New Complexity Results for Classical Planning Benchmarks

Malte Helmert
Institut für Informatik, Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee, Gebäude 052
79110 Freiburg, Germany
helmert@informatik.uni-freiburg.de

Abstract

The 3rd and 4th International Planning Competitions have enriched the set of benchmarks for classical propositional planning by a number of novel and interesting planning domains. Although they are commonly used for the purpose of evaluating planner performance, the planning domains themselves have not yet been studied in depth. In this contribution, we prove complexity results for the decision problems related to finding *some plan*, finding an *optimal sequential plan*, and finding an *optimal parallel plan* in these planning domains. Our results also provide some insights into the question *why* planning is hard for some of the domains under consideration.

Introduction

It is hard to deny the fact that the International Planning Competitions (IPCs), starting from their first incarnation at AIPS 1998 (McDermott 2000), have had a marked impact on classical planning research in recent years. This is true not just of the competitions themselves, but also of the planning domains used within them.

Evidence for this is manifold. First, significant work has been published that discuss properties of, or algorithms for, some of the competition domains (Slaney & Thiébaux 2001; Thiébaux & Cordier 2001; Helmert 2003). (Some domains, such as PSR, have only been used in the competitions *after* being the topic of research papers, but this only reinforces our point that they are interesting in their own right.)

Second, we observe that many developments in classical planning are motivated by weaknesses of earlier planning approaches in benchmark domains. For example, the Goal Agenda Management technique used in IPP and FF (Koehler *et al.* 1997; Hoffmann & Nebel 2001) is motivated by the BLOCKSWORLD domain and the CG heuristic used by Fast Downward (Helmert 2004) is inspired by "transportation domains" such as LOGISTICS and MYSTERY.

Finally, it has become a rare occurrence that a paper discussing new techniques for classical planning is published without presenting performance results for some of the competition domains. Indeed, in the ICAPS 2005 proceedings (Biundo, Myers, & Rajan 2005), there are eight papers on domain-independent deterministic planning, of which seven

IPC3	DEPOT	**IPC4**	AIRPORT
	DRIVERLOG		PIPESWORLD
	FREECELL		PROMELA
	ROVERS		PSR
	SATELLITE		SATELLITE
	ZENOTRAVEL		

Figure 1: IPC3 and IPC4 domains.

exclusively use IPC domains for evaluation purposes, while the remaining one uses two competition domains and the towers of Hanoi.

Whether or not this focus on the competition domains constitutes a healthy trend is a matter of some debate. It can be argued that current practice in classical planning research focuses too much on raw benchmark performance, and too little on original ideas. This has led to a certain uniformity of planning approaches, with three of the five best-performing planners in the non-optimizing propositional track of IPC4 (Hoffmann & Edelkamp 2005) being some variation of Hoffmann's FF. In any case, it is evident that the competition domains have become such a staple in current planning research that it is worthwhile putting some effort into understanding them well.

Working towards this goal, this paper analyzes the computational complexity of planning in the propositional planning domains of IPC3 (Long & Fox 2003) and IPC4 (Hoffmann & Edelkamp 2005), shown in Fig. 1. We focus on IPC3 and IPC4 because the domains of the first two competitions have already been addressed (Helmert 2003). We focus on propositional (i. e., non-numerical, non-temporal) domains because they are more commonly used than their non-propositional counterparts; indeed, no ICAPS 2005 paper includes experiments with the latter. However, this is not to say that the non-propositional domains are without interest, and indeed a follow-up investigation of those is underway.

Studies of decision complexity are certainly not the be-all and end-all of research on planning domains. We rather see them as a first step, where other steps should involve approximability properties, phase transition behavior, and (as a reference for comparison) domain-dependent optimal planning algorithms. All these steps have been undertaken for the BLOCKSWORLD domain (Slaney & Thiébaux 2001). To allow the other benchmark domains to catch up a little bit, a companion paper discusses the approximability of the competition benchmarks (Helmert, Mattmüller, & Röger 2006).

We proceed as follows. In the next section, we provide an overview of the problems addressed in this paper, and provide some first results. This is followed by four sections investigating certain planning domains in depth, namely AIRPORT, PIPESWORLD, PROMELA, and PSR. Finally, we discuss our findings and conclude.

The problems

The propositional planning domains of IPC3 and IPC4 are shown in Fig. 1. For each of these domains, we are interested in the complexity of the following decision problems:

- *Plan existence*: Is a given planning task solvable?
- *Bounded plan existence*: Is a given planning task solvable using no more than a certain number of actions?

Plan existence is closely related to the problem of *generating a plan* for the task, while bounded plan existence is related to *generating an optimal plan* for a task, i. e., a plan consisting of a minimal number of actions. If it is hard to decide the decision problem, then the planning problem must also be hard. While the converse does not hold universally, it does hold for the domains we analyze: All our proofs of easiness for the decision problems are constructive, providing a planning algorithm for the domain under consideration.

For many planning domains, plan existence can be decided in polynomial time whereas bounded plan existence is **NP**-complete. In these cases, it is natural to ask how difficult it is to generate *approximate solutions*, plans which are not much worse than optimal, for some definition of "not much worse." We answer this question for the competition domains in a companion paper (Helmert, Mattmüller, & Röger 2006), which includes classifications of approximability for the DEPOT, DRIVERLOG, ROVERS, SATELLITE and ZENOTRAVEL domains. Because that paper contains an in-depth study of these domains, we only discuss them very briefly here.

In all five domains, non-optimal plans can be easily generated in polynomial time by addressing one subgoal at a time. NP-hardness of bounded plan existence for DEPOT, DRIVERLOG and ZENOTRAVEL follows because each of these domains generalizes the MICONIC-STRIPS domain, for which bounded plan existence is **NP**-hard (Helmert 2003). For ROVERS and SATELLITE, we can reduce from an **NP**-hard set covering problem. The mappings are shown in the companion paper (Helmert, Mattmüller, & Röger 2006), where they are used to show limits of approximability. Membership in **NP** follows because shortest plans are only polynomially long and hence guess-and-check algorithms are applicable.

This leaves us with five domains to investigate. One of these, FREECELL, was already part of IPC2 and has been shown to have **NP**-complete plan existence and bounded plan existence problems (Helmert 2003). We dedicate the following sections to the remaining four domains, starting with AIRPORT.

AIRPORT

The AIRPORT domain models ground traffic on an airport, i. e., movement of aircraft along taxiways and runways. Unlike other route planning domains, AIRPORT tasks are heavily space-constrained: Not only can any given location (called a *taxiway segment*) only be occupied by one aircraft at a time, there even exist mutual exclusion constraints between segments to the effect that at most one of them may be occupied at a given time. The purpose of these constraints is to model realistic safety conditions. Indeed, the AIRPORT domain is firmly grounded in real-world planning tasks (Hatzack & Nebel 2001), and some of the IPC4 benchmarks are faithful translations of realistic data from Munich Airport. (Note that the formulation in Hatzack and Nebel's paper is quite different from the IPC4 version, leading to different complexity results.)

