Modelling and Solving Industrial Production Tasks as Planning-Scheduling Tasks*

Andrii Nyporko[a,b], Lukáš Chrpa[b,*]

[a]*Faculty of Electrical Engineering, Czech Technical University in Prague, Jugoslávských partyzánů 1580/3, Prague, Czechia,*
[b]*Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, Jugoslávských partyzánů 1580/3, Prague, Czechia,*

Abstract

Industrial production planning or manufacturing concerns the selection of activities that can produce a desired product and scheduling them on resources that perform these activities. To deal with such problems techniques in the fields of *Automated Planning* and *Scheduling* might be leveraged, which are usually pursued separately even though they are (very) complementary. In manufacturing, the activities represent elementary steps in the production and each activity requires a specific input in order to produce a desired output. From that perspective, activities resemble actions in planning as they can capture such a requirement. Selecting proper activities including their (partial) ordering can be understood as a planning task while allocating the activities to the resources can be understood as a scheduling task.

This paper formalizes the concept of "combined" planning and scheduling tasks by defining *planning-scheduling tasks* that are suitable for problems concerning industrial production or manufacturing. In particular, we define two types of activities – *production* and *maintenance* activities – where the former describes elementary production tasks while the latter modifies attributes of the resources (e.g. changing the configuration of reconfigurable machines). We introduce an extension of Planning Domain Definition Language (PDDL), a well-known language for describing planning tasks, to sup-

*Cite as: Andrii Nyporko, Lukás Chrpa: Modelling and solving industrial production tasks as planning-scheduling tasks. Data Knowledge Engineering 157: 102415 (2025)

*Corresponding Author

Email addresses: nyporand@cvut.cz (Andrii Nyporko), chrpaluk@cvut.cz (Lukáš Chrpa)

port modelling of planning-scheduling tasks. To tackle planning-scheduling tasks we propose two compilation schemes, one into temporal planning (in PDDL 2.1) and one into classical planning. We evaluated our approaches in three use cases of industrial production planning – Reconfigurable Machines, Woodworking, and Tube Factory domains. The results showed that solving planning-scheduling tasks by compiling them into planning tasks in order to use off-the-shelf planning engines is suitable as it scales reasonably well with the size of the actual tasks (although the resulting solutions are suboptimal).

Keywords: Planning-scheduling Tasks, Industrial Production Planning, Planning, Scheduling, Compilation

---

1. Introduction

In the context of industrial production planning or manufacturing, one can specify a set of production activities that, generally speaking, describe elementary means to produce (mid)products. These production activities require input material or (mid)products to produce other (mid)products. That behavior resembles actions in planning and it is often the case that a subset of (partially) ordered production activities is required to achieve the final product(s). These activities have to be executed on resources (e.g. machines) that might need to be configured for these activities (jobs) which is the domain of scheduling.

To address the problems concerning industrial production planning or manufacturing, one can leverage techniques in the fields of *Automated Planning* and *Scheduling*, which despite being complementary are usually pursued separately. In a nutshell, Automated Planning deals with finding sequences of actions to achieve a required goal. Scheduling, on the other hand, deals with the problem of allocating activities (or jobs) to (limited) resources.

In scheduling, it is usually the case that activities (or jobs), as well as their (partial) ordering, have to be explicitly specified upfront. Scheduling with alternatives that allows choosing which activities to schedule has been studied in recent past [1, 2], however, still requires an explicit specification of alternatives and ordering of activities that, on the other hand, is inherent to planning.

In planning, some domains that describe manufacturing process planning have been designed for International Planning Competitions (IPC). For

instance, the Schedule domain was introduced in IPC 2000 and the Woodworking domain was introduced in IPC 2008 (the latter domain is elaborated in detail later in the paper). These domains give an intuition that automated planning can be leveraged to solve production planning (or manufacturing) problems, however, there does not exist a principled approach to how such domains can be modelled.

This paper formalises the concept of "combined" planning and scheduling tasks into *planning-scheduling* tasks, introduced by Nyporko and Chrpa [3], that are suitable for addressing problems concerning industrial production planning or manufacturing. In planning-scheduling tasks, we introduce two types of activities – production and maintenance activities – that can be represented similarly to actions in planning that are then scheduled on (limited) resources. Note that in contrast to Nyporko and Chrpa, we consider "lifted" representation for planning-scheduling tasks. Furthermore, we propose an extension of Planning Domain Definition Language (PDDL), which is a well-known language for modelling planning tasks [4], that can model introduced planning-scheduling tasks. The idea of extending PDDL follows the recent work of Long et al. [5]. In contrast to Long et al., we consider a richer formalism for planning-scheduling tasks. To solve planning-scheduling tasks, we propose two compilation approaches – the first for compiling planning-scheduling tasks into classical planning tasks and the second for compiling them into temporal planning tasks. Such compilations are designed to be compatible with the PDDL (and PDDL2.1) language [4] in order to leverage off-the-shelf planning engines. We have specified three domains that belong to the category of industrial production planning. namely Reconfigurable Machines [6, 7], Tube Factory [8] and Woodworking (taken from the International Planning Competition), on which we show that our compilation approaches scale reasonably well with the size of the tasks, albeit at the cost of being suboptimal.

2. Related Work

Even though planning and scheduling are traditionally being developed separately, the idea of combining these two areas is not very new as, for instance, demonstrated by the work of Dean et al. [9] that extended hierarchical planning with "scheduling aspects" such as resources and deadlines. As argued by Tan and Khoshnevis [10] it can be very beneficial to integrate planning and scheduling into a single reasoning process.

Space and planet exploration has provided a lot of challenging problems that were addressed by approaches combining planning and scheduling. Inspired by his recent work on transportation planning [11], Muscettola has proposed the HSTS system that was used for scheduling for Hubble Space Telescope [12]. HSTS has inspired the EUROPA system that was used for Deep Space One mission [13] or Mars Rover mission [14].

IPSS [15] is another example of a system integrating planning and scheduling that can manage more complex temporal constraints (Allen primitives) as well as resource usage. Phanden et al. [16] also proposed a hybrid modular system that integrates scheduling and planning in the form of several independent modules (plan selection, scheduling, schedule analysis, and process plan modification modules). Ruml et al. [17] proposed a scheduling extension to planning that aimed at (near) real-time online planning and scheduling.

From more scheduling point of view, Laborie [18] introduced resource and capacity constraints into planning and scheduling that are part of the IBM ILOG CP solver [19]. Banerjee [20] also considered Constraint Modelling for dealing with problems with both planning and scheduling aspects. Scheduling with alternatives that allows choosing which activities to schedule considers aspects of planning in the form of explicitly specified alternatives of ordered activities [1, 2].

One of the most recent approaches [5] proposes a scheduling extension to PDDL (similarly to us) and provides a tool for modelling and solving hybrid problems from planning and scheduling. In contrast to that work, our extension allows a richer representation of activities and states of resources. On top of that, we provide a formal definition of *planning-scheduling* tasks and provide compilations to classical and temporal fragments of PDDL in order to leverage existing planning engines.

3. Preliminaries

In this section, we introduce the terminology from planning and scheduling that will be used in the paper.

3.1. Classical Planning

In *classical planning*, a static, deterministic, and fully observable environment in which actions have immediate effects is considered. In a nutshell, the task is to find a plan, a sequence of actions, that transforms the environment from a given initial state to a state that satisfies the goal condition.

In the (lifted) STRIPS representation, the environment is specified via a set of first-order logic *predicates* P, where a predicate $p = pred\_name(x_1, \ldots, x_n)$ consists of *pred_name* being a unique predicate name and $x_1, \ldots x_n$ being variable symbols. Grounded predicates, whose variable symbols are substituted with constants - problem-specific objects, are called *atoms*. Then a *state* of the environment s is a subset of atoms, meaning that all atoms the state contains are considered true while the atoms not contained in the state are considered false.

An *operator* o is a quadruple $o = (name(o), pre(o), del(o), add(o))$, 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 operators.[1] An action a is *applicable* in a state s if and only if $pre(a) \subseteq s$, and the application of a (if possible) *results* in a state $s' = (s \setminus del(a)) \cup add(a)$.

We define a (lifted) *classical planning task* as a tuple $\Pi = (P, O, C, I, G)$, where P is a set of predicates, O is set a of operators, C is a set of (task-specific) constants, I is an initial state and G is a set of goal atoms. A *plan* π for a classical planning task Π is a sequence of actions, grounded instances of operators from O, $\pi = \langle a_1, \ldots, a_n \rangle$ such that $a_1, \ldots, a_n$ are consecutively applied (it must be possible), starting in the initial state I and resulting in a goal state s_g ($G \subseteq s_g$).

3.2. Temporal Planning

In contrast to classical (STRIPS) planning, *temporal planning* explicitly reason with time, and hence action execution takes time and actions have to be scheduled to be executed (rather than being ordered in sequences). Here we consider the PDDL2.1 semantics for representing temporal planning tasks [4].

A *durative operator* is a tuple $o = (name(o),\ dur(o),\ pre^s(o),\ pre^o(o),\ pre^e(o),\ del^s(o),\ del^e(o),\ add^s(o),\ add^e(o))$, 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), $dur(o)$ represents

[1] Note that in literature, the notions "action" and "operator" are used interchangeably. Here, for the sake of clarity, we use the term "operator" for lifted actions.

o's *duration*, $pre^s(o), pre^o(o), pre^e(o)$ are sets of predicates representing o's *at-start, overall, at-end precondition*, respectively, $del^s(o), del^e(o)$ are sets of predicates representing o's *at-start, at-end delete effects*, respectively, and $add^s(o), add^e(o)$ are sets of predicates representing o's *at-start, at-end add effects*, respectively. Similarly, to classical planning, *durative actions* are grounded instances of durative operators.

In temporal planning, states evolve in time (rather than in steps between actions as in classical planning). States are also modified by action application such that an applied action creates two events - one shortly after the action starts and one when it finishes. Let $s(t)$ denote a state s in time t. If a durative action a is applied in t, then it modifies the state as $s(t+\delta) = (s(t) \setminus del^s(a)) \cup add^s(a)$ and $s(t + dur(a)) = (s(t + dur(a) - \delta) \setminus del^e(a)) \cup add^e(a)$. Note that δ is a small positive number. With regards applicability, a can be applied in t if and only if $pre^s(a) \subseteq s(t), pre^e(a) \subseteq s(t + dur(a) - \delta)$ and $\forall t' \in [t, t + dur(a)] : pre^o(a) \subseteq s(t')$.

We define a (lifted) *temporal planning task* as a tuple $\Pi = (P, O, C, I, G)$, where P is a set of predicates, O is set a of durative operators, C is a set of (task-specific) constants, I is an initial state and G is a set of goal atoms. A *plan* π is a set of pairs (action,time) $\pi = \langle (a_1, t_1), \ldots, (a_n, t_n) \rangle$ such that starting in the initial state $s(0) = I$ each (durative) action a_i is applicable and applied in t_i and the state achieved after the last event is a goal state (contains all propositions from G).

3.3. Job-shop Scheduling

The *job-shop scheduling* aims at allocating jobs (or activities) on resources (or machines) such that all constraints are met while optimising a given objective function (usually makespan, i.e., the time needed to complete all the jobs).

