Deterministic Planning in the Fifth International Planning Competition: PDDL3 and Experimental Evaluation of the Planners

Alfonso E. Gerevini* Patrik Haslum° Derek Long# Alessandro Saetti*
Yannis Dimopoulos+

* Dipartimento di Elettronica per l'Automazione, Università degli Studi di Brescia, Brescia, Italy
° NICTA & The Australian National University, Canberra, Australia
Department of Computer and Information Sciences, University of Strathclyde, Glasgow, UK
+ Department of Computer Science, University of Cyprus, Cyprus
*{gerevini,saetti}@ing.unibs.it °patrik.haslum@anu.edu.au #derek.long@cis.strath.ac.uk +yannis@cs.ucy.ac.cy

Abstract

The international planning competition (IPC) is an important driver for planning research. The general goals of the IPC include pushing the state of the art in planning technology by posing new scientific challenges, encouraging direct comparison of planning systems and techniques, developing and improving a common planning domain definition language, and designing new planning domains and problems for the research community. This paper focuses on the deterministic part of the fifth international planning competition (IPC5), presenting the language and benchmark domains that we developed for the competition, as well as a detailed experimental evaluation of the deterministic planners that entered IPC5, which helps to understand the state of the art in the field.

We introduce an extension of PDDL, called PDDL3, allowing the user to express strong and soft constraints about the structure of the desired plans, as well as strong and soft problem goals. We discuss the expressive power of the new language focusing on the restricted version that was used in IPC5, for which we give some basic results about its compilability into PDDL2. Moreover, we study the relative performance of the IPC5 planners in terms of solved problems, CPU time, and plan quality; we analyse their behaviour with respect to the winners of the previous competition; and we evaluate them in terms of their capability of dealing with soft goals and constraints, and of finding good quality plans in general. Overall, the results indicate significant progress in the field, but they also reveal that some important issues remain open and require further research, such as dealing with strong constraints and computing high quality plans in metric-time domains and domains involving soft goals or constraints.

Keywords: Keywords: Automated Planning, Planning Systems, PDDL, Planning Languages, Knowledge Representation in Planning, Preferences in Planning, Plan Constraints, International Planning Competition, Benchmarks for Planning, Experimental Evaluation of Planning Systems.

1 Introduction

The international planning competition (IPC for short) is an important driver for research in AI planning that is biennially held in conjunction with the International Conference on Automated Planning and Scheduling. The general goals of the IPC include pushing the state of the art in planning technology by posing new scientific challenges, encouraging and conducting direct comparison of planning systems and techniques, developing and improving a common planning domain definition language, PDDL [29, 36, 41], and designing new planning domains and problems for the research community that are increasingly realistic. This paper focuses on the deterministic part of the fifth international planning competition (IPC5 for short), which is the classical part of the competition addressing planning problems where the initial state is completely specified and the relevant effects of the available actions are

deterministic. We present the language and benchmark domains developed for the competition, and a detailed experimental evaluation of the deterministic planners that entered IPC5.

While IPC5 shares the same general goals of the previous planning competitions, it has some important novel features making this event significantly different from the previous competitions [1, 41, 50, 51]. In particular, the deterministic track of IPC5 emphasises the importance of plan quality, which is crucial in many applications, but which previously did not receive sufficient attention. Motivated by a desire to capture plan quality, a significant new version of PDDL, called PDDL3, has been designed. PDDL3 includes some new constructs that can better characterise plan quality by allowing the user to express both strong and soft constraints on the structure of the desired plans. PDDL3 also includes soft problem goals through which we can express over-constrained planning problems (called "over-subscription" problems in [15, 23, 62]).

Plan trajectory constraints are particular linear temporal logic formulae expressing constraints on possible actions in the plans and on intermediate states reached by the plans (such constraints are also known as "temporally extended goals" [2, 5]). Soft goals and constraints are preferences that we wish to satisfy in order to generate a good plan, but that do not have to be achieved in order for the plan to be correct. Strong plan constraints, in contrast, express properties that the acceptable plans must satisfy. Moreover, they allow the user to provide control knowledge to domain-independent planners supporting the extended PDDL language. By adding them as goal conditions, we can prevent a planner from exploring parts of the plan space, as, e.g., in [3, 48], possibly making its exploration faster or guiding the planner towards better quality solutions. This is not the way plan constraints were used in IPC5, but such possible use is another motivation for introducing them into PDDL.

Dealing with (strong or soft) plan trajectory constraints and soft goals poses a new challenge to fully automated planning. While soft constraints have been extensively studied in the CSP literature (e.g., [9, 24, 60]), only recently has the planning community started to investigate them [14, 15, 21, 53, 62, 63]. When using soft constraints and goals, it can be useful to give different importance to them. For this purpose, PDDL3 allows the domain modeler to assign different penalties to violated constraints and unachieved goals.

In order to make the language extensions more accessible for the competitors, IPC5 used a first version of PDDL3, called PDDL3.0, where we have imposed some simplifying restrictions, such as a limited form of modal operator nesting in the specification of trajectory constraints. While there is more than one way to specify the importance of a soft constraint or goal, as a first attempt to tackle this issue, in PDDL3.0 we have chosen a simple quantitative approach: each soft constraint and goal is associated with a numerical weight representing the cost of its violation in a plan (and hence also its relative importance with respect the other specified soft constraints and goals). Weighted soft constraints and goals are part of the plan metric expression, and the best quality plans are those optimising such an expression. Using this approach we can express that certain plans are preferred to others.

In order to evaluate the performance of the competing planners, the organisers of IPC5 developed several new planning domains and a large collection of new benchmark problems over these domains, that can also serve as a reference for future research. Some of the new domains are inspired by new applications of planning technology, e.g., to problems of molecular biology, or to known problems that have been investigated in other fields of computer science, such as the travelling purchaser problem studied in operations research.

A total of twelve planners entered IPC5. Even though they did not all attempt all of the problems, the size of the resulting data set is substantial. Given the limited amount of time available during the competition for analysing these results and assigning the awards, the organisers of IPC5 used an informal evaluation method similar to the one used for the previous competition [41], with the main difference that the evaluation criteria focused on the number of solved problems and plan quality, rather than CPU time and scalability.[1] The winners of IPC5 were: MAXPLAN and SATPLAN (version 2006) for the propositional optimal planning subtrack, and SGPLAN5 for satisficing (sub-optimal) planning

[1] A detailed description of the IPC5 evaluation criteria used to assign the IPC5 awards is available on the competition website: `ipc5.ing.unibs.it`.

subtrack [13].[2] In this paper, we analyse the performance of the IPC5 planners more rigorously, and much more in detail, in terms of their relative performance, advancement with respect to the state-of-the-art in fully-automated deterministic planning systems, and qualities of the solutions found for the IPC5 benchmarks.

In summary, the main contributions of our work are:

- An extension of the PDDL language that supports soft goals and soft and strong state trajectory constraints representing temporally extended goals;
- Some basic results about the expressiveness of PDDL3.0 and its compilability into the previous versions of the language;
- A detailed evaluation of the relative performance of the twelve IPC5 planners, for each domain category involving different fragments of PDDL3.0;
- An evaluation of the performance of the IPC5 winners with respect to the winners of the previous IPC and of the quality of solutions they computed;
- A collection of new benchmarks for testing planning algorithms and systems for problems specified with both PDDL3.0 and PDDL2.

The paper is organised as follows. Section 2 introduces PDDL3 focusing especially on PDDL3.0, and it gives some basic results about the compilability of the new features of PDDL3.0. In Section 3, we present the test domains that we developed for IPC5. In Section 4, after a very brief description of the IPC5 planners, we analyse in detail their performance. Finally, in Section 5, we summarise the results and give the conclusions.

2 The PDDL3 Language

The planning domain description language, PDDL, was first proposed by Drew McDermott for the first international planning competition in 1998 [36]. The language was based on Lisp syntax, using a structure based on the widely used variants of STRIPS notations. Establishing a common standard language has had a similar impact on planning research as the introduction of standards in other areas of research: it opens the route to stronger collaboration, exchange of tools, techniques and problems and provides a platform for comparative evaluation of approaches. The language has been, from the outset, strongly linked to the competition series, with developments in the language being seen as drivers for the direction of the competition challenges.

PDDL has been extended in several stages, in order to capture more expressive variants. The significance and impact of these changes is described below, in Section 2.1. There have been several explorations of the expressive power of the different variants of PDDL, mainly concentrated on the notion of compilability. Recent results include a demonstration that temporal features can be compiled away in polynomial work, subject to certain constraints on the forms of concurrency that can appear in the problem [59], while others have examined the compilability of conditional effects, timed initial literals and domain axioms [28, 54, 65].

For the fifth planning competition, PDDL was extended to include two important new features [31, 32]. The first is the ability to express goals that apply not only to the final state of the trajectory of states visited by a plan, but also to the intermediate states. These goals take the form of trajectory constraints, familiar from work on temporal logics. The second extension is the ability to express soft constraints, or preferences.

Both of these extensions to the language are motivated by the desire to see planning bridge the gap between research and application. Many real problems require the specification of goals that are more complex than be easily expressed in earlier versions of PDDL. These include constraints on the states that a plan visits as well as on the state in which it finishes. It can also be important to specify the

[2] The term "satisficing", introduced for planning in [41], has been largely adopted in the planning community for planners that do not offer any guarantee about the quality of the plans they compute. While some satisficing planners aim to find plans of good quality, many others ignore the quality aspect completely, aiming only to find a solution plan as quickly as possible.

relative benefits of different, perhaps conflicting, desirable conditions that a plan should satisfy, so that a plan might be constructed to evaluate these benefits against the costs of achieving them.

2.1 A Brief Review of PDDL

In order to provide the background that is required to place the discussions that follow in context, this section contains a short overview of PDDL. The key details of syntax and semantics of PDDL can be found in [29, 41].

PDDL allows actions to be described in terms of pre- and postconditions. The expressive levels of the language are associated with tags that are used to label domain files: the addition of a tag to a domain file indicates that the domain may use the corresponding syntax layer of the language. Preconditions can be simple conjunctions of atoms (or literals, if negative preconditions are allowed), or even arbitrary formulae (if quantification and ADL are allowed). Postconditions can contain add and delete effects and may use conditional effects, if allowed, and also quantification.

PDDL2.1 [29] extended the language to include number-valued fluents (with a corresponding "requirements" tag). A variant of these was included in the original PDDL specification, but had not been adopted. Two other important extensions were added in PDDL2.1, both relying on the use of numbers: plan metrics, which can be used to specify the way in which plans are to be evaluated in a specific problem instance, and durative actions. Durative actions are actions that execute over an interval of time. These can be of constant duration, of a duration determined by the state in which the action is executed or, most complex of all, of variable duration, which may be selected by the planner, possibly subject to constraints.

The use of durative actions implies that plans are embedded on a metric time line and, therefore, a plan must specify the time at which an action is to be executed. The structure of a durative action is equivalent to two standard (instantaneous) actions, one at the start of the durative action and one at the end, combined with an additional constraint — the action invariant. The start and end of the durative action can therefore have pre- and postconditions, each with the same semantics as the standard instantaneous actions. The start is applied at the time specified in a plan using the action and the end is then applied at the appropriate interval following this. The invariant is a logical condition (constrained by the same syntax limitations as preconditions) that must remain true throughout the interval over which the durative action is executing.

The introduction of durative actions into PDDL required that a decision be made about the structure of plans that do not use durative actions. It was proposed that, in all cases, plans would take the same form: time-stamped actions. Thus, STRIPS plans, in which time matters only for the ordering of actions, can be represented by simply labelling each of the actions with its position index in the plan, starting from 1. The consequence of this decision is that all PDDL plans are considered to be embedded in a real time line. This observation extends to plans for simple STRIPS problems which allows plans to have parallel actions. This semantics, which achieves the same effect as the semantics of GRAPHPLAN plans [10], is not the one that had been traditional for STRIPS plans before GRAPHPLAN, where even partial order plans were generally interpreted in terms of the set of possible serialisations of the partial orders.

The semantics of PDDL2.1 is discussed in detail in [29]. Essentially, a PDDL2.1 plan describes a trajectory of states, where states are valuations on the propositional and metric variables of the problem. The initial state is as specified for the planning problem. Transitions are caused by *happenings*, which are the collections of instantaneous actions (either simple actions in the domain or else the start or end points of durative actions) that occur at the same time points. It is worth emphasising that the semantics is uniform in its treatment of the end points of durative actions and instantaneous actions, so that the two kinds of actions can be mixed freely in a single plan, if it is considered appropriate to model a domain in this way. Each happening causes a state change according to the effects of the actions that occur at the corresponding time point. Invariants are checked in the intervals between happenings, where durative actions are executing.

In both cases (plans with STRIPS actions and plans with durative actions), it is possible for happenings to contain multiple instantaneous actions occurring together. In order to ensure that the behaviour is well-defined, it is required that these simultaneous actions be non-interfering. A simple paradigm is used to define the concept of interference, based on the observation that action effects can be seen as

analogous to data-base updates affecting the state: mutex locks. The idea is that, to access a particular variable, an action requires a lock — a read-lock if it simply needs to access the value of the variable (to check the satisfaction of a condition) and a write-lock if it must update the value (for an effect). Write-locks are mutually exclusive with any other kinds of lock by any other actions, while multiple read-locks are possible without inconsistency. Two consequences of this are that an action requires a write-lock even if the update it performs does not actually change the original value of the corresponding variable and two actions are considering interfering if they both update the same variable, even if they agree about the new value it should take. The only exception to this rule is in the use of certain commutative arithmetic effects: `increase` and `decrease` effects, in particular, are not considered to require independent write-locks to update a metric variable. The reason for this is discussed in detail in [29] and is not important to the remainder of the discussion in this paper.

For the purposes of IPC3, PDDL2.1 was considered to be split into "levels". These were not defined as part of the syntax of the language and are, essentially, identified with certain combinations of requirements tags. The levels that were used correspond to: simple STRIPS (level 1), domains with numeric fluents (level 2) and durative actions with discrete durations (level 3). Finally, level 4 contains the simple continuous process model that was proposed as part of PDDL2.1, although never used.

PDDL2.2 [26] extended the language still further, adding axioms, which allow derived propositions to be inferred from the satisfaction of logical formulae in a state, and timed initial literals, which specify effects that are triggered at predetermined times during the execution of the plan. These allow simple deterministic exogenous events to be modelled, such as sunrise and sunset at certain predefined times.

2.2 State Trajectory Constraints

State trajectory constraints assert conditions that must be met by the entire sequence of states visited during the execution of a plan. They are expressed through temporal modal operators over first order formulae involving state predicates. In this section we present the syntax and semantics of the extensions introduced in PDDL3.0. As will become clear, certain constraints have been placed on the ways in which the syntax can be exploited, in particular, in the nesting of modalities. Ultimately, the development of PDDL is a compromise between the goals of convenient expressive power, the needs of the competition, and the limits of the planning technology available at the time of the competition. One of the consequences of this compromise is that it is sometimes appropriate to add constraints that limit the problems that a planner must contend with, even if there are natural ways to allow the expressive power to be extended.

2.2.1 Syntax and Intended Meaning

The basic modal operators used in IPC5 are: `always`, `sometime`, `at-most-once` and `at end`. The last of these is used to identify conditions that must hold in the final state when a plan has executed, making them equivalent to traditional goal conditions. For convenience, therefore, unadorned goal conditions are assumed to be "at end" conditions. This assumption serves to preserve the standard meaning for existing goal specifications. The semantics of these modalities is given below (Section 2.2.2) along with examples of their use, but we will provide brief illustrations here to support intuitions about their use. For example, `(always (clear A))` expresses a condition that an object, A, must remain clear throughout a plan, `(sometime (clear A))` expresses that A must be clear at some point in the plan (not necessarily at the end) and `(at-most-once (clear A))` expresses that A can only be clear in at most one single unbroken period during execution of the plan.

The operator `within` is included to be used to express deadlines. For example, `(within 10 (clear A))` specifies that A must be clear by time 10. In addition, rather than allowing arbitrary nesting of modal operators (in the competition, at least), some specific combinations are encoded in explicit operators. These are: `sometime-before`, `sometime-after`, `always-within`. Other modalities could be added, but these are sufficiently powerful for an initial level of the sublanguage modelling constraints. Examples of the use of these are: `(sometime-before (clear A) (clear B))` specifies that if A is ever clear during the execution of a plan, then B must also have been clear before that point; `(sometime-after (clear A) (clear B))` is similar, except that it requires B to be clear *after* the

point at which A is clear. Finally, `(always-within 5 (clear A) (clear B))` specifies that every time A is made clear, B must be clear within 5 time units of that point in the execution of the plan.

Modal expressions can be combined in propositional formulae, but we limit their combination to conjunctions and universally quantified expressions (which can be considered equivalent to conjunctions since the models are all finite). PDDL3.0 does not support any syntactic nesting of modal operators. Allowing arbitrary nesting, or even depth-bounded nesting, of modalities creates a very rich collection of different constraints, most of which are unnecessary for the expression of very interesting problems. However, allowing them within the language would force the designer of a PDDL3.0 planner to consider how to deal with them. In order to arrive at an appropriate compromise between modelling expressiveness and competition challenge, it was decided that a collection of additional modalities, equivalent to specific nested structures of primitive modal operators, should be included as PDDL3.0 expressions. Thus, the limitation on nesting is a pragmatic decision intended to make the task for the competition entrants more tightly defined. An example of an expression that it is not possible to capture without nesting of modalities is `(sometime-after p (sometime-before q r))`, which asserts that *if* p is ever true in a state *and* q is true in a subsequent state, then r must be true in some state before the one in which q becomes true. This constraint cannot be captured using the existing modalities without nesting, unless additional encoding tricks are exploited that directly modify the actions of the domain. The extent to which the restrictions on the use of modalities limit what can be conveniently expressed is difficult to assess, since there is very little practical experience in the use of the language to express plan constraints. All that we can say is that the design of the benchmark problems, and the examples we considered, was not hindered in any way by the constraints we impose.

It should be noted that, by combining modalities with *timed initial literals* (defined in PDDL2.2 [41]), we can express further goal constraints. In particular, one can specify the interval of time when a goal should hold, or the lower bound on the time when it should hold. Since these are interesting and useful constraints, we introduce two modal operators as "syntactic sugar" over the basic language: `hold-during` and `hold-after`.

Trajectory constraints are specified in the planning problem file in a new field, called `:constraints`, that will usually appear after the goal. Constraints may also be specified in the action domain file. This is convenient for the expression of constraints that apply to all plans produced for a particular domain — perhaps legal or safety conditions on operating procedures. The use of trajectory constraints (in the domain file or in the goal specification) implies the need for the `:constraints` tag in the `:requirements` list.

No temporal modal operator is allowed in preconditions of actions. That is, all action preconditions are with respect to a state (or time interval, in the case of `over all` action conditions — the action invariants described earlier). This decision ensures that the set of actions applicable at any state is determined entirely by the state itself (which, of course, can contain a record of relevant parts of history in memory) and is not affected by the trajectory of states that precede or succeed this state. This "Markovian" requirement is consistent with our own view of what is an appropriate model of the way that actions are constrained by causal relationships in practice. However, there is also a very significant benefit which is to simplify the task, for a planner, of determining what choice of actions is open to it in a state. Without this constraint, the general problem of determining whether an action is applicable in a (fully specified) state is as hard as planning, since the conditions for execution could require arbitrary goals to be achieved in the past or the future of the current state. Indeed, without placing the state in the context of a trajectory, it is not clear whether the question of applicability of actions with modal preconditions even makes sense.

The following is a fragment of the grammar describing the new modalities of PDDL3.0 for expressing constraints (`con-GD`) (the full BNF grammar is given in [31, 33]):

```
<con-GD> ::= (at end <GD>) | (always <GD>) |
             (sometime <GD>) | (within <num> <GD>) |
             (at-most-once <GD>) |
             (sometime-after <GD> <GD>) |
             (sometime-before <GD> <GD>) |
             (always-within <num> <GD> <GD>) |
             (hold-during <num> <num> <GD> |
```

```
(hold-after <num> <GD> | ...
```

where <GD> is a goal description (a first order logic formula), <num> is any numeric literal (in STRIPS domains it will be restricted to integer values). In the interpretation of `within` and `always-within` when considering STRIPS plans (and similarly for `hold-during` and `hold-after`) the numeric bounds are counted in terms of plan *happenings*. For instance, (`within 10` ϕ) means that ϕ must hold within ten happenings. These can be happenings of one action or of multiple actions, depending on whether the plan is sequential or parallel.

Trajectory constraints allow specification of problems of a very different character to those captured by simple goal specifications in the same domain. For example, in the Blocks World, it is clear that there is a path from any state to any other state in a number of steps that is linear in the size of the problem specification. However, it is possible for a planner to be faced with trajectory constraints that prune the legal paths in such a way as to force exponential length plans to be required for some pairs of states. This can be seen as follows: suppose there are $n + 3$ blocks in a problem instance, named A, B and C and $b1, ..., bn$. By adding the constraints:

```
(and (always (on A table)) (always (on B table)) (always (on C table))
     (forall (?x - block)
         (always (or (= ?x A) (= ?x B) (= ?x C) (not (on ?x table)))))
     (always (not (on b1 b2))) (always (not (on b1 b3))) ...
     (always (not (on b1 bn))) (always (not (on b2 b3))) ...
     (always (not (on b3 b4))) ... )
```

to a Blocks World problem, we can force it to behave like the n-disc Towers of Hanoi problem, with blocks A, B and C playing the roles of the pegs and $b1...bn$ the discs. This problem only admits exponential solutions, but is captured in a collection of constraints that is quadratic in the size of the set of blocks. The semantics of the modal operator `always` is given formally below, but its use is consistent with intuition: the formula to which it is applied must hold in every state in order for the modal formula to hold over the trajectory.

A brief comment is required about the important decision not to include a modal operator for "next", which is used to significant effect in modal logics supported by existing planners, TALPLANNER [48] and TLPLAN [2]. In those planners formulae are often expressed using a "next" modality to trigger conditions at the point of change in a proposition, e.g., $(p \wedge \text{next}\, \neg p) \Rightarrow \text{next}\, \Phi$. Two problems led us to avoid adding the "next" modality to the language. Firstly, the fact that we do not allow nested modalities severely limits the context in which the "next" modality might be useful (the example just described is only really useful if it is nested inside an "always" modality). The second problem with attempting to capture "next" in PDDL3.0 is a consequence of the necessary separation of the time point at which the formula is evaluated and the time point to which the "next" modality is a reference: concurrent strands of activity can affect the state between these time points. This means that the next state change following the achievement of a particular condition could well be caused by a happening that is entirely irrelevant to the condition of interest. For example, we might consider attempting to express a constraint that when a truck arrives with a package at the destination of the package then in the next state the package should be unloaded. On the real time line, the next state change following arrival of the truck at its destination could be caused by a happening that affects an aircraft, say, in an entirely different part of the world and, most importantly, this could be an entirely appropriate next state, despite having no relevance to the next actions involving the truck. This observation does not prevent "next" being given a consistent semantics, but it does make its use in modelling less intuitive.

2.2.2 Semantics

The semantics of goal descriptors in PDDL2.2 determines that they should be evaluated only in the context of a single state (the state of application for action preconditions or conditional effects and the final state for top level goals). In order to give meaning to temporal modalities, which assert properties of trajectories rather than individual states, it is necessary to extend the semantics to support interpretation with respect to a finite trajectory (generated by a plan). The semantics of the modal operators is consistent with that used for modal operators in LTL and other treatments of modal temporal logic [52, 56].

Recall that a *happening* in a plan for a PDDL domain is the collection of all the instantaneous (start or end points of) actions that occur at the same time. This time is then the time of the happening, and a happening can be "applied" to a state by simultaneously applying all effects in the happening (which is well defined because no pair of such effects may interfere). The association of a real-valued continuous variable representing the time at which a state begins is an important difference from some treatments of temporal logics. The semantics is still based on the familiar conditions over sequences of states, but several modalities also depend on the values of these times.

Definition 1 *Given a* PDDL *domain D, a plan π and an initial state I, π generates the trajectory*

$$\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle$$

iff $S_0 = I$ and for each happening h generated by π, with h at time t, there is some i such that $t_i = t$ and S_i is the result of applying the happening h to S_{i-1}, and for every $j \in \{1 \ldots n\}$ there is a happening in π at t_j.

Note that there is intentionally no happening at time 0. The initial state holds at this time and must persist for a non-zero period of time, so the first happening is at time $t_1 > 0$.

Definition 2 *Given a* PDDL *domain D, a plan π, an initial state I, and a goal G, π is valid iff the trajectory it generates, $\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle$, where $S_0 = I$, satisfies the goal:*

$$\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models G.$$

This definition contrasts with the original semantics of goal satisfaction [29], where the requirement is that $S_n \models G$. The contrast reflects precisely the requirement that goals are now interpreted with respect to an entire trajectory. Action preconditions may not include modal operators, and therefore their interpretation continues to be relative to the single state in which the action is applied. The interpretation of simple formulae, ϕ (containing no modalities), in a single state S is unchanged and continues to be denoted $S \models \phi$. In the following definition we rely on context to make clear where we are using the interpretation of non-modal formulae in single states, and where we are interpreting modal formulae in trajectories.