Planning domain 1 AIRPORT
*The set of **movement modes** for aircraft is defined as* $M := \{\mathit{pushing}, \mathit{taxiing}, \mathit{airborne}, \mathit{parked}\}$.

An AIRPORT task is given by the following components:

- *finite sets of **aircraft** A and **taxiway segments** S,*
- *a **taxiway relation**, **pushback relation** and **blocking relation** $R_{\mathrm{T}}, R_{\mathrm{P}}, R_{\mathrm{B}} \subseteq S \times S$,*
- *an **initial mode function** $m_0 : A \to \{\mathit{pushing}, \mathit{taxiing}\}$ and **initial segment function** $s_0 : A \to S$, and*
- *a **goal mode function** $m_\star : A \to \{\mathit{airborne}, \mathit{parked}\}$ and **goal segment function** $s_\star : A \to S$.*

*The digraphs $G_{\mathrm{T}} = (S, R_{\mathrm{T}})$, $G_{\mathrm{P}} = (S, R_{\mathrm{P}})$ and $G_{\mathrm{B}} = (S, R_{\mathrm{B}})$ are called the **taxiway graph**, **pushback graph** and **blocking graph** of the task.*

*The task is called **undirected** iff R_{T}, R_{P} and R_{B} are symmetric, **planar** iff $(S, R_{\mathrm{T}} \cup R_{\mathrm{P}})$ is a planar digraph, and **regularly constrained** iff $R_{\mathrm{T}} = R_{\mathrm{P}} = R_{\mathrm{B}}$.*

*States of the task are pairs $(m, s) \in (A \to M) \times (A \to S)$, where $m(a)$ is called the **current mode** and $s(a)$ is called the **current segment** of aircraft $a \in A$. The initial state is (m_0, s_0), the only goal state is the state $(m_\star, s_\star)$.*

*Any state (including initial and goal state) must satisfy the following **blocking constraints**:*

- *If two aircraft share the same current segment, at least one must be airborne.*
- *If an aircraft located at segment u is pushing or taxiing and another aircraft located at segment v is pushing, taxiing or parked, then $(u, v) \notin R_{\mathrm{B}}$.*

There are five kinds of actions:

- *An aircraft can change its current segment from u to v if it is taxiing and $(u, v) \in R_{\mathrm{T}}$ (**move** actions), or if it is pushing and $(u, v) \in R_{\mathrm{P}}$ (**push** actions), unless the resulting state violates the blocking constraints.*
- *An aircraft can change its current mode from pushing to taxiing (**start up** actions), from taxiing to airborne (**take off** actions), and from taxiing to parked (**park** actions).*

The AIRPORT tasks of IPC4 obey two further restrictions not captured by our definition. First, there are no aircraft whose initial mode is *pushing* and goal mode is "parked".

This would make little sense as *pushing* mode is associated with outbound aircraft only and *parked* mode is associated with inbound aircraft only. Second, the pushback graph is always a subgraph of the taxiway graph with all arcs reversed. Neither restriction has an impact on the complexity of the problem because our hardness results already hold if there are no pushing aircraft at all.

We also made some deviations from the PDDL definition of the domain (Hoffmann & Edelkamp 2005) to simplify presentation. Most importantly, the PDDL definition distinguishes between the *location* and *facing* of an aircraft, while we only consider its current *segment*. Compilations between these two representations are straightforward.

Other differences involve the modeling of take-off actions. In the PDDL definition, aircraft "leave the map" when taking off, and they can only take off from specific runway segments. We ignore airborne aircraft for blocking purposes, which amounts to the same thing as having them leave the map, and while we allow take-off everywhere, it never makes sense to take off from a non-goal segment, and goal segments of outbound aircraft are always runway segments. The PDDL domain also contains a minor modeling flaw that allows airplanes to park immediately before take-off. However, this is never a useful thing to do and hence cannot affect complexity.

Finally, the PDDL domain allows the blocking relation to depend on the aircraft (via *airplane types*), but none of the existing benchmarks makes use of this feature. Again, modeling it would not make a difference in complexity, as even without airplane types, we can prove **PSPACE**-hardness.

Theorem 2 AIRPORT *planning is* PSPACE*-complete.*
Plan existence and bounded plan existence for AIRPORT *tasks are* **PSPACE***-complete. This is true even if we only allow undirected, planar, regularly constrained tasks where all aircraft are taxiing initially and must be parked in the goal.*

Proof: For a graph $G = (V, E)$ and a set of *tokens* T, we define a *legal placement* of T on G as an injective function $\pi : T \to V$ such that no two tokens are placed on adjacent vertices. A legal placement π' is a *successor* of another legal placement π iff they differ on exactly one token $t \in T$, for which we have $\{\pi(t), \pi'(t)\} \in E$. In other words, to obtain a successor of a legal placement, move a single token along an edge and verify that this results in another legal placement.

We show **PSPACE**-hardness of AIRPORT plan existence by polynomially reducing from the following (**PSPACE**-complete) variation of the Sliding Tokens puzzle (Hearn & Demaine 2005): Given a planar graph G, set of tokens T and legal placements π_0, $\pi_\star$ of T on G, is there a sequence of legal placements $\pi_1, \dots, \pi_M$ such that π_i is a successor of π_{i-1} for all $i \in \{1, \dots, M\}$ and $\pi_M = \pi_\star$?[1]

We now describe the mapping of puzzle instances to AIRPORT tasks. Given graph $G = (V, E)$, tokens T and placements π_0 and $\pi_\star$, we generate an AIRPORT task with segment set V, aircraft set T, taxiway graph G, pushback graph G and blocking graph G, initial segment function π_0 and goal segment function $\pi_\star$. All aircraft are taxiing in the initial state and must be parked in the goal state. Clearly, the mapping can be computed in polynomial time.

No solution to the planning task can ever contain *push*, *start up* or *take off* actions, so we only need to consider *move* and *park* actions. If the planning task has a solution, then the sequence of *move* actions in such a solution defines a solution to the puzzle instance. Note that if a taxiing aircraft ever moved to a segment which is adjacent to the current segment of another (taxiing or parked) aircraft, this would violate the blocking constraints. Similarly, from a solution to the puzzle we can obtain a sequence of actions that move each aircraft to its goal location without violating the blocking constraints, and from that state the task is solved by parking all aircraft. Therefore, the mapping is indeed a reduction.

