22nd International Conference
on Automated Planning and Scheduling
June 25-29, 2012, Atibaia – Sao Paulo – Brazil

WS-IPC 2012

Proceedings of the 3rd Workshop on the International Planning Competition

Edited by
Amanda Coles, Andrew Coles, Ángel García Olaya, Sergio Jiménez, Carlos Linares López and Scott Sanner

Organization

Program Committee

Foreword

The International Planning Competition (IPC) was created in 1998 to set a common ground for comparing different planning techniques. Nowadays the IPC is considered a reference source when building a planner and most of the new planning techniques presented at ICAPS, are evaluated regarding the languages, benchmarks and metrics defined in the competition. However, several critiques have been raised concerning the necessity and usefulness of several aspects of the competition.

Given the relevance of IPC and continuing with the lineage of the workshops organized at ICAPS 2003 and 2007, this workshop aims to review the current status of the IPC, and to help to determine/sketch/prepare the forthcoming competition, the Eight International Planning Competition.

Amanda Coles, Andrew Coles, Ángel García Olaya,
Sergio Jiménez, Carlos Linares López, Scott Sanner
Workshop Organizers
June 2012

The edition of these proceedings has been partially supported by the Universidad Carlos III de Madrid, Spain

Contents

The Academic Advising Planning Domain

Joshua T. Guerin, Josiah P. Hanna, Libby Ferland, Nicholas Mattei, and Judy Goldsmith
Department of Computer Science
University of Kentucky
Lexington, KY 40506-0633
jtguer2@uky.edu, jpha226@g.uky.edu, libby.knouse@uky.edu, nick.mattei@uky.edu, goldsmit@cs.uky.edu

Abstract

The International Probabilistic Planning Competition is a leading showcase for fast stochastic planners. The current domains used in the competition have raised challenges that the leading deterministic-planner-based MDP solvers have been able to meet. We argue that in order to continue to raise challenges and match real world applications, domains must be generated that exhibit true stochasticity, multi-valued domain variables, and concurrent actions. In this paper we propose the academic advising domain as a planning competition domain that exhibits these characteristics. We believe that this domain can build upon the success of previous contests in pushing the limits of MDP planning research.

Introduction

Generating new problem sets to challenge state-of-the-art probabilistic planners is often difficult. Planners have evolved in leaps and bounds in a very short time, opening the door to whole new classes of problems that can realistically be solved by computer programs. As a driver in planning research, the International Probabilistic Planning Competition (IPPC) has presented a number of domains meant to continue pushing the field in new and exciting directions.

As the IPPC has continued, domains have become increasingly complex and stochastic. In recent contests, strong planners that solve deterministic instances of stochastic domains (or "replanners" (Little and Thiébaux 2007)) have continued to outstrip their inherently probabilistic counterparts. This indicates to us that the domains that have been used are still not stochastic enough to closely model the complexities of real world planning problems. We suggest that current real world domains can be modified to present more difficult problems, and that new domains requiring different problem solving strategies can help provide more challenges for planners.

Current competition domains have been designed to represent real world scenarios for probabilistic planners. In the Elevator domain, for example, a controller coordinates a bank of elevators to pick up passengers when requests are received, and delivers them to the destination floor. This domain assumes that each elevator is acting separately and actions are performed sequentially. This makes the state space less complex, allowing deterministic replanners to produce strong solutions. In most real world problems, however, the most comprehensive models are highly stochastic and consider actions taken concurrently. This domain can be modified by assuming that the controller is directing elevators in concurrent pairs or groups, instead of individually and sequentially. This introduces joint actions that greatly increase the size and complexity of the state space. As the state space grows more complex, it becomes more difficult for deterministic planners to produce a good solution, and probabilistic planners using concurrent action planning become better choices due to a more complete consideration the domain (Sanner 2008). We believe that many other competition domains could be represented in this fashion, and that continuing to introduce domains requiring different planning methods provides interesting and valuable challenges to competitive planners.