Definition 3 *Let ϕ and ψ be atomic formulae over the predicates of the planning problem plus equality (between objects or numeric terms) and inequalities between numeric terms, and let t, u_1 and u_2 be any real constant values. The interpretation of the modal operators is as specified in Figure 1.*

Note that this interpretation exploits the fact that modal operators are not nested. A more general semantics for nested modalities is a straight-forward extension of this one. Note also that the last four expressions in Figure 1 are expressible in different ways if one allows nesting of modalities and use of the standard LTL modality until. Taking (until ϕ ψ) to mean that there is a state in which ψ is true and in all states before this (if any) ϕ is true. The modality weak-until is also occasionally used, where (weak-until ϕ ψ) is taken to mean that ϕ is true in all states before some state in which ψ is true, *if there is one* (otherwise ϕ is always true). The following equivalences can be proved (amongst many others — indeed, until is sufficient to capture all other modalities that do not have numeric arguments [19]):

$$\begin{gathered}
(\text{weak-until } \phi\ \psi) \equiv (\text{until } \phi\ (\psi \vee\ (\texttt{always}\ \phi))) \\
(\texttt{always-within}\ t\ \phi\ \psi) \equiv (\texttt{always}\ (\phi \rightarrow (\texttt{within}\ t\ \psi))) \\
(\texttt{sometime-before}\ \phi\ \psi) \equiv (\text{weak-until } (\neg\phi \wedge \neg\psi)\ (\psi \wedge \neg\phi)) \\
(\texttt{at-most-once}\ \phi) \equiv (\texttt{always}\ (\phi \rightarrow (\text{weak-until } \phi\ (\texttt{always}\ \neg\phi)))) \\
(\texttt{sometime-after}\ \phi\ \psi) \equiv (\texttt{always}\ (\phi \rightarrow (\texttt{sometime}\ \psi)).
\end{gathered}$$

The constraint `at-most-once` is satisfied if either its argument is never true (so the implication in the above equivalence is trivially satisfied because the antecedent never holds) or else, once it becomes true, it remains true until a state is reached in which the proposition becomes *and remains* false. That is, once the proposition first becomes false, after having been true, it must remain false thereafter, allowing at most one *interval* in the plan over which the argument proposition is true. An example of the use of this modality is in the following: "Each truck should visit each city *at most* once":

$\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{at end}\ \phi)$
iff $S_n \models \phi$

$\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models \phi$
iff $S_n \models \phi$

$\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{always}\ \phi)$
iff $\forall i : 0 \leq i \leq n \cdot S_i \models \phi$

$\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{sometime}\ \phi)$
iff $\exists i : 0 \leq i \leq n \cdot S_j \models \phi$

$\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{within}\ t\ \phi)$
iff $\exists i : 0 \leq i \leq n \cdot S_i \models \phi$ and $t_i \leq t$

$\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{hold-after}\ t\ \phi)$
iff if $t_n > t$ then $\exists i : 0 \leq i \leq n \cdot S_i \models \phi$ and $t_i > t$,
if $t_n \leq t$ then $S_n \models \phi$

$\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{hold-during}\ u_1\ u_2\ \phi)$
iff if $t_n > u_1$ then
$\forall i \cdot 0 \leq i \leq n \cdot$ if $u_1 \leq t_i < u_2$ then $S_i \models \phi$,
$\forall j \cdot 0 \leq j < n \cdot$ if $t_j \leq u_1 < t_{j+1}$ then $S_j \models \phi$
if $t_n \leq u_1$ then $S_n \models \phi$

$\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{at-most-once}\ \phi)$
iff $\forall i : 0 \leq i \leq n \cdot$ if $S_i \models \phi$ then $\exists j : j \geq i \cdot \forall k : i \leq k \leq j \cdot S_k \models \phi$
and $\forall k : k > j \cdot S_k \models \neg\phi$

$\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{sometime-after}\ \phi\ \psi)$
iff $\forall i \cdot 0 \leq i \leq n \cdot$ if $S_i \models \phi$ then $\exists j : i \leq j \leq n \cdot S_j \models \psi$

$\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{sometime-before}\ \phi\ \psi)$
iff $\forall i \cdot 0 \leq i \leq n \cdot$ if $S_i \models \phi$ then $\exists j : 0 \leq j < i \cdot S_j \models \psi$

$\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{always-within}\ t\ \phi\ \psi)$
iff $\forall i \cdot 0 \leq i \leq n \cdot$ if $S_i \models \phi$ then $\exists j : i \leq j \leq n \cdot S_j \models \psi$ and $t_j - t_i \leq t$

Figure 1: Semantics of the basic modal operators in PDDL3.0. ϕ and ψ stand for arbitrary (syntactically valid) goal formulae of PDDL3.0; t, u_1 and u_2 are real values.

```
(:constraints
    (and (forall(?t - truck ?c - city) (at-most-once (at ?t ?c))) ...))
```

To satisfy this constraint, each truck may visit each city and stay there any length of time, but once it leaves it cannot return during the execution of the plan.

Of the constraints `hold-during` and `hold-after`, `(hold-during` t_1 t_2 ϕ`)` states that ϕ must be true in every state during the interval $[t_1, t_2)$, while `(hold-after` t ϕ`)` states that ϕ must be true in some state after time t. The first can be expressed by using timed initial literals to specify that a dummy timed literal `d` is true during the time window $[t_1, t_2)$ together with the goal `(always (implies d` ϕ`))`. A variant of `hold-during` where ϕ must hold *exactly* during the specified interval could be easily obtained in a similar way. The `hold-after` modality can be expressed by using timed initial literals to specify that `d` is true (only) from time t, together with the goal `(sometime (and d` ϕ`))`.

The modal operators `within` and `always-within` are of particular interest. An example of a constraint using `always-within` is the following: "Whenever the energy of a rover is below 5, it should be at the recharging location within 10 time units":

```
(:constraints
  (and (forall (?r - rover)
       (always-within 10 (< (energy ?r) 5) (at ?r recharging-point))) ...))
```

This modality is interesting because it highlights the way that the semantics relies on the time associated with the achievement of individual states. Another example is the following:

```
(:constraints
    (and (forall (?t - truck ?p - package ?l - location)
          (always-within 10 (and (at ?t ?l) (in ?p ?t) (destination ?p ?l))
                            (at ?p ?l))) ...))
```

This condition requires that any time a truck carrying a package arrives at the location which is the destination of the package, then the package must be delivered within ten time units. The time limit can be manipulated to ensure that the only behaviour possible is to immediately unload the truck following its arrival at a particular location.

2.3 Soft Constraints and Preferences

A soft constraint is a condition on the trajectory generated by a plan that the user would prefer to see satisfied, but is prepared to accept might not be satisfied because of the cost involved, or because of conflicts with other constraints or goals. While soft constraints have been extensively studied in the constraint-satisfaction literature [9, 24, 60]), the planning community has started to consider them only relatively recently (see, for example, [8, 14, 15, 21, 53, 62, 63]).

There is still contention about the best way to capture and handle preferences, with some advocating a reward-based approach (e.g., [12]) and others advocating a qualitative approach (e.g., [37]). In particular, where a user has multiple soft constraints, there is a need to determine which of the various constraints should take priority if there is a conflict between them, or if it should prove costly to satisfy them. This can be expressed using a qualitative approach, for example by describing a partial order on the conditions that are preferred. The advantage of this approach is that it is intuitive and consistent with the demands of many potential applications. Unfortunately, it is also highly inconsistent with the demands of straightforward comparative evaluation of planner performance, since the use of a partial order introduces the complication of there being many incomparable plans, each maximally preferable. To avoid this problem (which is particularly acute in a competition context), PDDL3.0 uses quantitative preferences.

An example of the expressions we wish to capture is the following: "We prefer that every fragile package is insured while it is loaded in a vehicle".

```
(:constraints
   (and (forall (?p - package)
         (preference P1 (always (implies (and (fragile ?p) (loaded ?p))
                                                 (insured ?p))))) ...))
```

This example illustrates the power of combining preferences and trajectory constraints.

2.3.1 Syntax and Intended Meaning

The syntax for soft constraints falls into two parts. Firstly, there is the identification of the soft constraints, and secondly there is the description of how the satisfaction, or violation, of these constraints affects the quality of a plan.

Goal conditions, including action preconditions, can be labelled as preferences, meaning that they do not have to be true in order to achieve the corresponding goal or precondition. Thus, the semantics of these conditions is simple, as far as the correctness of plans is concerned: they are all trivially satisfied in any state. The role of these preferences is apparent when we consider the relative quality of different plans. In general, we consider plans better when they satisfy soft constraints and worse when they do not. Complications arise, however, when comparing two plans that satisfy different subsets of constraints (where neither set strictly contains the other). In this case, we rely on a specification of the violation costs associated with the preferences.

The syntax for labelling preferences over goal descriptors is simple: `(preference [name] <GD>)` (similarly for preferences over trajectory constraints). The definition of a goal description can be extended to include preference expressions. However, expressions in which preferences appear nested inside any connectives, or modalities, other than conjunction and universal quantifiers, are prohibited in PDDL3.0. Preferences appearing in the condition of a conditional effect are also invalid. Where a named preference appears inside a universal quantifier, it is considered to be equivalent to a conjunction (over all legal instantiations of the quantified variable) of preferences all with the same name.

The use of preferences in a domain or problem implies the need for the requirements tag `:preferences`. Preferences over state trajectory constraints are expressed in the `(:constraints ...)` field, while

preferences over goals are expressed in the `(:goal ...)` field. If a preference involves both a constraint and a goal, it is expressed in the `:constraints` field. Goal preferences expressed in the `:goal` field are implicitly interpreted under the `at end` modality.

Preference names can be used to refer to the preference in the construction of penalties for the violated constraint. Preferences with the same name share the same penalty.

Penalties for violation of preferences are calculated using the expression `(is-violated <name>)`, where `<name>` is a name associated with one or more preferences. This expression takes a value equal to the number of distinct preferences with the given name that are not satisfied in the plan. In PDDL3.0 there are no degrees of satisfaction of a soft constraint — a constraint is satisfied or not. The violation count includes every separate instance of a constraint with the same name. This means that:

```
(preference VisitParis (forall (?x - tourist) (sometime (at ?x Paris))))
```

yields a violation count of 1 for `(is-violated VisitParis)`, if at least one tourist fails to visit Paris during a plan, while

```
(forall (?x - tourist) (preference VisitParis (sometime (at ?x Paris))))
```

yields a violation count equal to the number of people who failed to visit Paris during the plan. The intention behind this is that each preference is considered to be a distinct preference, satisfied or not independently of other preferences. The naming of preferences is a convenience to allow different penalties to be associated with violation of different constraints.

Plans are awarded a value through the plan metric, introduced in PDDL2.1. The constraints can be used in weighted expressions in a metric. For example:

```
(:metric minimize (+ (* 10 (fuel-used)) (is-violated VisitParis)))
```

would weight fuel use as ten times more significant than violations of the `VisitParis` constraint.

The violation of a preference in the preconditions of an action is counted as often as the action occurs in the plan. For instance, suppose that `p` is the name of a preference in the precondition of an action a, and that a occurs three times in plan π, with the preference unsatisfied in each case. If the plan metric evaluating π contains the term `(* k (is-violated p))`, then this term will contribute $3k$ to the plan metric, since each instance of the action is considered to introduced a distinct instance of the preference.

Anonymous constraints (constraints for which no name is provided) are automatically considered to be weighted 1, and are included as an implicit additional additive term in the metric, positively if the metric is to be minimised and negatively if is to be maximised. This ensures that a plan that satisfies more constraints will be better than one that satisfies fewer, all else being equal. The default treatment of anonymous constraints can be avoided simply by naming the constraints — a named constraint contributes to the plan quality value only if it appears explicitly as a term in the metric.

2.3.2 Semantics

The expression:

$\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models (\texttt{preference}\ \Phi)$

is always true, so this allows preference statements to be combined in formulae expressing goals without changing the states in which the goals are true. A preference is a soft constraint, so failure to satisfy it is not considered to falsify the goal formula. In the context of action preconditions, $S_i \models (\texttt{preference}\ \Phi)$ is always true, too, for the same reason.

A preference `(preference Φ)` is *satisfied* iff $\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models \Phi$ and *violated* otherwise. To illustrate the interpretation of preferences take, as an example, the goal:

```
(and (at package1 london) (preference (clean truck1)))
```

which leads to the following interpretation (the lack of any explicit modality for the proposition in the preference means that it is to be interpreted as a required condition of the final state):

$\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models$ `(and (at package1 london) (preference (clean truck1)))`

iff $\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models$ `(at package1 london)` and
$\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models$ `(preference (clean truck1))`

iff $S_n \models$ `(at package1 london)`

iff `(at package1 london)` $\in S_n$, since the preference is always interpreted as true. In addition, the preference would be *satisfied*

iff $\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models$ `(at end (clean truck1))`

iff `(clean truck1)` $\in S_n$.

Now suppose that we have the following preferences and plan metric:

```
(preference p1 (always (clean truck1)))
(preference p2 (and (at end (at package2 paris)) (sometime (clean truck1))))
(preference p3 (at most once (in packeage2 truck1)))

(:metric minimize (+ (* 10 (is-violated p1)) (* 5 (is-violated p2))
                     (is-violated p3))).
```

Suppose we have two plans, π_1, π_2, and π_1 does not satisfy preferences p1 and p3 (but it satisfies preference p2) and π_2 does not satisfy preferences p2 and p3 (but it satisfies preference p1), then the metric for π_1 would yield a value (11) that is higher than that for π_2 (6), and we would say that π_2 is better than π_1.

The task of determining whether a preference is violated is simplified by a restriction in the language that allows preferences to appear only in conjunctions or universally quantified formulae. To see why this constraint is necessary, consider the example formulae: `(or` Φ `(preference` Ψ`))` and `(preference (or` Φ Ψ`))`. Under one natural interpretation, these formulae are equivalent both in terms of the satisfaction of the formulae and also in terms of whether the preference is satisfied. This happens if we consider the first formula to mean that Φ should be true but, failing that, it is *preferable* that Ψ be true (rather than not true). With this interpretation, in the state in which Φ holds but Ψ does not there is no violation, since the preference is irrelevant once Φ is satisfied. This interpretation has the property that it makes distinct the meanings of `(or` Φ `(preference` Ψ`))` and `(and` Φ `(preference` Ψ`))`. This apparently natural interpretation would lead to a situation in which the violation count for the preference, Ψ, would be incremented only if Φ were false. Unfortunately, it opens up a significant complication: to be consistent, the expression `(or (preference p1` Φ`) (preference p2` Ψ`))` should mean that only one of the two preference violation counts should be incremented. The problem is to decide which. One possibility would be to assign the violation to the least costly preference, measured according to the plan metric, but it seems a decidedly less natural interpretation to require to take into account the plan metric in order to decide which preference has been violated. Since disjunctions involving preferences, and formulae that are equivalent to disjunctions including preferences, do not behave intuitively, they have been excluded from the language.

The same interpretation of preferences is applied to action preconditions that include them. Formally, *a preference precondition is satisfied if the state in which the corresponding action is applied satisfies the preference*. The restriction on where preferences may appear in precondition formulae and goals, together with the fact that they are excluded from conditional effects, means that this definition is sufficient: the context of their appearance will never make it ambiguous whether it is necessary to determine the status of a preference. Similarly, a goal preference is satisfied if the proposition it contains is satisfied in the final state. Finally, an invariant (`over all`) condition of a durative action is satisfied if the corresponding proposition is true throughout the duration of the action — once the invariant is violated, the preference is unsatisfied, regardless of whether it is then resatisfied or violated again in multiple disconnected intervals.

In some cases it can be hard to combine preferences with an appropriate weighting to achieve the intended balance between soft constraints and other factors that contribute to the value of a plan (such as plan makespan, resource consumption and so on). For example, to ensure that a constraint takes priority over a plan cost associated with resource consumption (such as makespan or fuel consumption) is particularly tricky: a constraint must be weighted with a value that is higher than any possible consumption cost and this might not be possible to determine. With non-linear functions it is possible to achieve a bounded behaviour for costs associated with resources. For example, if a constraint, C, is to be considered always to have greater importance than the makespan for the plan then a metric could be defined as follows:

```
(:metric minimize (+ (is-violated C) (- 1 (/ 1 (+ 1 (total-time))))))
```

This metric will always prefer a plan that satisfies C, but will use makespan to break ties.

2.4 On the Expressiveness of PDDL3.0

The question of whether an extension of a planning language increases the expressive power of the original language can be addressed by studying the compilability of the extended language into the original one. As argued by Nebel and others [4, 54, 65], a compilation scheme should preserve solution existence, and it is theoretically important if it does not increase the size of the problem description more than polynomially or the size of the smallest solution by more than a constant: if a compilation scheme satisfying these conditions exists, then, from a theoretical point of view, we can say that the new language constructs do not add expressive power, and hence are merely "syntactic sugar". Of course, they might nevertheless be *useful*, by, for example, making it easier to model or solve certain kinds of problems.

The question of whether the new constructs introduced in PDDL3.0 increase the (theoretical) expressiveness of the language is not trivial. Several methods for compiling different forms of state trajectory constraints and preferences have appeared in the literature [5, 20, 25, 45, 58]. Indeed, some planners participating in the competition took this approach to handling the extended language. However, while such compilations preserve the existence of plans in the traditional sense, i.e., finite sequences of actions, we will show they do not, in fact *can not*, preserve existence of other forms of plans. Additionally, details of the complexity of the different compilation schemes proposed (size of the input planning problem description and of the output solution plans) have not been analysed.

This section contains some basic results about the compilability of PDDL3.0 state trajectory constraints and preferences. For non-temporal domains, where the actions are instantaneous and time corresponds to the happenings determined by the occurrence of actions in the plan, we show that this fragment of PDDL3.0 can be compiled into PDDL2 with a polynomial increase in problem size and constant increase in plan length. Thus, as argued earlier, we may claim that, for non-temporal domains, these constructs do not add expressive power to the PDDL language. However, we also show that this, and other, compilation schemes preserve only the existence of finite, sequential plans, i.e., there exist planning problems, with state trajectory constraints, that have, for example, plans with parallel actions but no sequential plan. Hence, in this particular case, we can also say that PDDL3.0 adds expressive power to PDDL2. Regarding state trajectory constraints for temporal domains, we outline a possible compilation scheme, which, however, increases the plan size linearly. In this case a compilation into PDDL2 preserving plan size exactly seems impossible. Moreover, we show how preferences (soft goals and soft state trajectory constraints) can be restated using numeric state variables (fluents) or in a more restricted form using action costs. Finally, in the last part of this section, we discuss some practical aspects of the usefulness of compiling PDDL3.0 constraints versus not compiling them.

2.4.1 Compiling State Trajectory Constraints for non-Temporal Domains

State trajectory constraints for a non-temporal domain can be restated as formulae in Linear Temporal Logic (LTL), which can be compiled into equivalent Büchi automata [18, 35]. Since PDDL3.0 constraints are normally evaluated over finite trajectories, the Büchi acceptance condition, that "an accepting state is visited infinitely often", reduces to the standard acceptance condition that the automaton is in

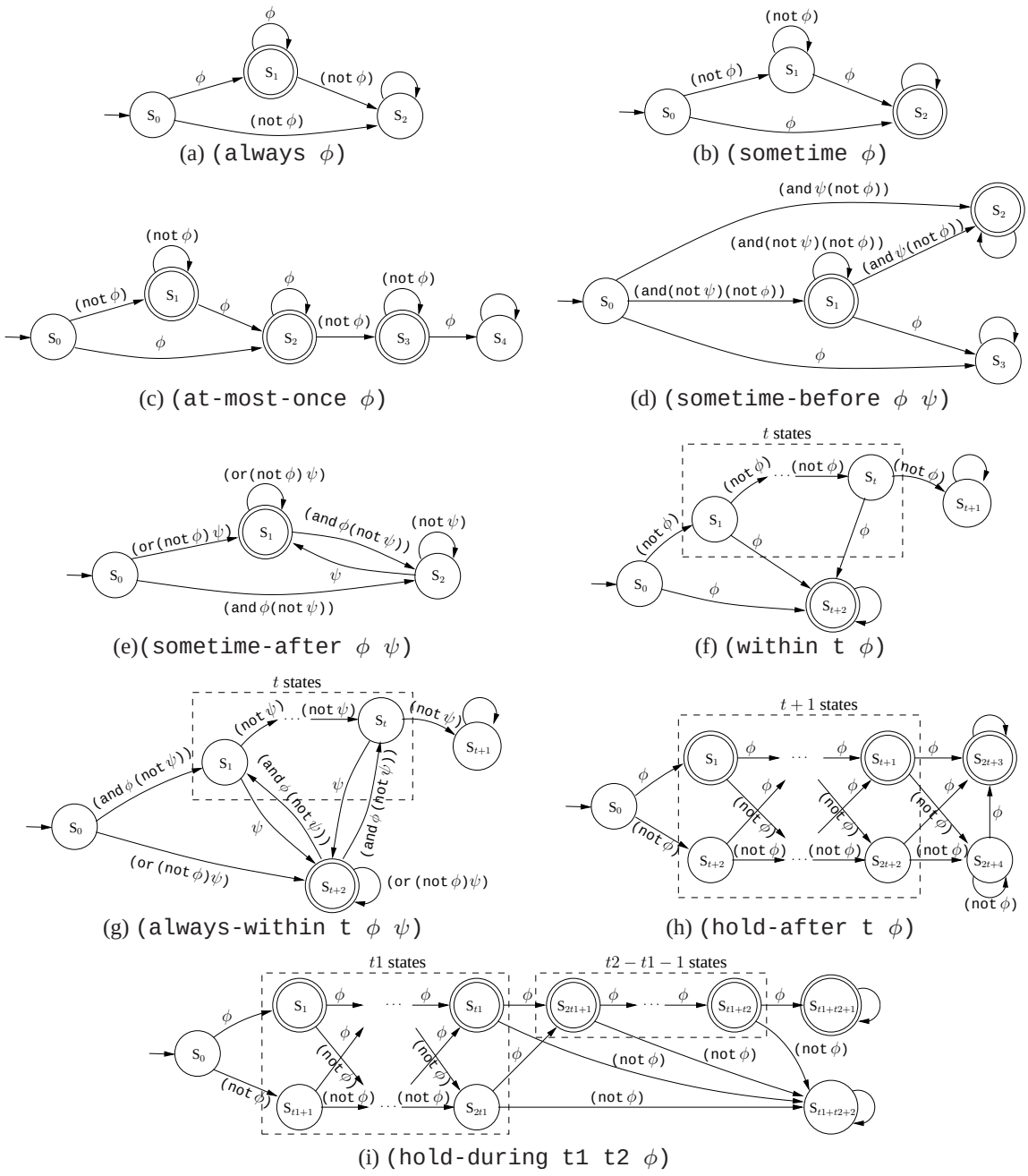

Figure 2: Automata corresponding to the basic PDDL3.0 state trajectory constraints. ϕ and ψ are arbitrary PDDL formula. t, t_1 and t_2 are real values.

an accepting state at the end of the trajectory. A straightforward approach to compiling away PDDL3.0 state trajectory constraints is thus to compile them into finite automata, and ensure that any valid plan correctly simulates the automata. The compilation schemes used by participants in IPC5 [5, 25] are variants of this idea.

The scheme we use is inspired by, and very similar to, the IPC5 planners MIPS-XXL [25] and HPLAN-P [5]. However, since our purpose is to study the expressiveness of PDDL3.0, we use a different encoding of automata, which enables us to derive explicit bounds on the growth in size of the compiled problem and its solutions.

In general, the compilation of LTL formulae may produce exponentially larger automata. However, because PDDL3.0 does not allow arbitrary nesting of modal operators, the automata corresponding to each of the basic plan constraints have fixed forms, which depend only on the modal operator: Figure 2 shows the automata for PDDL3.0 modal operators for non-temporal domains. The `within`, `always-within`, `hold-after` and `hold-during` operators are special, in that the number of states and transitions of the corresponding automata grow with the integer parameter t (resp. $t1$ and $t2$). These automata can be reformulated as finite automata augmented with finite-range binary counters [39], of size proportional to $\log(t)$ (resp. $\log(t1) + \log(t2 - t1 - 1)$) and with a constant number of distinct transitions. Below, we describe the compilation scheme only for automata without counters, since the encoding of finite integer counters by propositions make action conditions and effects more complex.

Without loss of generality, we assume that the problem description contains two distinguished actions, `start` and `finish`, that must appear first and last, respectively, in any valid plan.[3] The effects of the `start` action assert the initial facts of the problem, while the precondition of the `finish` action includes the problem goal. To enforce a trajectory constraint, the planning problem is modified in such a way that any valid plan simulates the execution of the corresponding automaton on the state sequence, and ensures that it ends in an accepting state. Let A be an automaton: the state of A is represented by a predicate `(state-A ?s)`, whose argument is drawn from a collection of additional constants. The `start` action asserts `(state-A s0)`, and the goal requires A to be in an accepting state. Since an automaton can have more than one accepting state, to avoid using disjunction in the goal, we also add a predicate `(accepting-A)`, which is made true whenever A is in an accepting state and is false otherwise. To ensure that the automaton is correctly updated throughout the plan, each action in the (original) planning problem and the special `finish` action is equipped with a set of conditional effects, one for each (non-looping) transition in A,

```
(when (and (state-A si) ‘‘TRANS-LABEL’’) (and (not state-A si) (state-A sj)))
```

where `TRANS-LABEL` is the formula labelling the transition from state s_i to state s_j of the automaton. For transitions to an accepting (resp. non-accepting) state, we also add the extra effect `(accepting-A)` (resp. `(not (accepting-A))`). Because the formula labels of transitions out of each state are mutually exclusive and exhaustive, exactly one of the conditional effects will take place whenever the action is performed. As an example, consider the state trajectory constraint `(sometime (at Plane NY))` in the well-known `Zenotravel` domain [50, 55]. To simulate the corresponding automaton (an instance of the one in Figure 2(b)), all actions in the domain are augmented with three conditional effects:

```
(when (and (state-A S0) (not (at Plane NY)))
 (and (not (state-A S0)) (state-A S1) (not (accepting-A))))
(when (and (state-A S1) (at Plane NY))
 (and (not (state-A S1)) (state-A S2) (accepting-A)))
(when (and (state-A S0) (at Plane NY))
 (and (not (state-A S0)) (state-A S2) (accepting-A))).
```

In the compiled problem, the state of the automaton will be updated to reflect the planning world state before the action takes place, i.e., the automaton will be "one step behind". This is because the automaton transitions simulated by the execution of a plan action are triggered by the world state where the action is executed, not by the world state modified by the effects of the action.[4] To ensure that the complete state sequence is indeed accepted by the automaton, the updating conditional effects are added also to the special `finish` action, and the condition that the automaton is in an accepting state placed in the problem goal rather than the precondition of this action.

Note that the conditional effects updating the states of the automata also make each action mutually exclusive with every other action (according to PDDL2 definition of mutex actions [29]), and hence force the plan to be sequential.

[3] This can be ensured by the addition of three dummy propositions, `(init)`, `(goal)` and `(active)`, such that `(init)`, and nothing else, holds in the initial state, and is required and deleted by `start`. `(goal)` is added by `finish` and required to hold in any goal state. `(active)` is added by `start`, deleted by `finish`, and is a precondition of every action except `start`.

[4] In the context of our `Zenotravel` example, assume that `(at Plane Boston)` holds in the problem initial state; if `(fly Plane Boston NY)` is the first plan action, this action updates the state of the automaton to `S1` and not to the accepting state `S2`.

Because modalities are not nested, the number of states and transitions in the automaton corresponding to a single basic constraint is bounded by a constant (assuming automata corresponding to constraints involving explicit time steps are reformulated with binary counters). Thus, the only place where the constraint formula enters the automaton is in the transition labels, and therefore the PDDL encoding of the automata outlined above grows only linearly with the size of the formula. However, the PDDL encoding of automata with counters also grows linearly with the number of bits required to represent the counters (i.e., logarithmically with the integer parameters t, resp. t_1 and t_2).

To extend the construction to universally quantified constraints, while keeping growth polynomial, it is sufficient to make two observations: First, given a universally quantified basic constraint, the construction can be "lifted", i.e., the predicates representing the automaton state are parameterised by the quantified variables and the updating conditional effects are universally quantified over the same set of variables (this was noted also by Baier & McIlraith [5]). Second, given a conjunction of several (possibly quantified) basic constraints, the updating conditional effects relating to different (possibly parameterised) automata are non-interfering, and therefore can all be carried out in parallel, by adding all the effects to each action (including the special `finish` action). In this way, all ground instances of the automaton are simulated in parallel. (This lifting also requires a universally quantified initialisation of the automata states, which can be encoded by a universally quantified effect of the special `start` action, and universally quantified goals for the compiled problem imposing that every automaton is in an accepting state.)