Thus, plan existence for restricted AIRPORT tasks is **PSPACE**-hard, which implies that bounded plan existence is also **PSPACE**-hard. Moreover, both problems must belong to **PSPACE** because PDDL planning in any fixed propositional domain is in **PSPACE**. This concludes the proof. ∎

Clearly, the result equally applies to the parallel planning framework, as all **PSPACE**-completeness results for PDDL domains do. The result also implies that there exist AIRPORT tasks for which the shortest plan consists of exponentially many actions (or sets of parallel actions). For example, the shortest solution to the puzzle corresponding to a QBF formula with n quantifier alternations consists of $\Omega(2^n)$ many steps (Hearn & Demaine 2005), leading to an AIRPORT task with a similarly bounded sequential solution length. The optimal parallel solution can only be shorter by a linear amount, because only $O(n)$ many actions can be executed in parallel (one per aircraft). This also implies that any AIRPORT planning algorithm requires exponential time for writing down the solution for such instances.

We have also proved another polynomial reduction (not included in this paper) from the halting problem for polynomially space-constrained Turing Machines. This reduction only generates *deterministic* AIRPORT tasks, where at most one action is applicable at any time. Therefore, AIRPORT planning is **PSPACE**-complete even if no branching is involved. However, the tasks generated by this reduction are not undirected, planar, or regularly constrained.

PIPESWORLD

The PIPESWORLD domain models the flow of oil-derivative liquids through *pipeline segments* connecting *areas*, and is inspired by applications in the oil industry (Milidiú, dos Santos Liporace, & de Lucena 2003). Liquids are modeled as *batches* of a certain unit size. A segment must always contain a certain number of batches (i. e., it must always be full). Batches can be pushed into pipelines from either side, leading to the batch at the opposite end "falling" into the incident area. Batches have associated *product types*, and batches of certain types may never be adjacent to each other in a pipeline. Moreover, areas may have constraints on how

[1]In the original Sliding Tokens puzzle, tokens are indistinguishable and the goal has a different form. Only simple adjustments to the hardness proofs are needed for the modified version; cf. Theorem 23 and Corollary 6 in the reference (Hearn & Demaine 2005).

many batches of a certain product type they can hold.

Planning domain 3 PIPESWORLD
*$P := \{lco, gasoline, rat\text{-}a, oca1, oc1b\}$ is the set of **products**. Two products $p, p' \in P$ are called **compatible** unless $p = rat\text{-}a$ and $p' \in \{oca1, oc1b\}$ or vice versa.*

A PIPESWORLD task is given by:

- *finite sets of **areas** A and **pipeline segments** S,*
- *a finite set of **batches** B, each with a **product type** $b^P \in P$,*
- *for each pipelines segment $s \in S$, a **start area** $s^- \in A$ and **end area** $s^+ \in A$ and a **segment length** $|s| \in \mathbb{N}_1$,*
- *an **area capacity function** $c : A \times P \to \mathbb{N}_0$,*
- *a **goal contents function** $C_G : A \to 2^B$ such that for each batch $b \in B$, we have $b \in C_G(a)$ for at most one area $a \in A$, and*
- *an (arbitrary) initial state,*

*where a state of the task is defined by an **area contents** function $C_A : A \to 2^B$ and a **pipeline segment contents** function $C_S : S \to B^*$ such that*

- *for each batch $b \in B$, either $b \in C_A(a)$ for exactly one area $a \in A$, or $b \in C_S(s)$ for exactly one segment $s \in S$,*
- *for all areas $a \in A$ and products $p \in P$, $C_A(a)$ contains at most $c(a, p)$ batches of product type p, and*
- *for all pipeline segments $s \in S$, $|C_S(s)| = |s|$ and any two adjacent batches in $C_S(s)$ can interface, i. e., have compatible product types.*

A state is a goal state iff $C_G(a) \subseteq C_A(a)$ for all $a \in A$.

*The only actions in the task are **push** actions. If $s \in S$ is a pipeline segment with contents $b_1 \dots b_{|s|}$ and $b \in C_A(s^-)$ is a batch that can interface with b_1, then b can be pushed into s. This results in a state where the new contents of segment s are $bb_1 \dots b_{|s|-1}$, b is no longer in $C_A(s^-)$, and $b_{|s|}$ is in $C_A(s^+)$. Similarly, $b \in C_A(s^+)$ can be pushed into s if it can interface with $b_{|s|}$, leading to a state where the contents of s are $b_2 \dots b_n b$, b is no longer in $C_A(s^+)$, and b_1 is in $C_A(s^-)$. Pushing a batch into a pipeline segment is not allowed if the resulting state would violate the area capacity constraints.*

General PIPESWORLD tasks are sometimes referred to as PIPESWORLD-TANKAGE tasks. Note that with our definition (as in the IPC4 benchmarks), the set of products and their compatibility relation is fixed. We will see that we can prove hardness already for this fixed compatibility relation.

Planning domain 4 PIPESWORLD-NOTANKAGE
A PIPESWORLD-NOTANKAGE task is a PIPESWORLD task where the area capacity for each area and product type is equal to the total number of batches of that product type.

Our definition of PIPESWORLD faithfully captures the PDDL specification except for one modeling flaw of the latter: In some situations, the PDDL definition allows pushing batches through a pipe even though this violates the area capacity constraints on the receiving end of the pipe, making some unsolvable tasks solvable. This minor difference does not affect the applicability of our results because these already hold for the PIPESWORLD-NOTANKAGE domain, where area capacities can be ignored.

Theorem 5 PIPESWORLD *is* NP*-hard*
Plan existence and bounded plan existence in the domains PIPESWORLD-TANKAGE and PIPESWORLD-NOTANKAGE are **NP***-hard problems.*

Proof: We prove that PIPESWORLD-NOTANKAGE has an **NP**-hard plan existence problem, so that the other results follow. The reduction is from satisfiability of propositional CNF formulae where clauses contain at most four literals and each variable occurs in at most three clauses (and at most once per clause). This problem is known to be **NP**-hard (Garey & Johnson 1979, LO1). (We could limit clauses to three literals, but then Fig. 3 would look less symmetric.)

Let χ be the formula, and let V and C be its variable and clause sets. Throughout the proof, we refer to batches of type *rat-a* as *white* batches, batches of type *oca1* as *black* batches, and batches of type *gasoline* as *gray* batches. Observe that white batches may not interface with black batches, while gray batches may interface with anything.

The generated PIPESWORLD-NOTANKAGE instance is assembled from components shown in Figs. 2 and 3, where edges with differently decorated endpoints distinguish between different kinds of pipeline segments. The key to these decorations is shown in Fig. 4. The first five kinds of segments all have length $6|V| + 1$, while the sixth has length 3. The first kind of segment is filled with $3|V|$ black batches, then a gray one to interface between black and white, and then $3|V|$ white batches. The second and third kind are completely filled with one product type, and the fourth and fifth are like the second and third except that the first batch is gray. The sixth kind is like the fifth, but only contains three batches.