Our domain, the academic advising domain, represents a real world model in which concurrent actions must be considered in order to reach an optimal solution. Academic advising, when treated as a probabilistic planning domain, is rich with the qualities used to classify domains as probabilistically interesting (Little and Thiébaux 2007). The presence of avoidable dead-ends, near-identical trajectories with distinct outcomes, multiple distinct trajectories, and mutually exclusive actions, as well as a high degree of unpredictable outside influences, makes for a challenging environment for state-of-the-art planners today. Planners in this domain must consider factors such as past performance history, courses completed, and above all else the student's ability to take multiple courses concurrently — all with the aim of maximizing the student's GPA as well as satisfying the requirements for graduation with a specific area of study.

This domain presents a large, complex, and easily scalable state space, ideal for a challenging competition. Most importantly, while deterministic planning may eventually produce a solution, optimal solutions can only be found using stochastic, concurrent-action planning. We believe that this quality could make the academic domain a good addition to the domains already used in competition, and help introduce even more "spice" into an already challenging problem set (Sanner 2008). Competitive domains have always been used to test the very best in planners, and it is our hope that the academic domain might be able to continue in this tradition.

In the section "The Academic Domain", we describe the real-world advising domain, and in the section "The Academic Domain Model Generator Problem Structure", we briefly sketch the proposed domain generator. Note that others have considered MDP-based advising domains. Both Khan et al. and Dodson et al. have used hand-constructed models of advising to motivate work on explanation generation for MDP policies (Khan, Poupart, and Black 2009; Dodson, Mattei, and Goldsmith 2011).

The Academic Domain

Good academic advising in the American educational system involves many decisions. In most degree programs, students choose electives to satisfy multiple, sometimes overlapping requirements. Their choice of electives, the order in which courses are taken, and the choices of which courses to take together, all have enormous effects on the student's success and their enjoyment of their academic career.

We can model advising as a factored MDP. A student's course history and grades determine their current state. Based on this state, a student can select one or more courses to take during the next semester. More formally, states of the MDP are student transcripts. Each variable represents a possible course, with values (let us say) HIGH, LOW, FAIL and NOT TAKEN. An action specifies a course to take. The courses available for selection at any stage depend on the fulfillment of prerequisites. Long prerequisite chains mean a policy with a time bound on the number of semesters (or a discounted reward) must begin the long chains early enough to ensure the later courses can be taken.

Note that students who have taken the same courses can be in very different states if one has received high grades and the other has received low or failing grades. Two students with the same GPA may be in different states because they took different courses.

The effects of actions are stochastic. Grading scales are less than precise measures, and small exogenous events (something seen on the way to an exam, a girlfriend with tutoring skills, a fight with a friend) can have large effects on grades.

A policy specifies sets of concurrent actions at each time step, since students take sets of courses each semester. While the goal of the policy can vary, almost all academic domain policies will seek to avoid states that involve failed classes.

Probabilistic Outcomes

The effects of sets of actions in the academic domain are uncertain. Adding to the uncertainty, the number of possible next states is large for even a small number of concurrent actions. Consider a set of four classes that a student has taken. There are 3^4 possible outcomes, ranging from HIGH, HIGH, HIGH, HIGH to FAIL, FAIL, FAIL, FAIL. Each outcome, or at least each set of FAILs, requires different responses. Required courses must be repeated. Prerequisites and predictors of success *should* be repeated. In some cases, a LOW should be repeated, but not always. For instance, if it were shown that students who had taken first-semester calculus performed better in discrete mathematics then a policy that had a student take calculus before discrete math would be more likely to result in a higher grade for the discrete math class, and a LOW grade in calculus might skew likely grades in several later courses toward LOW or FAIL.

As a result of this combinatorial explosion, the underlying planning problem is not easy to determinize.