Consider again our `Zenotravel` example, and the quantified constraint `(forall (?x - aircraft) (sometime (at ?x NY)))`. Then, action `fly` is augmented by three quantified conditional effects

```
(forall (?x - aircraft)
 (when (and (state-A ?x S0) (not (at ?x NY)))
  (and (not (state-A ?x S0)) (state-A ?x S1) (not (accepting-A ?x)))))
(forall (?x - aircraft)
 (when (and (state-A ?x S1) (at ?x NY))
  (and (not (state-A ?x S1)) (state-A ?x S2) (accepting-A ?x))))
(forall (?x - aircraft)
 (when (and (state-A ?x S0) (at ?x NY))
  (and (not (state-A ?x S0)) (state-A ?x S2) (accepting-A ?x))))
```

In summary, the increase in the size of the compiled problem is at most proportional to $C \cdot log_2(t) \cdot M \cdot N \cdot O$, where C is a constant (the number of transitions in the largest automaton corresponding to a basic constraint), t the maximum integer parameter appearing in a `within`, `always-within`, `hold-during` or `hold-after` constraint, M the size of the (largest) formula appearing inside a basic constraint, N the number of basic constraints (conjuncts) in the problem and O is the number of operators in the domain. A shortest plan for the compiled problem is exactly 2 actions longer than the length (number of actions) of a shortest plan for the original problem. This increase in length is due to the introduction of the special `start` and `finish` actions.

2.4.2 Two Non-Compilability Results

It is not difficult to see that the compilation scheme outlined in the preceding section preserves the existence of plans, in the sense that if there exists a finite executable action sequence satisfying the constraints (and goals) of the original planning problem, then there exists also such a valid plan for the compiled problem. However, if we consider a slightly wider notion of plan, we find that there are temporally extended goals expressible in PDDL3.0 that can not be stated in the STRIPS/ADL fragment of PDDL: one, perhaps not so interesting, example is goals that can be satisfied only by infinite plans, but another, perhaps more relevant, example is goals that can only be satisfied by plans in which some actions happen in parallel.

The first example, a goal requiring an infinite plan, is a well known example in LTL: $\Box\Diamond p \wedge \Box\Diamond\neg p$, which can be expressed in PDDL3.0 as

```
(and (sometime-after (p) (not (p))) (sometime-after (not (p)) (p))).
```

This constraint is satisfied by a state sequence where a state in which p is true is always (eventually) followed by a state where p is false, and vice versa. Since p can not be both true and false in the same state, only an infinite sequence of states alternating between p and $\neg p$ can satisfy it. That a goal requiring infinite plans can not be expressed in STRIPS/ADL is obvious, since the goal can only refer to the final state reached by the plan.

Non-temporal PDDL domains have the property that any linearisation of a valid parallel plan is also a valid plan. This implies that if a problem has a solution plan, it also has a plan that is strictly sequential. However, the same is not true for propositional PDDL3.0: *using state trajectory constraints, it is possible to specify problems having a plan that involves parallel actions, but no sequential plan.* Intuitively, this is because constraints are evaluated over the sequence of "intermediate states" generated by a plan, and a linearisation of a parallel plan can pass through some states that the parallel plan does not. For a simple example, consider a planning problem with the following two actions:

```
(:action a1 :precondition (p1) :effect (and (not (p1)) (q1)))
(:action a2 :precondition (p2) :effect (and (not (p2)) (q2)))
```

where `(p1)` and `(p2)` are initially true and the goal is `(and (q1) (q2))`. Clearly, the two sequences $\langle$`a1`, `a2`$\rangle$ and $\langle$`a2`, `a1`$\rangle$ are both valid plans, as is the plan that executes `a1` and `a2` in parallel. Now consider the plan constraint

```
(always (or (and (p1) (p2)) (and (q1) (q2)))).
```

This constraint is violated by both the sequential plans for the above problem, but is satisfied by the parallel plan.

2.4.3 Compiling State Trajectory Constraints for Temporal Domains

In a temporal planning domain (a domain with durative actions), state trajectory constraints not involving explicit time points (i.e., those of type `sometime`, `always`, `sometime-after`, `sometime-before` and `at-most-once`) can be compiled away using the scheme shown in the previous section with only a minor modification: the collection of conditional effects must be added to both the start and end effects of each action. As noted above, this prevents any pair of such effects from occurring at the exact same time, which means that in the compiled problem, no pair of actions may start or end concurrently. Note however, that it is only the endpoints of actions that need to be separated (and only by the infinitesimal amount ϵ required by PDDL2.1 semantics); actions themselves may still overlap. Thus, the compilation does not change minimal plan makespan by more than $O(\epsilon)$.

Constraints of type `within`, `hold-after` and `hold-during` can be easily encoded in PDDL2.2 by using *timed initial literals* (TILs), representing predictable (deterministic) exogenous events [25, 31, 41], which in turn can be encoded in PDDL2.1 by the compilation scheme described in [41]. Intuitively, `within`, `hold-after` and `hold-during` can be compiled into TILs because the exact absolute times when the formula appearing inside these constraints must be true is defined by their semantics independently from the plan where they should hold. However, this does not hold for constraints of type `always-within`, which require a more intricate encoding. Intuitively, the sources of this difficulty are that in the encoding we have to verify conditions over continuous time, and PDDL does not admit temporal constraints in action conditions.

In the following we outline a possible compilation scheme for `always-within` constraints in the context of temporal domains.

Each `always-within` constraint is represented by a timed automaton (see Figure 3). As for non-temporal domains, the automata execution is simulated by the execution of the plan, but in this case we synchronise the execution with the happenings of the plan, instead of with the plan actions. Hence, in the compilation of an `always-within` constraint, instead of augmenting the domain actions with conditional effects representing the transitions of the timed automaton, we add a new dummy action having these conditional effects, and we force such an action to happen immediately after each happening of the plan (this can be done by using the technique based on the "clip" actions introduced by Fox and colleagues [28]). Moreover, for each `always-within` constraint with metric time t, we add another

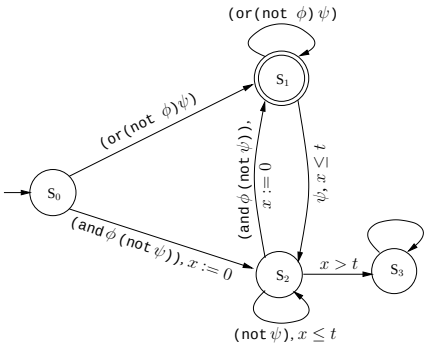

Figure 3: Timed automaton for representing `(always-within` $t\,\phi\,\psi$`)` constraints in temporal domains.

special action with duration t increasing a numerical fluent y (initialised to zero) by one at the beginning of the action and decreasing it by one at its end, and we force the action to occur in the plan at each time when an automaton transition resets the clock to zero. Essentially, this special action is used to deal with the temporal constraints labelling the automaton transitions, which in PDDL cannot be explicitly represented as action preconditions: in any state, the value of y is the number of clock resets that have occurred in the last t time units. Thus, if $y > 0$, the time elapsed since the last clock reset is less than or equal to t (i.e. condition $x \leq t$ labelling transitions of Figure 3 holds), while if $y = 0$, it is greater than t (i.e. condition $x > t$ in Figure 3 holds).

It can be shown that the outlined compilation scheme increases the size of the problem description polynomially. However, it does not preserve the number of plan actions exactly. Intuitively, a plan of the compiled problem can reset the automaton clock $O(H)$ times (up to once every two consecutive happenings), where H is the number of the plan happenings. Since in a plan with K actions we have $O(K)$ happenings, the number of additional actions in a solution plan of the compiled problem is proportional to $C \cdot K$, where C is the number of state trajectory constraints in the original problem. This increase could make constructing the plan computationally more expensive. On the other hand, since actions may overlap in the compiled problems as well, their makespan in unaffected (except for an $O(\epsilon)$ quantity, as noted above).

2.4.4 Compilation of Soft Goals and Constraints

PDDL3.0 preferences allow a plan metric to be expressed in terms of the satisfaction of soft goals, state trajectory constraints and action preconditions. As described above, the impact of the violation of a preference on the plan metric is specified by means of the expression `(is-violated` p`)`, which evaluates to the number of violations of preferences with name p. Thus, a plan metric involving preferences can be restated in terms of set of corresponding numeric fluents, by making sure these perform the same function, i.e., counting the number of preference violations.

For each preference name, p, we introduce a fluent `(is-violated-`p`)`, assigned zero in the initial state. (We can assume without loss of generality that all preferences are named, since if some are not, we can introduce a new name and assign it to all such anonymous preferences.) A preference `(preference` p ϕ`)` appearing in an action precondition translates into the conditional effect

`(when (not` ϕ`) (increase (is-violated-`p`) 1))`,

which is added to the effects of the action. To evaluate preferences in the problem goal, we introduce again a special action `finish`, constrained to appear last in any valid plan, and add the corresponding conditional effects to this action. Note that if the problem contains preferences over state trajectory constraints, we need two distinct `finish` actions, where the first performs the final update of automata corresponding to trajectory constraints as described above and the second evaluates the preferences, which in the compiled problem refer to the acceptance predicates of these automata. For preferences appearing inside a universal quantifier, the corresponding conditional effects are also quantified. For instance, let

```
(forall (?x - aircraft) (preference P1 (sometime (at ?x NY))))
```

be a quantified preference for our `Zenotravel` example; then, the special start action has the extra numerical effect `(assign (is-violated-P1) 0)` and the second finishing action has the quantified conditional effect

```
(forall (?x - aircraft)
  (when (not (accepting-A ?x)) (increase (is-violated-P1) 1))
```

Finally, we note that if the impact of preference violation on the plan metric is restricted to be linear (i.e., the metric is a sum of weighted preference expressions, plus possibly some other term), preferences can be reduced to *additive action costs*, by compiling the conditional effects into multiple action instances (this was also noted by Benton & Kambhampati [7]). Although PDDL does not have any special construct for expressing action costs, relying on numeric variables to specify such a metric, there is a growing number of planners that focus on optimising additive action costs (see e.g., [13, 30]), which makes this an interesting special case.

2.4.5 Discussion: Some Practical Considerations

Even though the hard and soft constraints permitted by PDDL3.0 can be compiled away, i.e., expressed in a reduced language such as PDDL2, there are potential advantages to introducing them anyway. From a knowledge engineering point of view, the new language constructs may make it possible to formulate some aspects of a domain or problem in a more natural, easily understandable, or modular way. From a computational perspective, having trajectory constraints or soft goals explicitly identified may simplify implementing more efficient strategies for dealing with them.

The compilation methods that have so far been described in the literature ([6, 25], as well as in this paper) are all based on simulating automata that track the status of trajectory constraints. These methods have obvious weaknesses, such as, for example, not dealing well with large numbers of constraints. As an example, the following constraint, taken from domain `Storage` (described in Section 3), states that any two crates stored in adjacent areas must be of a compatible nature:

```
(forall (?c1 ?c2 - crate ?s1 ?s2 - storearea)
  (always (imply (and (on ?c1 ?s1) (on ?c2 ?s2)
                 (not (= ?c1 ?c2)) (connected ?s1 ?s2))
           (compatible ?c1 ?c2))))
```

Using the compilation scheme outlined above, this constraint would be converted into a quantified conditional effect attached to each action. For a problem with 5 crates and 10 areas (a medium-sized benchmark problem in this domain), the corresponding ground problem would have actions with several hundred conditional effects (even after effects with statically false conditions have been removed). That is very likely to render it effectively unsolvable even by the best current classical planners. In fact, we have confirmed that is the case for both FF and SGPLAN5 with constraints that are trivially satisfiable, i.e., that are satisfied in the solution generated by the planner when ignoring the constraints [33].

However, a trajectory constraint of type `always` can also be enforced by adding to the precondition of any action that may possibly falsify it the regression of the constraint formula through the action. In the case of the above constraint, that amounts to adding

```
(forall (?c2 - crate ?s2 - storearea)
  (imply (and (connected ?s1 ?s2) (not (compatible ?c1 ?c2)))
         (not (on ?c2 ?s2))))
```

to the precondition of the `drop` action (as this is the only action that makes `(on ?c1 ?s1)` true). As the `connected` and `compatible` predicates are static, the resulting addition to the preconditions of the corresponding grounded actions would be only a conjunction of literals (albeit a fairly large number of them). It is likely that this would not significantly slow down a planner, at least when the constraint can be trivially or "easily" satisfied.

SGPLAN5 is, so far, the only planner to handle problems with PDDL3.0 trajectory constraints in a manner other than by compiling the constraints away. Among the IPC5 benchmark problems, it solves some that have in excess of 2000 ground constraints, so clearly it does not suffer from the same kinds of issues as current compilation methods. However, we have also observed that SGPLAN5's mechanism for dealing with trajectory constraints has its own problems. For example, SGPLAN5 solves no IPC5 benchmark problem in the Pipesworld domain with constraints (described in Section 3), and we have observed that in some cases adding just a single (satisfiable) constraint to a problem causes SGPLAN5 to fail whenever that constraint forces the solution plan to be different from the one SGPLAN5 finds for the original problem (regardless of whether the problem had other constraints or not).

Thus, it is clear that effective and general handling of PDDL3.0 trajectory constraints is still an open research question. Ultimately, the question of whether this is best done by compiling the constraints away, and if so what the compilation scheme should look like, or if constraints can be better dealt with in a more direct way, may depend both on the particular planner and characteristics of the constraint formulas of interest.

3 The Benchmark Domains

The benchmark domains used in IPC5 were derived from a variety of sources: some are inspired by (potential) applications of planning technology; some are encodings of benchmark problems used in other areas of computer science and operations research; and some were created for the explicit purpose of trying out the new language features offered by PDDL3.0. As in previous planning competitions, domains were designed in several "versions", each using a different subset of PDDL3.0 features. In most cases, however, these different versions encode radically different problems and should properly be considered to be different domains, sharing only a common theme. The name of each domain version indicates the language category it belongs to: *Propositional* domains use only constructs of level 1 of PDDL2 [29, 41]; *MetricTime* domains also use constructs of level 2 or 3 of PDDL2.1; *SimplePreferences* domains extend the propositional or metric-time variants with preferences over the problem goals; *QualitativePreferences* domains include preferences over action preconditions and preferences over state trajectory constraints; *MetricTimeConstraints* domains extend the metric-time variant with strong state trajectory constraints; and, finally, *ComplexPreferences* domains use the full power of PDDL3.0. Note that all domains are not represented in every language category.

In line with the aim to emphasise plan quality in the evaluation of competing planners, many of the domains encode optimisation problems, in which it is significantly easier (in some cases completely trivial) to find a plan that only satisfies the hard goals and constraints of a problem instance (indeed, in some domains there are no hard goals!), and the true difficulty lies in finding a plan that also has high quality. For the same reason, we also, for some domains, designed the problem instances very carefully. Creating problems by simply assigning random values to costs/penalties runs a high risk of resulting in problems that are simple, in the sense that optimal solutions lie at an extreme point where one objective is ignored in favour of maximising satisfaction of another. This situation we wanted to avoid. Moreover, for most domains, the problems instances were designed to have many solutions with significantly different qualities and requiring the planner find a good compromise among the different (possibly conflicting) terms in the objective function to optimise.

Three domains (`Rovers` STRIPS, `Pipesworld` Tankage-Nontemporal and `Pipesworld` Tankage-Temporal) were recycled from previous competitions, as a way to measure advancement in the field. However, new versions of these domains, with preferences and constraints, were also created. In all, we developed 32 new domains, or new versions of existing domains, and 978 problem instances. Most of these were automatically generated.[5]

3.1 Openstacks

The Openstacks domains are all, to a greater or lesser degree, based on the "minimum maximum open stacks" combinatorial optimisation problem, which can be stated as follows: A manufacturer has a

[5] The problem generation tools are available from the IPC5 website `http://ipc5.ing.unibs.it/`.

product sequence:	2	3	4	5	1	1	2	3	5	4
order 1 ($\{1, 2\}$):	X	–	–	–	X	X	X			
order 2 ($\{1, 3\}$):		X	–	–	X	X	–	X		
order 3 ($\{2, 4\}$):	X	–	X				X	–	–	X
order 4 ($\{3, 5\}$):		X	–	X				X	X	
order 5 ($\{4, 5\}$):			X	X					X	X
# open stacks:	2	4	5	4	2	2	3	3	3	2

Figure 4: Illustration of how the number of open stacks is calculated for two different production sequences. An "X" denotes that the order includes a request for the corresponding product; a "–" that the order is open at a point in the sequence, even though it does not include a request for the product made at that point. For the first production sequence (2, 3, 4, 5, 1) the maximum number of simultaneously open stacks is 5, while for the second sequence (1, 2, 3, 5, 4) it is 3, which is also the optimal value for this problem instance.

number of orders, each for a combination of different products. Only one product can be made at a time, but the total required quantity of that product is made at that time. From the time that the first product requested by an order is made to the time that all products included in the order have been made, the order is said to be "open" and during this time it requires a "stack" (a temporary storage space). The problem is to order the making of the different products so that the maximum number of stacks that are in use simultaneously, *i.e.*, the number of orders that are in simultaneous production, is minimised.

Figure 4 illustrates the relationship between a set of orders, two different production sequences, and the number of open stacks for a small example problem.

This and several related problems have been studied in operations research (see, e.g., Fink & Voss, [27]). It is a pure optimisation problem: for any instance of the problem, every ordering of the making of products is a solution, which at worst uses as many simultaneously open stacks as there are orders. The problem is known to be NP-hard [49]. Recently, it was posed as a challenge problem for the constraint programming community (see Smith & Gent, [61]).

3.1.1 Openstacks Propositional

The Openstacks Propositional domain is a direct encoding of the openstacks problem. There are two different formulations of the domain. In the *plain* formulation, the encoding is done in such a way that the length of a plan equals the maximum number of open stacks plus a problem-specific constant (equal to twice the number of orders plus the number of products). Thus, minimising the number of actions in the plan also minimises the objective function, *i.e.*, the maximum number of open stacks. However, because no plan quality metric can be specified in the propositional (STRIPS/ADL) fragment of PDDL, a different formulation had to be used in the competition: in this, the *sequenced* formulation, additional action preconditions and effects ensure that no two actions can be executed in parallel, so that minimising the number of parallel steps is equivalent to minimising the number of actions. The constant offset between the number of steps and the maximum number of open stacks is larger in the sequenced domain (equal to twice the number of orders plus twice the number of products).

As a result of the 2005 Constraint Modelling Challenge, a large library of instances of the openstacks problem, as well as data on the performance of a number of different solution approaches, is available. The instances used in IPC5 comprise 25 problems from this set, selected mainly for variety, plus five extra instances of trivially small size.

3.1.2 Openstacks SimplePreferences

The Openstacks SimplePreferences (SP) domain models a problem similar to, yet radically different from, the original openstacks problem. The main ingredients are the same: a set of products to be made, a set of orders, each for some subset of products, and the constraint that an order is "open", and requires a "stack", from the point where the first product requested by the order is made to the point where the last such product is made. The difference lies in the objective function: in this problem, the number of

stacks that may be used is fixed to a (instance-dependent) constant, and the constraint that all requested products must be included in each order is *soft*, *i.e.*, it does not have to be satisfied for a plan to be valid, but the plan is given a penalty for each violation. The objective is to minimise the total penalty for unsatisfied product requests. Put another way, given an infeasible (due to the limited number of stacks) openstacks problem, the planner is asked to find the maximal (weighted) subproblem that is solvable.

Instances of the Openstacks SP domain were constructed from standard openstacks problems (the same as selected for the propositional domain, except the five trivial and the five largest) by choosing two additional parameters: (1) a penalty function for unsatisfied product requests and (2) a limit on the number of stacks available. Two different models for the penalty associated with unsatisfied product requests were used, each in roughly half the instances: in one, the objective is simply to minimise the number of unsatisfied requests, while in the other, products requested by each order were weighted according to an (arbitrarily chosen) *order of importance*. Most instances do not have a sufficient number stacks to permit solutions with zero penalty, but a few (problems 15–18) unintentionally do.

3.1.3 Openstacks QualitativePreferences

The Openstacks QualitativePreferences (QP) domain combines the objective functions of the Openstacks Propositional and Openstacks SP problems in a weighted sum. That is, a solution may use any number of stacks and may drop any set of product requests, but must minimise the sum of a price per stack used and the total penalty for unsatisfied requests.

Problem instances of this domain were constructed from instances of the Openstacks SP domain, by simply assigning a price to stacks. The price per stack was set to the total penalty for unsatisfied product requests divided by the optimal (or, in the case of problems 15–18, best known at the time) number of stacks required to accommodate all requests, with the aim of making the two extreme solutions roughly equal in value.

3.1.4 Openstacks Time and MetricTime

The Openstacks Time and MetricTime (MT) domains again have the same elements as the original openstacks problem but very different objective functions. In the Openstacks Time domain, the objective is to minimise plan makespan. Making each product takes a different amount of time, but any number of products can be made in parallel (as long as all orders requesting the products are simultaneously open). In the Openstacks Time domain the maximum number of stacks in use is fixed, while in the MT domain it is unlimited, and the objective function is a weighted combination of makespan and the number of stacks used. There are no soft goals.

Problem instances were created from standard openstacks problems (again the same set of problems as used for the Openstacks SP and QP domains) by assigning (in part random) action durations, attempting to ensure that the scheduling of the product-making actions dominates plan makespan. The fixed number of stacks available in instances of the Openstacks Time domain was set close to the upper bound (number of orders). For the MT domain, the price per stack was determined by comparing the makespan of the best plans found with different fixed numbers of stacks, and choosing a value equal to the average decrease in makespan per stack added, following again the principle of trying to make the extreme points on the spectrum of trade-offs roughly equal in value.

3.2 Rovers

The Rovers domain, introduced in IPC3 [50], models the problem of planning for one or more autonomous rovers performing planetary exploration in order to obtain samples of rocks or soils from certain waypoints, or having images of some objects. In IPC5, we reused the Strips and Numeric versions of this domain, as Propositional and MetricTime, respectively.[6] We also created two new domains that are, very loosely, based on the Rovers domain.

[6] The problem set for the Rovers MetricTime domain extends the IPC3 Rovers Time set with some very large instances.

	#1	#2	#3	#4	#5
(#1) `(sample wp2)`	39.5	+23.7	±0	−26.9	−39.5
(#2) `(sample wp5)`		38.4	±0	+15.4	+23.7
(#3) `(sample wp7)`			116.2	±0	±0
(#4) `(image obj7)`				31.2	−26.9
(#5) `(image obj8)`					39.5

Figure 5: Example of goal cost relations for a (very) small Rovers problem. Entries on the diagonal give, for each goal, the optimal cost of achieving that goal alone, while each off-diagonal entry (i, j) shows the *difference* between the optimal cost of achieving both goals #i and #j and the sum of the costs of achieving each of them alone. A negative value represents a synergy effect between goals #i and #j, while values greater than zero indicate the goals are interfering.

3.2.1 Rovers MetricSimplePreferences

The Rovers MetricSimplePreferences (MSP) domain models a *net benefit maximisation* problem, in which the task of the planner is not to plan for all given goals but to select and plan for a subset of goals so as to maximise the net benefit, defined as the sum of the values of goals achieved by the plan minus the sum of the (independent and constant) costs of actions in the plan. In the domain used in the competition, the net benefit maximisation objective was reformulated as a minimisation objective. Net benefit maximisation and other cost-benefit trade-off problems have been studied in OR and scheduling, and have also attracted interest among planning researchers recently [62, 22]

Instances of the Rovers MSP domain were created by a general method aimed at generating "interesting" problems, having balanced costs and values for each subset of goals and thus non-obvious optimal solutions. The steps involved are: (1) Generating (random) base problem instances, with (random) actions costs and a relatively large number of potential goals. (2) Finding out the real cost of achieving small sets of goals (single goals and pairs of goals), by optimally solving the corresponding planning problems. (3) Calculating base values for each goal (and some pairs of goals), using the known costs to estimate the kind and strength of interaction between goals. (4) Randomising goal values by adding or subtracting a random percentage.

The calculation of goal base values aims to make the achievable net benefit of all goal sets roughly equal: the base value of a single-atom goal that has no interactions with other goals equals the optimal cost of achieving the goal. A goal that has only *synergy* relations with other goals, meaning the cost of achieving the set of goals together is less than the sum of the cost of achieving each of them individually, has this base value reduced by half the average synergy effect, while for a goal that has only the opposite, *interference*, relations with other goals, it is increased by the corresponding amount. Goals with mixed relations are treated as goals with only synergies, but the conjunction of any pair of such goals that are in an interference relation is given an additional value, equal to the interference effect.

As an example, Figure 5 shows the optimal cost of achieving each single goal and each pair of goals in a small Rovers instance. The goal of obtaining a sample from waypoint `wp5` has only interference relations with other goals: its base value is the cost of achieving the goal alone (38.4) plus half the average interference effect (i.e., $((23.7 + 15.4 + 23.7)/3)/2$). The goal to take an image of object `obj8` has synergy relations to two other goals (with an average synergy effect of 33.2), but also an interference relation, with the goal `(sample wp5)`: the base value of this goal is the cost of achieving it alone (39.5) minus half the average synergy effect. However, the *goal pair* {`(sample wp5)`,`(image obj8)`} is given an extra base value of 23.7 (the interference effect between the two). Final values for goals (and goal pairs that have a base value) are obtained by adding or subtracting a random percentage (in the range [-100%,+100%]) of the base value to it.

The set of instances of this domain forms three groups: in the first (problems 1 to 7), all goals have only synergy relations to other goals; in the second (problems 8 to 13), goals have only interference relations; and in the third (problems 14 to 20), there is a mix of the two kinds of goal relationships.[7]

[7] Because only actions that move the rovers have non-zero cost in this version of the domain, it turned out that some goals in some of the problems we generated could be achieved by zero cost plans, and therefore got a base value of zero. This problem was fixed by assigning such goals a small value, 1%–10% of the total goal value.

The method is not fool-proof: a final test run, using a simple optimal planner for net benefit problems, was made to filter out problems that were too easy or that appeared too hard.

3.2.2 Rovers QualitativePreferences

The Rovers QP domain is also based on the IPC3 Rovers domain but, again, models a very different problem. This domain was designed explicitly to test competing planners' ability to trade off soft state trajectory constraints against one another. Constraints in the Rovers QP domain are all soft, *i.e.*, a plan does not have to satisfy them, but is given a penalty for each unsatisfied constraint. Problems also have regular hard goals (same as in the original Rovers domain). Plan constraints may contradict each other, or the hard goals: an optimal solution in this domain is one that selects a jointly achievable set of constraints with maximum value (the "value" of a constraint being the penalty avoided by satisfying it).

