3.5216789580317283,\\
V(\text{d2}XT) &= -2.4366569612259945,\\
V(\text{d1}XT) &= -1.5964107708929134,\\
V(\text{p2}X1) &= -1.0
\end{aligned}$$

The actual cost of the task is 5, while the predicted cost from the starting state to the goal state will be 2.8. The cost of the abstracted task would have been 2, which is smaller than using our method. We can also observe that the close a state is to the goal state, the lower the cost, proving that our predictions can be used as a heuristic.

Chapter 5

Experiments

In order to evaluate the effectiveness of our proposed method, we will conduct empirical studies using carefully designed experiments that will allow us to collect and analyze data on various relevant metrics, such as accuracy, efficiency, and scalability.

Specifically, we will select a range of benchmark problems that are representative of the types of real-world planning tasks that our method is intended to solve, and we will compare the performance of our approach against that of other state-of-the-art planning techniques. By conducting these experiments, we aim to provide empirical evidence that supports the effectiveness of our method and demonstrates its potential to improve state of the art in planning and decision-making in complex domains.

Additionally, we will examine the strengths and weaknesses of our approach in various scenarios and explore potential avenues for further improvement and development. Overall, our goal is to demonstrate our approach's practical utility and effectiveness in addressing real-world planning challenges and provide a basis for future research in this area.

5.1 Implementation details

Our implementation [7] is implemented in Python using Numpy for the MDP. A third-party library[3] has been used to solve the Value Iteration. Since the MDP library doesn't support MDPs where not all actions are applicable in a state, a workaround has been used, where if an action is not applicable, the cost is set to a large number so that the action is not picked during the value iteration.

All tests have been run on an i5-12600KF 3.70 GHz CPU with 32 GB DDR5 RAM.

5.2 Gamma importance

In almost all tests, the higher the discount factor γ, the closer to the actual cost was the approximation given by the value function. Figures 5.1 and

5.2 show two tasks with different projections, where we can see that the projection does not strongly affect the relationship between γ and the actual cost.

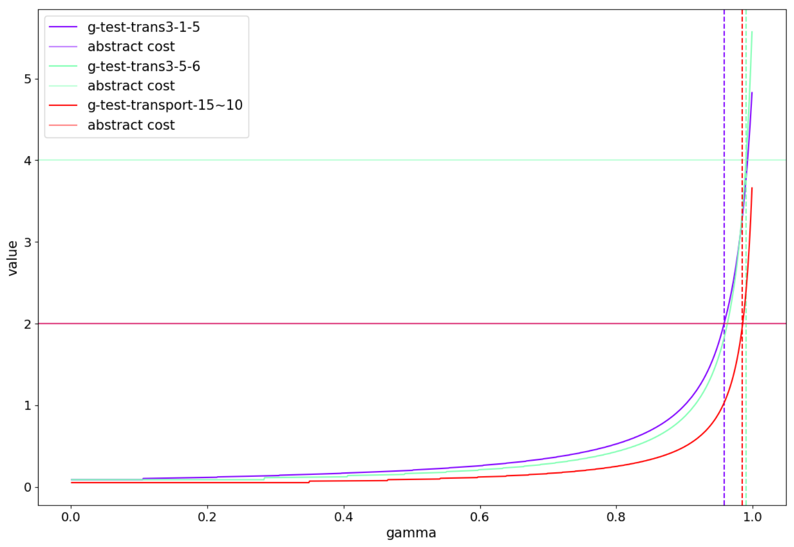

Figure 5.1: Gamma test on the transport domain from the IPC 2008

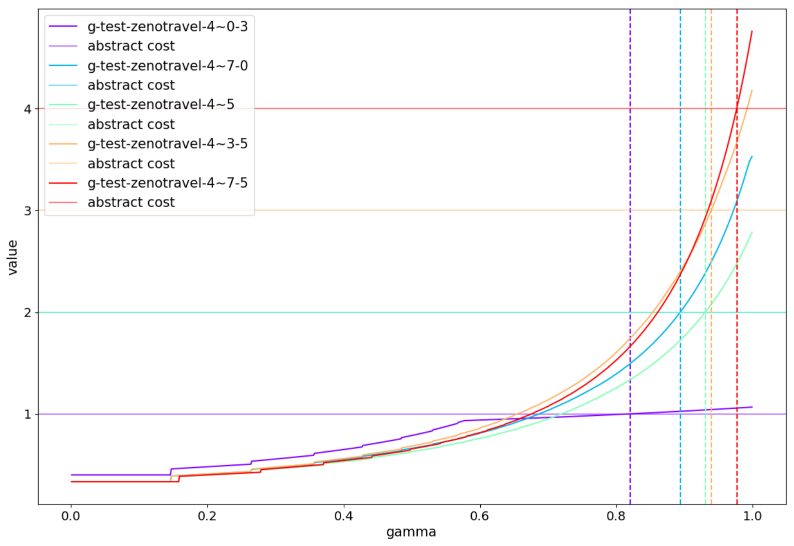

Figure 5.2: Gamma test on the zenotravel domain from the IPC 2002

Figures 5.1 and 5.2 show that having a larger γ close to 1 will always be better and that its predicted cost is always higher than the cost in the abstraction.