Technically, a *job-shop scheduling task* consists of a set of *jobs* $J = \{J_1, \ldots, J_n\}$, where each job J_i is composed by a set of *operations* (or *activities*) O_i that have to be run in a given order, and a set of *resources* (or machines) $R = \{R_1, \ldots, R_k\}$ on which activities are scheduled. Each activity (operation) has a defined duration to complete and might be scheduled only on a specific resource (or a subset of resources).

A *schedule* is a set of triples (time,activity,resource) that for each activity determines its start time and resource on which it is run. A schedule that is a *solution* of a job-shop scheduling task schedules all the activities such that each activity is scheduled on a required resource, all preceding activities

finished before the activity starts, and at most one activity is scheduled on an individual resource at the same time.

4. Planning-Scheduling Task Specification

Industrial production, in a nutshell, involves both planning and scheduling aspects since it might be needed to select (and order) activities that have to be performed to get the desired product, and these activities need to be allocated to resources (e.g. machines, workers) that are responsible for completing them. The planning part describes the mechanics of production, that is, what elementary activities require, what they consume, and what they produce. The scheduling part is responsible for allocating these activities to resources that are capable of completing them.

We distinguish between "planning" predicates (P) that describe the environment such as available material or (semi)products and "attribute" predicates (L) that describe states relevant to particular resources (e.g. configuration of a machine) and between "static" predicates (S) that describe persistent relations between objects (e.g. a certain product requires certain configuration). Then, we define two types of activities – *production* and *maintenance* activities – that can be understood as "jobs" in scheduling as well as they resemble actions in planning since they modify the environment (production activities) or the attributes of resources (maintenance activities).

Definition 1. *A production activity is a* 7*-tuple* $a = (name(a),\ dur(a),\ ATTR(a),\ STAT(a),\ pre(a),\ del(a),\ add(a))$*, where* $name(a) = act\_name(x_1, \ldots, x_k)$ *(act_name is an unique activity name and* $x_1, \ldots x_k$ *are variable symbols (arguments) appearing in the activity), dur(a) represents the duration of* a*,* $ATTR(a)$ *is a non-empty set of sets of attributes from* L *that* a *requires,* $STAT(a)$ *is a set of predicates from* S*, which specify permitted relations between objects, and* $pre(a)$*,* $del(a)$*,* $add(a)$ *are sets of predicates from* P *representing* a*'s precondition, delete and add effects, respectively.*

For describing production activities, we consider semantics that takes into account temporal aspects (in contrast to classical planning) while not being as expressive as the temporal (PDDL2.1) semantics. In particular, delete effects are considered "at-start", add effects "at-end", $pre(a) \cap del(a)$ "at-start" and $pre(a) \setminus del(a)$ "overall". The rationale behind the simplified semantics is that production activities consume required material (which

can be assumed that it happens shortly after a production activity starts) in order to produce some product (which happens when a production activity finishes) while items that are required but not deleted should be available during the run of a production activity. The cardinality of the $ATTR(a)$ set determines how many resources the production activity a requires (since $ATTR(a)$ is non-empty, a requires at least one resource).

Definition 2. *A maintenance activity is a* 7*-tuple* $m = (name(m),\ dur(m),\ res(m),\ ATTR(m),\ STAT(m),\ rem(m),\ add(m))$*, where* $name(m) = act\_name(x_1, \ldots, x_k)$ *(act_name is an unique activity name and* $x_1, \ldots x_k$ *are variable symbols (arguments) appearing in the activity),* $dur(m)$ *represents the duration of* m*,* $res(m) \in R$ *represents a resource on which* m *is performed,* $ATTR(m)$ *is a set of sets of attributes from* L *that* m *requires (might be empty),* $STAT(m)$ *is a set of predicates from* S*, which specify permitted relations between objects,* $rem(m) \subseteq L$ *and* $add(m) \subseteq L$ *are sets of attributes associated with* $res(m)$ *or global that will be removed or added, respectively.*

We assume that before m is applied, all attributes from $rem(m)$ must hold. The rationale behind this rule is to capture how exactly the attributes (global or of a specific resource) are modified (e.g. for reconfigurable machines it might take a different time to change from one configuration to another). Similarly to the production activity case, the cardinality of the $ATTR(m)$ set determines how many additional resources (different from the maintained resource) the maintenance activity m requires (here $ATTR(m)$ might be empty).

Both production and maintenance activities can be grounded in a similar way as operators in planning. Static predicates $STAT(\cdot)$ determine permissible combinations of arguments in grounded instances (e.g. a required machine configuration for a given type of product). With regards to grounded "attribute" predicates, we distinguish between global ones (e.g. availability of power supply), and resource-specific ones (e.g. a machine mode), i.e., with a set of resources $\{r_1, \ldots, r_k\}$ we split the set of all grounded "attribute" predicates from L, denoted as $attr$ into pairwise disjoint sets $attr(r_1), \ldots, attr(r_k), attr(*)$ (the latest denotes global attributes).

Definition 3. *A planning-scheduling task (or PS task, for short) as a tuple* $\Pi = (P, L, S, A, R, M, C, I, I_L, I_S, G)$*, where* P *is a set of predicates describing the environment,* L *is a set of resource attributes,* S *is a set of static*

predicates, A is a set of production activities, R is a set of resources, M is a set of maintenance activities, C is a set of task-specific constants (or objects), I is an initial state (over instances of P), I_L are initial attributes (over instances of L), I_S are initial instances of static predicates (from S), and G is a goal (over instances of P).

PS tasks, in a nutshell, deal with selecting and scheduling activities (both production and maintenance) on resources such that the goal is achieved and all the constraints are met. Besides satisfying preconditions for all planned activities, it has to be taken into account that at most one activity can be running on a single resource at the same time, and the activities whose application time intervals overlap should not interfere with each other (by deleting a proposition that the other activity requires or produces). The following definition formally introduces the concepts behind how *solutions* of PS tasks look like. In the following definitions, we consider a PS task $\Pi = (P, L, S, A, R, M, C, I, I_L, I_S, G)$.

Definition 4. *Let $\pi = \{(t_1, x_1, R_1), \ldots, (t_n, x_n, R_n)\}$ be a set of triples (timestamp,grounded activity,set of resources), where $x_i \in A \cup M$ and $R_i \subseteq R$ $(1 \leq i \leq n)$. We say that π is consistent if for each x_i, x_j $(i \neq j)$ it is the case that if the time intervals $[t_i, t_i + dur(x_i)]$ and $[t_j, t_j + dur(x_j)]$ are not disjoint, then $R_i \cap R_j = \emptyset$ and if $x_i, x_j \in A$, then $del(x_i) \cap (pre(x_j) \cup add(x_j)) = \emptyset$ and $del(x_j) \cap (pre(x_i) \cup add(x_i)) = \emptyset$.*

Definition 5. *The state trajectory respecting I and π, which must be consistent, is a mapping from non-negative time to sets of states (over instances of P). Initially, $s(0) = I$ and for each production activity $a \in A$ being scheduled at time t in π the state changes in $t + \delta$ (where δ is a small non-negative number) such that $s(t + \delta) = s(t) \setminus del(a)$ and in $t + dur(x)$ such that $s(t + dur(x)) = s(t + dur(x) - \delta) \cup add(a)$.*

Analogously, for each resource $r \in R$ we define a configuration trajectory as a mapping from non-negative time to sets of attributes (from instances of L). The initial attributes are $attr(0) = I_L$ and for each maintenance activity $x \in M$ associated with r and scheduled in time t the set of attributes changes such that $attr(t + \delta) = attr(t) \setminus rem(x)$ and $attr(t + dur(x)) = attr(t + dur(x) - \delta) \cup add(x)$.

Definition 6. *For an activity $x \in A \cup M$ (either production or maintenance) scheduled in time t in π, we say that a set of resources $R_x \subseteq R$ is admissible*

for x in t, if there exists a bijective mapping $\rho_x : ATTR(x) \rightarrow R_x$ such that for each $q \in ATTR(x)$ it is the case that $q \subseteq attr(\rho_x(q))$. We say that a maintenance activity $m \in M$ is applicable on a resource r in time t iff $STAT(m) \subseteq I_S$, $rem(m) \subseteq attr(r)(t) \cup attr(*)(t)$ and there exists a set of resources R_x that is admissible for m in t and $r \notin R_x$. We also say that a production activity $a \in A$ is applicable in time t iff $STAT(m) \subseteq I_S$, $pre(a) \subseteq s(t)$ and there exists a set of resources R_x that is admissible for m in t.

Definition 7. *We say that π is a solution of a PS-task P iff π is consistent, for each $(t_i, x_i, R_i) \in \pi$ it is the case that x_i is applicable in t_i and $R_i \subseteq R$ is admissible for x_i in t_i, and $G \subseteq s(\max_{i=1}^{n}(t_i + dur(x_i)))$.*

In this paper, we consider two objective functions for optimising PS-task solutions. One objective function minimises *makespan*, i.e., a time needed to achieve the goal, that is, $\max_{i=1}^{n}(t_i + dur(x_i))$. The other objective function we consider minimises the total cost of activities present in a PS-task solution. We introduce a cost function $cost : (A \cup M) \rightarrow \mathbb{R}_0^+$ that assigns a non-negative cost to each activity (both production and maintenance). Then, $\sum_{i=1}^{n} cost(x_i)$ is minimised. We would like to note that objective functions that can be formulated in PDDL 2.1 can be straightforwardly used for PS tasks as well as the PDDL extension for PS tasks is derived from PDDL 2.1. (see Section 6).

4.1. PS tasks vs Planning

PS tasks are more expressive than job-shop scheduling tasks, including variants that support configuring resources [6], since PS tasks implicitly support selecting appropriate production activities for achieving given goals (or products). PS tasks are also more expressive than classical planning tasks. We can translate classical planning operators into production activities as the name, preconditions, and add and delete effects can correspond to each other. Duration of production activity can be set to some value (e.g. 1) and we can specify a "dummy" attribute of a (single) resource that is always true. Hence the solution of such a PS-task would correspond to a sequential plan that is a solution of the classical planning task. Of course, PS tasks consider temporal aspects that classical planning tasks do not. With regards to temporal planning, PS tasks are, in contrast, less expressive than temporal planning tasks (in the PDDL 2.1 semantics) as they, for instance, do not support "at start" add effects.

Even though it might be argued that PS tasks can be directly encoded as temporal planning tasks, PS tasks, besides distinguishing between "planning" and "scheduling" parts in their design, do not require concurrency in terms of that one activity has to run in parallel with some other activity. Note that "required concurrency" does not prohibit solutions with concurrent activities, on the other hand, it allows to generate totally-ordered sequences of activities that are later rescheduled. Hence PS-tasks can be tackled by classical planners (as we show in this paper) or by temporal planners that do not support required concurrency, which are often more efficient [21, 22].

5. Case Studies

In this section, we describe three use cases that can be modelled as PS tasks - Reconfigurable Machines, Woodworking, and Tube Factory domains.

5.1. Reconfigurable Machines Domain

As one of the case studies, we had considered production planning on *Reconfigurable Machines* that are becoming more widespread nowadays [6]. In a nutshell, each production activity requires a specific configuration that can be changed by maintenance activities.

In our terminology, reconfigurable machines are resources with their possible configurations as their attributes. The maintenance activities are responsible for changing configurations on machines (if such configurations are supported) that are required by production activities that have to be scheduled.

Each production activity produces a specific product, which is its add effect. Some production activities require one product (in their preconditions) that has been produced by a different production activity (represented as a static attribute of the particular object). Each production activity requires a specific configuration, which is a resource attribute, and needs to be scheduled on a single resource (having the required attribute set).