University curricula can lead to an enormous state space, and real-world weakly coupled MDPs. Courses that are required for Human Ecology majors may have little impact on Computer Science majors. There is a wealth of potential challenges in modeling actual curricula, and building planners that can recognize and leverage the compartmentalization of individual programs and majors. However, a realistic model of transition probabilities can be built using data mining techniques on grade data from students to have taken courses previously (Guerin and Goldsmith 2011).

Concurrent Actions

Within the academic domain, students take courses in sets rather than one at a time. The result of this is a much larger set of choices for each stage of the plan. The number of courses taken in a semester can vary so that the policy does not need to have a uniform number of courses each semester. Constraints can be specified for the domain so that a student must remain full time (establishing a minimum number of courses for each semester) and/or to establish a maximum number of courses. This can be further complicated by having the number of courses taken affect the probability of success in each course. For example, a student taking four courses is more likely to achieve higher grades than a student taking the same four as well as three additional courses. It should be noted that this planning domain does not have to involve concurrent action planning if the number of courses to be taken at a time is limited to one.

We can model the transition probabilities for each course, based on statistical predictors (CS II grades probabilistically depend, let us say, on the grade in CS I and on grades in prior attempts on CS II). However, actions taken concurrently can affect each other's outcome probabilities, either synergistically or destructively. For instance, taking a compilers course at the same time as models of computation tends to improve grades in both, because the compilers course motivates interest in regular and context-free grammars, while the theory course reinforces computational techniques. On the other hand, taking two courses with the same schedule of assignments, exams, and projects can be detrimental to grades. This means that optimal planning must consider actions concurrently rather than sequentially. This holds in a fully realized model that takes into account concurrency effects, but it also holds in simpler models. Taking multiple courses concurrently has a different effect than taking them in sequence, and thus concurrent actions should not be modeled sequentially.

Goal States

For each school and each program within that school, requirements define a set of goal states. Individual students have preferences on the types of electives, professors, semester schedules, grade point average, time to graduation, etc. The goal of the academic planner is to optimize the student's total expected utility and keep them moving toward graduation. For a small scale realistic version of the problem, the goal can be to earn an academic minor. A larger scale version is planning for completion of a degree. Besides varying in size, goals also depend on what is valued in the program.

A policy can be developed with a simple reward based on time to the goal or on maximizing grades. With the former, an optimal policy would only be concerned with the student passing courses, while a GPA-driven policy might recommend more time be taken if it ensured better grades. Other criteria can be considered for optimization such as enjoyment (different rewards for different courses or for certain distributions of courses in each semester) or uniformity of number of courses each semester.

Real-World Features of the Academic Domain

The features of the academic advising domain reflect aspects of many real world problems. The three features described above allow the academic domain to capture interesting planning problems. These are:

- planning under uncertainty,
- planning with concurrent actions, and
- planning with multiple reward criteria.

First, the stochastic nature of human endeavors, particularly humans of college age, introduces uncertainty into any plan. For this reason a classical plan does not suffice; rather, a policy is needed.

Secondly, the academic domain models concurrent action planning, because most students take multiple courses each semester. Concurrent action planning has applications from space exploration to pharmaceuticals (Mausam and Weld 2004). An example application for this research could be designing drug regimens. For complex medical conditions, a doctor has a choice of different drug treatments. Different drugs may be taken simultaneously and there are uncertainties associated with the effects and side effects of each drug. For instance, a person might be taking medication for migraines, rheumatoid arthritis, allergies, and digestive disorders. Side effects may lead to the prescription of more drugs which will have their own uncertain effects. This scenario can be modeled as a concurrent action MDP. Solving it could provide huge benefits to doctors prescribing drug regimens and to patients trying to understand the utility of their medicines. But first we need to develop concurrent-action MDP solvers.

Finally, the academic domain has different criteria to optimize and could, therefore, be used for planning with multi-criteria optimization (Perny and Weng 2010). This is relevant to many scenarios in which there are multiple values to optimize, such as balancing risk and reward with investments or prescribing radiation to kill tumors and not harm healthy organs (Ehrgott 2000). The academic domain is structured in a way that allows planners to be built that must account for any or all of these features.