State trajectory constraints in the Rovers QP domain are artificial, in the sense that they do not encode any real preferences on plans. As in the case of the Rovers MSP domain, constraints (and their associated penalties) for the problem instances were generated by a general method, with the aim of producing problems with non-obvious optimal solutions. Given a base problem, a set of candidate constraints is found by mining a set of plans for the problem: candidates are constraints satisfied by at least one plan, but not by all. The strategy for assigning penalties to constraints is again to calculate a base value, in a manner intended to make the values of all maximal satisfiable sets of constraints roughly equal, and determine final values by randomly adding or subtracting a percentage of the base value. The joint satisfiability of sets of constraints is approximated by looking at the set of plans.

The base problems and plans used to create instances of the Rovers QP domain were the instances of the Rovers domain (STRIPS version) used in IPC3 and the plans submitted by planners participating in that competition.

3.3 Pathways

The Pathways domains are inspired by the field of molecular biology, specifically the study of biochemical pathways. "A pathway is a sequence of chemical reactions in a biological organism. Such pathways specify mechanisms that explain how cells carry out their major functions by means of molecules and reactions that produce regular changes. Many diseases can be explained by defects in pathways, and new treatments often involve finding drugs that correct those defects." [64] The function of a pathway, at an abstract level, can be modelled as a planning problem. Actions represent some of the different chemical reactions that can appear in a pathway (association reactions, association reactions requiring catalysts, and synthesis reactions). The problem goal is to construct a sequence of reactions that produces one or more substances. Goals are generally disjunctive (in the domain used in the competition, however, these disjunctions were compiled away). The plan must also choose a limited number of substances to use as input for the sequence of reactions, i.e., some aspects of the initial state of the problem are left to the planner. This feature was introduced mainly to make the problems non-trivial to solve.

The Pathways domains created for IPC5 are based on the pathways of the Mammalian Cell Cycle Control as described in [47] and modelled in [16]. Figure 6 shows an example of a small part of the network of reactions.

3.3.1 Pathways Propositional

The Pathways Propositional domain uses a simple qualitative encoding of chemical reactions, where only the presence/absence of a substance is modelled and not the quantity that is available. The goals are conjunctions of binary disjunctions (compiled into actions with disjunctive preconditions).

As an example, consider the network of reactions depicted in Figure 6, and suppose we seek a pathway producing either `RAF-RAFK` or `MEK-{p1,p2}` using at most one of the substances `RAF`, `RAFK` and `MEK-RAF-{p1}` as input. Without the restriction to using at most one input substance, finding a solution to this problem would be a trivial task: simply triggering all possible (chains of) reactions of the pathway generates all producible substances. However, selecting a limited number of input substances that can generate the desired output is more challenging. Note that in the network shown in Figure 6,

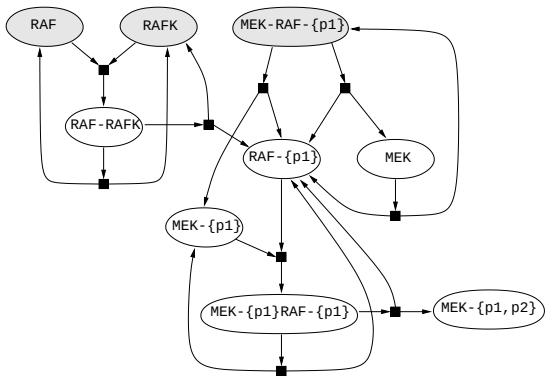

Figure 6: An example of a small biochemical reaction network. Ellipses represent substances, squares represent reactions, and edges indicate substances consumed/produced by them. The shaded nodes are substances that can be chosen as inputs.

producing RAF-RAFK from a single input substance is not possible, so we are forced to satisfy the other disjunct, synthesising MEK-{p1,p2}. If, on the other hand, the number of input substances was not limited, producing RAF-RAFK would be easier.

3.3.2 Pathways SimplePreferences

This domain has the same basic structure as the propositional version, with the difference that both goals (products that must be synthesised by the pathway) and the initial state constraints (maximum number of input reactants) are soft. The plan metric is a weighted sum of preference violations. Problems in this domain do not admit solutions that satisfy all preferences; in particular, in order to synthesise the desired products some input reactants must be used.

The penalties associated with preferences for desired outputs of the pathway were computed using estimates of the minimum number of required initial reactants, with the aim of ensuring that the trade-off between the two kinds of preferences is non-trivial (i.e., that preferences of one kind do not completely dominate the cost function).

3.3.3 Pathways MetricTime and ComplexPreferences

The Pathways MetricTime domain models chemical reactions at a greater level of detail, with reactions consuming and producing certain quantities of substances, and taking a certain amount of time. Goals are expressed as sums of substance concentrations that must be generated by the reactions of the pathway. The objective function is to minimise a linear combination of the number of input substances used by the pathway and plan makespan.

The ComplexPreferences domain adds numerous preferences concerning the concentration of substances along the pathway and the order in which substances are produced. The metric is a combination of penalties for violations of these preferences, the number of substances used and plan makespan.

3.4 Pipesworld Domains

The Pipesworld domain was introduced in IPC4 [42]. It models the problem of transporting batches of petroleum products in a network of pipelines, with or without restrictions on “tankage” (space in intermediate storage tanks). In IPC5, the Tankage-Nontemporal (Strips) and Tankage-Temporal versions of this domain were reused, and two new domain versions were created.

3.4.1 Pipesworld TimeConstraints

The Pipesworld TimeConstraints (TC) domain is based on the IPC4 Pipesworld Notankage-Temporal domain. Like several other IPC5 domains, it adds hard deadlines for the achievement of subgoals.

In the context of the planning competition, the main difficulty in constructing problem instances with hard deadlines is to ensure that those deadlines can in fact be met. One might expect that determining if given deadlines are feasible should be within the capabilities of temporal planners. In practice, however, most temporal planners cannot do this. In particular, none of the temporal planners participating in the full PDDL3.0 subtrack of IPC5 could do so. Therefore, including unsolvable problems in the competition set would not have served any purpose.

To ensure deadline goals were feasible, we made use of existing plans (specifically, solution plans submitted by planners competing in IPC4), simply selecting for each problem one solution plan, with a preference for plans achieving goals quickly, and extracting deadlines from that plan. A similar approach was used by for the construction of some problems with time-windows for IPC4 [42]. In addition to deadlines for the achievement of goal atoms (encoded using `within` constraints), problems in this domain also have (a fairly large number of) conditional deadlines, modelled by `always-within` constraints. In retrospect, this was perhaps somewhat excessive, since most of the problems with only subgoal deadlines are already too hard for the competing planners.

3.4.2 Pipesworld ComplexPreferences

The Pipesworld ComplexPreferences domain is very similar to Pipesworld TC, with the difference that in this domain, deadlines are soft, i.e., preferences instead of hard constraints. Deadlines are specified only for goal atoms, but each goal can have several (increasing) deadlines with different (increasing) penalties for missing them.

The method for selecting deadline goals and penalty values for instances of Pipesworld CP has similarities to those used for the Rovers MSP and Rovers QP domains. Given a base instance of the Pipesworld (Notankage-Temporal) domain, upper and lower bounds on the time required to reach each subset of goal atoms were derived: upper bounds from a collection of valid solution plans for the problem instance (the plans found by competitors in IPC4) and lower bounds using various admissible makespan heuristics. The set of distinct values appearing as lower or upper bounds define a set of "interesting" time points.[8] Each goal atom p and interesting time point t, such that t is not less than the lower bound on the time required to achieve p, defines a potential deadline goal, `(within` t p`)`, of which a random subset was selected. The base value of a selected deadline goal is 1, plus 1 for every other selected deadline goal such that the pair of them is known to be unachievable. Final penalty values were chosen by randomly adding or subtracting a percentage of the base value.

3.5 Storage

The Storage domains model a transportation problem involving a kind of spatial reasoning, similar to that found in some kinds of puzzle domains (e.g., Sokoban or the $(n^2 - 1)$-Puzzle). The goal is to unload crates from one or more containers and deposit them in storage spaces ("depots") using hoists. Space inside each depot is divided into "areas": hoists can only move between adjacent areas, and can only enter and leave the depot to/from certain areas. Crates, once deposited in an area, block further movement through the area. Thus, in a plan to store more than one crate in a depot, leaving the first crate just inside the door is not going to work. Movement outside depots (in the "loading area") is unrestricted.

Figure 7 shows a small example of Storage instance. Note that access to the depot is only through area `D32`, and that thus, depositing a crate there prevents moving other crates into the depot. Likewise, after putting a crate in area `D22`, only the area closest to the door will be reachable. Thus, since the goal in this example is to stow four crates, and there is only one depot to store them in, the first two must be placed in areas `D11`, `D12`, or `D21`. This situation can be particularly problematic for delete-relaxed plan heuristics [11] : in the delete relaxation, the optimal (and also the easiest) solution is to store all crates in area `D32`, which is clearly not very indicative of the real plan.

[8] Two time values are not considered distinct if they differ by less than a given tolerance, τ. This filtering is necessary because upper bounds are derived from plans, which tend to have action start times shifted by (often wildly different) epsilon values.

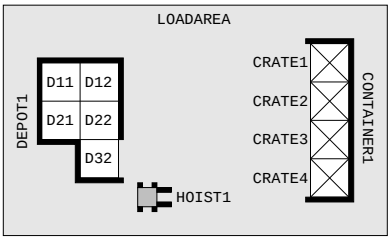

Figure 7: Example of a small Storage problem.

The different Storage domain versions add action durations, preferences and trajectory constraints. Altogether, they involve almost all the new features of PDDL3.0. There is no numeric version of this domain.

3.5.1 Storage Propositional and Time

The Storage Propositional domain encodes the basic problem, as described above. In the Storage Time domain actions have non-unit duration and the objective is to minimise plan makespan, but otherwise it is identical to the Propositional version.

3.5.2 Storage SimplePreferences and QualitativePreferences

The Storage SimplePreferences domain differs from the Propositional version in that goals are soft. Additionally, in this domain some crates are incompatible with each other, and preferences specify that only compatible crates are stored in the same depot or, failing that, that incompatible crates stored in the same depot are located at non-adjacent areas. There are also preferences for keeping certain areas clear, and for having the hoists located in depots different from those where crates are stored at the end of the plan.

The QualitativePreferences version extends Storage SP with preferences over trajectory constraints, which concern the use of available hoists for moving crates and the order in which crates are stored in the depots.

In both domains, plan quality is measured by the sum of the weighted preference violations. In general, preferences may contradict each other, so that there is no plan satisfying all of them, forcing the planner to make a trade-off.

3.5.3 Storage TimeConstraints

This domain extends Storage Time with trajectory constraints, imposing that a crate can be moved at most once and that every hoist is used at least once, constraints on the order in which certain crates are stored, deadlines for storing crates, and a maximum time that a hoist can stay outside a depot. Besides the goal of storing all crates, there are end-state constraints imposing that incompatible crates are not stored in adjacent areas and that all hoists are inside a depot.

There are three groups of instances, using different modal operators for state trajectory constraints: ten instances contain only constraints of type `always` and `sometime`; ten instances contain these two plus `sometime-before` and `within`; and ten instances further extended with constraints of type `at-most-once` and `always-within`.

Plan quality is measured by makespan. Note, however, that, due to the additional constraints, the solution that is optimal for corresponding Storage Time instances may not be valid.

3.5.4 Storage ComplexPreferences

This domain extends Storage Time with preferences over the goal state and over state trajectory constraints. Trajectory constraints are similar to those found in the TimeConstraints version, but in this version there are many more (soft) constraints, which frequently contradict each other so that there

is no plan satisfying all preferences. The plan metric is a weighted sum of preference violations. However, since some constraints impose deadlines or conditional deadlines (using the `within` and `always-within` modal operators), time also plays a part in determining plan quality.

3.6 TPP (The Travelling Purchaser Problem)

The TPP domains are inspired by the Travelling Purchaser Problem, which is a known generalisation of the Travelling Salesman Problem. The problem can be defined as follows: We have a set of products and for each product a known demand. We also have a set of markets, each of which can provide a known limited amount of each product at a known price. The purchaser must select a subset of markets such that the given demand for each product can be purchased, and construct a tour which starts and finishes at a distinguished location (called the depot) and visits all the selected markets. The objective is to minimise a combination of the travel cost (sum of known costs for each leg of the tour) and purchase cost (sum over all products and markets of the quantity of the product purchased at the market times the price at which it is offered).

The problem is NP-hard and arises in several applications, mainly in routing and scheduling contexts. Computing optimal or near optimal solutions for the TPP is a topic of active research in operations research (see, e.g., [57]).

3.6.1 TPP Metric

The TPP Metric domain encodes the original Travelling Purchaser Problem. There are two different purchasing actions, `buy-all` and `buy-allneeded`: the first buys at a certain market the whole amount of a product sold at that market, while the second buys at a market the amount of a product that is needed to satisfy the remaining demand.[9] Travel between any two locations (markets and the depot) incurs a travel cost. Travel costs are symmetric.

Figure 8 shows an example of a small TPP instance with two markets and three types of goods; the available amount and price for each type of good is shown in the table next to each of the markets, and the travel cost on the edges of the graph. The purchaser (here called `truck`) is at the depot. The goal in this instance is to acquire 10 units of `goods1`, 100 units of `goods2` and 10 units of `goods3`, and return the truck to the depot. The optimal plan for this instance is a tour passing both markets, buying `goods1` and `goods3` at `market1` and `goods2` at `market2`. Compared to the simplest plan, i.e., the one with the fewest actions, this saves 780 units of currency. Thus, it is likely that a planner that considers only the distance to the goal and not the plan metric will come up with quite a poor plan.

3.6.2 TPP Propositional

This domain simplifies the original TPP by discretising the amounts of goods into "levels" and assuming prices are the same for all products at all markets. Travel costs are coarsely approximated by making the map of connections between locations a less than complete graph. The goal is still to acquire a certain total amount of (some subset of) the different goods.

Since this is a propositional domain, it has the default objective of minimising the number of parallel steps. Because of this, most of the instances of this domain have more than one depot and more than one purchaser ("truck"), which allows the number of plan steps to be reduced by parallelising operations.

3.6.3 TPP SimplePreferences

This domain is similar to the Propositional domain version, with the difference that goals are all soft (i.e., preferences). Besides a general preference for maximising the amount of goods acquired, there are also preferences over the relative amounts of some kinds of goods that is acquired. For example,

[9] This encoding avoids the need for a purchase action with a numeric argument (the amount to purchase), which is not permitted in PDDL (for reasons detailed in [29]). It does, however, introduce a constraint that is not present in the original problem, *viz.*, that the market from which a fraction of the available good is purchased must be visited last. This additional coupling between the problems of optimising purchase and travel costs means that there are instances of the TPP for which the optimal solution can not be represented by any plan.

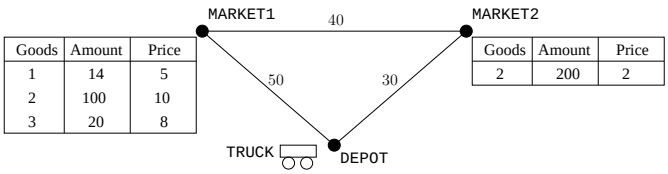

Figure 8: Example of a simple TPP world.

```
(preference p3A (imply (stored goods2 l1) (stored goods1 l2)))
```

indicates that when 1 unit of `goods2` is purchased, 2 units of `goods1` should be. These preferences may conflict with the general preference for acquiring as much as possible, since the total amounts available of different kinds of goods may be different. For example, if there is only one unit available of both `goods1` and `goods2`, buying the one unit of `goods2` leads to a violation of the preference above (while not buying it violates the general preference for buying everything). This forces planners to trade-off the satisfaction of the two kinds of preferences.

The plan quality metric is composed solely of weighted preference violations. The relative weights of preferences were set so that plans storing certain levels of goods would be better than the empty plan. There are also some preferences for "sensible plans", e.g., a truck not leaving a market without having loaded purchased goods, or not unloading goods from the truck by the end of the plan. These preferences are never in conflict with preferences of the other kinds and therefore do not affect the best achievable plan quality. To some extent, they are a substitute for action costs, which can not be modelled in this PDDL fragment.

3.6.4 TPP QualitativePreferences

This domain version extends the SP version with preferences over trajectory constraints. These include constraints about which truck to use for different kinds of goods and constraints imposing the use of every truck. Plan quality is measured by weighted violations of the soft goals, soft constraints and soft action preconditions. Similarly to the SP version, instances of this domain generally do not admit a solution satisfying all the preferences.

3.6.5 TPP MetricTime

This domain version extends the Metric version with action durations. It has explicit actions for loading and unloading goods (not present in the Metric version), whose duration depend on the amount loaded/unloaded. The objective function is a linear combination of plan makespan and the sum of purchase and travel costs. Similar to the propositional version, instances can have more than one truck, making it possible to reduce makespan by parallelising operations.

This domain also has an additional twist, in that the action buying the entire quantity of a product sold at a market gets a (known) "rebate rate", i.e. a lower price. This rate, like the ordinary price, may vary between markets, and some markets may not offer it at all.

3.6.6 TPP MetricTimeConstraints

The TPP MetricTimeConstraints (MTC) extends the MT version with several hard constraints. Domain-wide constraints impose that in the final state, all purchased goods are stored in a depot (i.e., not left lying in the market or in a truck), that every market can be visited by at most one truck at the same time, and that every truck is used (loaded) at some point in the plan. Moreover, instances of the domain have additional constraints concerning the relative amounts of different types of goods stored in a depot, the number of times a truck can visit a market, the order in which goods should be stored, the order in which some type of goods should be stored and another bought, and deadlines for delivering goods once they have been loaded in a truck. Plan quality is a linear combination of makespan and the total cost of travelling and purchasing.

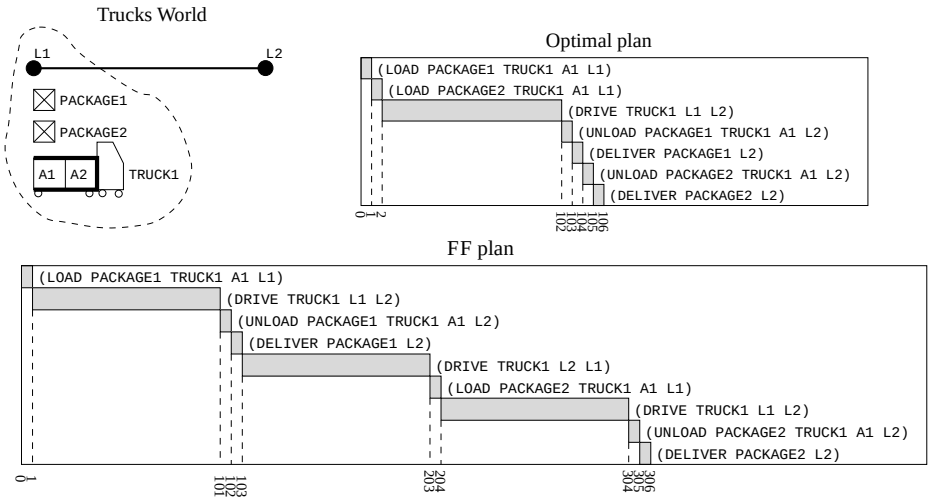

Figure 9: Gantt charts of the optimal plan and the plan computed by FF for moving `package1` and `package2` to `L2` by `truck1` from the sketched initial world state.

There are three groups of instances, using different modal operators for state trajectory constraints: ten instances contain only constraints of type `always` and `sometime`; ten instances contain these two plus `at-most-once` and `sometime-before`; and ten instances further extended with constraints of type `always-within`.

3.6.7 TPP ComplexPreferences

This domain extends the MT version with various preferences both over the final state and over state trajectory constraints. Trajectory constraints are similar to those in the MTC domain version, and the plan metric is a weighted sum of preference violations. However, this domain also has the same hard goals as the MT version, i.e., that the requested amount of each type of good is stored at the end of the plan.

3.7 Trucks

The Trucks domains are all (single-vehicle-type) transportation domains with two additional constraints. The first is that the cargo space in each truck is limited and divided into areas, similarly to the space inside a depot in the Storage domain, and a package can be loaded into or unloaded from an area of a truck only if *all* areas between this area and the truck door are unoccupied. In other words, the storage space in a truck functions like a stack: last in, first out. The second constraint is that packages must be delivered by a deadline.

3.7.1 Trucks TimeConstraints and Time

The TimeConstraints version of the Trucks domain is the one in which the additional constraints are most naturally expressed. The goal is to have packages at their destination by certain deadlines. Solution quality is measured by plan makespan. However, the most difficult aspect of the problem is meeting the deadlines, which were determined so that a valid plan must exploit truck capacity well (though not necessarily fully). The durations of actions that move trucks are generally much greater than those of other actions (such as loading, unloading and delivering), so that efficient routing is a primary concern.

For example, consider the following simple instance of the domain: There are 2 objects, `package1` and `package2`, that need to be transported from their initial location `L1` to `L2` using one truck with two load areas `A1` and `A2`, each of which can carry one package. The travel time between the two locations is 100 time units, whereas loading and unloading takes 1 time unit. The optimal plan for this problem is depicted in Figure 9. First both packages are loaded on the truck at location `L1`, then the truck moves

to location L2 where the packages are unloaded and delivered. The bottom chart in Figure 9 depicts the plan that is generated by first running FF on the propositional version of the domain (obtained by removing action durations) and then scheduling the actions, taking into account their actual durations. Note that this plan is suboptimal. This can be attributed to the fact that the relaxed plan heuristic used by FF does not distinguish between the usage of the two load areas, and (accidentally) generates a plan where the first package is loaded in the area closer to the door, thus blocking the use of the inner load area. If there is the deadline that package1 must be delivered to location L2 by time 200, then the upper chart in Figure 9 represents the only valid plan, and planning can become more difficult.

In this domain version, deadlines were specified by within constraints. However, we also created an equivalent version in which the constraints are compiled into timed initial literals.

The Trucks Time domain is the same as the TC version but without deadlines for package deliveries.

3.7.2 Trucks ComplexPreferences Domain

The Trucks ComplexPreferences domain has the same basic structure as the TC version, but the deadline constraints are soft, i.e., modelled as preferences. (Eventually delivering all packages is still a hard goal, though.) In addition, there are preferences over other state trajectory constraints, imposing a partial consistent ordering on the delivery of packages, constraints about the use of storage areas in trucks, and constraints imposing that a package can be loaded at most once.

Plan quality is measured by the sum of weighted preference violations. In general, preferences concerning delivering packages within deadlines are the highest weighted, but preferences about which load areas inside trucks are used and how are also important.

For all instances of this domain, there exists no plan which satisfies all preferences. In the example shown in Figure 9, the following preference

```
(forall (?p - package ?t - truck) (preference p1A (always
   (forall (?a - truckarea) (imply (in ?p ?t ?a) (closer ?a a2))))))
```

indicates the desire that, if a package is in a truck, it is on an area nearer to the truck door than the loading area a2 (i.e., in the example of Figure 9 on the nearest area a1). On the other hand, the preferences

```
(preference p2A (within 120 (delivered package1 l2)))
(preference p2B (within 150 (delivered package2 l2)))
```

express the desire that two particular packages are delivered within 120 and 150 time units, respectively. In order to satisfy the second set of preferences, these packages must be loaded together into the same truck, and hence the previous preference will be violated. Which option leads to a higher quality plan depends on the precise weights associated with the preferences.

3.7.3 Trucks Propositional

The Trucks Propositional domain differs from the TC version mainly in that time is modelled as a discrete resource (with a fixed number of levels). Only actions driving trucks consume time, and, due to the encoding, such actions cannot be executed concurrently.

The deadlines for package delivery mean instances of the domain tend to have many deadend states, i.e., states from which some undelivered packages cannot be delivered in time. We have observed that this causes the enforced hill-climbing strategy of FF to fail in many problem instances.

3.7.4 Trucks SimplePreferences and QualitativePreferences

The Trucks SimplePreferences domain has the same basic structure as the propositional version, but has soft rather than hard deadlines. For each package, there is a hard goal to deliver the package, and a sequence of increasing soft deadlines with increasing penalties for violating them (similar to the Pipesworld CP domain). For example, suppose the deadline for delivering package1 is 3. The set of preferences

```
(preference p1B (exists (?t - time)
    (and (delivered package1 l1 ?t) (less-or-equal ?t t4))))
(preference p2B (exists (?t - time)
    (and (delivered package1 l1 ?t) (less-or-equal ?t t5))))
(preference p3B (exists (?t - time)
    (and (delivered package1 l1 ?t) (less-or-equal ?t t6))))
```

expresses increasing penalties for late delivery of `package1` (up to the limit three penalty units). The plan metric in this domain is the number of violated preferences, so delays, i.e., the difference between required and actual delivery time for each package, should be minimised. This encodes a simplified form of the "sum tardiness" optimisation criterion, frequently used in scheduling.

The QualitativePreferences domain version extends the SP version with additional preferences over state trajectory constraints, similar to those used for the CP version. Violation of these preferences is combined with the soft deadlines in the plan quality metric.

4 Experimental Analysis of the Performance of the IPC5 Planners

In this section, after a very brief presentation of the planners that entered the deterministic part of IPC5, we experimentally investigate their performance in detail. We conducted an extensive analysis using the data from the competition, as well as additional results obtained by further experiments. The analysis has two main related aims: comparing the relative performance of the IPC5 planners, and studying their effectiveness more generally.

For the first aim, we analyse the data from the competition in different ways according to the domain categories involving different fragments of PDDL3 (Propositional, MetricTime, SimplePreferences, QualitativePreferences, ComplexPreferences, MetricTimeConstraints). First we consider the overall problem solving success ratio and the number of problems solved by each planner with respect to an increasing CPU-time limit. Then, for each domain variant, we give scatterplots showing a general comparison of each planner w.r.t. the overall best performing planner(s) using the benchmarks domains altogether. (The detailed plots of the results for each different domain are available from the IPC5 website.) The planners are compared in terms of CPU time required for generating a valid plan and quality of the computed plan (measured using the specified plan metric expression).

To get a better estimate of the significance of differences in performance observed among the compared planners, we also make use of a statistical test, the Wilcoxon sign-rank test [67].[10] This test applies to a set of paired observations (a sample from a larger population), and tells us if it is plausible to assume that there is no correlation between the pairwise observed quantities. In our case, these paired observations are, e.g., the runtimes of two planners on the same problem instance, and "no correlation" between them means it is equally likely that we will see one planner solving a problem faster than the other as it is that we will see the opposite on a random sample of problems. The sample is our set of problem instances for some IPC5 domain category. Obviously, instances in each of the IPC5 problem sets are not drawn uniformly at random from the set of all problems in the corresponding domain: although for many domains there was an element of randomness in the problem generation process, many parameters, for example, problem size, were also selected, systematically within some fixed range, by us. However, it is not inaccurate to say that our problem sets are random samples, albeit drawn according to some *unknown distribution* (by which it is much more likely to draw problems that we consider reasonable and interesting or representative of application problems).[11] In particular, the construction and selection of problems was done without knowledge of how the competing planners would behave on them: thus, we can at least say that the sample distribution is not (intentionally) skewed to favour or disfavour any planner. The Wilcoxon test is appropriate because it does not require us to know the sample distribution, and makes no assumptions about this distribution. That is, we have no way to know a priori how hard a planning problem is, and hence we have no distribution of the performance of the competing planners for these problems. As a consequence, it is critical that we use a non-parameterised

[10] This test was also used by the organisers of IPC3 to analyse the results of that competition [50].