It should be noted that duration of activities depends on the configuration it is run (in case of production activities) or from which to what configuration a machine is changed (in case of maintenance activities). The similar applies to the cost of the activities (both production and maintenance).

5.2. Woodworking Domain

Woodworking domain is one of the classical planning benchmarks that was introduced in the IPC 2008. Woodworking domain describes a production planning problem in a carpenter's workshop in which the task is to produce a collection of wooden parts from different types of wood, sizes, and kinds of treatment.

The domain considers several different types of resources (resource type can be understood as a resource attribute, in our terminology), namely – saw, highspeed saw, glazer, planer, immersion varnisher, spray varnisher, and grinder. Highspeed saw has additional attributes referring to whether a given board of wood is loaded into it, or whether the saw is empty. Glazers and both types of varnishers have to have a specific colour to paint wooden parts, which is represented by (static) resource attributes.

Two types of maintenance activities are considered, specifically loading a wooden board into a highspeed saw (must be empty beforehand) and unloading the wooden board from the highspeed saw.

Production activities, in a nutshell, describe how a given part of wood can be produced and treated. In particular, wooden boards can be cut by either a (normal) saw or by highspeed saw (the wooden board must be loaded into it) in order to produce wooden parts (of different sizes). To make the surface of the wooden part smoother, one can use a grinder (if the surface is rough) or a planer (if the surface is smooth to make it very smooth). A glazer is used to glaze wooden parts (that are untreated) and to paint them by the colour that the glazer has. Both types of varnishers also paint (untreated) wooden parts with a colour that the given varnisher has and, in contrast to glazing, make the wooden parts varnished.

Durations and cost of activities differ by type of activity (e.g. what resource is used, how large a wooden part is). Each activity requires one resource to be run on.

5.3. Tube Factory Domain

Another use case we had considered concerns the production of Tubes, described in [8]. It is about optimisation of the production of plastic tubes in a factory while having different options of how they can be produced. The goal is to produce a certain amount of tubes, where each tube can be produced automatically (on a machine), semiautomatically (by a human server operating a semiautomated machine), or manually (by a human server).

Each machine has to be set up for a specific task that is represented by machine attributes. Also, a fully automatic machine, a semi-automatic machine, and a (human) server are types of resources. These resource types can be represented by (static) attributes.

Maintenance activities are changing the machine setup for required tasks. To do so, a (human) server is required. Hence each maintenance activity besides the machine it serves "consumes" a (human) server, who has to do the job.

Production activities are responsible for producing tubes. A production activity might run on a fully automated machine (if properly set up), on a semiautomated machine (if properly set up) with a human server, or executed solely by a human server.

How production activity is performed affects its duration and cost. In contrast to the other use cases, here the production activities are not interdependent (i.e., none of the production activities requires an output from some other production activity).

6. Representing PS Tasks: PDDL Extension

The high-level idea of how to represent a PS task is to extend the "planning" environment with scheduling-specific elements such as resources and attributes. In particular, we extend the current PDDL specification (we consider the PDDL 2.1 version [4]) with the elements (such as production and maintenance activities) that are introduced by PS tasks. The EBNF (Extended Backus–Naur Form) description of the proposed language is in the following subsections. Note that we also consider "types", which is a feature of PDDL, that categorize objects into (hierarchical) classes. Types, in a nutshell, restrict what objects can be substituted for free variables while grounding.

6.1. Domain Specification

The domain part consists of, in a nutshell, the specification of the environment, attributes, and activities. The formal description of the grammar of our extension of the PDDL language follows. Note that expressions in square brackets are optional, i.e., such an expression either appears once or

not at all:

$$\begin{aligned}\langle \textit{domain} \rangle \ := \ & (\textit{define} \ (\textit{domain} \ \langle \textit{name} \rangle) \\ & \langle \textit{require-definition} \rangle \\ & \langle \textit{types-definition} \rangle \\ & [\langle \textit{constants-definition} \rangle] \\ & \langle \textit{attributes-definition} \rangle \\ & \langle \textit{predicates-definition} \rangle \\ & [\langle \textit{functions-definition} \rangle] \\ & \langle \textit{statics-definition} \rangle \\ & \langle \textit{structure-definition} \rangle)\end{aligned}$$

The block – $\langle \textit{require-definition} \rangle \ := \ (\textit{:require} \ \langle \textit{require-key} \rangle +)$ – consists of the requirements needed to express the domain (at least one). On top of (standard) PDDL requirements, we specify the requirement – *:ps-task* – for our PS task specification. For example, we can specify (*:requirements :strips :fluents :ps-task*).

The block – $\langle \textit{type-definition} \rangle \ := \ (\textit{:types} \ \langle \textit{typedlist}(\textit{name}) \rangle +)$ – contains information about the type hierarchy of objects (at least one). In PS tasks, we define three base types, *object*, *resource*, and *attribute*, from which the other types are inherited.

The rule $\langle \textit{typedlist}(x) \rangle \ := \ x+ \ - \ \langle \textit{name} \rangle$ is used to declare type for a list of entities x. Multiple entities could be assigned to one type simultaneously.

The block – $\langle \textit{constants-definition} \rangle \ := \ (\textit{:constants} \ \langle \textit{typedlist}(\textit{name}) \rangle +)$ – contains information about domain-specific constants, i.e., all tasks in that domain have these constants.

The following blocks declare (environment) predicates (P), attribute predicates (L), and static predicates (S). On top of that, "functions" that have the same syntax as predicates while representing a numeric value con be declared. Such value can be declared in the task description (but cannot be changed by any activity) and can be used to represent activity duration or cost.

⟨*attributes-definition*⟩ := (*:attributes* ⟨*atomic formula*(*variable*)⟩)
⟨*predicates-definition*⟩ := (*:predictates* ⟨*atomic formula*(*variable*)⟩)
⟨*functions-definition*⟩ := (*:functions* ⟨*atomic formula*(*variable*)⟩)
⟨*statics-definition*⟩ := (*:static* ⟨*atomic formula*(*variable*)⟩)
⟨*atomic formula*(*x*)⟩ := (⟨*name*⟩ ⟨*typedlist*(*x*)⟩+)
⟨*variable*⟩ := ?⟨*name*⟩

Furthermore, we show how production and maintenance activities can be declared.

⟨*structure-definition*⟩ := ⟨*pr_activity-definition*⟩ [⟨*structure-definition*⟩]
⟨*structure-definition*⟩ := ⟨*m_activity-definition*⟩ [⟨*structure-definition*⟩]

Production activities are declared as follows. It can be seen that the elements from Definition 1 correspond to the blocks. Note that the optional block ":cost" refers to the cost of the production activity that, in Section 4, is mentioned as a function that maps activities into non-negative numbers (rather than be part of Definition 1).

⟨*pr_activity-definition*⟩ := (*:pr_activity* ⟨*name*⟩
(*:parameters* ⟨*typedlist*(*variable*)⟩+)
(*:attributes* ⟨*attr-description*⟩)
(*:duration* ⟨*time-description*⟩)
[(*:cost* ⟨*cost-description*⟩)]
(*:static* ⟨*pred-conjunction*⟩)
(*:precondition* ⟨*pred-conjunction*⟩)
(*:del-eff* ⟨*pred-conjunction*⟩)
(*:add-eff* ⟨*pred-conjunction*⟩)

$$
\begin{aligned}
\langle \textit{pred-conjunction} \rangle &:= (\textit{and}\ \langle \textit{predicate}\ (\textit{term}) \rangle +) \\
\langle \textit{pred-conjunction} \rangle &:= () \\
\langle \textit{attr-description} \rangle &:= (\textit{and}\ \langle \textit{attr-element} \rangle +) \\
\langle \textit{attr-element} \rangle &:= (\textit{for}\ (\langle \textit{typedlist}(\textit{variable}) \rangle +) \langle \textit{pred-conjunction} \rangle \\
\langle \textit{predicate}(x) \rangle &:= (\langle \textit{name} \rangle\ x+) \\
\langle \textit{term} \rangle &:= \langle \textit{variable} \rangle
\end{aligned}
$$

Note that for predicates being within the body of production activities, types of variables that are declared in the ":parameters" block, or in the "for" part of the ":attributes" block, are not explicitly stated. On top of that, these predicates might contain both variables and domain-specific constants (declared in the ":constants" block in the domain definition).

$$
\begin{aligned}
\langle \textit{time-description} \rangle &:= (= ?\textit{duration}\ \langle \textit{numeric} \rangle) \\
\langle \textit{cost-description} \rangle &:= (= ?\textit{cost}\ \langle \textit{numeric} \rangle) \\
\langle \textit{numeric} \rangle &:= \langle \textit{number} \rangle \\
\langle \textit{numeric} \rangle &:= \langle \textit{predicate}\ (\textit{term}) \rangle
\end{aligned}
$$

Both activity duration and cost are declared analogously. Whereas duration declaration follows the PDDL2.1 syntax (except for not allowing arithmetical expressions within the declaration), cost declaration deviates from the PDDL syntax. For the sake of clarity of our PDDL extension, we decided to declare activity cost analogously to the declaration of activity duration.

Maintenance activities are declared as follows. It can be seen that the elements from Definition 2 correspond to the blocks. Again, the optional block ":cost" refers to the cost of the maintenance activity that, in Section 4, is mentioned as a function that maps activities into non-negative numbers

(rather than be part of Definition 2).

$$
\begin{aligned}
\langle \textit{m\_activity-definition} \rangle \ := \ & (\textit{:m\_activity} \langle \textit{name} \rangle \\
& (\textit{:parameters } \langle \textit{typedlist } (\textit{variable}) \rangle) \\
& (\textit{:resource } \langle \textit{variable} \rangle - \langle \textit{name} \rangle) \\
& [(\textit{:attributes } \langle \textit{attr-description} \rangle)] \\
& (\textit{:duration } \langle \textit{time-description} \rangle) \\
& (\textit{:cost } \langle \textit{cost-description} \rangle) \\
& (\textit{:static } \langle \textit{pred-description} \rangle) \\
& (\textit{:rem-eff } \langle \textit{pred-description} \rangle) \\
& (\textit{:add-eff } \langle \textit{pred-description} \rangle)
\end{aligned}
$$

Note that the correct semantics of the language follows Definition 3. In particular, only static predicates can be listed in the ":static" block of both types of activities, only "attribute" predicates can be listed in maintenance activities (except the ":static" block) and the ":attributes" block in production activities, and in the remaining "predicate" blocks of production activities, only "planning" predicates can be listed. Introduced variables in the "for" parts of the ":attributes" block have to be of type "resource" (or its subtype).

To get an intuition of how a domain specification of a PS task looks like in our extension of PDDL, we provide the specification of the Reconfigurable Machines domain in Appendix B.