The pipe network contains one copy of the *variable gadget* structure shown in Fig. 2 for each variable $v \in V$, and one copy of the *clause gadget* structure shown in Fig. 3 for each clause $c \in C$. The open ends to the right of the variable gadgets (dotted) are connected to the open ends to the left of the clause gadgets. In particular, if clause c contains the positive literal v, then area v' of the variable gadget is connected to any of the dangling pipeline segments of the clause gadget for c. Similarly, for negative literals in c, area $\neg v'$ is connected to a dangling pipeline segment of the clause gadget. Because every clause contains at most four literals, there are sufficiently many pipes to make these connections. Because every variable occurs in at most three clauses, at most three new pipeline segments are connected to either v' or $\neg v'$. Any pipeline segments left dangling (for clauses of size three or less) are removed.

The areas in the variable gadgets labeled 3 are the only areas that are not initially empty, each of them containing three black batches. In each clause gadget, the goal requirement is to move the last (rightmost) black batch in the pipe connecting areas c and G into area G. We call these pipeline segments *goal pipes*.

This completes the description of the mapping. Clearly, the PIPESWORLD task can be generated in polynomial time. We will now show that it has a solution iff χ is satisfiable.

First assume that χ is satisfiable, and that α is a satisfying assignment to V. For each variable $v \in V$, we push the three black batches in area 3 of the corresponding variable

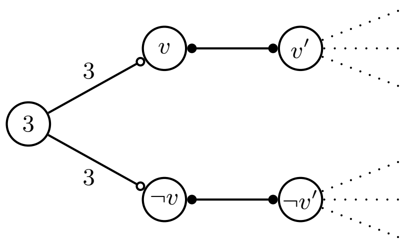

Figure 2: Variable gadget.

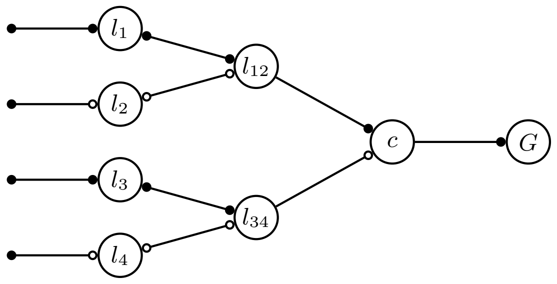

Figure 3: Clause gadget.

gadget into the pipe leading to the area denoted by a literal l satisfied by α (i. e., to area v if $\alpha(v) = 1$ and to area $\neg v$ otherwise). This pushes three batches into this area, one of which is gray. We push the gray batch, then the two white batches into the pipe leading to area l', making three black batches available there. We then push one batch into each of the pipes connecting l' to clause gadgets.

Due to the way variable gadgets are connected to clause gadgets and because α satisfies χ, this places at least one batch in one of the areas l_1, l_2, l_3 or l_4 of each clause gadget. In each clause gadget, we choose one such batch and push it into the pipe leading to l_{12} or l_{34}, placing a batch in one of these areas. This batch is then pushed into the pipe leading to c, releasing a batch there which is pushed into the goal pipe, satisfying the goal for this clause. We thus satisfy the goal in each clause gadget, which shows that the task is solvable.

Now assume that the task has a solution. Obviously, this requires that more batches are pushed into each clause gadget than pushed out of them. It is never possible to push any batch out of a clause gadget unless this batch has been

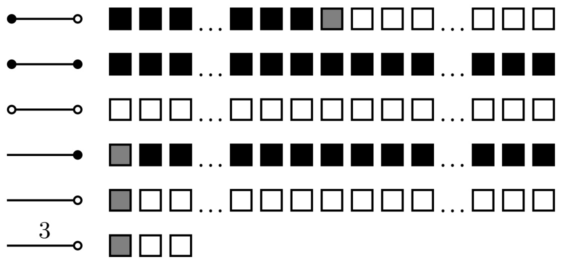

Figure 4: Key to Figs. 2 and 3.

previously pushed into the clause gadget through the same pipe. This is because batches moved into l_{12} from l_1 have the wrong color to be pushed into l_2 and vice versa, and similarly there cannot be a flow between l_3 and l_4 via l_{34} or between l_{12} and l_{34} via c. (Note that there are not sufficiently many batches on the left side of the network to push a gray batch out of the pipes leading to l_1, l_3 or c.)

We can thus treat the segments connecting variable gadgets to clause gadgets as "one-way pipes", which simplifies the analysis because we can consider each variable gadget in isolation. The important property for variable gadgets is that we can either push batches into area v' or into area $\neg v'$, but never both. To see this, note that to push even a single batch into area v', we must push *all three* batches from area 3 into area v; otherwise we obtain only white batches in area v, which cannot be pushed into the pipe connecting to v'. Moreover, to push a batch into v', we must make use of the gray batch from the pipe between 3 and v' and without pushing that gray batch back into v (which requires emptying area v'), we cannot push anything back into area 3.

Therefore, if there is a solution, then there is one where for each variable only one of the areas v' and $\neg v'$ ever contains a batch. Define the truth assignment α so that $\alpha(v) := 1$ if area v' ever contains a batch, and $\alpha(v) := 0$ otherwise. Then a batch can only be pushed into the clause gadget area l_i if α satisfies l_i. Because at least one batch must be pushed into each clause gadget, α satisfies at least one literal in every clause, and hence χ is satisfiable. This concludes the proof. ■

Obviously, **NP**-hardness of plan existence also implies that bounded parallel plan existence is **NP**-hard. Also note that the proof works just as well if batches of the same product type are indistinguishable, and hence the goal is expressed in term of *product types*, not batches, which is a more realistic model of the underlying application problem.

Unfortunately, we cannot provide an **NP** membership result, and indeed the question whether plan existence for PIPESWORLD is in **NP** is open.

However, we do know that PIPESWORLD-NOTANKAGE without interface constraints (all product types are compatible) admits polynomial planning algorithms (Pessoa 2004), although bounded plan existence is still **NP**-complete for this PIPESWORLD variant. Due to space limitations, we do not prove this result.

PROMELA

PROMELA (*Process* or *Protocol Meta Language*) is the input language used by the SPIN model checker (Holzmann 1997). The PROMELA planning domain (Edelkamp 2003) encodes a subset of PROMELA in PDDL2.2, allowing the application of planning technology to a certain class of model-checking problems. We first introduce and discuss the general PROMELA planning domain, then the restricted subclasses PROMELA-PHILOSOPHERS and PROMELA-OPTICALTELEGRAPH, which were part of the IPC4 benchmark set.

A PROMELA task defines a distributed system consisting of a set of *processes*, modeling individual components of a distributed system, and *queues*, used for communication between processes. The goal is always to find a *deadlock* state, in which no process is able to continue its operation.

Planning domain 6 PROMELA
A PROMELA *task is given by finite sets of* ***processes*** P, ***queues*** Q *and* ***messages*** Σ, *a* ***capacity*** *function* $c : Q \to \mathbb{N}_1$, *and for each process* $p \in P$:

- *a finite set of* ***states*** $S(p)$,
- *an* ***initial state*** $s_0(p) \in S(p)$,
- *a set of* ***reading transitions*** $R(p) \subseteq S(p) \times Q \times \Sigma \times S(p)$,
- *a set of* ***writing transitions*** $W(p) \subseteq S(p) \times Q \times \Sigma \times S(p)$.