An interesting note is that γ can even be 1, which usually is not allowed in value iteration. In our case, we can have it 1 since it will never self-loop because the rewards are negative or zero. In the following sections, we will usually assume that $\gamma = 0.999$ if not stated otherwise.

5.3 Projection size and Speed/Precision

Our projection's size and variable selection directly affect our predictions' precision since we lose information by decreasing the projection's size. But in some cases, we can get the same or even better predictions with a smaller projection.

In Figure 5.3, we have picked an example where all projections have the same final value, yet the time needed for calculation differs quite a lot. Note that the time axis uses the log scale since the difference between times is so big.

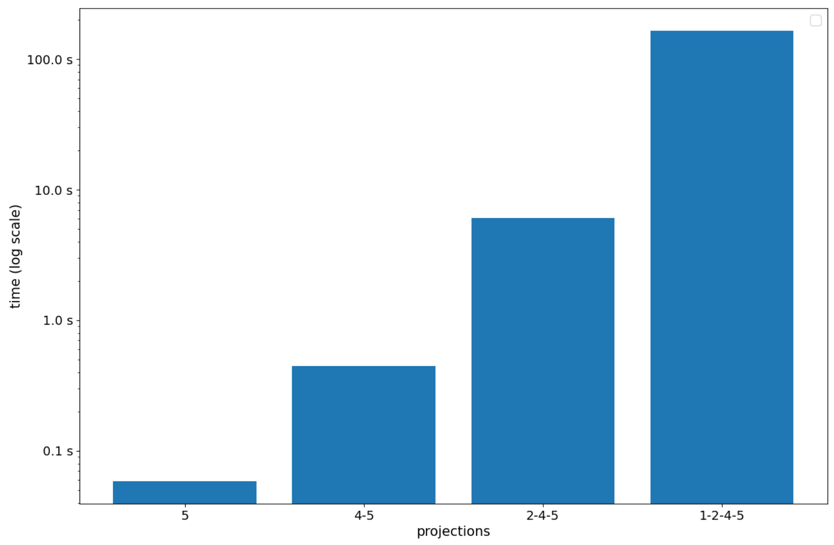

Figure 5.3: Zenotravel-4, value is -2.7848

In Figure 5.4, we can observe that increasing the projection can increase the precision, but that is not always the case. Since some variables do not play a big role in the final solution, there is no point in using them.

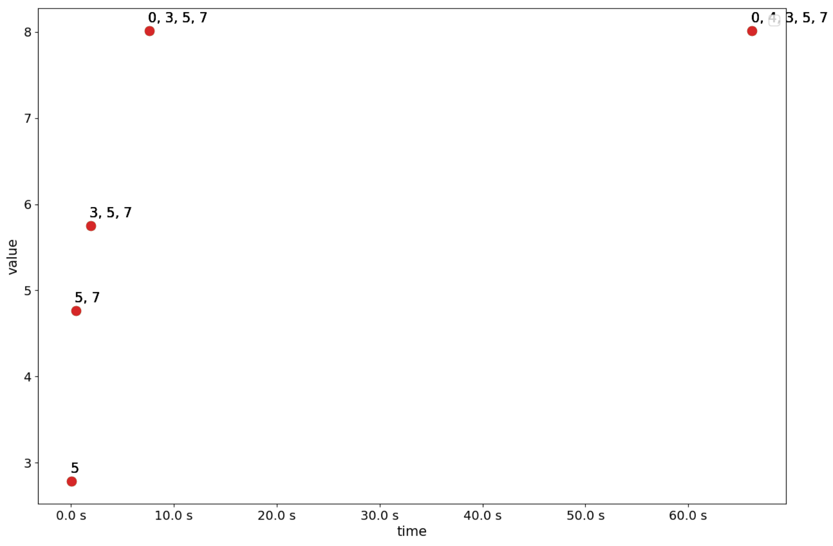

Figure 5.4: Zenotravel 4, speed/precision

Here are all possible combinations of all variables, with the maximum projection length being 4. Some jitter has been added to see the points. All points with 0 value have been removed.