6.2. Problem Specifications

The problem-instance part of the PS-task description consists of task-specific constants (C) and resources (R), an initial state of all types of predicates (I, I_L, I_S), and a goal (G). The formal grammar specification follows.

$$
\begin{aligned}
\langle \textit{problem} \rangle \ := \ & (\textit{define } (\textit{problem } \langle \textit{name} \rangle) \\
& (\textit{:domain } \langle \textit{name} \rangle) \\
& (\textit{:objects } \langle \textit{typedlist}(\textit{name}) \rangle +) \\
& (\textit{:init } \langle \textit{init-element}(\textit{name}) \rangle *) \\
& (\textit{:goal } (\textit{and } \langle \textit{predicate}(\textit{name}) \rangle +) \\
& [(\textit{:metric minimize } \langle \textit{criterion} \rangle)] \\
&)
\end{aligned}
$$

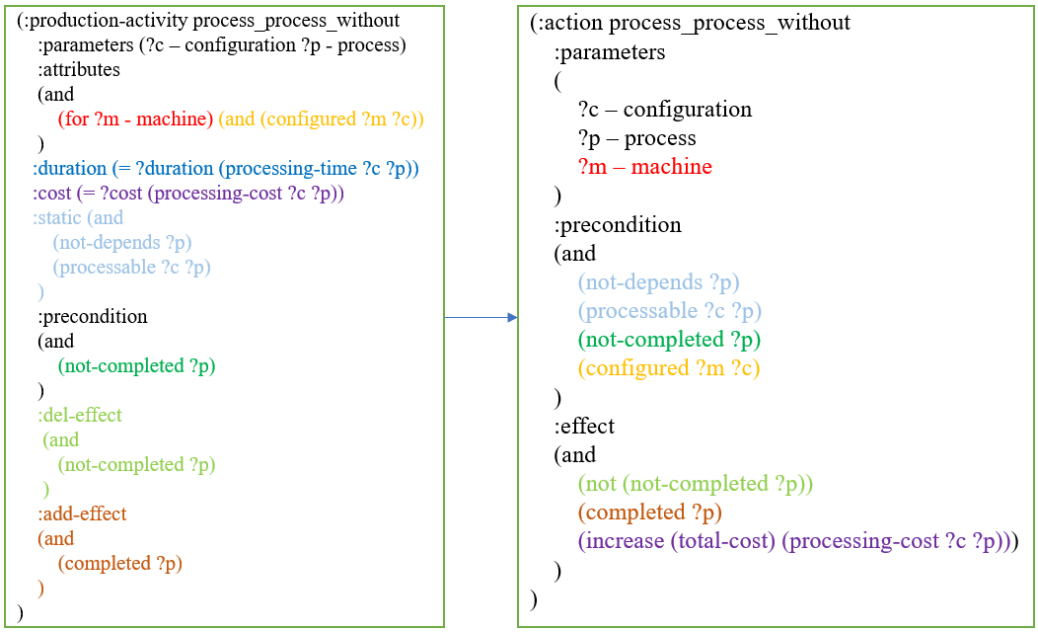


Figure 1: Schematic compilation of PS-task production activity into Classical Planning action. Corresponding elements in the compilation are highlighted in the same color.

$$\begin{aligned}
\langle \textit{init-element}(x) \rangle &:= \langle \textit{predicate}(x) \rangle \\
\langle \textit{init-element}(x) \rangle &:= (= \ \langle \textit{predicate}(x) \rangle \ \langle \textit{number} \rangle) \\
\langle \textit{criterion} \rangle &:= (\langle \textit{name} \rangle)
\end{aligned}$$

Note that at this moment we use two criteria for optimisation "make-span" and "total-cost" that are also defined in PDDL.

7. Compiling PS Tasks into Classical Planning

The idea of how to represent a PS task as a planning task is to merge the "planning" environment with the resource-specific attributes and to represent both types of activities as actions. Although classical planning does not explicitly support reasoning with time, it can still provide a sequence of actions (activities) that can be (according to the given ordering) scheduled afterward because in PS-tasks there is no required concurrency of two or more

activities due to their semantics as both types of activities achieve predicates or attributes only as they finish. Hence an activity that depends on some other activity can be only scheduled after the latter activity finishes.

Let $\Pi = (P, L, S, A, R, M, C, I, I_L, I_S, G)$ be a PS-task. In the following lines, we show how P can be compiled into a classical planning task $T = (P_T, O_T, C_T, I_T, G_T)$.

Firstly, we show how a production activity $a = (name(a),\ dur(a),\ ATTR(a),\ STAT(a),\ pre(a),\ del(a),\ add(a))$ can be compiled into a (classical) operator $o = (name(o), pre(o),\ del(o),\ add(o))$.

$$\begin{aligned} name(o) &= name(a) \\ pre(o) &= pre(a) \cup STAT(a) \cup \bigcup_{z \in ATTR(a)} z \\ del(o) &= del(a) \\ add(o) &= add(a) \end{aligned}$$

Secondly, we show how a maintenance activity $m = (name(m),\ dur(m),\ res(m),\ ATTR(m),\ STAT(m),\ rem(m),\ add(m))$ can be compiled into a (classical) operator $q = (name(q), pre(q),\ del(q),\ add(q))$.

$$\begin{aligned} name(q) &= name(m) \\ pre(q) &= rem(m) \cup STAT(m) \cup \bigcup_{z \in ATTR(m)} z \\ del(q) &= rem(m) \\ add(q) &= add(m) \end{aligned}$$

The set of classical operators O_T is the union of all classical actions compiled from all production (A) and maintenance activities (M). Note that without loss of generality, we assume that classical operators have the same names and arguments as their production or maintenance activity counterparts.

The set of predicates is the union of all types of predicates from the PS tasks, i.e., $P_T = P \cup L \cup S$, and the set of task-specific constants is the union of task-specific constants of the PS-task and resources, i.e., $C_T = C \cup R$. The initial state is a union of all types of PS-task's initial states, i.e., $I_T = I \cup I_L \cup I_S$. The goal is the same as for the PS task, i.e., $G_T = G$.

A solution of a complied planning task Π is a sequence of actions that are grounded instances of operators from O_T. Since we consider the same names and arguments as for activities, then the action directly corresponds to a grounded instance of a respective activity. Although classical planning relaxes out the temporal aspect of the PS-tasks, one can reconstruct a timestamp in which a planned activity can be applied by analysing Simple Temporal Networks [23] as it is done by temporal planners that transform classical plans into temporal ones such as LPG [24], which we use in our experiments.

To get an intuition of how a domain specification of a PS task is compiled to a classical planning domain (in PDDL) see Figure 1, in which it is illustrated how each element of a production activity is compiled into a classical action. The whole Reconfigurable Machines domain and its compilation to classical planning is provided in Appendix B and Appendix C, respectively.

8. Compiling PS Tasks into Temporal Planning

Temporal planning, in contrast to classical planning, takes into account action duration and temporal constraints between actions while reasoning. Hence, temporal plans can directly provide application time for planned (and scheduled) activities. The idea of how to compile PS tasks into temporal planning tasks follows the idea of compilation into classical planning tasks in the sense of compiling both types of activities into operators. Compilation of activities into durative operators follows their semantics, so the add effects (attributes) take place "at end" while delete effects (or removing attributes) take effect "at start". In contrast to the classical planning case, applying more activities at the same time on an individual resource has to be prevented, which is done by introducing "available" predicates that are consumed when an activity starts on a resource and released when that activity finishes.

Let $\Pi = (P, L, S, A, R, M, C, I, I_L, I_S, G)$ be a PS-task. In the following lines, we show how P can be compiled into a temporal planning task $T = (P_T, O_T, C_T, I_T, G_T)$. We introduce an "available" predicate that has one argument (variable symbol), *available(r)* (without loss of generality we assume that a predicate with such name is not defined in Π).

Firstly, we show how a production activity $a = (name(a),\ dur(a),\ ATTR(a),\ STAT(a),\ pre(a),\ del(a),\ add(a))$ can be compiled to a durative operator $o = (name(o),\ dur(o),\ pre^s(o),\ pre^o(o),$

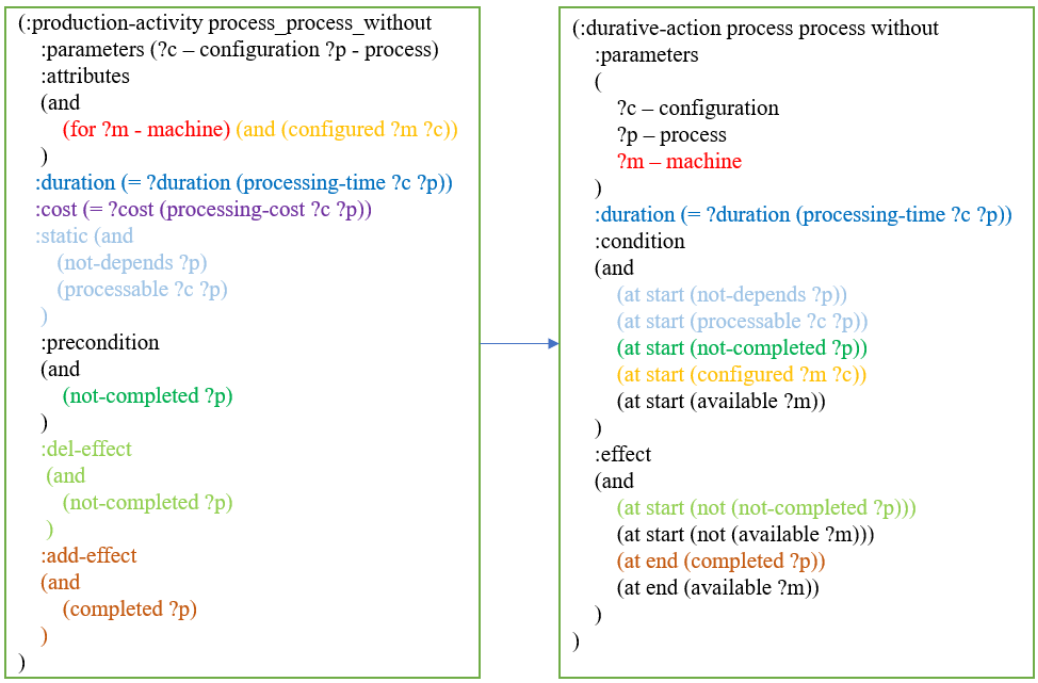


Figure 2: Schematic compilation of PS-task production activity into Temporal Planning action. Corresponding elements in the compilation are highlighted in the same color.

$pre^e(o)$, $del^s(o)$, $del^e(o)$, $add^s(o)$, $add^e(o)$). We denote the set of a's arguments representing resources as $\rho(a)$.

$$
\begin{aligned}
name(o) &= name(a) \\
dur(o) &= dur(a) \\
pre_s(o) &= (pre(a) \cap del(a)) \ \cup \{available(x) \mid x \in \rho(a)\} \\
pre_e(o) &= \emptyset \\
pre_o(o) &= (pre(a) \setminus del(a)) \cup STAT(a) \cup \bigcup_{z \in ATTR(a)} z \\
del_s(o) &= del(a) \cup \{available(x) \mid x \in \rho(a)\} \\
del_e(o) &= \emptyset \\
add_s(o) &= \emptyset \\
add_e(o) &= add(a) \cup \{available(x) \mid x \in \rho(a)\}
\end{aligned}
$$

Secondly, we show the compilation of maintenance activity $m = (name(m),\ dur(m),$

$res(m)$, $ATTR(m)$, $STAT(m)$, $rem(m)$, $add(m)$) into a durative operator $q = (name(q),\ dur(q),\ pre^s(q),\ pre^o(q),\ pre^e(q),\ del^s(q),\ del^e(q),\ add^s(q),\ add^e(q))$. Again, we denote the set of m's arguments representing resources as $\rho(m)$.

$$
\begin{aligned}
name(q) &= name(m) \\
dur(q) &= dur(m) \\
pre_s(q) &= rem(m) \cup \{available(x) \mid x \in \rho(m)\} \\
pre_e(q) &= \emptyset \\
pre_o(q) &= STAT(m) \cup \bigcup_{z \in ATTR(m)} z \\
del_s(q) &= rem(m) \cup \{available(x) \mid x \in \rho(m)\} \\
del_e(q) &= \emptyset \\
add_s(q) &= \emptyset \\
add_e(q) &= add(m) \cup \{available(x) \mid x \in \rho(m)\}
\end{aligned}
$$

The set of durative operators O_T is the union of all the sets of durative operators compiled from all production (A) and maintenance activities (M).