[11] For most of the domains, the criteria we used for selecting the benchmark problems are coded in the fully-automated problem generation tools that are available from the IPC5 website.

Planner	Planning Capability					Plan Quality Measure	
	D	N	SG	C	SC	Propositional	Others
Optimal							
CPT2	√	–	–	–	–	#Steps	Makespan
FDP	–	–	–	–	–	#Actions	–
IPPLAN-1SC	–	–	–	–	–	#Steps	–
MAXPLAN	–	–	–	–	–	#Steps	–
MIPS-BDD	–	–	–	–	–	#Actions	–
SATPLAN	–	–	–	–	–	#Steps	–
Satisficing							
DOWNWARD04SA	–	–	–	–	–	#Actions	–
IPPLAN-G1SC	–	–	–	–	–	#Actions	–
HPLAN-P	–	–	√	–	√	–	Problem metric
MIPS-BDD	–	–	√	–	√	–	Problem metric
MIPS-XXL	√	√	√	√	√	#Actions	Problem metric
SGPLAN5	√	√	√	√	√	#Actions	Problem metric
YOCHANPS	√	–	√	–	–	#Actions	Problem metric

Table 1: Summary of the capabilities (columns 2–6) and measures of plan quality (columns 7–8) of planners participating in IPC5. "D" means durative actions, "N" numeric fluents, "SG" soft goals (i.e., preferences over atoms in the goal state), "C" trajectory constraints and "SC" soft trajectory constraints. The plan quality measures are those indicated by the IPC5 teams for their planners.

test. When the statistical test indicates a significant difference, this means that it is quite likely that we would have seen a similar result if we had taken a different sample according to the same distribution. In our context, this means that when we find that, say, planner A is faster than planner B in some domain category, then it is highly likely that if we were to generate more problems in this domain category – following the same construction method and selection criteria as we did when generating the IPC5 problem set for the domains in this category – planner A will be faster than planner B on most of those problems too.

For the second general aim of our experimental investigation, we compare the IPC5 planners with the best performing planners of the previous competition (IPC4). Moreover, for a selection of the benchmark problems, we compare the quality of the solutions produced by the IPC5 planners with respect to (a) the corresponding optimal solutions (when we could obtain such solutions), (b) the best sub-optimal solutions that we obtained by running other planners that did not enter IPC5 or other "ad hoc" methods, or (c) lower and upper bounds on the quality of the optimal solutions. Finally, for a selection of PDDL3 domains involving preferences, we evaluate the behaviour of the IPC5 planners with and without the preferences in the test problems.

All compared planners were run on the same machine. The CPU-time limit was 30 minutes and for each process at most 1 Gbytes of RAM was allowed.

4.1 The IPC5 planners

Participating in IPC5 were six optimal planners: CPT2, FDP, IPPLAN-1SC, MAXPLAN, MIPS-BDD and SATPLAN; and seven satisficing planners: DOWNWARD04SA, IPPLAN-G1SC, HPLAN-P, MIPS-BDD, MIPS-XXL, SGPLAN5 and YOCHANPS.

Table 1 briefly summarises the capabilities of the competitors. As can be seen, SGPLAN5 and MIPS-XXL are the only planners supporting all language features used across the competition domains. HPLAN-P and MIPS-BDD support soft goals and trajectory constraints, YOCHANPS soft goals and durative actions, and CPT2 supports durative actions, while the remaining competitors are limited to the propositional subtrack only.

The table also shows the plan quality measure optimised by each of the competitors. Most of the optimal planners are optimal only w.r.t. the parallel plan length, i.e., number of steps, but two of them, FDP and MIPS-BDD, optimise the number of actions in the plan and one of them, CPT2, is able to optimise makespan in problems with durative actions (which reduces to parallel length in the case of plain propositional problems). Among the satisficing planners, some try to find plans of good quality

according to the metric function specified in the problem definition, while some always aim to minimise the number of actions in the plan, and some may not consider plan quality at all, focusing only on finding a plan quickly. For the purpose of evaluating plans found by these planners, for propositional domains we measure plan quality in terms of number of actions (because, for these domains, all the satisficing planners use this criterion), while for the other domains we followed the principle of always evaluating them according to the problem metric.

In the rest of this section, we give brief descriptions of each of the competing planners. More details can be found in the short papers by the planners' authors that were collected in the IPC5 booklet [13].

CPT2 (Vincent Vidal & Sebastien Tabary)

CPT2 is the new version of the CPT planner that participated in IPC4, which combines a partial-order causal-link branching scheme with a powerful pruning mechanism based on constraint propagation. The planner handles durative actions and is optimal w.r.t. makespan. In the new version, the constraint formulation has been extended with several new pruning rules and the underlying CP engine has been replaced with a new, more efficient, implementation.

FDP (Stephane Grandcolas & Cyril Pain-Barre)

Like CPT2, FDP is also based on CP mechanisms, but designed for optimal sequential planning instead of temporal planning. FDP uses a planning graph-like structure to represent partial plans, a number of filtering rules to remove inconsistent possibilities from this structure, and a branching rule based on the deletion/preservation of each atom at each step to decompose the problem. By incrementally extending the size of the plan structure, FDP ensures that the plans it finds are optimal w.r.t. the number of actions.

IPPLAN (Menkes van den Briel, Subbarao Kambhampati & Thomas Vossen)

IPPLAN reformulates the planning problem as an integer programming (IP) problem and solves it using the CPLEX solver, combining and extending ideas from several previous IP encodings. It supports different IP formulations, some of which ensure optimality w.r.t. the parallel length in the plan while others don't. Thus, IPPLAN participated in two versions, one as an optimal (IPPLAN-1SC) and one as a satisficing planner (IPPLAN-G1SC). IPPLAN uses the STRIPS-to-SAS translator component from the FAST DOWNWARD planner [40] to convert problems from PDDL to a multi-valued state variable representation.

MAXPLAN (Zhao Xing, Yixin Chen & Weixiong Zhang)

MAXPLAN, similarly to SATPLAN, converts the planning problem into a series of propositional satisfiability problems and relies on a SAT solver to answer these. Like SATPLAN, MAXPLAN finds plans that are optimal w.r.t. the parallel length. However, MAXPLAN differs from SATPLAN in several important respects: The search for a shortest plan starts from an upper bound on the length and works downward until the last solution has been proven optimal, and the encoding into SAT incorporates information learned while solving previous SAT problems, as well as additional mutex constraints. The SAT solver is also modified to take advantage of the special structure of SAT problems that result from the encoding.

MIPS-BDD (Stefan Edelkamp)

MIPS-BDD is based on symbolic exploration of the state space, using BDDs to compactly represent sets of states. It handles PDDL3.0 trajectory constraints, which are compiled into Büchi automata and preferences. In propositional problems, it finds plans of minimal length (i.e., optimal w.r.t. the number of actions). In problems with preferences, the planner still searches for plans of increasing length and records the best (w.r.t. the problem metric) plan found so far, thus ensuring that when the search space has been completely exhausted, the value of the current best plan is optimal. In the competition, however, for problems with preferences, MIPS-BDD outputs the best plan found within the available CPU time, and therefore did not guarantee optimality.

SATPLAN (Henry Kautz, Bart Selman & Jörg Hoffmann)

The 2006 version of SATPLAN is an updated version of the SAT-based planner that participated in IPC4. It is optimal w.r.t. the parallel plan length. The main differences from the previous version are the use of a different SAT encoding (using variables for both actions and fluents) and the use of limited mutex propagation on the planning graph that forms the basis of the encoding.

DOWNWARD04SA (Malte Helmert)

DOWNWARD04SA is almost identical to the FAST DOWNWARD planner that participated in IPC4. It translates the PDDL problem specification into a multi-valued state variable representation ("SAS+") and searches for a plan using a heuristic derived from the causal graph constructed from the SAS+ representation. The main improvement compared to the IPC4 version of the planner is the addition of *safe abstraction*, a form of problem simplification that allows the planner to solve certain kinds of simple problems without search.

MIPS-XXL (Stefan Edelkamp, Shahid Jabbar & Mohammed Nazih)

MIPS-XXL uses a combination of several heuristic search methods, including an extension of METRIC-FF's search and a best-first search using external memory (disk). The planner handles PDDL3.0 trajectory constraints and preferences, by compiling them into Büchi automata and numerical fluents, respectively, as well as problems with durative actions and timed initial literals. Similarly to MIPS-BDD, the planner is "optimal in the limit", i.e., after exploring the complete state space, but was in the competition configured to output the best plan found within the available CPU time, with no guarantee of optimality.

SGPLAN5 (Chih-Wei Hsu, Benjamin W. Wah, Ruoyun Huang & Yixin Chen)

SGPLAN5 is the new version of the SGPLAN planner that also participated in IPC4. Features in the new version include a new heuristic, similar to the causal graph heuristic used by FAST DOWNWARD, for planning at the subgoal level, and extensions to handle all the new features of PDDL3.0. Like several other planners, for problems with preferences SGPLAN5 employs a strategy of iteratively searching for better plans after the first plan has been found. Unlike other competing planners, however, trajectory constraints are not compiled away but handled directly by the search.

HPLAN-P (Jorge Baier, Jeremy Hussell, Fahiem Bacchus & Sheila McIllraith)

HPLAN-P is a heuristic search planner for problems with preferences, built on top of the TLPLAN system. It also handles a subclass of trajectory constraints, by compiling these into parameterised finite state automata. The heuristic guiding the search combines estimates of the cost of reaching the goals, the cost of satisfying preferences, and different estimates of the final plan metric value.

YOCHANPS (J. Benton & Subbarao Kambhampati)

The YOCHANPS planner translates PDDL3.0 problems with preferences into so called *net benefit* (partial satisfaction) problems, and solves them using the SAPA^{PS} planner [23]. The main difference between problems as expressed in PDDL3.0 and net benefit problems is that while in the former, the objective is to minimise a penalty for violated (goal and precondition) preferences, in the latter the objective is to maximise the utility of achieved goals and minimise the cost of actions in the plan. SAPA^{PS} solves net benefit planning problems by heuristic search, using an inadmissible heuristic based on cost propagation over a planning graph.

4.2 Summary of the Main Results

Our experimental analysis contains many results. At a general level, we can derive at least the following eleven interesting observations. With respect to the *relative performance* of the *optimal propositional* IPC5 planners, we note that:

1. In terms of the number of problems solved within the competition CPU-time limit, considering the entire propositional problem set, MAXPLAN and SATPLAN perform similarly (although SATPLAN is, in general, better with respect to lower CPU-time limits) and both are significantly better that the other competing optimal planners. However, there is at least one domain in which both these planners are outperformed by some other IPC5 planner.

With respect to the *relative speed* of the *satisficing* IPC5 planners, we note that:

2. For every domain category, SGPLAN5 performs significantly better than the other IPC5 planners both in terms of CPU time and number of problems solved within any CPU-time limit up to 30 minutes.

With respect to the *plan quality* of the *satisficing* IPC5 planners, we note that:

3. In the propositional domain category, SGPLAN5 produces better quality plans than the other compared planners, except IPPLAN-G1SC, which, however, solves far fewer problems.

4. In the metric-temporal domain category, YOCHANPS performs better than the other compared planners but solves far fewer problems than SGPLAN5.

5. Across the simple and complex preferences domain categories, with respect to plan quality, SGPLAN5 performs generally better than the other IPC5 planners, except for MIPS-BDD in the simple preferences domain category, which, however, solves far fewer problems. For the qualitative preferences category, SGPLAN5 performs better than HPLAN-P and similarly to the other planners.

6. In domains with strong state trajectory constraints, MIPS-XXL is slightly better than SGPLAN5 in terms of plan quality (these two are the only planners competing in this domain category). However, the performance of both planners, in terms of the total number of problems solved in this domain category, is quite poor. Domains of this kind clearly present an open challenge for future research.

Finally, concerning the performance of the IPC5 planners in general, we note that:

7. A comparison of the IPC5 optimal propositional planners with the winner of the propositional optimal track at IPC4 (the 2004 version of SATPLAN) shows that the 2006 version of SATPLAN is significantly faster than the previous winner, while the performance of MAXPLAN and CPT2 is similar to that of the IPC4 winner. Moreover, CPT2, the only optimal temporal planner of IPC5, performs significantly better than the winner of the optimal temporal track of IPC4 (CPT).

8. A comparison of SGPLAN5 with the winners of the satisficing propositional and metric-temporal tracks at IPC4 (FAST DOWNWARD [40] and SGPLAN4 [17], respectively), shows that SGPLAN5 performs better with respect to both CPU time and plan quality.

9. A study of plan quality for a subset of propositional domains and problems shows that the solutions computed for these problems by the IPC5 satisficing planners are in general very good.[12]

10. A study of the quality of plans found by IPC5 satisficing planners for a large subset of metric-temporal domains indicates that these are generally of rather poor quality.

11. A study of the quality of plans found by the IPC5 satisficing planners for a subset of problems with preferences shows that (a) the planners are, in general, doing significantly better than "blind luck", i.e., than the expected value of plans found while disregarding the preferences, but also that (b) the planners often find plans of rather poor quality, compared to what is known to be achievable.

[12] The subset of domains and problems considered for this analysis and for the analysis described in the next two items are those for which we were able to compute optimal solutions or lower/upper bounds on the optimal solutions.

Problems	CPT2	FDP	IPPLAN-1SC	MAXPLAN	MIPS-BDD	SATPLAN
Solved/attempted	53 / 210	46 / 240	23 / 140	84 / 240	54 / 240	83 / 210
Success ratio	23.1%	19.1%	16.4%	35.0%	22.5%	39.5%

Table 2: Number of solved/attempted benchmark problems in the propositional IPC5 domains and success ratio for CPT2, FDP, IPPLAN-1SC, MAXPLAN, MIPS-BDD and SATPLAN.

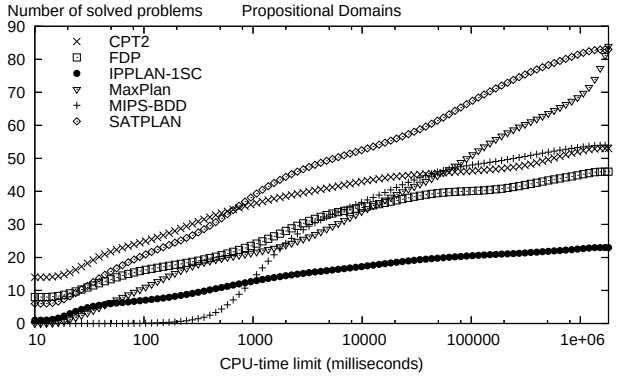

Figure 10: Number of problems solved by the IPC5 optimal planners with respect to an increasing CPU-time limit (logarithmic scale) for propositional domains.

4.3 Relative Performance of the Optimal Planners

Since the IPC5 optimal planners, CPT2, FDP, IPPLAN-1SC, MAXPLAN, MIPS-BDD and SATPLAN, produce optimal quality solutions, we compare them only in terms of number of solved problems and CPU time.[13] Although these planners are optimal with respect to two different measures, number of actions and number of parallel steps, here we disregard this difference and treat them all equally. All the IPC5 optimal planners attempted the (seven) propositional versions of the benchmark domains, while only one planner, CPT2, attempted the temporal version. For this reason we focus the analysis in this section on only propositional domains.

Number of solved problems

Table 2 shows the number of (propositional) benchmark problems attempted and solved by the IPC5 optimal planners, as well as their corresponding overall success ratio. We consider a problem non-attempted by a planner if the domain variant to which it belongs was not attempted by the planner; we consider a domain variant non-attempted by a planner if it contains no problem that was solved by the planner within the CPU-time limit of the competition (30 minutes). MAXPLAN solves more problems than any other compared planner, although the gap with respect to SATPLAN consists of only one problem, while SATPLAN is the planner with the best success ratio among those compared.

In Figure 10, the optimal planners are compared in terms of number of solved problems within a CPU-time limit ranging from 10 milliseconds to 30 minutes. When the CPU-time limit is very low (about half of a second), CPT2 solves more problems than any other competitor; for CPU-time limits higher than half a second SATPLAN solves more problems than the other optimal planners, except for the highest considered CPU time where MAXPLAN solves one problem more than SATPLAN, and both these two planners solve many more problems than every other compared planner. Note that although the number of problems solved by MAXPLAN increases significantly for CPU-time limits near the competition limit, we experimentally observed that with 30 additional CPU minutes this planner solves only two additional problems.

[13] During the evaluation of the competition results we realised that for a few problems MAXPLAN produced sub-optimal solutions. This was most probably due to an implementation bug. In the evaluation of an IPC5 optimal planner, each problem with a known sub-optimal solution is considered unsolved.

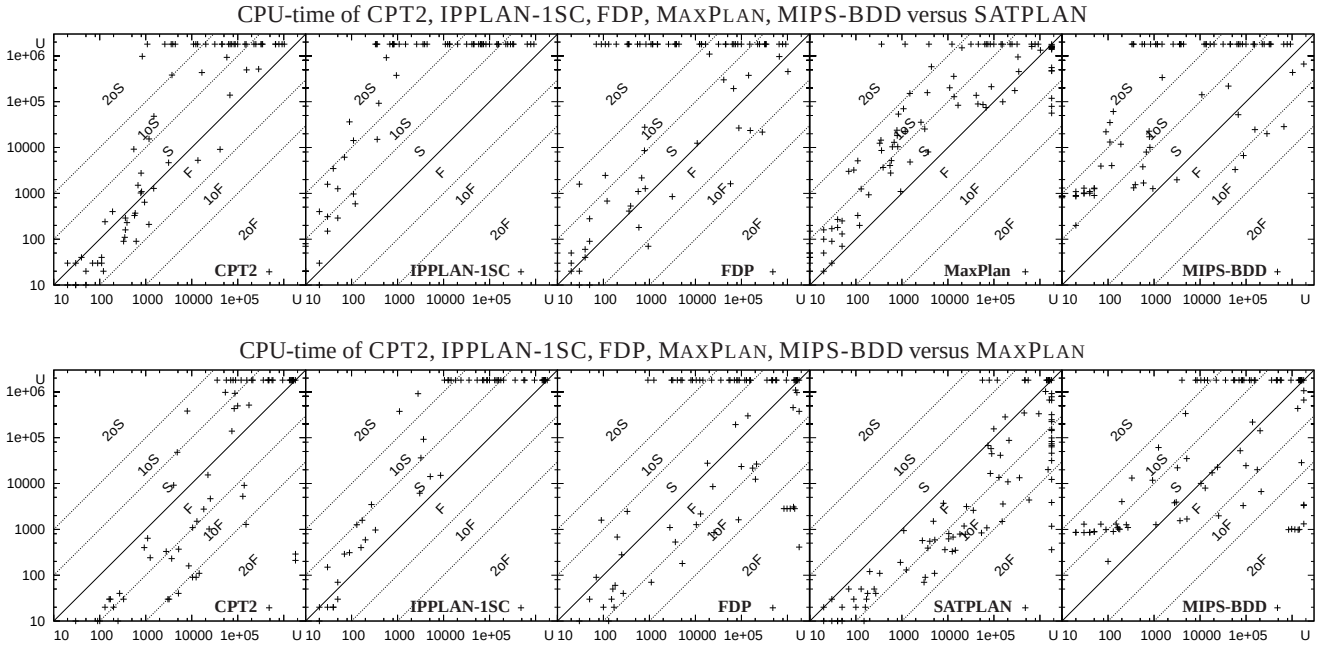

Figure 11: Performance of the optimal planners with respect to SATPLAN (plots on the top) and MAXPLAN (plots on the bottom) in terms of CPU time for propositional domains. In the plots on the top (bottom) of the figure, on the x-axis there are the CPU milliseconds of SATPLAN (MAXPLAN).

CPU-time performance relative to SATPLAN and MAXPLAN

In order to give a compact graphical representation of the overall performance of the IPC5 planners, we use scatterplots comparing the performance results of pairs of planners. For analysing the relative CPU time of the optimal planners, as well as of the satisficing planners in the next sections, we consider all the problems *attempted* by both the compared planners *and* solved by at least one of them.

The two sets of scatterplots in Figure 11 compare SATPLAN (plots on the top of the figure) and MAXPLAN (plots on the bottom), respectively, with the other IPC5 optimal planners. On the x-axis there is the performance of the reference planner (either SATPLAN or MAXPLAN), on the y-axis the performance of the other compared planner. For instance, consider the plot concerning the performance of CPT2 versus SATPLAN, each cross symbol indicates the CPU time used by CPT2 to solve a particular test problem (y-value) w.r.t. the time used by SATPLAN (x-value). When a cross appears above (under) the main diagonal of a scatterplot, CPT2 is slower (faster) than the reference planner; the distance of the cross from the main diagonal indicates the performance gap (the greater the distance, the greater the gap). The scatterplots have additional parallel lines dividing the picture into sectors. A cross under the line labelled "1oF" (over the line labelled "1oS") corresponds to a problem where CPT2 is at least one order of magnitude faster (slower) than the reference planner. Similarly, the lines labelled "2oF" ("2oS") and "3oF" ("3oS") identify sectors of scatterplots corresponding to problems where the compared planner is at least two and three, respectively, orders of magnitude faster (slower) than the reference planner. Crosses with "U" on the y-axis correspond to problems solved by CPT2 and unsolved by the reference planner; crosses with the "U" value on the x-axis correspond to problems solved by the reference planner and unsolved by CPT2.

In general, as is consistent with the analysis of Figure 10, the crosses with the "unsolved value" on the y-coordinate are much more dense than the crosses with the "unsolved value" on the x-coordinate, indicating that the reference planners solve more problems than the other compared planners.

In the plots on the top of Figure 11 most of the crosses are above the main diagonal, indicating that SATPLAN is generally faster than the other compared planners. The plots on the bottom part of the

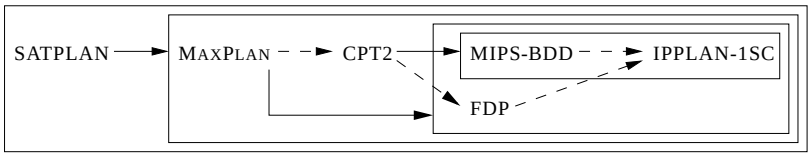

Figure 12: Partial order of the performance of the optimal IPC5 planners in terms of CPU time according to the Wilcoxon test for the propositional domains. A solid arrow indicates that a planner performs better than the other planner (or cluster of planners) with confidence level 99.9%; a dashed arrow indicates that a planner performs better with confidence level 98.1%.

figure give a less clear indication for MAXPLAN, since there are many crosses above the diagonal but also many below it.

The scatterplots of Figure 11 give a general visual indication of the relative performance of the compared planners considering all test problems, where in some cases the crosses corresponding to different problems cannot be distinguished because they appear overlapped. In order to have a somewhat more specific indication, we counted the number of crosses in each region of each plot. For the sake of brevity, here we omit these numerical data, which are available in [33], but, when useful, we present a qualitative assessment of the compared planners based on such data. According to this analysis, with respect to FDP, IPPLAN-1SC and MIPS-BDD, MAXPLAN is more often faster than slower, and the number of problems for which it is much faster is greater than the number of problems for which it is also much slower. On the other hand, CPT2 is often faster but solves fewer problems than MAXPLAN (see Table 2).

Statistical analysis

Figure 12 shows the results of the Wilcoxon sign-rank test comparing every possible combination of pairs of optimal IPC5 planners. We consider all the test problems attempted by both the compared planners and that are solved by at least one of them.

The data for carrying out the Wilcoxon test are derived as follows. For each planning problem we compute the difference between the CPU times of the two planners being compared. When a planner does not solve a problem, the corresponding CPU time is twice the competition CPU-time limit (i.e., 60 minutes).[14] This defines the samples of the test for the CPU-time analysis. The absolute values of these differences are then ranked by increasing numbers, starting from the lowest value. (The lowest value is ranked 1, the next lowest value is ranked 2, and so on.) Then we sum the ranks of the positive differences, and we sum the ranks of the negative differences. If the performance of the two compared planners is not significantly different, then the number of the positive differences is approximately equal to the number of the negative differences, and the sum of the ranks in the set of the positive differences is approximately equal to the sum of the ranks in the other set. Intuitively, the test considers a weighted sum of the number of times one planner performs better than the other. The sum is weighted because the test uses the performance gap to assign a rank to each performance difference.

The Wilcoxon test is characterised by a probability value, which represents the level of significance of the performance gap. In our analysis we use a default confidence level equal to 99.9%; hence, if the probability-value is greater than 0.001, then we refuse the hypothesis that the performance of the compared planners is statistically similar, and accept the alternative hypothesis that their performance is statistically different. Otherwise, there is no statistically significant evidence that they perform differently; so we consider that, on the evidence we have, they perform pretty much similarly. For the sake of conciseness, this paper contains only a general description of the statistical results; the interested reader can find more details in [33].

[14] This is the minimum value such that the performance gap for a problem solved by one planner and unsolved by the other compared planner is bigger than the performance gap for any problem solved by both the compared planners. An alternative choice would have been (1) using the competition limit, which, however, would have given less importance to the planner ability of solving a problem within the CPU-time limit, or (2) considering only the problems solved by both the planners, which in some cases could have significantly reduced the data for performing statistical test.

Category	P	DOW.04SA	IPPG.	MIPSB.	MIPSX.	SGPLAN5	HPLAN.	YOCH.
P	240	180 / 240	51 / 240	–	68 / 240	217 / 240	–	75 / 160
D	130	–	–	–	39 / 130	110 / 130	–	58 / 80
N	40	–	–	–	8 / 40	40 / 40	–	–
N+D	130	–	–	–	23 / 130	119 / 130	–	12 / 40
SG	110	–	–	29 / 110	43 / 110	110 / 110	–	34 / 90
N+SG	20	–	–	–	6 / 20	20 / 20	–	20 / 20
D+C	50	–	–	–	8 / 50	29 / 50	–	–
N+D+C	50	–	–	–	6 / 50	18 / 30	–	–
SC	100	–	–	16 / 80	12 / 80	100 / 100	70 / 100	–
N+D+SC	108	–	–	–	22 / 88	105 / 108	–	–
Total	978	180 / 240	51 / 240	45 / 190	235 / 938	868 / 958	70 / 100	199 / 390
Success %		75.0%	21.3%	23.7%	25.1%	90.6%	70.0%	51.0%

Table 3: Total number of IPC5 benchmark problems (column "P") and number of problems solved/attempted by DOWNWARD04sa, IPPLAN-G1SC, MIPS-BDD, MIPS-XXL, SGPLAN5, HPLAN-P and YOCHANPS (3rd–9th columns) for different domain versions (the names of the planners are abbreviated). "D" indicates domains with durative actions, "N" with numeric fluents, "SG" with soft goals, "C" with strong constraints on state trajectories, and "SC" with soft constraints on state trajectories. "–" indicates that the corresponding domains were not attempted by the planner.