A state of the task defines:

- *for each process* $p \in P$, *a* ***process state*** $s(p) \in S(p)$, *initially* $s(p) = s_0(p)$,
- *for each queue* $q \in Q$, *the* ***queue contents*** $C(q) \in \Sigma^*$, *initially* $C(q) = \epsilon$.

There is only one kind of actions of the task, ***applying local transitions****. Process* $p \in P$ *can apply local transition* $t = (s, q, a, s') \in R(p) \cup W(p)$ *iff* $s(p) = s$ *and either* t *is a reading transition and the first element of* $C(q)$ *is the message* a, *or* t *is a writing transition and* $|C(q)| < c(q)$. *As a result of the action, the local state of process* p *changes to* s' *and the first element of* $C(q)$ *is removed (if* t *is a reading transition), or message* a *is appended to* $C(q)$ *(if* t *is a writing transition). All other state components are unaffected.*

A state is a goal state iff no action is applicable.

Our definition of the PROMELA domain differs from the PDDL definition in some minor ways that do not limit the applicability of our results. These are discussed towards the end of the section.

Processes can be naturally described by labeled directed graphs, where vertices correspond to process states and arcs to transitions. For a transition $t = (s, q, a, s')$, the graph contains an arc from s to s' with the label $q : a?$ if t is a reading transition and $q : a!$ if t is a writing transition. Fig. 5 shows an example process from PROMELA-PHILOSOPHERS. The process corresponds to a single philosopher, the queues L and R to the forks to his left and right. The intuition behind the model is that writing a message corresponds to putting a fork on the table, and reading a message corresponds to picking it up. Initial process state 1 is a set-up state in which each philosopher puts one fork on the table. After leaving this state, philosophers follow a deterministic strategy of repeatedly requesting the two forks they require in a certain order, then putting them down again.

Communicating processes are a very expressive formalism for modeling computations. This makes planning for general PROMELA tasks hard.

Theorem 7 PROMELA *planning is* PSPACE*-complete.*
Plan existence and bounded plan existence for PROMELA *tasks are* **PSPACE***-complete. This is true even if all queues have capacity 1 and the tasks are deterministic, i. e., at most one action is applicable in any reachable state.*

Proof: We provide a reduction that maps space-restricted Turing Machines to PROMELA tasks such that the task has a solution iff the Turing Machine halts (starting from a blank tape).

Let M be a Turing Machine with state set Z, including initial state $z_0 \in Z$ and accepting state $z_* \in Z$, tape alphabet Γ, including blank symbol $\square \in \Gamma$, and transition function $\delta : (Z \setminus \{z_*\}) \times \Gamma \to Z \times \Gamma \times \{-1, +1\}$. We assume that the machine has n tape cells, starts at the left-most one, and that attempts to move past the end of the tape in either direction is an error that terminates computation (just like reaching the accepting state).

The corresponding PROMELA task has one *tape cell process* p_i and one queue q_i for each tape cell $i \in \{1, \dots, n\}$. The set of messages is the set of Turing Machine states Z. All queues have capacity 1, all tape cell processes have state set $\Gamma \cup (\Gamma \times Z)$, and the initial state of each process is $\square$ except for process p_1 with initial state $(\square, z_0)$.

For each Turing Machine transition $\delta(z, a) = (z', a', \Delta)$ and each tape position $i \in \{1, \dots, n\}$ where $i + \Delta \in \{1, \dots, n\}$, tape cell process p_i has the following transitions:

- A transition from a to (a, z) which reads z from q_i.
- A transition from (a, z) to a' which writes z' to $q_{i+\Delta}$.

There is a straightforward correspondence between configurations of the Turing Machine and states of the corresponding PROMELA task. If after k computation steps, the Turing Machine reaches state z with current tape position i and tape contents $a_1 \dots a_n$, then after $2k$ steps in the PROMELA task, process p_i is in state (a_i, z) and each process $p_j \neq p_i$ is in state a_j. Moreover, all queues are empty.

We prove this inductively. Clearly, the statement is true for $k = 0$. Assume that it is true for k. We can assume that $z \neq z_\star$ and $\delta(z, a_i) = (z', a', \Delta)$ with $i + \Delta \in \{1, \dots, n\}$, since otherwise the Turing Machine computation stops and there is nothing to prove. In this case, the only possible local execution in the PROMELA task is by process p_i, since all other processes are in states that require reading from a queue, and all queues are empty by the inductive hypothesis. Process p_i is in state (a_i, z), which has only one outgoing transition, writing z' to queue $q_{i+\Delta}$ and changing the process state of p_i to a'. In the next step, all processes are in a state that requires reading, but only process $p_{i+\Delta}$ can read from a non-empty queue, so this process acts next. The only applicable transition is the one that reads message z' and changes state from $a_{i+\Delta}$ to $(a_{i+\Delta}, z')$. After these two steps, all queues are empty again, and the local process states again correspond to the Turing Machine configuration as required, concluding the inductive proof.

This shows that if the Turing Machine does not halt, we cannot reach a deadlock in the PROMELA task. On the other hand, if the Turing Machine halts, it either does so by attempting to go past the tape boundaries or by reaching state $z_\star$. In both cases, the PROMELA task reaches a deadlock, because no local executions are possible in the state corresponding to the last Turing Machine configuration after reaching $z_\star$ (or before going past the tape boundaries).

Thus, plan existence for PROMELA tasks is **PSPACE**-hard, which implies that bounded plan existence is also **PSPACE**-hard. Moreover, both problems must belong to

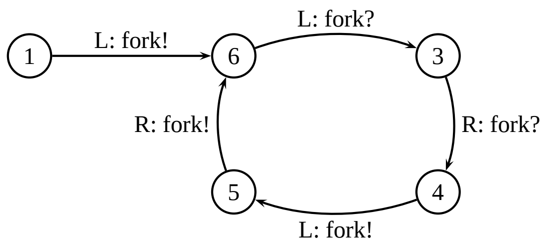

Figure 5: PROMELA-PHILOSOPHERS transition graph. State numbers follow the PDDL specification.

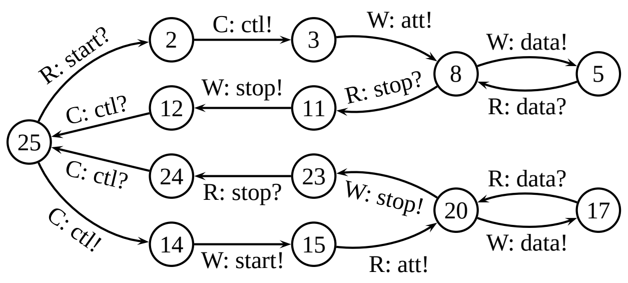

Figure 6: PROMELA-OPTICALTELEGRAPH transition graph. State numbers follow the PDDL specification.