The set of predicates is the union of all types of predicates from the PS tasks and the "available" predicate, i.e., $P_T = P \cup L \cup S \cup \{available(r)\}$, and the set of task-specific constants is the union of task-specific constants of the PS-task and resources, i.e., $C_T = C \cup R$. The initial state is a union of all types of PS task's initial states and all instances of the "available" predicate, i.e., $I_T = I \cup I_L \cup I_S \cup \{available(r) \mid r \in R\}$. The goal is the same as for the PS task, i.e., $G_T = G$.

A solution of a complied temporal planning task Π is a set of grounded (durative) actions with timestamps of their application. Sets of resources on which the activities are scheduled are determined in the same way as in the classical planning case and timestamps of activity application are the same as the timestamps for corresponding actions in the temporal plan.

To get an intuition of how a domain specification of a PS task is compiled to a temporal planning domain (in PDDL 2.1) see Figure 2, in which it is illustrated how each element of a production activity is compiled into a durative action. The whole Reconfigurable Machines domain and its compilation to temporal planning is provided in Appendix B and Appendix D, respectively.

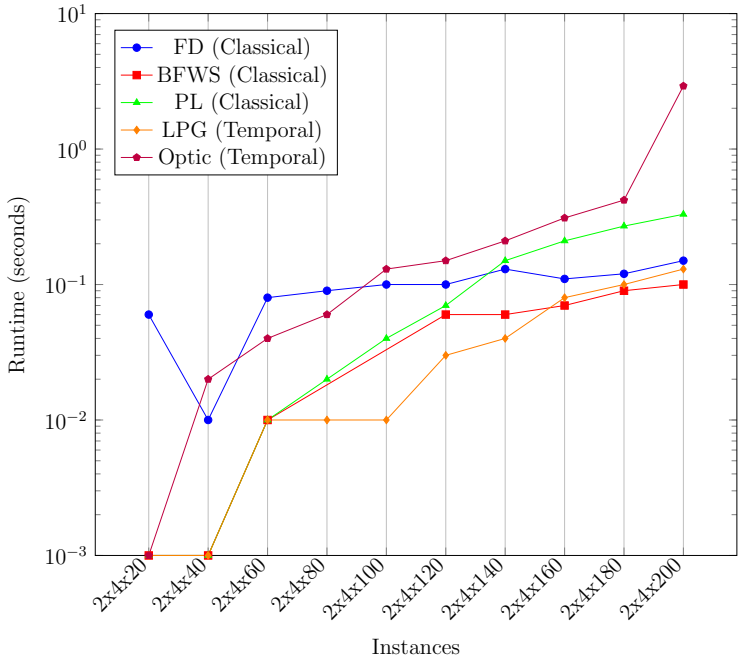


Figure 3: Runtime Comparison in Reconfigurable Machines domain

9. Experimental Evaluation

The experiments aim to show how the presented compilations of PS tasks scale and how good solutions in terms of minimising makespan and total action cost. We considered the introduced three domains and for each domain, we considered both classical and temporal models. For the temporal compilations, we considered makespan, and for the classical compilations, total action cost optimisation metrics, respectively. For classical compilations, the FastDownward (FD) [25], BFWS [26, 27] and Powerlifted (PL) [28] planners were selected and for temporal models, the LPG-TD [24], and Optic [29]. FD was configured to use the lazy-greedy search with the FF and CEA heuristics, Powerlifted - with best-first width search and additive heuristics. Since LPG-TD is a randomised planner, we set the seed to 0 to ensure reproducibility of the results. To cut off local search, the option *onlybestfirst* was used for all instances, except the last five of the Woodworking domain. To determine makespan of classical plans, we have used the plan repair feature of LPG-td that can produce temporal plans from sequential ones. Also, to obtain better quality results (for classical compilations) we used the LAMA planner [30]

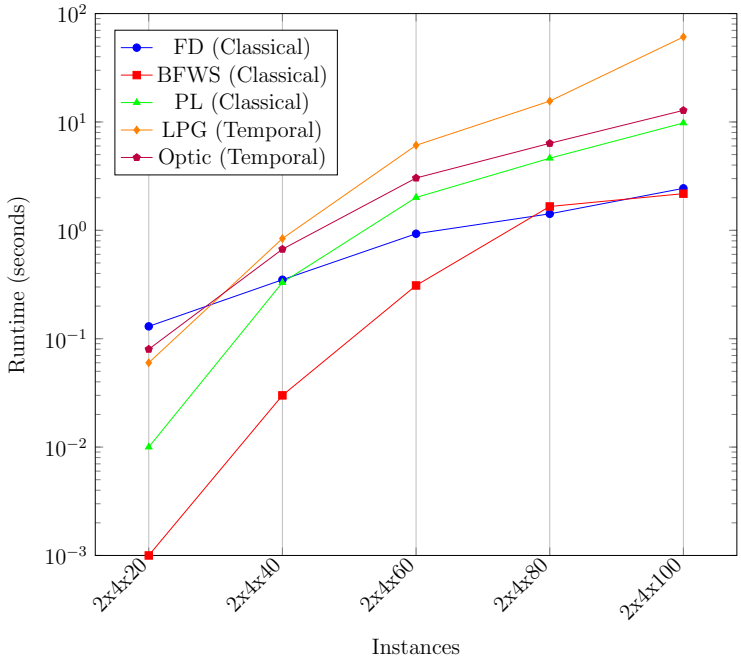


Figure 4: Runtime Comparison in Tube Factory Domain

in an anytime mode (i.e., improving solutions until no better solution can be found, or the time limit is reached). In the case of temporal compilations, LPG-TD was also used in anytime mode by enabling the local search mode.

The timeout for each instance was set to 3600 seconds. All experiments were conducted on the machine with 3.2 GHz processor with 6 cores and 32GB RAM[2].

The runtime results for the Reconfigurable Machines, Tube Factory, and Woodworking domains are depicted in Figures 3, 4 and 5, respectively. As a general trend, it can be observed that classical planners perform better than temporal ones (in terms of runtime), which is expectable as temporal planning is more expressive than classical planning. Classical planners were able to generate solutions within a second in many cases, yet in several cases (especially in Tube Factory) the runtime grew to several seconds, and in Woodworking in several cases the planner ran out of time or memory.

[2]Source code and benchmark data are provided at `https://gitlab.com/xankrig/planning_scheduling_2023_benchmark`.

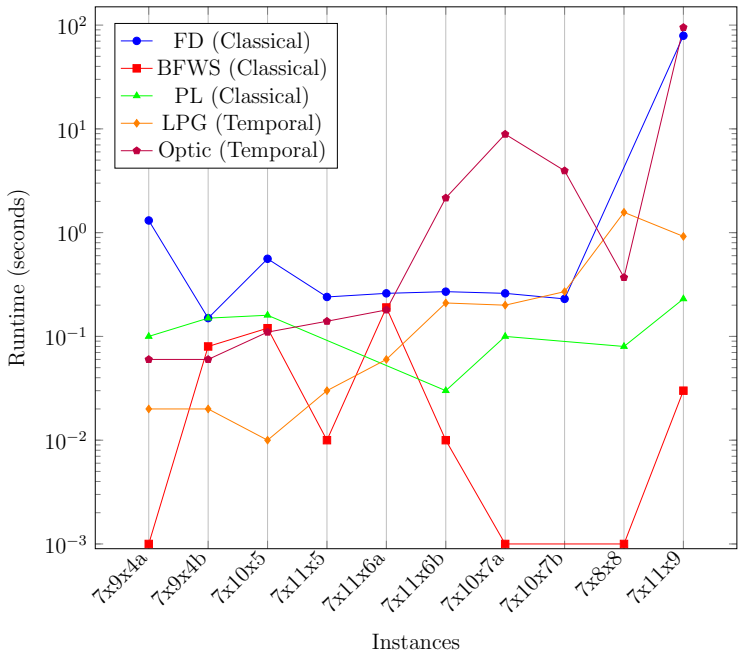
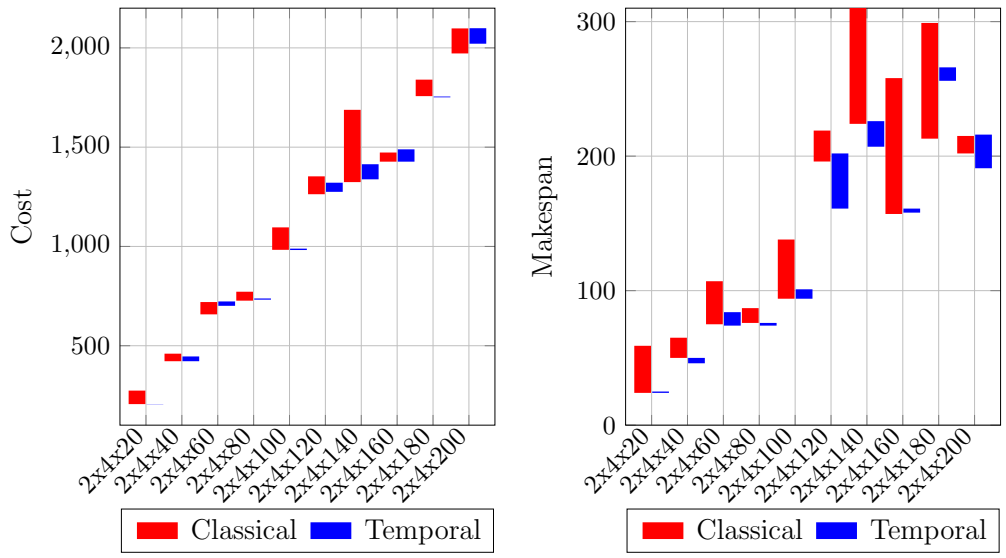


Figure 5: Runtime Comparison in Woodworking Domain

BFWS has shown to be the most "reliable" classical planner in terms of runtime as it solved all problems in all domains within a second except the two hardest problems in Tube Factory. Regarding temporal planners, LPG outperforms Optic (in terms of runtime) in Reconfigurable Machines and Woodworking domains, while underperforms Optic in Tube Factory. Notably, LPG is competitive with classical planners in Reconfigurable Machines and Woodworking. Detailed results are presented in the Appendix in Tables A.1, A.2 and A.3.