Figure 12 contains a graphical summary of the Wilcoxon results about the relative performance of the optimal planners in terms of CPU time. A solid arrow from a planner A to a planner (or a cluster of planners) B indicates that the performance of A is statistically different from the performance of (every planner in) B, and that A performs better than (every planner in) B. A dashed arrow from A to B indicates that A is better than (every planner in) B a significant number of times, but there is no significant Wilcoxon relationship between A and (any planner in) B with a confidence level equal to 99.9%; on the other hand, the relationship does hold with a confidence level slightly less than 99.9%, which will be indicated in every statistical comparison (e.g., for the analysis in Figure 12 it is 98.1%). When there is no arrow connecting two (clusters of) planners, we consider these (clusters of) planners having a similar performance.

According to the results of the Wilcoxon test, in terms of CPU time SATPLAN performs statistically better than MAXPLAN, CPT2, FDP, MIPS-BDD and IPPLAN-1SC, while MAXPLAN performs better than FDP, IPPLAN-1SC and MIPS-BDD. MAXPLAN also performs better than CPT2, although with a confidence level equal to 98.1%. Note that this result is not inconsistent with the results in the scatterplot of Figure 11 indicating that CPT2 is generally faster than MAXPLAN for the problems solved by *both* planners. This is because for the Wilcoxon test we also consider the problems that are unsolved by one of the two compared planners (using twice the CPU-time limit for each unsolved problem). Hence, when the number of the problems solved by two compared planners is significantly different, like for MAXPLAN and CPT2 (see Table 2), the result of the Wilcoxon test can be different from the observations that we can make from the corresponding scatterplot. In fact, if we consider only the subset of the problems solved by both these planners, the results of the Wilcoxon test is that CPT2 performs statistically better than MAXPLAN.

4.4 Relative Performance of the Satisficing Planners

We compare the performance of the IPC5 satisficing planners in terms of number of solved problems, CPU time and plan quality. Table 3 shows the number of problems solved/attempted by the compared planners for the different versions of the benchmark domains, as well as their overall success ratio. As previously noted, the only planners that support all the planning capabilities used in the competition are MIPS-XXL and SGPLAN5, but with a very different performance. In particular, for each domain version, SGPLAN5 solves a much higher number of problems than any other satisficing planner, and the success ratio of this planner is the highest among the compared planners.

In the rest of this section, we evaluate the performance of the satisficing planners for different domain categories.

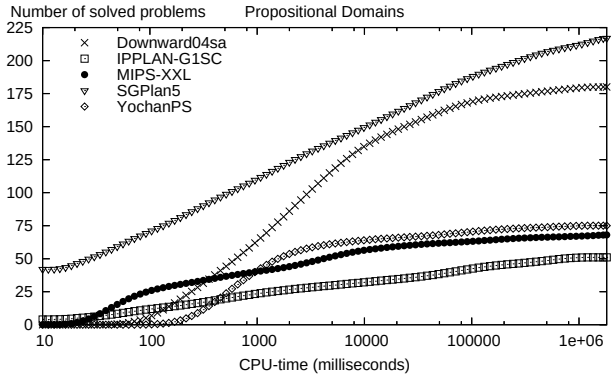

Figure 13: Number of IPC5 propositional benchmark problems solved by the IPC5 satisficing planners with respect to an increasing CPU-time limit (logarithmic scale).

4.4.1 Propositional Domains

Five of the IPC5 satisficing planners attempted the (seven) propositional versions of the benchmark domains: DOWNWARD04SA, IPPLAN-G1SC, MIPS-XXL, SGPLAN5 and YOCHANPS. In this section we analyse the relative performance of these planners in more detail.

Number of solved problems

Figure 13 shows the number of (propositional) benchmark problems solved by the IPC5 satisficing planners within an increasing CPU-time limit, which ranges from 10 milliseconds to 30 minutes. Regardless of the CPU-time limit, SGPLAN5 solves more problems than the other compared planners. For CPU-time limits greater than about one second, DOWNWARD04SA solves many more problems than IPPLAN-G1SC, MIPS-XXL and YOCHANPS, and for CPU-limits between 10 and 100 seconds it performs almost as well as SGPLAN5.

CPU time and plan quality relative to SGPLAN5

Figure 14 gives a compact representation of the overall performance of the IPC5 satisficing planners w.r.t. SGPLAN5 in terms of CPU time and plan quality for all the propositional benchmark problems. Concerning CPU time, in general, SGPLAN5 solves a problem more quickly than any other compared planner, and often the performance gap is at least one order of magnitude in favour of this planner. SGPLAN5 is almost always faster than IPPLAN-G1SC and MIPS-XXL. Compared to DOWNWARD04SA and YOCHANPS, for several problems SGPLAN5 is slower. However, the number of problems for which SGPLAN5 is faster is much higher than the number of problems for which it is slower, especially for performance gaps larger than one order of magnitude (the interested reader can find an exact count in [33]).

The plots on the bottom part of Figure 14 compare the performance of the IPC5 satisficing planners for propositional problems in terms of plan quality with respect to the performance of SGPLAN5. For this analysis, as well as for the following plan quality comparisons, we consider all the benchmark problems solved by *both* the compared planners. In each of these plots, a cross above (below) the main diagonal corresponds to a problem for which the plan computed by the compared IPC5 planner is worse (better) than the plan computed by SGPLAN5 for the same problem. The crosses above the diagonal labelled "2tW" (below the diagonal labelled "2tB") correspond to problems for which the plans computed by the compared IPC5 planner are at least two times worse (better) than the ones computed by SGPLAN5 for the corresponding problems. The plans computed by SGPLAN5 are generally better than those computed by DOWNWARD04SA, since most of the crosses in the corresponding plot appear above the main diagonal. However, the plan quality plots in Figure 14 do not give a very clear indication for the other pairs of compared planners. The numerical data in each plot help to better understand their relative performance: the number of problems for which SGPLAN5 computes plans with quality better

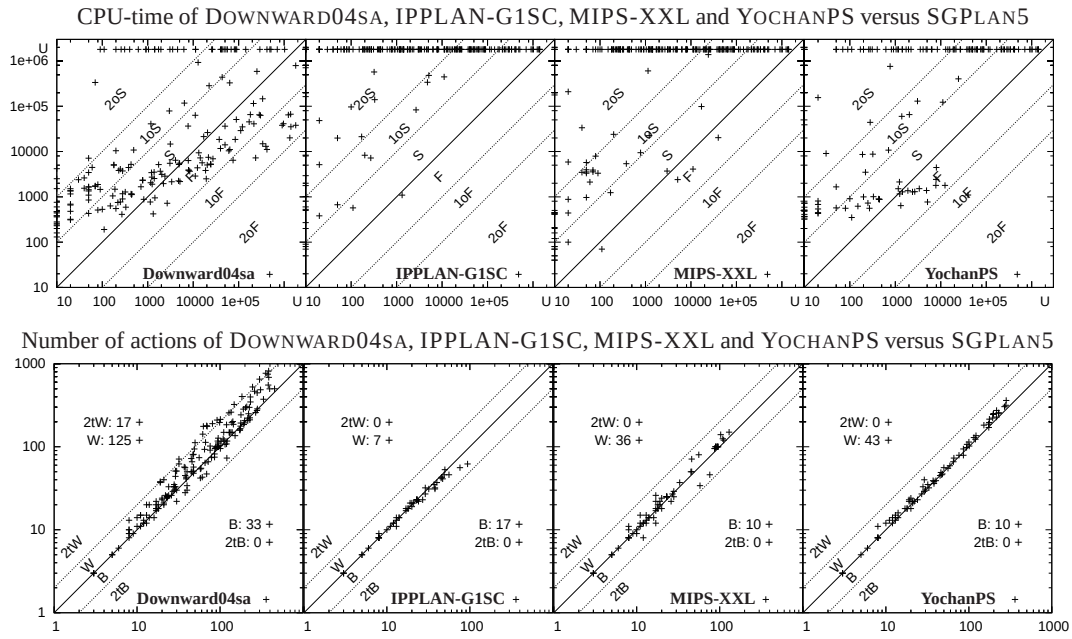

Figure 14: Performance of the IPC5 satisficing planners for propositional domains w.r.t. SGPLAN5 in terms of CPU time (plots on the top) and plan quality (plots on the bottom) for propositional domains. In the plots on the top (bottom) part of the figure, on the x-axis there is the CPU time (number of actions) of SGPLAN5; on the y-axis there is the CPU time (number of actions) of DOWNWARD04SA, IPPLAN-G1SC, MIPS-XXL and YOCHANPS, respectively.

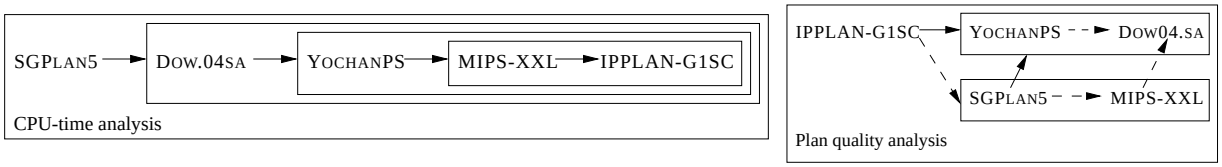

Figure 15: Partial order of the performance of the IPC5 satisficing planners in terms of CPU time and plan quality according to the Wilcoxon test for the IPC5 propositional domains. Dashed arrows indicate that the corresponding performance relationships hold with confidence level 97.9%.

than the plans generated by MIPS-XXL and YOCHANPS (crosses in the "W" and "2tW" sectors) is greater than the number of problems for which it computes worse plans (crosses in the "B" and "2tB" sectors). On the other hand, IPPLAN-G1SC produces better quality plans more often than SGPLAN5.

Statistical analysis

Like for the optimal planners, we use the Wilcoxon test to understand whether the performance gaps between two IPC5 satisficing planners are significant. For these planners, we test not only the difference in CPU time but also the difference in the quality of the plans they find. The test procedure for plan quality is essentially the same as the one previously described (in Section 4.3), but with two main differences: First, we normalise the difference by dividing it with the value of the better plan (so that, for example, if the value of the plan found by planner A is 200 and the value of the plan found by planner B is 220, the difference is 10%, in favour of planner A, if the objective is to minimise). Second, we limit the comparison to the set of problems solved by both planners.

Both these modifications stem from the same cause, which is that the magnitude of the value of a good plan may vary greatly between domains, or even between problems in the same domain (unlike

CPU time, which is measured on the same scale for every problem). This is particularly acute in PDDL2 (metric-temporal) and PDDL3.0 problems. For example, in the `Openstacks` Time domain quality is measured by plan makespan, with values of good plans ranging in the hundreds, while in `Openstacks` MetricTime the measure of quality is a sum of makespan and cost, and good plans have values of several thousands for larger instances. But it happens also for some propositional domains, even though the measure of quality is plan length for all of them. For example, the longest plan found for any instance in the `Storage` Propositional domain contains 80 actions, while in the `Openstacks` Propositional domain, more than half the instances have minimal plan lengths greater than that, and several containing over 450 actions.

Since the Wilcoxon test uses a ranking of the differences between values in each sample pair, if we compared the absolute plan quality values directly, without normalisation, such differences in the magnitude of values between domains could result in an unintended bias, with small relative differences in a domain with large values weighted as more important that larger relative differences in a domain with small values.[15] Normalisation helps to avoid this problem. However, the fairly simple normalisation scheme we apply is not perfect, as, for example, it does not take into account the amount of difference in plan quality that is possible (i.e., the difference between the optimal and worst possible plans), which may also be subject to variation between domains. Results should be interpreted in light of this.

Figure 15 gives a graphical summary of the Wilcoxon results about the relative performance of the IPC5 satisficing planners for the benchmark propositional problems of the competition. In terms of CPU time, SGPLAN5 performs statistically better than any other compared planner. In terms of plan quality, IPPLAN-G1SC performs better than DOWNWARD04SA and YOCHANPS, and it also performs better than MIPS-XXL and SGPLAN5 but with confidence level 97.9%.

4.4.2 Metric-Temporal Domains

The IPC5 metric-temporal domain versions consist of nine domains: one version of `TPP` involving numerical fluents but without action durations; a version for each of `Openstacks`, `Storage`, `Trucks` and `Pipesworld` involving action durations, but without numerical fluents; and a versions for each of `TPP`, `Openstacks`, `Pathways` and `Rovers` involving both action durations and numerical fluents. The IPC5 satisficing planners that attempted the benchmark problems in these domains are MIPS-XXL, SGPLAN5 and YOCHANPS.

Number of solved problems

Figure 16 shows the number of metric-temporal benchmark problems solved by the IPC5 satisficing planners within a CPU-time limit ranging from 10 milliseconds to 30 minutes. For every CPU-time limit that we considered, SGPLAN5 always solves many more problems than the other compared planners; while, in terms of solved problems, the performance of MIPS-XXL and YOCHANPS is similar. Remarkably, within 30 minutes SGPLAN5 solves 269 of the 300 benchmark problems, while YOCHANPS and MIPS-XXL only a much smaller percentage of them.

CPU time and plan quality relative to SGPLAN5

The scatterplots on the left side of Figure 17 give a compact graphical representation of the performance of MIPS-XXL and YOCHANPS w.r.t. SGPLAN5 in terms of CPU time for all metric-temporal benchmark problems. Since in each plot every cross appears above the main diagonal of the plot, it is easy to see that SGPLAN5 outperforms MIPS-XXL and YOCHANPS in terms of CPU time.

In the scatterplots on the right side of Figure 17, comparing the performance of MIPS-XXL and YOCHANPS relative to SGPLAN5 in terms of plan quality, many crosses are above the main diagonal, but there are also many of them that are below it. As indicated by the numerical data in these plots, which count the number of crosses in the different sectors, in terms of plan quality SGPLAN5 is not the best IPC5 satisficing planner that attempted the metric-temporal problems. The plans computed

[15] We have observed that this does indeed happen, and does influence the results of some statistical tests, though not in a large number of cases.

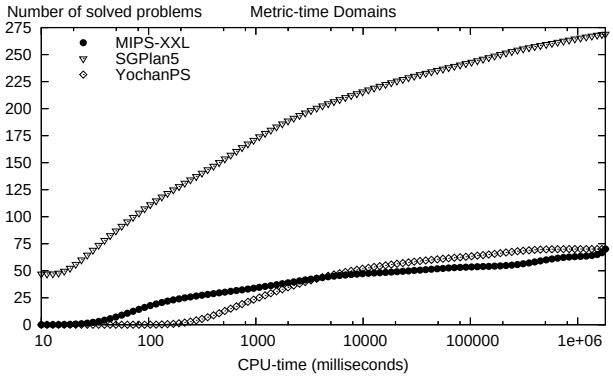

Figure 16: Number of problems solved by the IPC5 satisficing planners with respect to an increasing CPU-time limit (logarithmic scale) for metric-time domains.

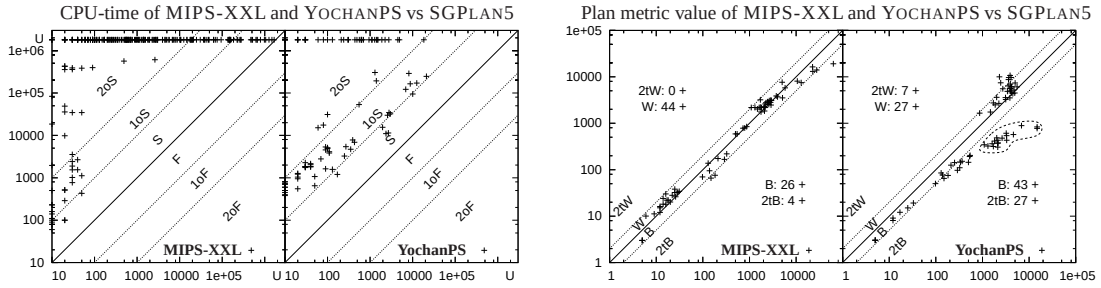

Figure 17: Performance of MIPS-XXL and YOCHANPS with respect to SGPLAN5 in terms of CPU time (left plots) and plan quality (right plots) for metric-time domains. In the plots on the left (right), on the x-axis there is the CPU time (number of actions) of SGPLAN5; on the y-axis there is the CPU time (number of actions) of MIPS-XXL and YOCHANPS.

by SGPLAN5 are more often better than worse with respect to the plans generated by MIPS-XXL. However, there is no SGPLAN5 plan that is at least two times better than the corresponding MIPS-XXL plan, while MIPS-XXL computes some plans that are significantly better than those computed by SGPLAN5. Moreover, in terms of plan quality, YOCHANPS tends to perform better than SGPLAN5: the plans generated by YOCHANPS are more often better than worse, and there is a large number of plans that are at least two times better.

Interestingly, YOCHANPS produces plans of very good quality (both compared to the other two planners and to known upper and lower bounds) for problems in the `Openstacks` Time domain (which correspond to the crosses inside the dashed region in the right side scatterplot of Figure 17). This is a pure makespan optimisation domain. All three planners use some form of post-scheduling of plans to improve makespan, but it appears that the schedulers used by SGPLAN5 and MIPS-XXL produce very poor results in this domain: by recovering the partial order of SGPLAN5's and MIPS-XXL's plans and rescheduling them optimally (using the simple critical path algorithm), we were able to improve the makespan of these plans significantly. However, the rescheduled plans are still worse than those produced by YOCHANPS, indicating that this planner is not only doing a better job of scheduling the sequential plan it finds, but that it is also better at finding plans that can be scheduled with low makespan, at least in this domain.

Statistical analysis

The qualitative results of the Wilcoxon test comparing the performance of the IPC5 satisficing planners for the metric-temporal benchmark problems are given in Figure 18 (further details are available in [33]). This analysis confirms that the performance gap between SGPLAN5 and the other compared

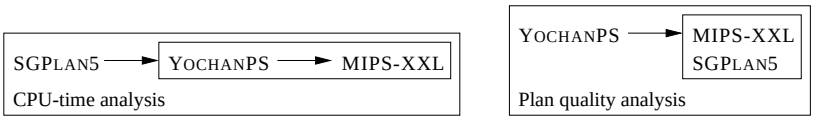

Figure 18: Partial order of the performance of MIPS-XXL, SGPLAN5 and YOCHANPS in terms of CPU time and plan metric value according to the Wilcoxon test for the IPC5 benchmark metric-temporal problems.

planners is significant in terms of CPU time and is in favour of SGPLAN5. While in terms of plan quality, YOCHANPS performs statistically better than the other compared planners.

4.4.3 Domains with SimplePreferences

The IPC5 SimplePreference domain category contains six domains: a version of each of `TPP`, `Openstacks`, `Pathways`, `Storage` and `Trucks`, all of which are propositional, and one version of `Rovers`, which also uses numeric fluents and effects (however, these are used only in a very simple manner to encode action costs). The IPC5 planners that attempted this category of benchmark domains are: MIPS-BDD, MIPS-XXL, SGPLAN5 and YOCHANPS. For most of these problems, computing a valid plan is very simple: 90 problems over 130 have only soft goals, and hence the empty plan is a solution for each of them. On the other hand, for these problems computing a plan with good quality is not a trivial task.

Number of solved problems

Figure 19 shows the number of problems solved by MIPS-BDD, MIPS-XXL, SGPLAN5 and YOCHANPS within a CPU-time limit ranging from 10 milliseconds to 30 minutes. For every CPU-time limit considered in this analysis, SGPLAN5 solves many more problems than the other compared planners, and within 30 CPU minutes it solves all problems. For CPU-time limits between about 1 second and 30 minutes, YOCHANPS solves more problems than MIPS-XXL and MIPS-BDD, while for lower limits these three planners perform similarly.[16]

Within the highest CPU-time limit considered (30 minutes), SGPLAN5 solves all the 130 benchmark problems of the SimplePreferences domain versions, while the number of problems solved by YOCHANPS, MIPS-XXL and MIPS-BDD is much lower. This is somewhat surprising, because the empty plan is a valid plan for 90 of these benchmark problems. Note that SGPLAN5 computes only one empty plan, YOCHANPS two, MIPS-BDD five, and MIPS-XXL nine.

CPU time and plan quality relative to SGPLAN5

Figure 20 gives a representation of the overall performance of MIPS-BDD, MIPS-XXL and YOCHANPS w.r.t. SGPLAN5 in terms of CPU time and plan quality for all IPC5 benchmark problems in the SimplePreferences domain versions. The distribution of the crosses in the plots on the top part of the figure show that SGPLAN5 is generally faster than the compared planners. Moreover, the plots on the bottom part of the figure indicate that SGPLAN5 performs better than MIPS-XXL and YOCHANPS in terms of plan quality as well. However, the comparison of the plans generated by SGPLAN5 and MIPS-BDD does not clearly indicate that one planner performs better than the other in terms of plan quality. The number of problems for which SGPLAN5 computes plans that are better than those generated by MIPS-BDD is slightly greater than the number of problems for which it computes worse plans, but there is no problem for which SGPLAN5 computes a significantly better plan.

[16] It is worth noting that MIPS-XXL solves several problems using a CPU-time limit near the limit of the competition. The reason is not fully clear, but we think it is due to the plan optimisation phase of the planner, which exploits the entire available CPU time and, probably because of an implementation bug, terminates slightly after the competition CPU-time limit. However, in our analysis we do not consider such plans because the CPU-time limit of IPC5 was 30 minutes, and we don't have data concerning plans produced by the other compared planners using additional CPU time.

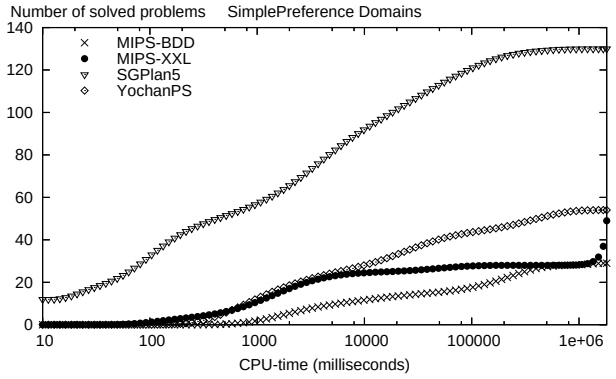

Figure 19: Number of problems solved by the IPC5 satisficing planners with respect to a given CPU-time limit (logarithmic scale) for the IPC5 benchmark SimplePreferences problems.

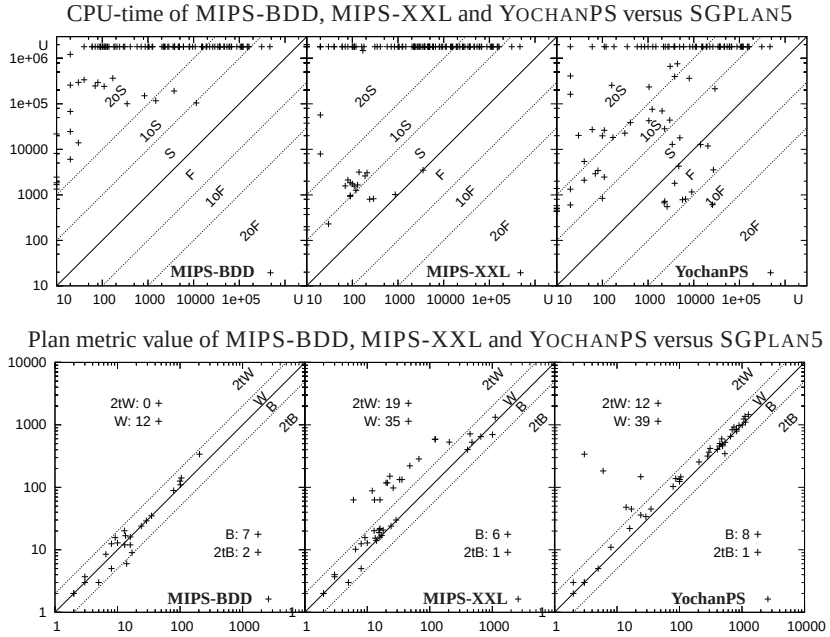

Figure 20: Performance of MIPS-BDD, MIPS-XXL and YOCHANPS with respect to SGPLAN5 in terms of CPU time (top plots) and plan quality (bottom plots) for the IPC5 benchmark SimplePreferences problems. In the plots on the top (bottom) part of the figure, on the x-axis there is the CPU time (plan metric value) of SGPLAN5; on the y-axis there is the CPU time (plan metric value) of MIPS-BDD, MIPS-XXL and YOCHANPS.

Statistical analysis

The results of the Wilcoxon test comparing the performance of the IPC5 satisficing planners for the benchmark SimplePreferences problems (Figure 21) confirm the general picture indicated by the previous analysis in Figure 20: SGPLAN5 performs statistically better than YOCHANPS and MIPS-XXL in terms of CPU time and plan quality, while it performs better than MIPS-BDD in terms of CPU time, and similarly to MIPS-BDD in terms of plan quality.

Figure 21: Partial order of the performance of MIPS-BDD, MIPS-XXL, SGPLAN5 and YOCHANPS according to the Wilcoxon test for the IPC5 SimplePreferences problems. A dashed arrow indicates that the performance relationship holds with confidence level 96.9%.

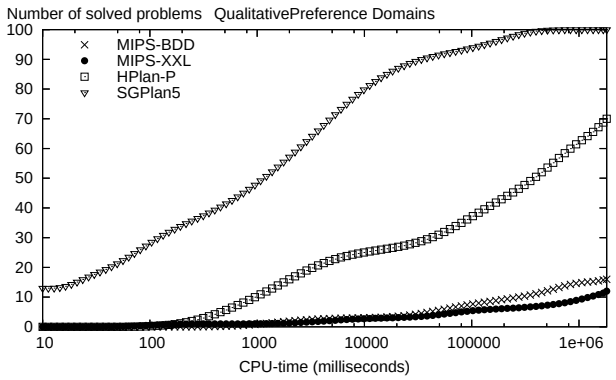

Figure 22: Number of problems solved by the IPC5 satisficing planners with respect to an increasing CPU-time limit (logarithmic scale) for the IPC5 problems in the QualitativePreferences domains.