PSPACE because PDDL planning in any fixed propositional domain is in **PSPACE**. This concludes the proof. ■

Having established the result for the general PROMELA domain, we now turn to the PROMELA-PHILOSOPHERS and PROMELA-OPTICALTELEGRAPH domains. These domains are special cases of PROMELA where each task is characterized by a single number. In the former domain, this number defines the number of philosophers in a dining-philosophers style problem. In the latter, it defines the number of optical telegraphs in a communication protocol.

Planning domain 8 PROMELA-PHILOSOPHERS
A PROMELA-PHILOSOPHERS *task is given by a natural number $n \geq 2$ and denotes a* PROMELA *task with message set $\{fork\}$, processes p_i and queues q_i (of capacity 1) for all $i \in \{1, \dots, n\}$. Throughout this section, process and queue indices of* PROMELA-PHILOSOPHERS *tasks are considered modulo n. States and transitions of process p_i are given by the directed graph in Fig. 5, where the initial process state is state 1, L denotes the queue q_i, and R denotes the queue q_{i+1}.*

Planning domain 9 PROMELA-OPTICALTELEGRAPH
A PROMELA-OPTICALTELEGRAPH *task is given by a natural number $n \geq 2$ and denotes a* PROMELA *task with message set $\{att, ctl, data, start, stop\}$, processes p_i^{d} and p_i^{u} and queues q_i^{c}, q_i^{d} and q_i^{u} (of capacity 1) for all $i \in \{1, \dots, n\}$. Throughout this section, process and queue indices of* PROMELA-OPTICALTELEGRAPH *tasks are considered modulo n. States and transitions of the processes are given by the directed graph in Fig. 6, where the initial process state is state 25. For process p_i^{d}, C denotes the queue q_i^{c}, R denotes the queue q_i^{d} and W denotes the queue q_i^{u}. For process p_i^{u}, C denotes the queue q_i^{c}, R denotes the queue q_{i+1}^{u} and W denotes the queue q_{i+1}^{d}.*

Because of their simple scaling structure, these benchmarks are much easier to solve than general PROMELA tasks.

Theorem 10 PROMELA-PHILOSOPHERS *is easy.*
In the PROMELA-PHILOSOPHERS *domain, optimal plans can be generated in polynomial time.*
Proof: To reach a goal state, apply the transitions from 1 to 6 to 3 in all processes. When all processes are in state 3, they are all blocked, so this is a solution of length $2n$, if n is the number of philosophers.

We now prove optimality. Because there is only one message type and queues have size 1, queues only have two configurations, *full* or *empty*. We can verify the following invariant: Queue q_i is full iff p_i is in state 5 or 6 and p_{i-1} is in state 1, 3 or 6. Therefore p_i cannot be deadlocked in state 1 or 4 (q_i is not full if p_i is in state 1 or 4) or in state 5 (q_{i+1} is not full if p_i is in state 5). Therefore, processes can only be blocked in states 6 or 3. However, if all processes are in state 3 or 6 and p_i is in state 6, then q_i is not full and hence p_i is not blocked. Therefore, for all processes to be blocked, all of them must be in state 3. The generated plan clearly is the shortest sequence of actions achieving this. ■

Theorem 11 PROMELA-OPTICALTELEGRAPH *is easy.*
In the PROMELA-OPTICALTELEGRAPH *domain, optimal plans can be generated in polynomial time.*
Proof: To reach a goal state, first apply the transitions from 25 to 14 to 15 in all processes p_i^{d}, then the transition from 25 to 2 in all processes p_i^{u}. Clearly, this leads to a deadlock. Optimality can be proved by similar arguments as for PROMELA-PHILOSOPHERS (details omitted). ■

All the results in this section easily generalize to *parallel planning*. Clearly, the general **PSPACE**-completeness result also applies to that setting, as **PSPACE**-completeness of plan existence implies **PSPACE**-completeness of bounded parallel plan existence for propositional PDDL domains. In the restricted domains, parallelism allows taking the transitions of each process simultaneously, so that the optimal parallel plan length for any PROMELA-PHILOSOPHERS task is 2, whereas the optimal parallel plan length for any PROMELA-OPTICALTELEGRAPH task is 3. In the latter case, note that process p_i^{u} can only transition to state 2 *after* p_{i-1}^{d} has transitioned to state 15.

Finally, some comments on the differences between our formalization and the actual PDDL domain. First, the PDDL definition seems to have a minor flaw in the formalization of writing to queues of capacity 2 or greater. The hardness proof does not require such queues and they do not occur in the competition domains, so this does not make a difference. Second, because of another flaw in the PDDL definition, processes can only be recognized as blocked in states with at most one outgoing transition; reaching a deadlock in which some process has two outgoing transitions in its current state is not considered a solution, even if all those transitions are blocked. This does not affect our proofs for the competition domains, but it does mean that the **PSPACE**-hardness proof is not immediately applicable to the PDDL specification. However, it is easy to adjust to work around the flaw. Finally, due to the difficulty of expressing the queue updates and dead-lock condition succinctly in PDDL, a single action in our model corresponds to a sequence of four actions in the PDDL model, and another action is needed at the end of the plan for each blocked process with an outgoing transition in the current state. Counting the number of PDDL actions, the $2n$ plan length for PROMELA-PHILOSOPHERS thus becomes $9n$ ($2n$ transitions, n processes), and the $3n$ plan length for PROMELA-OPTICALTELEGRAPH becomes $14n$ ($3n$ transitions, $2n$ processes). The optimal parallel plan lengths in the PDDL domains, following the PDDL definition of concurrency, becomes 9 for PROMELA-PHILOSOPHERS and 11 for PROMELA-OPTICALTELEGRAPH (it is not 14 since some of the "subactions" can be interleaved).

PSR

The PSR (*power supply restoration*) domain was originally introduced for planning under uncertainty (Thiébaux & Cordier 2001). At the 4th International Planning Competition, a deterministic and fully observable variant was one of the benchmark domains. The domain models a situation where parts of a power network, consisting of power sources (*circuit breakers*), switches and power lines, have turned faulty. Circuit breakers or switches can be open or closed, with open devices blocking the current. The objective of a PSR task is to reconfigure the network by opening and closing devices so that as many lines as possible are fed, while avoiding to feed any faulty lines (which immediately opens all power sources feeding them).