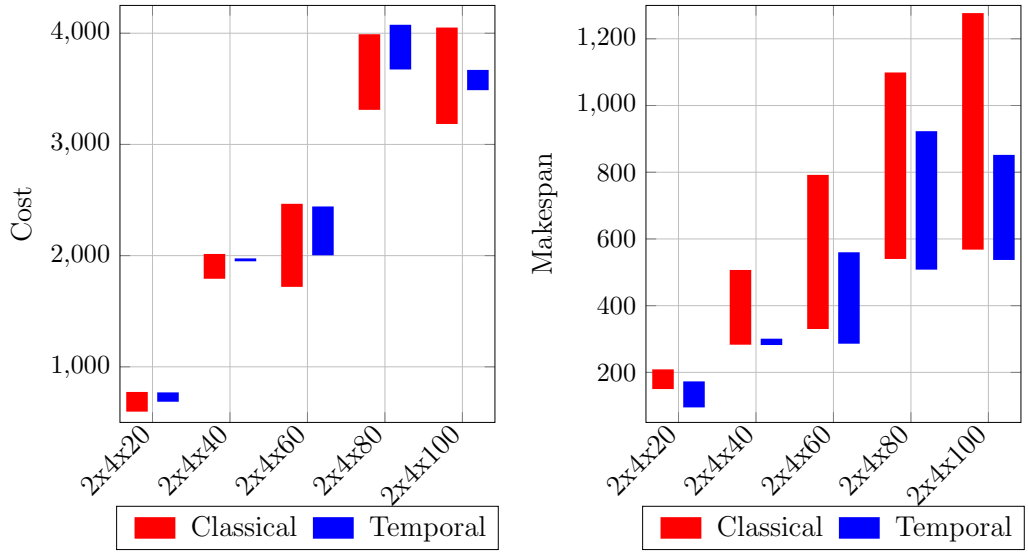
The results concerning solution quality in both considered metrics, total action cost and makespan, are depicted in Figures 6, 7, and 8 for Reconfigurable Machines, Tube Factory and Woodworking domains, respectively. In particular, the charts depict ranges of the cost or makespan values across both types of planners, i.e., classical and temporal ones. Therefore, longer bars refer to larger differences that are among the planners, which is most apparent in the Tube Factory domain while considering the makespan metric. The use of classical planning usually led to less expensive solutions (in terms of action cost) than temporal planning. In the Reconfigurable Machines domain, however, the best results achieved by temporal planners slightly



(a) Cost ranges

(b) Makespan ranges

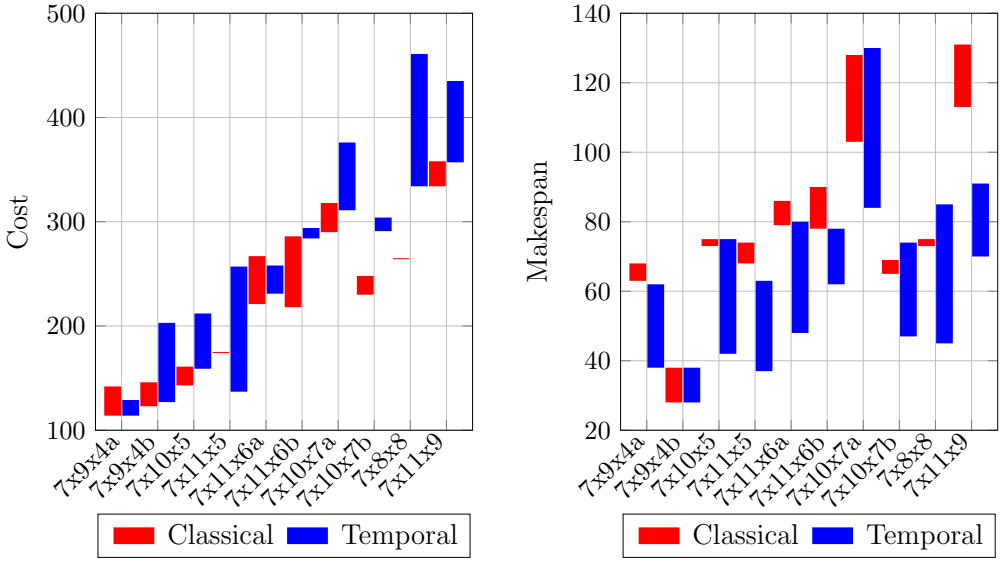
Figure 6: Solution quality comparison for the classical and temporal planners in the Reconfigurable Machines domain.



(a) Cost ranges

(b) Makespan ranges

Figure 7: Solution quality comparison for the classical and temporal planners in the Tube Factory domain.



(a) Cost ranges

(b) Makespan ranges

Figure 8: Solution quality comparison for the classical and temporal planners in the Woodworking domain.

outperformed the best results achieved by classical ones by 1.09%. For the other two domains, the best results achieved by classical planners were better by 12.08% in the Tube Factory domain, and by 16.8% in the Woodworking domain than the best results achieved by temporal planners. In terms of makespan, temporal planners were better (since classical planning does not take time into account). Number-wise, for Reconfigurable Machines, the average makespan of the Optic was better by 10.82%, for Tube Factory by 18.2%, and for Woodworking by 41.3%. Detailed results are presented in the Appendix in Tables A.1, A.2 and A.3. Table A.6 (also in Appendix), on top of that, provides averaged results across all problem instances in each domain. These results often correlate with the focus of a certain type of planners: temporal ones better cope with makespan optimization while classical ones better cope with cost optimisation.

For anytime classical planner, LAMA, the obtained results, in terms of average action cost, are for Reconfigurable Machine – 1074, for Tube Factory – 2120, and for Woodworking – 211. That slightly improves over the best results achieved by classical planners (by a few percent). Detailed results

are shown in Table A.4 (in Appendix). The results obtained by the anytime setting of LPG-TD (a temporal planner), in terms of average makespan, are for Reconfigurable Machines – 114, for Tube – 648, and for Woodworking – 56. In Reconfigurable Machines, the improvement was by 12% while for Woodworking the results were comparable to the best (suboptimal) temporal planner. In the Tube Factory domain, however, the results were not that encouraging. The reason is that the anytime mode of LPG-td is based on a local search that is heavily dependent on random choices during the search and might produce very suboptimal solutions (in contrast to solutions provided by systematic (best-first) search). In 40% of cases, the search was interrupted due to the inability to find a solution and then the planner turned on the *onlybestfirst* mode (so the results were the same as those previously obtained in Table A.2) without the possibility of further optimization of the solutions (as LPG-td does not support that). Detailed results are shown in Table A.5 (in Appendix).

To give a perspective on how good solutions generated by the considered planners are, we also used an optimal classical planner, FD with A*, and CEGAR heuristic, and an optimal temporal planner CPT [31]. The optimal variant of FD managed to solve 2 problems in Reconfiguration Machines and 4 problems in Woodworking. For such problem instances, the optimal solution was up to about 20% less expensive than the reported suboptimal ones, in one case the optimal solution was about 55% less expensive. CPT managed to solve one problem in Reconfigurable Machines and 4 problems in Woodworking. Notably, for two Woodworking problems, CPT found solutions with 22% and 26% shorter makespan compared to the best suboptimal temporal planner.

In summary, results have shown that solving PS tasks suboptimally by means of classical and temporal planning can scale reasonably well even in comparison to specialised scheduling approaches as can be seen in the Reconfigurable Machine domain [7] (albeit such a comparison has to be taken with a grain of salt as the settings are not the same). More complex domains such as Woodworking are not known to be tackled by scheduling approaches.

10. Conclusion

In this paper, we have formally introduced the concept of PS tasks that, in a nutshell, combines aspects of Automated Planning and Scheduling. The motivation for introducing PS tasks relates to industrial production planning

and manufacturing, where production activities might need to be selected from a larger pool and then (partially) ordered before being scheduled on resources (machines) to be run at. The selection and ordering of production activities can be formulated as a planning task while scheduling these activities on resources can be formulated as a scheduling task.

In this paper, we presented an extension of PDDL in which PS tasks can be modeled. Then, we presented two compilations of PS tasks - one into classical planning and the other into temporal planning - so off-the-shelf planning engines (classical and temporal) can be leveraged to generate solutions for PS tasks. We specified three use cases – Reconfigurable Machines, Tube Factory, and Woodworking – that are examples of industrial production planning problems that can be specified as PS tasks. Domains such as Woodworking involving choosing from (many) alternatives while producing required products are not known to be tackled by scheduling techniques.

The empirical results indicate that solving PS tasks by means of compiling them into classical or temporal planning tasks and then using off-the-shelf planning engines is a viable way as it scales reasonably well with the size of problems, although the solutions are suboptimal. In the Reconfigurable Machines domain, our domain-independent methods were comparable to specialized approaches [7] in terms of scalability (albeit the settings were not the same, so the results have to be considered with a grain of salt). The developed specification for PDDL can be understood as a link between planning and scheduling and can form a future basis for extending existing planning engines with scheduling heuristics. We believe that leveraging Mixed Integer Programming or Linear Programming for such heuristics as, for instance, in the flow heuristics [32] or potential heuristics [33] used in classical planning is a promising direction of research.

In the future, we plan to investigate how further constraints (e.g. deadlines, resource operation time windows) can be incorporated into the concept of PS tasks and develop more specialized solvers combining planning and scheduling techniques to tackle them.

11. Acknowledgments

This research is supported by Czech Science Foundation (project no. 23-05575S), by the European Union under the project ROBOPROX (reg. no. CZ.02.01.01/00/22_008/0004590). and by the Grant Agency of the Czech Technical University (project no. SGS24/115/OHK3/2T/37).

References

[1] R. Čapek, P. Šůcha, Z. Hanzálek, Production scheduling with alternative process plans, European Journal of Operational Research 217 (2) (2012) 300–311. doi:https://doi.org/10.1016/j.ejor.2011.09.018.

[2] Z. Hanzálek, R. Čapek, P. Šůcha, Total setup time minimisation in production scheduling with alternatives, in: V. Mařík, W. Wahlster, T. Strasser, P. Kadera (Eds.), Industrial Applications of Holonic and Multi-Agent Systems, Springer International Publishing, Cham, 2017, pp. 11–23.

[3] A. Nyporko, L. Chrpa, Towards an effective framework combining planning and scheduling [extended abstract], in: Sixteenth International Symposium on Combinatorial Search, SOCS 2023, July 14-16, 2023, Prague, Czech Republic, 2023, pp. 173–174.

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

[5] D. Long, J. Dolejsi, M. Stolba, Scheduling problems in PDDL, in: Workshop of Knowledge Engineering in Planning and Scheduling (KEPS), 2023.

[6] S. Borgo, A. Cesta, A. Orlandini, A. Umbrico, A planning-based architecture for a reconfigurable manufacturing system, in: Proceedings of the International Conference on Automated Planning and Scheduling, 26(1), 2016, pp. 358–366.

[7] B. Vahedi-Nouri, R. Tavakkoli-Moghaddam, Z. Hanzálek, A. Dolgui, Workforce planning and production scheduling in a reconfigurable manufacturing system facing the covid-19 pandemic, Journal of Manufacturing Systems 63 (2022) 563–574. doi:https://doi.org/10.1016/j.jmsy.2022.04.018.

[8] V. Heinz, A. Novák, M. Vlk, Z. Hanzálek, Constraint programming and constructive heuristics for parallel machine scheduling with sequence-dependent setups and common servers, Computers & Industrial Engineering 172 (2022) 108586. doi:https://doi.org/10.1016/j.cie.2022.108586.

[9] T. L. Dean, R. J. Firby, D. P. Miller, Hierarchical planning involving deadlines, travel time, and resources, Computational Intelligence 4 (1988).

[10] W. Tan, B. Khoshnevis, Integration of process planning and scheduling— a review, Journal of Intelligent Manufacturing 11 (2000) 51–63. doi:10.1023/A:1008952024606.
URL https://doi.org/10.1023/A:1008952024606

[11] R. E. Frederking, N. Muscettola, Temporal planning for transportation planning and scheduling, Proceedings 1992 IEEE International Conference on Robotics and Automation (1992) 1225–1230 vol.2.