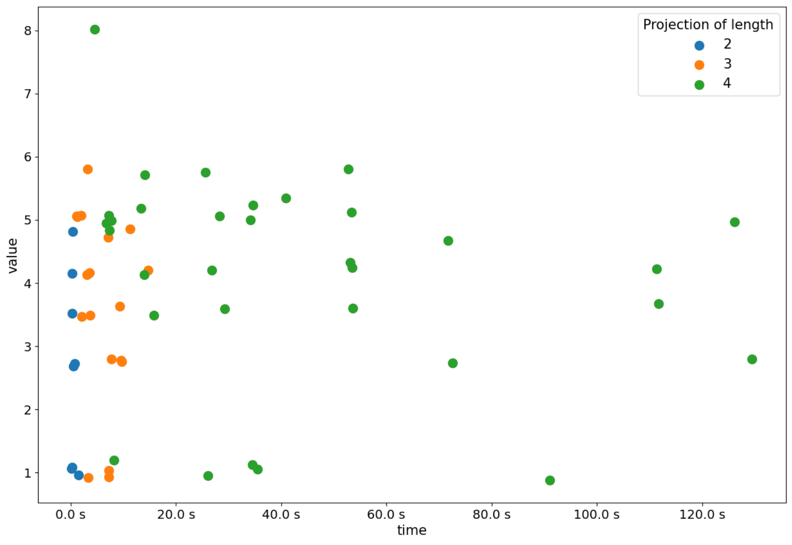

Figure 5.5: Zenotravel 4, 3 projection sizes

As you can see, there isn't even a need to have a big projection most of the time. The most important part is having the correct/important variables inside the projection to get a precise approximation. Having a lot of variables that are needed to satisfy the goal condition can significantly improve the precision.

5.4 Comparison to actual costs

For the following tests, the FastDownward planner has been used to find the actual cost of the solution, while our planner finds the cost in an abstracted state space and the predicted (expected) cost based on the value function. Twelve instances have been used from different domains from the IPC problem bank.

The results in Figure 5.6 are the comparison between the predicted costs and the actual costs. The projections had a constant size. Thus the predicted cost is generally the same. When comparing the results to Figure 5.7, we can observe that the predictions are much closer to the actual cost than the abstract costs. By increasing the projection size for bigger problems, we can further improve the precision. Figure 5.6 also demonstrates that the heuristic based on predicted costs is not admissible, in general.