Planning domain 12 PSR
A PSR *task is given by a finite set of* ***devices*** D*, partitioned into* ***circuit breakers*** $C \subseteq D$ *and* ***switches*** $D \setminus C$*, and by a finite set of* ***lines*** L*, some of which are* ***faulty lines*** $F \subseteq L$*.*

Devices are linked to lines by a connected, bipartite graph called the ***power network****. In the power network, each edge connects a device to a line. Circuit breakers have a degree of* 1*, switches a degree of* 1 *or* 2*. There is no restriction on the degree of lines. We say that a line is* ***feedable*** *iff there exists a path in the power network leading from a circuit breaker to that line which does not pass through any faulty lines (including the line itself).*

For each circuit breaker, there is a set of lines and devices called its ***feeder tree****, including the circuit breaker itself. The subgraph induced by a feeder tree must be a tree where all leaves are devices. Each line and circuit breaker is part of exactly one feeder tree, each switch part of one or two feeder trees. If it is part of two feeder trees, it must be a leaf in both induced graphs and is called a* ***joining switch****.*

A state of the task is given by a set $O \subseteq D$ *of* ***open*** *devices; non-open devices are called* ***closed****. Initially, the joining switches are open, all other devices closed. We say that a line* $l \in L$ *is* ***fed*** *by a circuit breaker* $c \in C$ *in a given state iff there exists a path in the power network from* c *to* l *which does not pass through any open device (including* c *itself). We say that a circuit breaker is* ***affected*** *iff a faulty line is fed by it. A state is* ***unsafe*** *iff there is an affected circuit breaker, and* ***safe*** *otherwise.*

There are three kinds of actions in the task:

- *The* ***wait*** *action is applicable iff the state is unsafe, and opens all affected circuit breakers.*
- *The* ***open*** *and* ***close*** *actions are applicable iff the state is safe. They open or close a single device.*

A state is a goal state iff it is safe and each feedable line is fed by some circuit breaker.

Differently to the original PDDL definition, we do not explicitly model the *earth* device, which is always open, but rather allow to have switches with a degree of 1, which leads to the same semantics.

Note that there is no requirement that a line be fed by only one circuit breaker, although this is true for the initial state due to the fact that joining devices are initially open.

Somewhat surprisingly, optimal plans in PSR can be generated in polynomial time.

Theorem 13 PSR *is easy*
In the PSR *domain, optimal plans can be generated in polynomial time.*
Proof: Solving PSR tasks requires maintaining safety and feeding all feedable lines. The safety property is monotonously increasing in the set of open devices, i. e., if $O \subseteq O'$ and state O is safe, then state O' must also be safe. (Recall that we identify states with the corresponding set of open devices.) The feeding property is monotonously decreasing in the set of open devices, i. e., if $O \subseteq O'$ and a certain line is fed in O', then it is also fed in O. Thus, the two aspects of solving a PSR task conflict in a certain way. However, as we shall see, it is possible to separate these aspects by ensuring safety first, then feeding all feedable lines.

We say that a circuit breaker is *dangerous* iff it is adjacent to a faulty line, and a switch is dangerous iff it is adjacent to a faulty line and to a feedable line.

If the initial state is a goal state, we return the empty plan. Otherwise, since all lines are fed initially by the circuit breaker in their feeder tree, the initial state must be unsafe, and the first action in any plan must be a *wait* action, which opens all dangerous circuit breakers. We then use *open* actions to open all dangerous switches. Like the initial *wait* action, these actions must occur in any solution (although not necessarily at this point), because switches can only be opened by *open* actions (rather than by waiting, as for circuit breakers), and dangerous switches must be open in a goal state: Assume dangerous switch d were closed in a goal state. By definition, it is adjacent to a feedable line l and faulty line l'. In a goal state, l must be fed by some circuit breaker c, so l' is also fed by c, and hence c is affected and the goal state unsafe, a contradiction. It is also evident that dangerous circuit breakers must be open in a goal state to ensure safety.

Interestingly, having all dangerous devices open is not just necessary for safety of a goal state, it is also *sufficient* for safety of any state. Assume that this were not the case and there were an unsafe state where all dangerous devices are open. By definition of safety, in this state there must be a path $\pi = d_1 l_1 \dots d_n l_n$ from circuit breaker d_1 to faulty line l_n where all devices d_i are closed. We can assume that l_n is the only faulty line on the path (if l_i for $i < n$ is faulty, we consider $\pi' = d_1 l_1 \dots d_i l_i$ instead). If $n = 1$, then the circuit breaker d_1 is dangerous and therefore not closed, a contradiction. If $n > 1$, then line l_{n-1} is feedable by the path $d_1 l_1 \dots d_{n-1} l_{n-1}$, and hence d_n connects a feedable line to a faulty line and is dangerous and therefore not closed, a contradiction.

Therefore, we can solve the task as follows:

- Wait, then open all dangerous switches.
- Compute a set of non-dangerous devices D_C of minimal cardinality such that closing D_C leads to all lines being fed. Close these devices.

We already saw that all actions in the first step must occur in any solution. Moreover, due to the monotonicity of feeding and due to the fact that closing a device requires a single action per device (unlike opening, which can in some cases be done more efficiently with the *wait* action), the generated plan is clearly optimal provided that safety is not violated by any of the closing actions. However this is ensured by the fact that having all dangerous devices open is sufficient for safety.

Thus, we only need to show how to calculate the set D_C in polynomial time. For this purpose, we apply some transformation to the power network. First, we remove all dangerous devices along with all lines and devices that become disconnected from the circuit breakers by this operation. Clearly, since all dangerous devices are open and we are not going to close them, this is a valid operation. This results in a graph where all lines are feedable and no devices are dangerous, so we can ignore wait or open actions in the following. Second, we introduce a new (closed) *main circuit breaker*, connect it to a new *main line*, and connect that line to all original circuit breakers in the network, which change status to switches (note that their degree is now 2 due to the edge from the main line). Again, this does not change the semantics of the *fed* predicate. It *does* change the semantics of affectedness for our network, but this is not a problem because our network contains no faulty lines. Third, we remove all switches with degree 1 (they are no use for solving the task) and replace all other switches with colored (i. e., labeled) edges connecting their two neighboring lines, using red edges for open switches and green edges for closed switches. Closing a switch thus corresponds to changing the color of an edge to green. A line is fed iff it is reached by a path from the main circuit breaker that does not pass through any red lines, and hence all lines are fed iff the subgraph obtained by removing all red lines is connected. To achieve this with a minimal number of close actions, we can compute a spanning tree with a minimal number of red edges, or equivalently a minimal spanning tree in the weighted graph obtained by assigning weight 1 to all red edges and weight 0 to all other edges. Computing a minimal spanning tree is a polynomial time operation. ■

Some comments are in order at this point. First, if we use Prim's algorithm (Cormen, Leiserson, & Rivest 1990) for computing minimum spanning trees, it is easy to verify that the complete PSR planning algorithm amounts to the following quite simple greedy strategy:

1. If the initial state is a solution state, return the empty plan; otherwise continue.
2. Wait.
3. Open all dangerous switches.
4. Until a goal is reached, close some non-dangerous device such that closing this device leads to at least one additional line being fed.