[12] N. Muscettola, HSTS: integrating planning and scheduling, Ph.D. thesis, Carnegie Mellon University (1993).

[13] A. Garrido, F. Barber, Integrating planning and scheduling, Applied Artificial Intelligence 15 (5) (2001) 471–491. doi:10.1080/088395101300125734.

[14] M. Ai-Chang, J. Bresina, L. Charest, A. Chase, J. Hsu, A. Jonsson, B. Kanefsky, P. Morris, K. Rajan, J. Yglesias, B. Chafin, W. Dias, P. Maldague, Mapgen: Mixed-initiative planning and scheduling for the mars exploration rover mission, IEEE Intelligent Systems 19 (2004) 8–12. doi:10.1109/MIS.2004.1265878.

[15] M. D. Rodríguez-Moreno, A. Oddi, D. Borrajo, A. Cesta, Ipss: A hybrid approach to planning and scheduling integration, IEEE Transactions on Knowledge and Data Engineering 18 (2006) 1681–1695.

[16] R. Phanden, A. Jain, R. Verma, An approach for integration of process planning and scheduling, International Journal of Computer Integrated Manufacturing 26 (2012) 1–19. doi:10.1080/0951192X.2012.684721.

[17] W. Ruml, M. Do, M. Fromherz, On-line planning and scheduling for high-speed manufacturing., in: ICAPS 2005 - Proceedings of the 15th International Conference on Automated Planning and Scheduling, 2005, pp. 30–39.

[18] P. Laborie, Algorithms for propagating resource constraints in ai planning and scheduling: Existing approaches and new results, Artificial

Intelligence 143 (2) (2003) 151–188. doi:https://doi.org/10.1016/S0004-3702(02)00362-4.

[19] P. Laborie, J. Rogerie, P. Shaw, P. Vilím, Ibm ilog cp optimizer for scheduling, Constraints 23 (2) (2018) 210–250. doi:10.1007/s10601-018-9281-x.

[20] D. Banerjee, Integrating planning and scheduling : A constraint-based approach, Ph.D. thesis, ANU College of Engineering & Computer Science, The Australian National University (2015).

[21] C. L. López, S. J. Celorrio, A. G. Olaya, The deterministic part of the seventh international planning competition, Artif. Intell. 223 (2015) 82–119.

[22] M. Vallati, L. Chrpa, T. L. McCluskey, What you always wanted to know about the deterministic part of the international planning competition (IPC) 2014 (but were too afraid to ask), Knowl. Eng. Rev. 33 (2018) e3.

[23] R. Dechter, I. Meiri, J. Pearl, Temporal constraint networks, Artificial Intelligence 49 (1) (1991) 61–95. doi:https://doi.org/10.1016/0004-3702(91)90006-6.

[24] A. Gerevini, A. Saetti, I. Serina, Planning through stochastic local search and temporal action graphs in LPG, J. Artif. Intell. Res. 20 (2003) 239–290. doi:10.1613/jair.1183.
URL https://doi.org/10.1613/jair.1183

[25] M. Helmert, The fast downward planning system, Journal of Artificial Intelligence Research 26 (2006) 191–246. doi:10.1613/jair.1705.
URL https://doi.org/10.1613/jair.1705

[26] N. Lipovetzky, H. Geffner, Best-first width search: Exploration and exploitation in classical planning, Proceedings of the AAAI Conference on Artificial Intelligence 31 (1) (Feb. 2017). doi:10.1609/aaai.v31i1.11027.
URL https://ojs.aaai.org/index.php/AAAI/article/view/11027

[27] N. Lipovetzky, H. Geffner, A polynomial planning algorithm that beats lama and ff, Proceedings of the International Conference on Automated Planning and Scheduling 27 (1) (2017) 195–199. doi:10.1609/icaps.v27i1.13822.

URL https://ojs.aaai.org/index.php/ICAPS/article/view/13822

[28] A. B. Corrêa, J. Seipp, Best-first width search for lifted classical planning, Proceedings of the International Conference on Automated Planning and Scheduling 32 (1) (2022) 11–15. doi:10.1609/icaps.v32i1.19780.
URL https://ojs.aaai.org/index.php/ICAPS/article/view/19780

[29] J. Benton, A. Coles, A. Coles, Temporal planning with preferences and time-dependent continuous costs, Proceedings of the International Conference on Automated Planning and Scheduling 22 (1) (2012) 2–10. doi:10.1609/icaps.v22i1.13509.
URL https://ojs.aaai.org/index.php/ICAPS/article/view/13509

[30] S. Richter, M. Westphal, The LAMA planner: Guiding cost-based anytime planning with landmarks, J. Artif. Intell. Res. 39 (2010) 127–177. doi:10.1613/JAIR.2972.
URL https://doi.org/10.1613/jair.2972

[31] V. Vidal, H. Geffner, Cpt: An optimal temporal pocl planner based on constraint programming, in: Booklet of the IPC, 2004.

[32] B. Bonet, M. van den Briel, Flow-based heuristics for optimal planning: Landmarks and merges, in: Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014, AAAI, 2014.

[33] F. Pommerening, G. Röger, M. Helmert, B. Bonet, Heuristics for cost-optimal classical planning based on linear programming, in: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, AAAI Press, 2015, pp. 4303–4309.

Appendix A. Tables with Detailed Results

In this appendix, we present three tables with full experimental results about runtime, cost and makespan of different instances of aforementioned FastDownward, BFWS, PL, LPG-TD, Optic and CPT planners for each of the domains, a table with the data about cost and makespan obtained by the anytime configuration of the LAMA planner, a table with the data about cost and makespan obtained for the anytime configuration of the LPG-TD planner for all domains, and finally, a table with averaged data, including standard deviation for cost and makespan, for each of the domains and each of the planners.

Table A.1: Planning time (in seconds), plan cost, and makespan per problem and planner for the Reconfigurable Machines domain. Instances are described in a format MxCxP that stands for the number of machines, configurations, and processes, respectively.

	Classical planners				Temporal planners			
Instances	Planners	Runtime	Cost	Makespan	Planners	Runtime	Cost	Makespan
2x4x20	FD	0.06	274	59	LPG	0.00	206	24
	BFWS	0.00	206	24	Optic	0.00	206	25
	PL	0.00	206	24	CPT	0.04	206	24
2x4x40	FD	0.01	422	50	LPG	0.00	422	50
	BFWS	0.00	460	65	Optic	0.02	446	46
	PL	0.00	460	65	CPT	TIMEOUT		
2x4x60	FD	0.08	658	77	LPG	0.01	723	74
	BFWS	0.01	720	75	Optic	0.04	701	84
	PL	0.01	690	107	CPT	TIMEOUT		
2x4x80	FD	0.09	772	87	LPG	0.01	738	76
	BFWS	0.00	727	76	Optic	0.06	732	74
	PL	0.02	742	81	CPT	TIMEOUT		
2x4x100	FD	0.1	1096	102	LPG	0.01	982	94
	BFWS	0.00	999	138	Optic	0.13	989	101
	PL	0.04	983	94	CPT	TIMEOUT		
2x4x120	FD	0.1	1263	196	LPG	0.03	1321	202
	BFWS	0.06	1353	219	Optic	0.15	1275	161
	PL	0.07	1353	219	CPT	TIMEOUT		
2x4x140	FD	0.13	1688	359	LPG	0.04	1338	226
	BFWS	0.06	1398	230	Optic	0.21	1414	207
	PL	0.15	1324	224	CPT	TIMEOUT		
2x4x160	FD	0.11	1427	258	LPG	0.08	1489	161
	BFWS	0.07	1438	251	Optic	0.31	1427	158
	PL	0.21	1473	157	CPT	TIMEOUT		
2x4x180	FD	0.12	1840	213	LPG	0.1	1751	266
	BFWS	0.09	1769	299	Optic	0.42	1755	256
	PL	0.27	1757	258	CPT	TIMEOUT		
2x4x200	FD	0.15	1972	202	LPG	0.13	2099	216
	BFWS	0.1	2098	215	Optic	2.92	2021	191
	PL	0.33	2098	215	CPT	TIMEOUT		

Table A.2: Planning time (in seconds), plan cost, and makespan per problem and planner for the Tube Factory domain. Instances are described in a format MxSxT that stands for the number of machines, servers, and tasks, respectively.

	Classical planners				Temporal planners			
Instances	Planners	Runtime	Cost	Makespan	Planners	Runtime	Cost	Makespan
2x4x20	FD	0.13	597	150	LPG	0.06	686	173
	BFWS	0.00	774	209	Optic	0.08	770	95
	PL	0.01	666	173	CPT		TIMEOUT	
2x4x40	FD	0.35	1792	431	LPG	0.84	1975	301
	BFWS	0.03	2015	507	Optic	0.67	1949	282
	PL	0.33	1993	283	CPT		TIMEOUT	
2x4x60	FD	0.93	1719	330	LPG	6.09	2442	560
	BFWS	0.31	2430	519	Optic	3.04	2003	286
	PL	2.01	2466	792	CPT		TIMEOUT	
2x4x80	FD	1.42	3311	540	LPG	15.57	4076	923
	BFWS	1.66	3991	1099	Optic	6.35	3674	508
	PL	4.63	3806	783	CPT		TIMEOUT	
2x4x100	FD	2.44	3184	568	LPG	60.87	3670	852
	BFWS	2.18	3675	842	Optic	12.79	3488	537
	PL	9.79	4051	1277	CPT		TIMEOUT	

Table A.3: Planning time (in seconds), plan cost and makespan per problem and planner for the Woodworking domain. Instances are described in a format MxBxP that stands for the number of machines, boards, and parts, respectively.

	Classical planners				Temporal planners			
Instances	Planners	Runtime	Cost	Makespan	Planners	Runtime	Cost	Makespan
7x9x4	FD	1.31	142	68	LPG	0.02	114	62
	BFWS	0.00	114	63	Optic	0.06	129	38
	PL	0.1	114	63	CPT	TIMEOUT		
7x9x4	FD	0.15	123	28	LPG	0.02	140	28
	BFWS	0.08	145	38	Optic	0.06	203	38
	PL	0.15	146	38	CPT	0.05	127	28
7x10x5	FD	0.56	161	75	LPG	0.01	159	75
	BFWS	0.12	143	73	Optic	0.11	187	42
	PL	0.16	155	73	CPT	0.5	212	42
7x11x5	FD	0.24	174	74	LPG	0.03	137	63
	BFWS	0.01	175	68	Optic	0.14	174	50
	PL	MEMOUT			CPT	0.9	257	37
7x11x6	FD	0.26	221	79	LPG	0.06	231	80
	BFWS	0.19	267	86	Optic	0.18	258	48
	PL	MEMOUT			CPT	TIMEOUT		
7x11x6	FD	0.27	218	78	LPG	0.21	294	78
	BFWS	0.01	286	90	Optic	2.16	284	62
	PL	0.03	273	85	CPT	TIMEOUT		
7x10x7	FD	0.26	318	103	LPG	0.2	376	130
	BFWS	0.00	290	128	Optic	8.89	311	84
	PL	0.1	290	120	CPT	TIMEOUT		
7x10x7	FD	0.23	230	65	LPG	0.27	304	74
	BFWS	0.00	248	69	Optic	3.95	291	47
	PL	MEMOUT			CPT	TIMEOUT		
7x8x8	FD	TIMEOUT			LPG	1.57	334	85
	BFWS	0.00	264	73	Optic	0.37	365	58
	PL	0.08	264	75	CPT	6.96	461	45
7x11x9	FD	79	358	113	LPG	0.92	435	91
	BFWS	0.03	349	122	Optic	95	357	70
	PL	0.23	334	131	CPT	TIMEOUT		