4.4.4 Domains with QualitativePreferences

The results of this experimental comparison concern the QualitativePreferences versions of five benchmark domains (`TPP`, `Openstacks`, `Rovers`, `Storage` and `Trucks`), which are propositional domains extended with soft state trajectory constraints as well as soft goals. Similar to the problems of the SimplePreferences versions of our benchmark domains, for many problems with qualitative preferences finding a valid plan is a simple task (in particular, 40 of the 100 benchmark problems that we used have no hard goal, and hence the empty plan is a valid plan for them), but computing a good quality plan can be much more difficult.

The IPC5 planners supporting PDDL3 qualitative preferences that we compare in this section are MIPS-BDD, MIPS-XXL, HPLAN-P and SGPLAN5.

Number of Solved Problems

Figure 22 shows the number of problems with preferences over action preconditions and state trajectory constraints that are solved by the compared planners within an increasing CPU-time limit ranging from 10 milliseconds to 30 minutes. Overall, for every CPU-time limit considered, SGPLAN5 solves more problems than the other planners; for CPU-time limits higher than about 100 milliseconds, HPLAN-P solves more problems than MIPS-XXL and MIPS-BDD, while MIPS-XXL and MIPS-BDD perform similarly.

It is worth noting that, within about 5 CPU minutes, SGPLAN5 solves all these IPC5 benchmark problems producing no empty plan. By contrast, a small percentage of the plans generated by the other three planners are empty. Most of these plans are computed for the `TPP` domain.

CPU time and plan quality relative to SGPLAN5

Figure 23 shows the performance of HPLAN-P, MIPS-BDD and MIPS-XXL w.r.t. SGPLAN5 in terms of CPU time and plan quality. As indicated by the distribution of the crosses in the plots of the top part of the figure, SGPLAN5 is always faster than MIPS-BDD and MIPS-XXL, and very often it

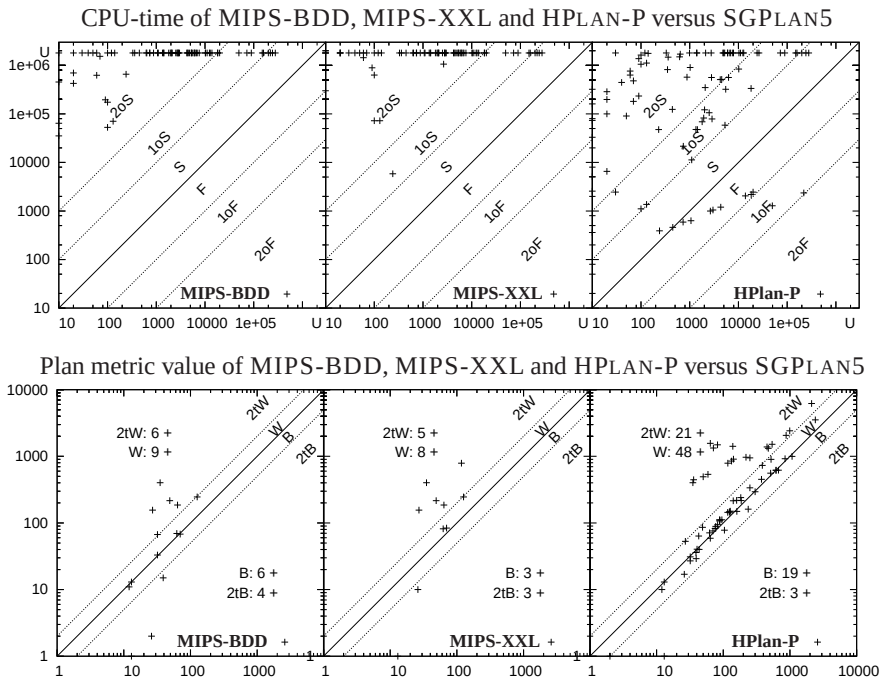

Figure 23: Performance of MIPS-BDD, MIPS-XXL and HPLAN-P w.r.t. SGPLAN5 in terms of CPU time (top plots) and plan quality (bottom plots) for the IPC5 benchmark problems in the QualitativePreferences domains. In the plots in the top (bottom) part of the figure, on the x-axis there is the CPU time (plan metric value) of SGPLAN5; on the y-axis there is the CPU time (plan metric value) of the other compared planners.

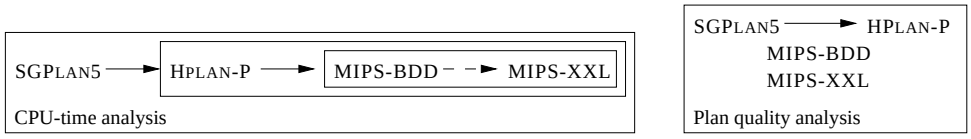

Figure 24: Partial order of the performance of MIPS-BDD, MIPS-XXL, HPLAN-P and SGPLAN5 according to the results of Wilcoxon test for IPC5 benchmark problems of the QualitativePreferences domains.

is faster than HPLAN-P as well (with the exception of several TPP problems, for which we observed that HPLAN-P generates empty plans).

The distribution of the crosses in the plots on the bottom part of the figure shows that SGPLAN5 also computes plans that are more often better than worse w.r.t. the plans generated by HPLAN-P, MIPS-BDD and MIPS-XXL. It is worth noting that the computed empty plans are again worse than the non-empty plans.

Statistical analysis

Figure 24 shows the results of the Wilcoxon test comparing the performance of the IPC5 planners for the benchmark problems in the QualitativePreferences domain versions. In terms of CPU time required for finding a valid plan, SGPLAN5 performs statistically better than the other compared planners. In terms of plan quality, SGPLAN5 performs statistically better than HPLAN-P, while it performs similarly to MIPS-BDD and MIPS-XXL.

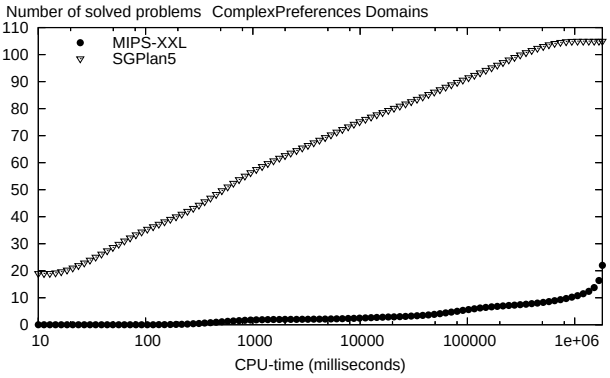

Figure 25: Number of problems solved by MIPS-XXL and SGPLAN5 with respect to an increasing CPU-time limit (logarithmic scale) for the IPC5 problems in ComplexPreferences domains.

4.4.5 Domains with ComplexPreferences

The results of this experimental comparison concern five metric-temporal domains extended with soft goals as well as soft state trajectory constraints. These are the ComplexPreferences versions of domains `TPP`, `Pathways`, `Pipesworld`, `Storage` and `Trucks`. Only two planners attempted this category of benchmark problems: SGPLAN5 and MIPS-XXL.

Number of solved problems

Figure 25 shows the number of problems solved by MIPS-XXL and SGPLAN5 within an increasing CPU-time limit ranging from 10 milliseconds to 30 minutes. For every CPU-time limit considered, SGPLAN5 solves many more problems than MIPS-XXL. SGPLAN5 solves all test problems except three large ones in domain `Pipesworld` ComplexPreferences. It is worth noting that, while most of these problems can be solved by the empty plan (because all goals are soft), neither of the compared planners generates empty plans.

CPU time and plan quality (direct comparison)

Figure 26 compares the performance of MIPS-XXL and SGPLAN5 in terms of CPU time and plan quality. Since in the plot on the left side of the figure all crosses are above the main diagonal, it easy to see that SGPLAN5 outperforms MIPS-XXL in terms of CPU time. In terms of plan quality (plot on the right side of the figure), SGPLAN5 again performs generally better than MIPS-XXL, although for few test problems it performs significantly worse. Note that there are two classes of test problems here, depending on whether the plan metric expression has to be minimised (crosses in the plot) or maximised (circles in the plot). For maximisation problems, when the circles appear *below* the main diagonal, it means that MIPS-XXL performs *worse* than SGPLAN5 (for minimisation problems it is the other way around).

Statistical analysis

The results of the Wilcoxon test comparing the performance of MIPS-XXL and SGPLAN5 for the IPC5 problems with complex preferences indicate that in terms of CPU time SGPLAN5 performs statistically better than MIPS-XXL, while in terms of plan quality it performs better with confidence level 98.1%.

4.4.6 Domains with MetricTimeConstraints

The only two IPC5 planners that support this category of benchmark problems are SGPLAN5 and MIPS-XXL. The results of their experimental comparison concern the MetricTimeConstraint version

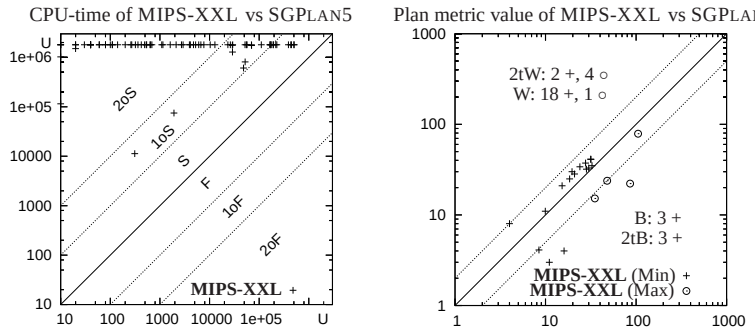

Figure 26: Performance of MIPS-XXL and SGPLAN5 in terms of CPU time (left plot) and plan quality (right plot) for the IPC5 problems in ComplexPreferences domains. In the plot on the left (right) side, on the x-axis there is the CPU time (plan metric value) of SGPLAN5; on the y-axis there is the CPU time (plan metric value) of MIPS-XXL. For plan quality, in case of plan metric maximisation, MIPS-XXL performs worse than SGPLAN5 when the circles appear *below* the main diagonal, while for plan metric minimisation problems, it performs worse when the crosses are *above* the main diagonal.

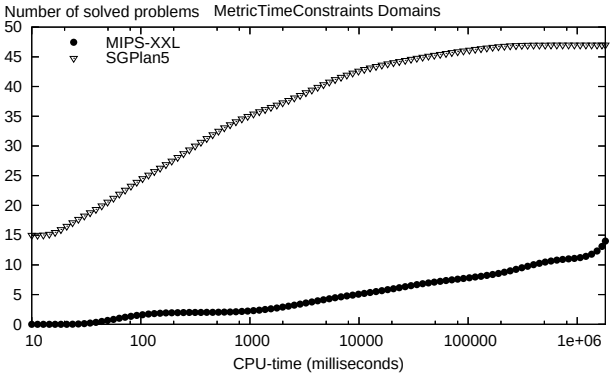

Figure 27: Number of problems solved by MIPS-XXL and SGPLAN5 with respect to an increasing CPU-time limit (logarithmic scale) for the IPC5 benchmark domains with strong plan trajectory constraints.

of four domains (`Pipesworld`, `Trucks`, `Storage` and `TPP`), involving various types of (strong) state trajectory constraints.

Number of solved problems

Figure 27 shows the number of problems solved by MIPS-XXL and SGPLAN5 within an increasing CPU-time limit ranging from 10 milliseconds to 30 minutes. For every CPU-time limit considered, SGPLAN5 solves many more problems than MIPS-XXL, but about 50% of the IPC5 problems in this domain category remain unsolved.

CPU time and plan quality (direct comparison)

Figure 28 shows the performance of MIPS-XXL and SGPLAN5 in terms of CPU time and plan quality for the IPC5 problems involving strong state trajectory constraints. Since all crosses in the plot on the left side of the figure are above the main diagonal, SGPLAN5 is always faster than MIPS-XXL. However, in terms of plan quality, the plot on the right side of the figure shows that often MIPS-XXL generates plans that are better than or similar to the corresponding plans computed by SGPLAN5.

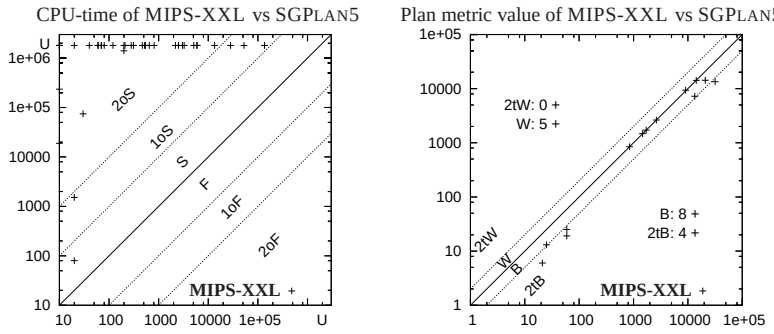

Figure 28: Performance of MIPS-XXL with respect to SGPLAN5 in terms of CPU time (left plot) and plan quality (right plot) for the IPC5 problems with strong state trajectory constraints. In the plot on the left (right) side on the x-axis there is the CPU time (plan metric value) of SGPLAN5, and on the y-axis there is the CPU time (plan metric value) of MIPS-XXL.

Statistical analysis

According to the Wilcoxon test comparing the CPU times of SGPLAN5 and MIPS-XXL for the IPC5 problems involving strong state trajectory constraints, as expected by observing the plot on the left of Figure 28, SGPLAN5 performs better than MIPS-XXL. Concerning plan quality, the number of problems for which both the planners compute a solution is too low for a significant statistical analysis of the results.

4.5 How Good is the Performance of the IPC5 Planners?

In the previous section, we have given a comparative evaluation of the performance of the IPC5 planners; in this section we analyse their CPU time and plan quality with respect to (a) the winners of the previous competition, and (b) exact or estimated lower/upper bounds on the distance from the optimal solutions of the IPC5 plans for a subset of the benchmark problems. For (a) we separately analyse optimal planners, satisficing propositional planners and satisficing metric-temporal planners; for (b) we separately analyse a subset of the IPC5 plans for PDDL2 and PDDL3 problems.

4.5.1 Performance Relative to the IPC4-Winner Optimal Planner

The optimal planners that won IPC4 are: for the propositional track, the 2004 version of SATPLAN [46] (here indicated with SATPLAN.ipc4); for the metric-time track, CPT [66] (here indicated with CPT.ipc4). *The analysis in this section shows that, overall, the optimal planners that won IPC5 improve on the performance of the optimal planners that won IPC4.*

The tables in Figure 29 compare the CPU times of SATPLAN.ipc4 and CPT.ipc4 with the CPU times of the IPC5 optimal planners for all the IPC5 propositional and temporal domains. For this analysis, as well as for the comparison of the best IPC4 satisficing planners with the IPC5 satisficing planners, we summarize the result of the experiment by counting the number of test problems in which the IPC4 winner planner is faster (slower), at least one order of magnitude faster (slower) and at least two orders of magnitude faster (slower) than the compared IPC5 planner (these values are lower bounds because for the unsolved problems here we consider the exceeded CPU-time limit, which is a lower bound of the actual solution time). Bold data emphasise the comparisons that are in favour of the IPC5 planners.

Overall, we have that for a large number of test problems, the IPC5 version of SATPLAN is faster than the IPC4 winner version, which is faster than the new version only for a few problems. Moreover, for many test problems the 2006 version of SATPLAN is at least one order of magnitude faster, while this is never the case for the IPC4 version. On the other hand, according to this analysis, we observe no significant improvement for the other optimal propositional planners of IPC5.

SATPLAN.ipc4 vs	2oS	1oS	S	F	1oF	2oF
CPT2	2	9	**34**	32	9	2
FDP	8	14	26	49	28	16
IPPLAN-1SC	0	0	0	50	30	10
MIPS-BDD	8	14	22	62	45	14
MAXPLAN	0	8	41	59	9	1
SATPLAN	**1**	**17**	**65**	8	0	0

CPT.ipc4 vs	2oS	1oS	S	F
CPT2	**3**	**7**	**19**	0

Figure 29: Numbers of IPC5 test problems for which SATPLAN.ipc4 and CPT.ipc4 are faster/slower than the IPC5 optimal propositional and temporal planners. The table columns distinguish the number of problems for which the reference planner is faster (slower), F(S)-columns, and the *minimum* number of problems for which it is at least one order of magnitude faster (slower), 1oF(S)-columns, and at least two orders of magnitude faster, 2oF(S)-columns.

DOWNWARD.ipc4 vs	2oS	1oS	S	F	1oF	2oF
DOWNWARD.04SA	0	1	**83**	51	2	1
IPPLAN-G1SC	0	4	14	164	144	111
MIPS-XXL	0	2	39	139	111	85
SGPLAN5	**26**	**114**	**165**	54	13	0
YOCHANPS	0	0	16	113	59	35

DOWNWARD.ipc4 vs	2tW	W	B	2tB
DOWNWARD.04SA	0	3	72	17
IPPLAN-G1SC	0	**29**	3	0
MIPS-XXL	**1**	**16**	13	0
SGPLAN5	0	**84**	65	0
YOCHANPS	0	14	41	0

Figure 30: Minimum numbers of IPC5 test problems for which DOWNWARD.ipc04 performs better/worse than the IPC5 satisficing propositional planners. The table on the left concerns CPU time (the meanings of the column labels is as in Figure 29); the table on the right gives the numbers of problems for which DOWNWARD.ipc4 produces better (worse) plans, B(W)-columns, and at least two times better (worse) plans, 2tB(W)-columns.

The results of the Wilcoxon test comparing the IPC4 version of SATPLAN and the IPC5 optimal propositional planners indicate that the only IPC5 planner that performs statistically better than SATPLAN.ipc4 is the IPC5 version of SATPLAN.

Concerning the optimal metric-temporal planners, the plot on the right side of Figure 29 indicates that CPT2, which was the only competing IPC5 planner of this category, significantly improves the previous (IPC4 awarded) version of CPT. The result of the Wilcoxon test confirms that CPT2 is statistically faster than CPT.ipc4.

4.5.2 Performance Relative to the IPC4-Winner Satisficing Propositional Planner

The satisficing planners that won IPC4 are: for the propositional track, FAST DOWNWARD [40] (here indicated with DOWNWARD.ipc4); for the metric-time track, SGPLAN4 [17] (here indicated with SGPLAN.ipc4). *The analysis in this section shows that, overall, the winner of the IPC5 satisficing track improves on the performance of the satisficing planners that won IPC4, both in terms of CPU time and plan quality.*

The tables in Figure 30 summarise the results of an experimental comparison about the performance of DOWNWARD.ipc4 with the IPC5 satisficing propositional planners, for all the IPC5 propositional benchmarks. Concerning CPU time, we have that SGPLAN5 clearly outperforms DOWNWARD.ipc4 for most of the problems. In many cases SGPLAN5 is at least one order of magnitude faster, and in several cases it is at least two orders of magnitude faster. On the other hand, in most cases DOWNWARD.ipc4 is faster that the other IPC5 planners, with the exception of the IPC5 version of DOWNWARD04SA.

Concerning plan quality (measured in terms of number of actions in the plans generated for the problems solved by both the compared planners), more than one planner performs generally better than the IPC4 winner. As the table on the right side of Figure 30 shows, the number of test problems for which IPPLAN-G1SC, MIPS-XXL and SGPLAN5 compute better quality plans is higher than the number of problems for which they produce worse quality plans. On the other hand, there are more test problems for which DOWNWARD04SA and YOCHANPS generate worse quality solutions (w.r.t. the solutions of the IPC4 winner) than test problems for which they produce better solutions.

SGPLAN.ipc4 vs	2oS	1oS	S	F	1oF	2oF
MIPS-XXL	0	0	1	128	111	85
SGPLAN5	**48**	**70**	**132**	39	8	2
YOCHANPS	0	4	7	61	53	14

SGPLAN.ipc4 vs	2tW	W	B	2tB
MIPS-XXL	**2**	**50**	7	0
SGPLAN5	**19**	**113**	1	0
YOCHANPS	**36**	**50**	2	0

Figure 31: Numbers of IPC5 test problems for which the SGPLAN.ipc4 performs better/worse than the IPC5 satisficing metric-temporal planners. The table on the left concerns CPU time: the table on the right concerns plan quality. The meanings of the column labels are as in Figures 29 and 30.

The main results of the Wilcoxon test comparing the performance of DOWNWARD.ipc4 and the IPC5 satisficing planners for propositional domains are:

- In terms of CPU time, the only planners that perform statistically better than the IPC4 winner are SGPLAN5 and DOWNWARD04SA (the latter with confidence level 99.5%). In terms of plan quality, SGPLAN5 performs better than the IPC4 winner with confidence level 98.9%, while DOWNWARD04SA performs worse;
- In terms of plan quality, the only planner that statistically performs better than the IPC4 winner is IPPLAN-G1SC, which however, as we have seen in Table 2, solves a small percentage of the test problems.

4.5.3 Performance Relative to the IPC4-Winner Satisficing Metric-Temporal Planner

We now analyse the performance of the satisficing IPC5 planners supporting metric-temporal domains with respect to SGPLAN.ipc4, the best IPC4 metric-temporal planner, for all the metric-temporal IPC5 domains.

Concerning plan generation speed, as indicated by the results in the table on the left hand side of Figure 31, SGPLAN5 is generally faster than SGPLAN.ipc4, and for many problems it is at least two orders of magnitude faster. On the other hand, in most cases, the other IPC5 planners considered in this analysis are slower than SGPLAN.ipc4.

Concerning plan quality, interestingly, we observed that all the compared IPC5 planners perform generally better than SGPLAN.ipc4.

Finally, the results of the Wilcoxon test comparing the performance of MIPS-XXL, SGPLAN5 and YOCHANPS with the performance of SGPLAN.ipc4 confirm the observation derived from Figure 31: SGPLAN5 is the only IPC5 satisficing metric-temporal planner which is statistically faster than SGPLAN.ipc4, while in terms of plan quality every IPC5 satisficing metric-temporal planner performs statistically better than SGPLAN.ipc4.

4.5.4 Quality of the Solutions for PDDL2 problems

In order to evaluate how good a plan for a problem is w.r.t. the specified plan metric, we first need to know the plan metric value of an optimal plan for the problem. In this section, we compare the plans generated by the IPC5 satisficing planners with the plans generated by the optimal IPC5 planners. Obviously, since the satisficing planners solve many more problems than the optimal ones, in this analysis only a subset of the solved problems can be considered. To extend the collection of optimal plans (w.r.t. number of actions) for the propositional domains, we also used a domain-specific solver to obtain optimal solutions for the `Openstacks` propositional problems. Since for metric-time PDDL2 domains the number of known optimal solutions is very limited, for this category of IPC5 benchmarks we also compare the solutions generated by the IPC5 planners with solutions that approximate the optimal ones.

Figure 32 summarises the results of this analysis for the propositional IPC5 benchmarks for which we know optimal solutions (28 from `Openstacks`, 4 from `Pathways`, 9 from `Pipesworld`, 7 from `Rovers`, 14 from `Storage`, 6 from `Trucks` and 8 from `TPP`). The measure of plan quality here is plan length, i.e., the number of actions. Overall, the satisficing IPC5 planners tend to perform well (an exception is DOWNWARD04SA, which overall has the worst behaviour in terms of distance from the optimal solution): the quality of most of the solutions examined is at most 10% worse than the optimal

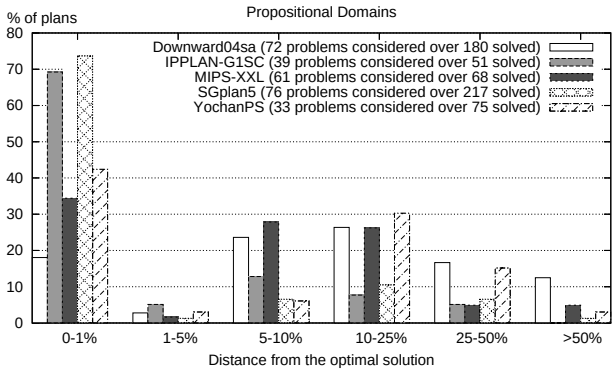

Figure 32: Plan quality distance of the solutions computed by DOWNWARD04SA, IPPLAN-G1SC, MIPS-XXL, SGPLAN5 and YOCHANPS from the optimal plan metric value for a subset of problems in every IPC5 propositional domain.

plan length, and there is a small percentage of the solutions with a quality that is 25% or more worse than the optimal plan length. Interestingly, most of the examined plans computed by IPPLAN-G1SC and SGPLAN5 are optimal or nearly optimal. On the other hand, we observe that only a small subset of the IPC5 benchmark problems are considered in this analysis and, moreover, most of them are small instances (those solved by the IPC5 optimal planners). The behaviour of the IPC5 satisficing planners may be different for larger instances.

A comparison of the solutions generated by CPT2, the only IPC5 optimal temporal planner, with those found by the metric-temporal IPC5 satisficing planners indicates that, contrary to the propositional case, for these problems the IPC5 satisficing planners often produce poor quality solutions.

Since CPT2 solves only 21 temporal problems, in order to have a more general analysis, Figure 33 shows the evaluation of the IPC5 plans in terms of lower bounds for their distance from the optimal solution. We computed these lower bounds by running the version of LPG described in [34] up to some CPU hours, and we compared the solutions of the IPC5 planners with the solutions computed by LPG.[17] For each IPC5 solution that is worse than the LPG solution, the distance between the qualities of the compared solutions provides a lower bound on the distance from the optimal solution. For each evaluated planner, the analysis does not consider the problems for which the planner computes a solution that is better than the one generated by LPG; these problems are a very small percentage of those solved by both the planners.

The analysis confirms that most of the generated plans are far from the optimal solutions: for at least 65% of the IPC5 metric-time benchmarks considered for this analysis, the solutions computed by SGPLAN5 and MIPS-XXL are at least 50% worse than the optimal solutions, with a distribution of their solutions over the lower bounds for the plan quality distance that tends to increase with the size of the bound.

The reason for the low plan qualities for SGPLAN5 is not completely clear, but we believe it is mainly because this planner optimises plan quality only under certain particular conditions which rarely occur in the considered test problems [44]. During search, SGPLAN5 optimises only the makespan when it runs best-first search, which is executed only when the main method based on enforced hill-climbing fails (the hill-climbing does not optimise plan quality). If the problem can be serially decomposed into some stages (called "subproblem level decomposition" [17]), SGPLAN5 tries different orders of these stages to get multiple feasible plans with different metric values. If SGPLAN5 employs neither subproblem-level decomposition nor best-first search, it never considers the plan metric during the search [44].

MIPS-XXL attempts to optimise the plan metric during search, however, evidently the implemented techniques are not very effective within 30 CPU minutes (the competition limit). Finally, although both

[17]The CPU-time limit for LPG was much higher than the one used in the competition, and this analysis is *not* intended to compare LPG with the IPC5 planners.