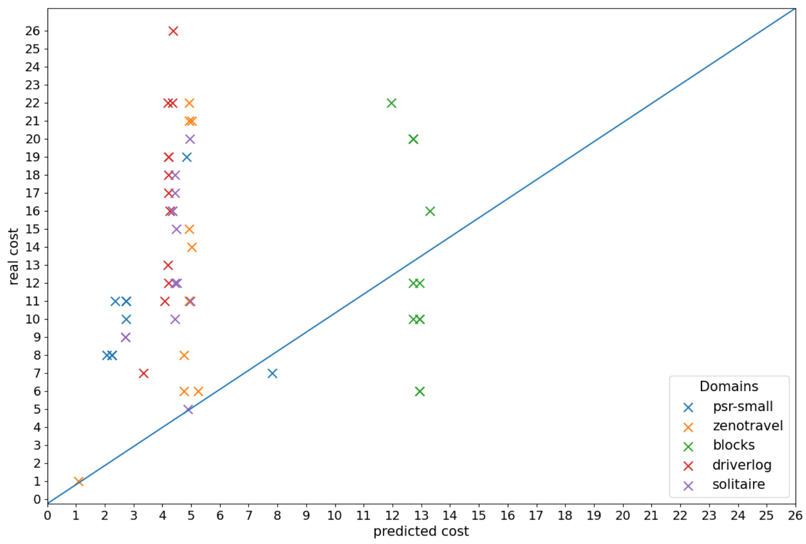

Figure 5.6: Real/Predicted costs

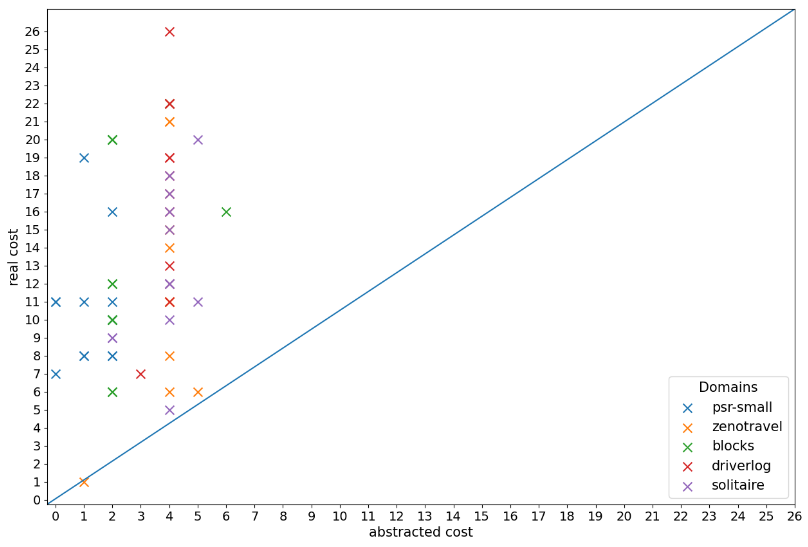

Figure 5.7: Real/Abstracted costs

In Figure 5.8, we compare the predicted and abstract costs. We can see that the predicted costs are always either higher or the same, so it directly improves upon using only abstractions and proves our point on real data.

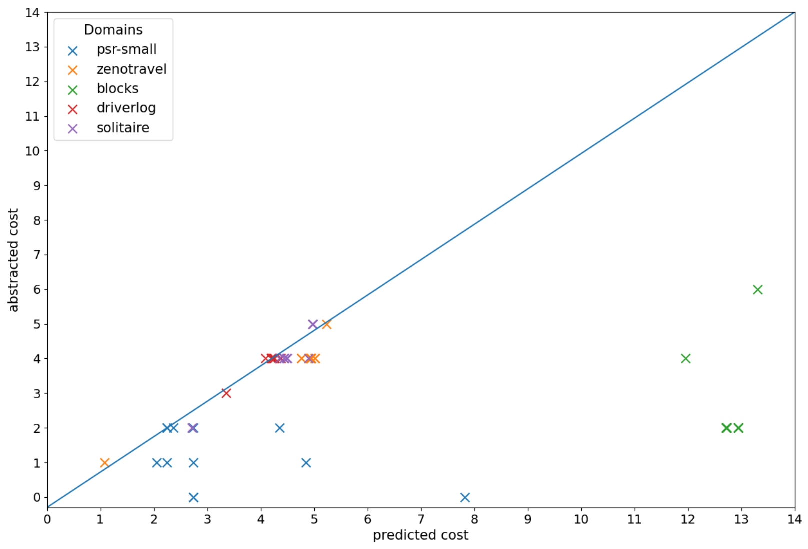

Figure 5.8: Abstracted/Predicted costs

Chapter 6

Conclusion

In summary, our method offers several important advantages compared to traditional abstraction processes. Instead of throwing away information, we use it by including transition probabilities, resulting in a significant improvement in precision compared to the usual raw abstraction technique.

Moreover, there are possibilities for further improvements, like using smart methods to choose the best projection based on specific problem characteristics. By tailoring the projection to each problem, we can enhance the performance and effectiveness of our method, as finding a suitable projection currently takes a lot of time.

Most importantly, our method allows for efficient solutions to complex problems that would otherwise require extensive computing power and time. This highlights its potential for practical applications and advancements in related fields.

Future research can focus on enhancing the implementation by rewriting it in a faster programming language like Julia.

By connecting statistical information with abstraction, our research provides valuable insights and techniques to the field. Our findings lay a strong foundation for further exploration in this direction.

In conclusion, our method offers a promising approach for more accurate and efficient problem-solving.

Bibliography

[1] Christer Bäckström and Bernhard Nebel. Complexity results for SAS$^+$ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[2] Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.

[3] Steven Cordwell and hiive. Markov decision process (mdp) toolbox for python. `https://github.com/hiive/hiivemdptoolbox`, 2022.

[4] Stefan Edelkamp. Planning with pattern databases. In A. Cesta and D. Borrajo, editors, *Proceedings of the 6th European Conference on Planning (ECP'01)*, pages 13–24, 2001.

[5] Richard E. Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. 2:189–208, 1971.

[6] Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM*, 61(3):16.1–16.63, 2014.

[7] Vasily Levitskiy and Rostislav Horčík. Heuristics finder using projections and statistical information for planning tasks. `https://github.com/HollyAssasin/MdpProjectionPlanner`, 2023.

[8] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (Third Edition)*. 2010.

[9] Josh Tenenberg. Planning with abstraction. In *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, USA, August 11-15, 1986. Volume 1: Science*, pages 76–80, 1986.