Table A.4: Plan cost and makepan for LAMA specifications in FastDownward planner for all domains. RMT stands for Reconfigurable Machines. The planning time limit is 600 seconds, the best-known solution (within the limit) is stated for each instance, and optimal results are highlighted with *

	RMT		Woodworking			Tube Factory		
Instances	Cost	Makespan	Instances	Cost	Makespan	Instances	Cost	Makespan
2x4x20	190*	41	7x9x4	114	62	2x4x20	597	150
2x4x40	398	47	7x9x4	122*	28	2x4x40	1792	431
2x4x60	613	68	7x10x5	143	76	2x4x60	1719	330
2x4x80	766	91	7x11x5	128	68	2x4x80	3311	540
2x4x100	962	128	7x11x6	189	76	2x4x100	3184	568
2x4x120	1339	226	7x11x6	312	111			
2x4x140	1365	257	7x10x7	279	114			
2x4x160	1295	241	7x10x7	207	68			
2x4x180	1770	229	7x8x8	261	72			
2x4x200	2047	238	7x11x9	356	120			

Table A.5: Plan cost and makepan for anytime specifications in LPG-TD planner for all domains. RMT stands for Reconfigurable Machines. The planning time limit is 600 seconds, and the best-known solution (within the limit) is stated for each instance

	RMT		Woodworking			Tube Factory		
Instances	Cost	Makespan	Instances	Cost	Makespan	Instances	Cost	Makespan
2x4x20	206	24	7x9x4	167	37	2x4x20	799	76
2x4x40	422	43	7x9x4	116	28	2x4x40	3679	462
2x4x60	673	65	7x10x5	182	42	2x4x60	4506	927
2x4x80	720	71	7x11x5	210	41	2x4x80	4076	923
2x4x100	1013	92	7x11x6	337	58	2x4x100	3670	852
2x4x120	1296	136	7x11x6	311	53			
2x4x140	1356	182	7x10x7	368	68			
2x4x160	1528	152	7x10x7	199*	60*			
2x4x180	1776	182	7x8x8	355	49			
2x4x200	2062	198	7x11x9	338	123			

Table A.6: Average planning time (in seconds), total cost, makespan, and their standard deviations (columns with StD suffix) per planner for all domains. RMT stands for Reconfigurable Machines.

Domains	Planners	Runtime	Cost	Makespan	Cost.StD	Makespan.StD
RMT	FD	0.095	1141.2	160.3	567.02	96.4
	BFWS	0.04	1116.8	159.2	566.6	90.15
	PL	0.11	1108.6	144.4	565.59	76.62
	LPG-TD	0.041	1106.9	138.9	567.81	80.89
	Optic	0.426	1096.6	130.3	552.01	71.72
Tube	FD	1.054	2120.6	403.8	1013.75	152.38
	BFWS	0.836	2577	635.2	1165.61	306.41
	PL	3.354	2596.4	661.6	1239.88	398.12
	LPG-TD	16.686	2569.8	561.8	1216.37	294.57
	Optic	4.586	2376.8	341.6	1079.07	163.29
Woodworking	FD	9.14	216.11	75.88	74.31	22.61
	BFWS	0.044	228.1	81	74.15	25.70
	PL	0.12	225.14	83.57	78.73	29.91
	LPG-TD	0.33	252.4	76.6	106.75	24.31
	Optic	11.09	255.9	53.7	75.85	14.11

Appendix B. Reconfigurable Machine Tools - Planning-Scheduling Domain

```
(define (domain ps_domain_rmt)
    (:requirements :strips :fluents :durative-actions
    :timed-initial-literals :typing :conditional-effects
    :negative-preconditions :duration-inequalities
    :equality :ps-task)
    (:types
        resource - object
        process - object
        attribute - object
        machine - resource
        configuration - attribute
    )
    (:attributes
        (configured ?m - machine ?c - configuration)
        (configurable ?m - machine ?c - configuration)
    )
    (:static
        (not-same ?c1 - configuration ?c2 - configuration)
        (processable ?c - configuration ?p - process)
        (depends-one ?p1 - process ?p2 - process)
        (not-depends ?p - process)
    )
    (:predicates
        (completed ?p - process)
        (not-completed ?p - process)
    )
    (:functions
        (reconfiguration-time ?c1 - configuration ?c2 - configuration)
        (reconfiguration-cost ?c1 - configuration ?c2 - configuration)
        (processing-time ?c - configuration ?p - process)
        (processing-cost ?c - configuration ?p - process)
    )
    (:maintenance-activity reconfigure_machine
        :parameters
        (
```

```
        ?c1 ?c2 - configuration
    )
    :resource
    (
        ?m - machine
    )
    :attributes
    (and
        (configured ?m ?c1)
    )
    :duration (= ?duration (reconfiguration-time ?c1 ?c2))
    :cost (= ?cost (reconfiguration-cost ?c1 ?c2))
    :static (and (configurable ?m ?c2) (not-same ?c1 ?c2))
    :rem-effect
    (and (configured ?m ?c1))
    :add-effect
    (and (configured ?m ?c2)
    )
)
(:production-activity process_process_without
    :parameters
    (
        ?c - configuration
        ?p - process
    )
    :attributes
    (and
        (for ?m - machine) (and (configured ?m ?c))
    )
    :duration (= ?duration (processing-time ?c ?p))
    :cost (= ?cost (processing-cost ?c ?p))
    :static
    (and
        (not-depends ?p) (processable ?c ?p))
    :precondition
    (and
        (not-completed ?p)
    )
```

```
    :del-effect
    (and
        (not-completed ?p)
    )
    :add-effect
    (and
        (completed ?p)
    )
)
(:production-activity process_process_with_1p
    :parameters
    (
        ?c - configuration
        ?p1 ?p2 - process
    )
    :attributes
    (and
        (for ?m - machine) (and (configured ?m ?c))
    )
    :duration (= ?duration (processing-time ?c ?p1))
    :cost (= ?cost (processing-cost ?c ?p1))
    :static
    (and
        (depends-one ?p1 ?p2)
        (processable ?c ?p1)
    )
    :precondition
    (and
         (completed ?p2) (not-completed ?p1)
    )
    :del-effect
    (and
        (not-completed ?p1)
    )
    :add-effect
    (and
        (completed ?p1)
    )
```

```
    )
)
```

Appendix C. Reconfigurable Machine Tools - Classical Domain

```
(define (domain ps_domain_rmt)
    (:requirements :strips :typing :conditional-effects
    :negative-preconditions :equality)
    (:types
        resource - object
        process - object
        attribute - object
        machine - resource
        configuration - attribute
    )
    (:predicates
        (completed ?p - process)
        (not-completed ?p - process)
        (configured ?m - machine ?c - configuration)
        (configurable ?m - machine ?c - configuration)
        (not-same ?c1 - configuration ?c2 - configuration)
        (processable ?c - configuration ?p - process)
        (depends-one ?p1 - process ?p2 - process)
        (not-depends ?p - process)
    )
    (:functions
        (reconfiguration-cost ?c1 - configuration ?c2 - configuration)
        (processing-cost ?c - configuration ?p - process)
        (total-cost)
    )
    (:action reconfigure_machine
        :parameters
        (
            ?c1 ?c2 - configuration
            ?m - machine
        )
        :precondition
        (and
```

```
        (configurable ?m ?c2)
        (not-same ?c1 ?c2)
        (configured ?m ?c1)
    )
    :effect
    (and
        (not (configured ?m ?c1))
        (configured ?m ?c2)
        (increase (total-cost) (reconfiguration-cost ?c1 ?c2))
    )
)
(:action process_process_without
    :parameters
    (
        ?c - configuration
        ?p - process
        ?m - machine
    )
    :precondition
    (and
        (not-depends ?p)
        (processable ?c ?p)
        (not-completed ?p)
        (configured ?m ?c)
    )
    :effect
    (and
        (not (not-completed ?p))
        (completed ?p)
        (increase (total-cost) (processing-cost ?c ?p))
    )
)
(:action process_process_with_1p
    :parameters
    (
        ?c - configuration
        ?p1 ?p2 - process
        ?m - machine
```

```
        )
        :precondition
        (and
            (depends-one ?p1 ?p2)
            (processable ?c ?p1)
            (completed ?p2)
            (not-completed ?p1)
            (configured ?m ?c)
        )
        :effect
        (and
            (not (not-completed ?p1))
            (completed ?p1)
            (increase (total-cost) (processing-cost ?c ?p1))
        )
    )
)
```

Appendix D. Reconfigurable Machine Tools - Temporal Domain

```
(define (domain ps_domain_rmt)
    (:requirements :strips :fluents :durative-actions
    :timed-initial-literals :typing
    :conditional-effects :negative-preconditions
    :duration-inequalities :equality)
    (:types
        resource - object
        process - object
        attribute - object
        machine - resource
        configuration - attribute
    )
    (:predicates
        (completed ?p - process)
        (not-completed ?p - process)
        (configured ?m - machine ?c - configuration)
        (configurable ?m - machine ?c - configuration)
        (not-same ?c1 - configuration ?c2 - configuration)
```

```
    (processable ?c - configuration ?p - process)
    (depends-one ?p1 - process ?p2 - process)
    (not-depends ?p - process)
    (available ?r - resource)
)
(:functions
    (reconfiguration-time ?c1 - configuration ?c2 - configuration)
    (processing-time ?c - configuration ?p - process)
)
(:durative-action reconfigure_machine
    :parameters
    (
        ?c1 ?c2 - configuration
        ?m - machine
    )
    :duration (= ?duration (reconfiguration-time ?c1 ?c2))
    :condition
    (and
        (at start (configurable ?m ?c2))
        (at start (not-same ?c1 ?c2))
        (at start (configured ?m ?c1))
        (at start (available ?m))
    )
    :effect
    (and
        (at start (not (configured ?m ?c1)))
        (at start (not (available ?m)))
        (at end (configured ?m ?c2))
        (at end (available ?m))
    )
)
(:durative-action process_process_without
    :parameters
    (
        ?c - configuration
        ?p - process
        ?m - machine
    )
```

```
    :duration (= ?duration (processing-time ?c ?p))
    :condition
    (and
        (at start (not-depends ?p))
        (at start (processable ?c ?p))
        (at start (not-completed ?p))
        (at start (configured ?m ?c))
        (at start (available ?m))
    )
    :effect
    (and
        (at start (not (not-completed ?p)))
        (at start (not (available ?m)))
        (at end (completed ?p))
        (at end (available ?m))
    )
)
(:durative-action process_process_with_1p
    :parameters
    (
        ?c - configuration
        ?p1 ?p2 - process
        ?m - machine
    )
    :duration (= ?duration (processing-time ?c ?p1))
    :condition
    (and
        (at start (depends-one ?p1 ?p2))
        (at start (processable ?c ?p1))
        (at start (completed ?p2))
        (at start (not-completed ?p1))
        (at start (configured ?m ?c))
        (at start (available ?m))
    )
    :effect
    (and
        (at start (not (not-completed ?p1)))
        (at start (not (available ?m)))
```

```
            (at end (completed ?p1))
            (at end (available ?m))
        )
    )
)
```