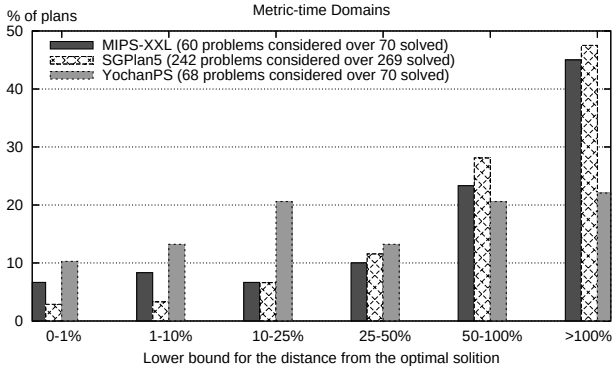

Figure 33: Percentage of the IPC5 solutions (satisficing planners) for a large subset of problems in every IPC5 metric-time domain with respect to increasing lower bounds for the plan quality distance from the optimal solution.

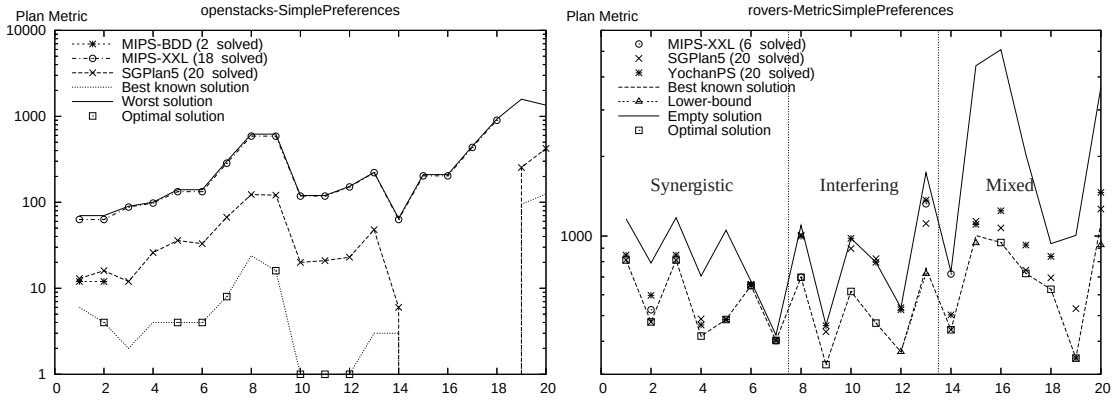

Figure 34: Plan quality evaluation for MIPS-BDD, MIPS-XXL, YOCHANPS and SGPLAN5 in `Openstacks` SimplePreferences and `Rovers` MetricSimplePreferences. On the x-axis there are the problem names simplified by numbers; on the y-axis there is the plan metric value in log scale (the lower the better).

these two planners schedule plan actions by a post-processing algorithm, these techniques do not derive significantly better plans. We conjecture this is because of two main reasons: the implemented post-processing step does not perform optimal action (re-)scheduling; the original plans do not allow good scheduling of the actions. In order to support the first conjecture, we ran a simple scheduling algorithm on the plans generated by MIPS-XXL and SGPLAN5 for the `Openstacks` Time benchmarks, obtaining considerably better plans.

Concerning YOCHANPS, this planner attempts to minimise the number of actions during search and performs a post-processing step for improving their scheduling. Somewhat surprisingly, this strategy allows YOCHANPS to perform slightly better than MIPS-XXL and SGPLAN5, with fewer plans having very poor quality and a more uniform distribution of the solutions over the plan quality distance bounds.

4.5.5 Quality of the Solutions for PDDL3 problems

In this section, we study the quality of the solutions computed by the satisficing IPC5 planners for problems involving preferences. The main observation that we can derive from the results of this analysis is that, while in many cases the IPC5 planners produce good quality solutions, there is also a large number of problems for which their solution is far from the optimal one. In the following, we analyse the

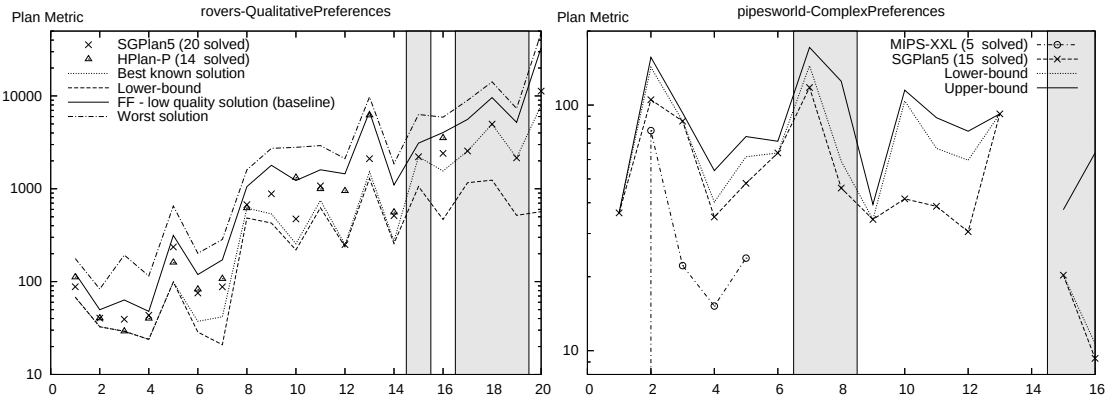

Figure 35: Plan quality evaluation for HPLAN-P, MIPS-XXL and SGPLAN5 in `Rovers` QualitativePreferences and `Pipesworld` ComplexPreferences. On the x-axis there are the problem names simplified by numbers. On the y-axis there is the plan metric value (log scale): for the plot on the left side, the lower the better; for the plot on the right side, the higher the better.

IPC5 solution plans with respect to the best known solutions or the optimal solutions, the worst plan metric values, and the lower/upper bound on the optimal solutions. For a minimisation (maximisation) problem, the lower bound is a plan metric value that is better (worse) than the optimal value, while the upper bound is a plan metric value that is worse (better) than the optimal value. (`Pipesworld` ComplexPreferences is the only IPC5 domain encoding a maximisation problem). The definition of the worst plan metric depends on the specific domain.

We derived the upper bound values using solutions that could be obtained "easily", meaning either as a side-effect of the construction of problem instances, or by a domain-specific polynomial procedure. Optimal solutions, best known solutions and lower bounds were obtained in a variety of ways, some by domain-specific methods and some by general planning techniques using a large amount of CPU time. For a more detailed description, see [38].

We consider two domains involving soft goals and two domains with preferences over state trajectory constraints. Figure 34 shows the evaluation of the plans computed by the IPC5 planners for `Openstacks` and `Rovers` with soft goals. In general, the plans for `Openstacks` SimplePreferences generated by SGPLAN5 and MIPS-XXL have low qualities. The solutions computed by MIPS-XXL are close to the worst plans and at least one order of magnitude worse than the optimal solutions; the plans computed by SGPLAN5 are better, but they are often still significantly distant from the optimal plans. (In this domain, each preference can be violated exactly once, so the worst plan quality is the sum of penalties over all preferences in the plan metric.)

The problems in `Rovers` MetricSimplePreferences form three groups (in Figure 34 denoted "synergistic", "interfering" and "mixed"), which differ in certain properties of the penalties associated with soft goals (see Section 3). In this domain, the worst possible solution quality is infinity. However, the empty plan is a valid plan for every instance of this domain, so we consider this as our baseline. Almost every plan computed by the IPC5 planners has quality better than the empty plan. The two planners that behave generally better are SGPLAN5 and YOCHANPS. For problems in the "synergistic" group, they generally compute good quality plans; for problems in the "interfering" group, the quality of their plans is very close to the quality of the empty plan; finally, for problems in the "mixed" group, in most cases SGPLAN5 and YOCHANPS compute plans that, in terms of plan quality, are closer to the optimal solution than to the empty plan.

Figure 35 shows the evaluation of the plans computed by the IPC5 planners for domains `Rovers` and `Pipesworld` with preferences over state trajectory constraints. Concerning `Rovers` QualitativePreferences, since problems have hard goals, as a baseline for analysing plan quality, we used the plans generated by FF [43] for solving the problems in this domain modified by omitting all preferences. The distances between the qualities of the best known solutions and the plan quality lower

Planner & Domain category	Probs	100%W	33%W	Worse	Better	33%B	100%B
HPLAN-P							
QualitativePreferences	70	0	0	1.42	75.0	37.0	21.0
MIPS-XXL							
SimplePreferences	49	0	2.0	2.0	79.0	26.0	18.0
QualitativePreferences	12	0	0	0	33.0	33.0	33.0
ComplexPreferences	22	4.0	9.0	59.1	36.0	31.0	18.0
MIPS-BDD							
SimplePreferences	29	0	0	0	82.0	48.0	44.0
QualitatativePreferences	16	0	0	0	68.0	50.0	37.0
SGPLAN5							
SimplePreferences	117	0	0	0.85	97.0	71.0	42.0
QualitativePreferences	85	0	0	1.17	98.0	95.0	77.0
ComplexPreferences	105	5.0	11.0	15.2	74.0	53.0	29.0
YOCHANPS							
SimplePreferences	54	1.0	3.0	13.0	79.0	44.0	29.0

Table 4: Percentages of problems for which the solutions computed by the IPC5 planners are worse/better than the solutions generated by these planners for the same problems without preferences in the plan-metric: at least two times worse (column "100%W"), at least 33% worse (column "33%W"), worse (column "W"), better (column "B"), at least 33% better (colum"33%B"), at least two times better (column "100%B"). Column "Probs" indicates, for each planner and domain category, the number of test problems considered for this analysis. The shaded value indicates the only case when the percentage of the better solutions was smaller than the percentage of the worse solutions.

bounds identify intervals containing the plan metric values of the (unknown) optimal solutions. Shaded areas indicate problems for which the qualities of the IPC5 plans are (a) close to (at most 30% worse than) the qualities of the corresponding best known solutions and (b) far from (at least 30% greater than) their lower bounds; in these cases, the results of this experiment are not very informative.

For most of the considered problems, the plan quality of FF is at least two times worse than the optimal plan quality. The worst plan quality is given by the sum of the preference weights in the plan metric. Interestingly, all the plans computed by SGPLAN5 and HPLAN-P are better than those computed by FF, although they are not very good plans: their qualities are often roughly in the middle between the quality of the plan computed by FF and the optimal plan quality.

The problems in `Pipesworld` ComplexPreferences require the satisfaction of the problem preferences be maximised, instead of their violation be minimised. The plans violating every problem preference are the worst solutions and they all have quality zero. The lower/upper plan quality bounds identify intervals containing the plan metric values of the (unknown) optimal solutions. Shaded areas indicate problems for which the IPC5 plan qualities are (a) far from (at least 30% lower than) the corresponding upper bounds and (b) are also close to (at most 30% lower than) their lower bounds; in these cases, the results of this experiment are not very informative. The planner with the best behavior is SGPLAN5, which generates good quality plans for the small problems. However, for medium-size problems it often computes plans with qualities roughly in the middle between the worst and the optimal ones.

4.5.6 Behaviour of the IPC5 Planners for the Benchmark Problems with/out Preferences

Since in a valid plan for a PDDL3 problem the preferences specified in the plan metric do not have necessarily to be satisfied, a planner that simply ignores them could accidentally produce a plan satisfying some or most of them, possibly obtaining a good-quality plan. In order to give a general experimental evaluation of the effectiveness of the methods implemented in the IPC5 planners to deal with preferences, we conducted the following experiment. We ran all IPC5 planners supporting PDDL3.0 preferences using the IPC5 benchmarks modified by *removing* the preferences, and we compared the quality of the plans for the modified problems with the plans for the corresponding original problems containing preferences. In case a modified test problem contains no classical goals, for every tested

planner, we used the empty plan. The results of this experimental analysis are given in Table 4. If we consider only the problems with hard goals similar results can be obtained.

In most cases, the techniques for dealing with the preferences implemented in the tested planners allow the planners to derive plans with better qualities. Remarkably, SGPLAN5 achieves the highest improvements (relative to its own solutions generated without considering preferences), with the best results for the problems involving soft qualitative state-trajectory constraints (called qualitative preferences): 77% of the solutions are at least 2 times better than the solutions generated for the problems with the preferences omitted. On the other hand, for every tested planner except MIPS-BDD, there are some problems for which ignoring the preferences leads to better quality plans.

5 Conclusions

Planning has been tackling increasingly difficult problems with greater success over recent years. An objective for the community is to move the focus of research towards the solution of problems with increasing relevance to application. In many application areas, the quality of plans is central to their usefulness. In IPC5, differently from the previous IPCs, plan quality was important, both in the planning language and in the evaluation of the competing planners.

In this paper we have presented a new version of PDDL, PDDL3, that was designed for the deterministic part of IPC5. PDDL3 includes new features that allow the user to specify plan quality in terms of constraints across the trajectories and in terms of preferences over such constraints as well as over goals. Although the concepts of constraints, both hard and soft, are not new, even to planning, the adoption of a common language and the basis for benchmarks plays a central role in promoting research into these areas. In order to make the new language more accessible to the IPC5 participants, a restricted version of PDDL3, PDDL3.0, was used for the competition. Several new planners supporting some of or most of the new features of PDDL3.0 entered the competition. Some methods for compiling state trajectory constraints and preferences have recently been developed (in particular by some competing teams of IPC5, e.g.,[25]), but these schemes were not designed with the purpose of studying the language theoretical expressiveness. Although a detailed study of the expressiveness of PDDL3 is outside the goals of this paper, we have given some new basic results about the compilability of PDDL3.0 state trajectory constraints and preferences.

PDDL3.0 as well as PDDL3 could be further extended in many ways. An interesting possibility would be to use an alternative way to define the importance of preferences that is more based on qualitative priorities rather than numerical weights, as outlined in [31]. However, the current version of PDDL is already a powerful language. As demonstrated by the results of IPC5 and previous competitions, current planners are not yet capable of dealing with many features of PDDL in a fully satisfactory.

Another contribution of our work is the development of a large collection of new benchmark domains and problems, specified with PDDL3.0 and PDDL2, which we have presented in this paper. The new benchmark domains were derived from a variety of sources: some are inspired by (potential) applications of planning technology; some are encodings of benchmark problems used in other areas of computer science and operations research; and some were created for the explicit purpose of trying out the new language features offered by PDDL3. In line with the aim to emphasise plan quality in the evaluation of competing planners, many IPC5 domains encode optimisation problems, in which it is significantly easier to find a plan that only satisfies the hard goals and constraints (if any) of a problem instance, and the true difficulty lies in finding a plan that also has high quality. For the same reason, the problem instances were designed very carefully so that, for example, they admit many solutions with significantly different qualities or require the planner to find a good compromise among the different (possibly conflicting) terms in the objective function. Although most of the domains and problems developed for the previous two IPCs [41, 50] are equipped with a plan metric function, only a few of them had an emphasis on optimisation, or the emphasis was split between time to plan and quality of plan in a way that left it unclear what aspect was intended to matter more.

Finally, we have presented the results of a large experimental investigation that includes a detailed analysis of the data from the deterministic part of IPC5, as well as additional experiments that we

conducted to better understand the effectiveness of the twelve compared planners. The main conclusions we can draw from this investigation are:

- The detailed analysis confirms that SATPLAN and MAXPLAN are the best (in terms of ability to solve problems quickly) propositional optimal planners of those participating in IPC5, which is consistent with the preliminary informal evaluation of the planners conducted during the competition. However, it also shows that SATPLAN is generally faster than MAXPLAN. Likewise, our analysis confirms that, overall, SGPLAN5 is the best satisficing IPC5 planner;
- The 2006 version of SATPLAN, CPT2 and SGPLAN5 each offers a significant improvement over the performance of the winner of the corresponding track of the previous competition. In this sense, we can say that they advance the state of the art in fully automated planning systems;
- An analysis of the quality of the plans generated by the satisficing IPC5 planners for a subset of the benchmark problems shows that: for propositional problems, they tend to find good solutions (as measured by the number of actions), while for metric-temporal problems and problems with preferences, the quality of the solutions they find is generally far from the best known to be achievable;
- An analysis of the behaviour of the IPC5 planners supporting PDDL3.0 preferences also shows that the techniques they use to deal with preferences are useful, in the sense that, for the most part, they find plans of quality better than what would be expected from blind luck, i.e. from completely disregarding preferences when solving these problems.

Overall, while we observed a clear advancement of the state-of-the-art in optimal propositional planning as well as in satisficing planning (in terms of CPU time, plan quality, and support for features of the language), finding high quality plans in metric-temporal domains and in domains with preferences remains an important open issue deserving further research effort. Moreover, most of the benchmark problems with hard state trajectory constraints are still unsolved, suggesting that there is considerable need for improved techniques for dealing with them.

Acknowledgements

We would like to thank the anonymous reviewers for many useful comments. The organisers of IPC5, Yannis Dimopoulos, Alfonso E. Gerevini, Patrik Haslum and Alessandro Saetti, would like to thank all participants of IPC5 and the consulting committee of the deterministic track of IPC5. Alfonso E. Gerevini and Derek Long would also like to thank Carmel Domshlak, Stefan Edelkamp, Maria Fox, Joerg Hoffmann, Ari K. Jonsson, Drew McDermott, Len Schubert, Ivan Serina, David E. Smith and Daniel S. Weld for some very useful discussions about PDDL3. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

[1] F. Bacchus. The AIPS '00 planning competition. *AI Magazine*, 22(3):47–56, 2001.

[2] F. Bacchus and F. Kabanza. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence*, 22(1-2):5–27, 1998.

[3] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116:123–191, 2000.

[4] C. Bäckström. Expressive equivalence of planning formalisms. *Artificial Intelligence*, 76:17–34, 1995.

[5] J. Baier and S. McIlraith. Planning with first-order temporally extended goals using heuristic search. In *Proc. of 21st National Conf. on Artificial Intelligence (AAAI'06)*, 2006.

[6] J. Baier and S. McIlraith. Planning with temporally extended goals using heuristic search. In *Proc. of 16th Int. Conf. on Automated Planning and Scheduling (ICAPS'06)*, 2006.

[7] J. Benton and S. Kambhampati. YochanPS: PDDL3 simple preferences as partial satisfaction planning. In *5th Int. Planning Competition Booklet*, 2006.

[8] J. Benton, M. van den Briel, and S. Kambhampati. A hybrid linear programming and relaxed plan heuristic for partial satisfaction planning problems. In *Proc. of 17th Int. Conf. on Automated Planning and Scheduling (ICAPS'07)*, 2007.

[9] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint solving and optimization. *Journal of ACM*, 1997.

[10] A. Blum and M.L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

[11] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[12] B. Bonet and H. Geffner. Heuristics for planning with penalties and rewards using compiled knowledge. In *Proc. of 10th Int. Conf. on Knowledge Representation (KR'06)*, 2006.

[13] B. Bonet, A.E. Gerevini, and R. Givan. Abstract booklet of the Fifth Int. Planning Competition. `http://ipc5.ing.unibs.it`, 2006.

[14] R. Brafman and Y. Chernyavsky. Planning with goal preferences and constraints. In *Proc. of 15th Int. Conf. on Automated Planning and Scheduling (ICAPS'05)*, 2005.

[15] M. Briel, R. Sanchez, M. Do, and S. Kambhampati. Effective approaches for partial satisfaction (over-subscription) planning. In *Proc. of 19th National Conf. on Artificial Intelligence (AAAI'04)*, 2004.

[16] N. Chabrier, 2003. `http://contraintes.inria.fr/BIOCHAM/EXAMPLES/~cell_cycle/cell_cycle.bc`.

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

[18] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[19] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[20] S. Cresswell and A. Coddington. Compilation of LTL goal formulas into PDDL. In *Proc. of 15th Eureopean Conf. on Artificial Intelligence, (ECAI'04)*, 2004.

[21] P. J. Delgrande, T. Schaub, and H. Tompits. A general framework for expressing preferences in causal reasoning and planning. In *Proc. of 7th Int. Symposium on Logical Formalizations of Commonsense Reasoning*, 2005.

[22] M. Do, J. Benton, M. van den Briel, and S. Kambhampati. Planning with goal utility dependencies. In *Proc. of 20th Int. Conf. on Artificial Intelligence (IJCAI'07)*, 2007.

[23] M.B. Do and S. Kambhampati. Partial satisfaction (over-subscription) planning as heuristic search. In *Proc. of 5th Int. Conf. on Knowledge Based Computer Systems (KBCS'04)*, 2004.

[24] D. Dubois, H. Fargier, and H. Prade. Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty. *Applied Intelligence*, 6:287–309, 1996.

[25] S. Edelkamp. On the compilation of plan constraints and preferences. In *Proc. of 16th Int. Conf. on Automated Planning and Scheduling (ICAPS'06)*, 2006.

[26] S. Edelkamp and J. Hoffmann. PDDL2.2: The language for the classic part of the 4th International Planning Competition. Technical Report 195, Institut für Informatik, Freiburg, Germany, 2004.

[27] A. Fink and S. Voss. Applications of modern heuristic search methods to pattern sequencing problems. *Computers & Operations Research*, 26:17–34, 1999.

[28] M. Fox, Long D., and Halsey K. An investigation into the expressive power of PDDL2.1. In *Proc. of 16th European Conf. on Artificial Intelligence (ECAI-04)*, 2004.

[29] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[30] H. Geffner, P. Haslum, M. Helmert, J. Hoffmann, V. Vidal, B. Bonet, and C. Domshlak. *Proceedings of the ICAPS-07 Workshop on Heuristics for Domain-independent Planning: Progress, Ideas, Limitations, Challenges*. 17th Int. Conf. on Automated Planning and Scheduling (ICAPS'07), 2007.

[31] A. Gerevini and D. Long. Plan constraints and preferences in PDDL3. Technical Report RT-2005-08-47, Dipartimento di Elettronica per l'Automazione, Universitá di Brescia, 2005.

[32] A. Gerevini and D. Long. Preferences and soft constraints in PDDL3. In *Proc. of ICAPS-2006 Workshop on Preferences and Soft Constraints in Planning*, 2006.

[33] A. Gerevini, A. Saetti, P. Haslum, D. Long, and Y. Dimopoulos. Deterministic planning in the fifth planning competition: PDDL3 and experimental evaluation of the planners. Technical Report RT-2008-02-59, Dipartimento di Elettronica per l'Automazione, Universitá di Brescia, 2008.

[34] A. Gerevini, A. Saetti, and I. Serina. An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artificial Intelligence*, 172(8-9):899–944, 2008.

[35] R. Gerth, D. Peled, M. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proc. of 15th Workshop on Protocol Specification, Testing and Verification*, 1995.

[36] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. Technical Report CVC TR98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[37] E. Giunchiglia and M. Maratea. Planning as satisfiability with preferences. In *Proc. of 22nd Conf. of Artificial Intelligence (AAAI'07)*, 2007.

[38] P. Haslum. Quality of solutions to IPC5 benchmark problems: Preliminary results. In *Proc. of ICAPS-07 Workshop on Int. Planning Competition: Past, Present & Future*, 2007.

[39] P. Haslum and P. Jonsson. Some results on the complexity of planning with incomplete information. In *Proc. of 5th European Conf. on Planning (ECP'99)*, 1999.

[40] M. Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[41] J. Hoffmann and S. Edelkamp. The deterministic part of IPC-4: An overview. *Journal of Artificial Intelligence Research*, 24:519–579, 2005.

[42] J. Hoffmann, S. Edelkamp, S. Thiébaux, R. Englert, F. Liporace, and S. Trüg. Engineering benchmarks for planning: the domains used in the deterministic part of IPC-4. *Journal of Artificial Intelligence Research*, 26:453–541, 2006.

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

[44] C. Hsu, B.W. Wah, and Y. Chen. Personal communication. Oct 2007.

[45] F. Kabanza and S. Thiébaux. Search control in planning for temporally extended goals. In *Proc. of 15th Int. Conf. on Automated Planning and Scheduling (ICAPS'05)*, 2005.

[46] H. Kautz. SATPLAN04: Planning as satisfiability. In *4th Int. Planning Competition Booklet*, 2004.

[47] K. Kohn. Molecular interaction map of the mammalian cell cycle control and DNA repair systems. *Molecular Biology of the Cell*, 10(8), 1999.

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

[49] A. Linhares and H.H. Yanasse. Connection between cutting-pattern sequencing, VLSI design and flexible machines. *Computers & Operations Research*, 29:1759–1772, 2002.

[50] D. Long and M. Fox. The 3rd International Planning Competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20:1–59, 2003.

[51] D. Long, H. Kautz, B. Selman, B. Bonet, H. Geffner, J. Koehler, M. Brenner, J. Hoffmann, F. Rittinger, C. Anderson, D. Weld, D. Smith, and M. Fox. The AIPS-98 planning competition. *AI Magazine*, 21(2):13–33, 2000.

[52] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer, 1992.

[53] I. Miguel, P. Jarvis, and Q. Shen. Efficient flexible planning via dynamic flexible constraint satisfaction. *Engineering Applications of Artificial Intelligence*, 14(3):301–327, 2001.

[54] B. Nebel. On the compilability and the expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research*, 12:271–315, 2000.

[55] S. Penberthy, J. *Planning with Continuous Change*. PhD thesis, University of Washington, 1993. Available as technical report UW-CSE-93-12-01.

[56] A. Pnueli. The temporal logic of programs. In *Proc. of 18th IEEE Symposium on Foundations of Computer Science*, 1977.

[57] J. Riera-Ledesma and J. Salazar-Gonzalez, J. A heuristic approach for the travelling purchaser problem. *European Journal of Operational Research*, 160(3):599–613, 2005.

[58] J. Rintanen. Incorporation of temporal logic control into plan operators. In *Proc. of 14th European Conf. on Artificial Intelligence (ECAI'00)*, 2000.

[59] J. Rintanen. Complexity of concurrent temporal planning. In *Proc. of 17th Int. Conf. on Automated Planning and Scheduling*, 2007.

[60] F. Rossi, K.B. Venable, and N. Yorke-Smith. Controllability of soft temporal constraint problems. In *Proc. of 10th Int. Conf. on Principles and Practice of Constraint Programming (CP'04)*, 2004.

[61] B.M. Smith and I.P. Gent. Constraint modelling challenge 2005. `http://www.dcs.st-and.ac.uk/~ipg/challenge/`, 2005.

[62] D. Smith. Choosing objectives in over-subscription planning. In *Proc. of 14th Int. Conf. on Automated Planning and Scheduling (ICAPS'04)*, 2004.

[63] C. Son, T. and E. Pontelli. Planning with preferences using logic programming. In *Proc. of 7th Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, 2004.

[64] P. Thagard. Pathways to biomedical discovery. *Philosophy of Science*, 70, 2003.

[65] S. Thiébaux, J. Hoffmann, and B. Nebel. In defense of PDDL axioms. *Artificial Intelligence*, 168:38–69, 2005.

[66] V. Vidal and H. Geffner. Branching and pruning: An optimal temporal POCL planner based on constraint programming. *Artificial Intelligence*, 170(3):298–335, 2006.

[67] F. Wilcoxon and R. A. Wilcox. *Some Rapid Approximate Statistical Procedures*. Lederle Laboratories, Pearl River, New York, USA, 1964.