Second, the proof critically relies on the fact that switches are connected to at most two lines, and circuit breakers only to one line. Eliminating the degree restriction for devices indeed leads to a more difficult domain, for which bounded plan existence is **NP**-complete. However, we do not prove this result here.

Finally, the problem remains easy in a parallel planning framework. In fact, according to the PDDL definition of the domain, no two PSR actions are concurrently executable, due to the conservative definition of mutexes in the presence of derived predicates (Hoffmann & Edelkamp 2005). Under a less strict notion of concurrency, it makes sense to allow opening several devices in parallel and closing several devices in parallel if that does not lead to any circuit breakers being affected. Using this notion, it is obvious that the optimal parallel solution length for any PSR task is 3, where the first step consists of a wait action, the second of a number of open actions, and the third of a number of close actions. This concludes our discussion of PSR, and of the IPC benchmarks in general.

Summary and discussion

Fig. 7 summarizes our results. Comparing these findings to the complexity properties of the earlier benchmark domains (Helmert 2003), one notable development is the advent of **PSPACE**-equivalent planning domains at IPC4. We believe that this is an effect of the competition organizers' focus on actively seeking for *realistic* and *structurally interesting* domains. Indeed, they mention including a **PSPACE**-equivalent benchmark as one of the desiderata for the IPC4 benchmark suite (Edelkamp & Hoffmann 2003). Another notable development is the advent of a *non-trivial* planning domain in which optimal solutions can be computed efficiently, namely PSR. We consider theoretically tractable, but non-trivial domains an import addition to the toolset of planner evaluation, and indeed solving PSR tasks has proven to be very challenging for state-of-the-art planning systems.

Realism and interestingness come with a cost in complexity, and this is not limited to decision complexity. In three out of the four IPC4 domains (or domain groups) we studied, the PDDL models are flawed, PSR being the only exception. Moreover, two of these three (PIPESWORLD and PROMELA) are modeled unnaturally in the sense that an atomic activity in the modeled domain (such as pushing a batch into a pipe, or taking a transition in a process) corresponds to several actions in the model. A similar need for splitting con-

optimal planning in **P**:
PROMELA-OPTICALTELEGRAPH, PROMELA-PHILOSOPHERS, PSR

planning in **P**, optimal planning **NP**-equivalent:
DEPOT, DRIVERLOG, ROVERS, SATELLITE, ZENOTRAVEL

planning **NP**-hard:
PIPESWORLD (both variants)

planning **PSPACE**-equivalent:
AIRPORT, PROMELA (general case)

Figure 7: Planning complexity for the IPC3 and IPC4 domains.

ceptually atomic activities into several PDDL operators has been identified in an application domain devised by Boddy et a. (2005). From these observations, we conclude that it would be useful to extend PDDL to allow modeling complex operators more naturally. Allowing operators to be defined as *sequential compositions* of other (not independently applicable) suboperators could go a long way towards addressing these issues, while still allowing for compilation techniques to current PDDL.

Apart from an increase of complexity, another interesting trend in the competition domains is the lack of a common theme or pattern in the IPC4 benchmark suite. Many of the IPC1 and IPC2 domains can be subsumed under the heading *transportation domains* (Helmert 2001). Many of the domains of IPC3 have a transportation or route-planning aspect, but except for ZENOTRAVEL, each of them significantly diverges from the theme in some way. For the IPC4 domains, no central theme can be identified at all. This is clearly a healthy development if we want planning technology to apply to a wide spectrum of application domains.

In the introduction, we mentioned how the competition benchmarks have influenced the development of propositional planning systems in the past. Our results show that the IPC4 domains present considerable difficulties from a complexity point of view. It will be interesting to see how planning technology will rise to this challenge.

Acknowledgments

Many thanks to Sylvie Thiébaux for identifying a number of mistakes in an earlier version of this paper.

References

Biundo, S.; Myers, K.; and Rajan, K., eds. 2005. *Proc. ICAPS 2005*.

Boddy, M.; Gohde, J.; Haigh, T.; and Harp, S. 2005. Course of action generation for cyber security using classical planning. In *Proc. ICAPS 2005*, 12–21.

Cormen, T. H.; Leiserson, C. E.; and Rivest, R. L. 1990. *Introduction to Algorithms*. The MIT Press.

Edelkamp, S., and Hoffmann, J. 2003. Quo vadis, IPC-4? — Proposals for the classical part of the 4th International Planning Competition. In *Proc. ICAPS 2003 Workshop on the Competition*.

Edelkamp, S. 2003. Limits and possibilities of PDDL for model checking software. In *Proc. ICAPS 2003 Workshop on the Competition*.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability — A Guide to the Theory of NP-Completeness*. Freeman.

Hatzack, W., and Nebel, B. 2001. The operational traffic control problem: Computational complexity and solutions. In *Proc. ECP 2001*, 49–60.

Hearn, R. A., and Demaine, E. D. 2005. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science* 343(1–2):72–96.

Helmert, M.; Mattmüller, R.; and Röger, G. 2006. Approximation properties of planning benchmarks. Technical Report 224, Albert-Ludwigs-Universität Freiburg, Institut für Informatik.

Helmert, M. 2001. On the complexity of planning in transportation domains. In *Proc. ECP 2001*, 349–360.

Helmert, M. 2003. Complexity results for standard benchmark domains in planning. *AIJ* 143(2):219–262.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *Proc. ICAPS 2004*, 161–170.

Hoffmann, J., and Edelkamp, S. 2005. The deterministic part of IPC-4: An overview. *JAIR* 24:519–579.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Holzmann, G. J. 1997. The model checker SPIN. *IEEE Transactions on Software Engineering* 23(5):279–295.

Koehler, J.; Nebel, B.; Hoffmann, J.; and Dimopoulos, Y. 1997. Extending planning graphs to an ADL subset. In *Proc. ECP 1997*, 273–285.

Long, D., and Fox, M. 2003. The 3rd International Planning Competition: Results and analysis. *JAIR* 20:1–59.

McDermott, D. 2000. The 1998 AI Planning Systems competition. *AI Magazine* 21(2):35–55.

Milidiú, R. L.; dos Santos Liporace, F.; and de Lucena, C. J. P. 2003. Pipesworld: Planning pipeline transportation of petroleum derivatives. In *Proc. ICAPS 2003 Workshop on the Competition*.

Pessoa, A. A. 2004. Planning the transportation of multiple commodities in bidirectional pipeline networks. In Fleischer, R., and Trippen, G., eds., *Algorithms and Computation, 15th International Symposium (ISAAC 2004)*, volume 3341 of *LNCS*, 766–777. Springer-Verlag.

Slaney, J., and Thiébaux, S. 2001. Blocks World revisited. *AIJ* 125(1–2):119–153.

Thiébaux, S., and Cordier, M.-O. 2001. Supply restoration in power distribution systems — a benchmark for planning under uncertainty. In *Proc. ECP 2001*, 85–95.