The Academic Domain Model Generator

Problem Structure

Although the common Bayes net representation of MDPs uses dynamic Bayes nets, we represent the temporal aspect of our models implicitly, and draw simple Bayes nets. The nodes of the network represent courses. The values of each course are NOT TAKEN, HIGH, LOW, and FAIL. Each node has a probability table for those values, conditioned on its parent nodes.

The domain generation module generates the structure and parameters of the network, a goal state, and additional constraints. The structure is based on a standard lattice. In order to create varied instances, we begin with a full lattice and then remove edges.

One feature of our domain instances, which is more typical of computer science programs than of many other programs at our university, is chains of prerequisites. These can be represented as explicit constraints, if the planners can handle constraints, or can be handled implicitly, through the construction of the conditional probability tables (CPTs): students are able to skip prerequisites or to go forward with a poor or failing grade in a prerequisite course, but the expectation of HIGH or LOW grades is significantly depressed. For now, we use the latter approach, so edges in the lattice are interpreted both as prerequisites and as parents for CPTs.

The instance generator will choose:

- the size of the lattice;
- where to randomly break the prerequisite chains;
- the conditional probability tables for each node;
- the utility/reward function, and/or goal state.

Prerequisite Hierarchy

Course prerequisites are specified by a prerequisite graph. The graphs we generate are based on lattices with complete connectivity between layers, which are then pruned to generate prerequisite models. A visual representation of the first three full lattices of this sequence are shown in Figures 1 and 2. Edges of the lattice will be pruned until the following conditions are met: (1) all courses have at least one prerequisite and (2) few courses have more than one prerequisite. The resulting graphs will bear strong resemblance to prerequisite hierarchies we have surveyed in actual academic departments.

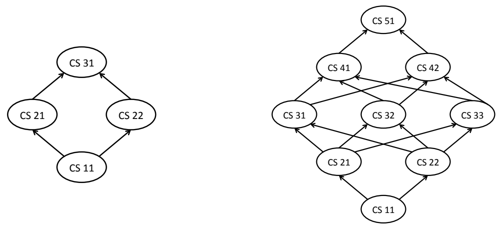


Figure 1: Full prerequisite lattices for small domains

The models we build, based on these lattices, have several interesting properties:

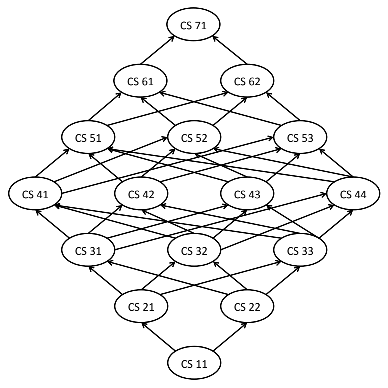


Figure 2: Full prerequisite lattice for a larger domain

1. there is an optimal policy, but even following that policy doesn't guarantee success;
2. the final prerequisite hierarchy specifies a partial ordering over action sequences which must be learned in order to find an optimal policy;
3. the number of *state variables* grows quadratically with the lattice size parameter;
4. the number of *states* grows exponentially in the number of state variables.

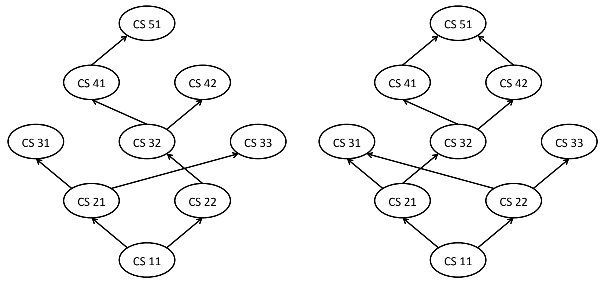


Figure 3: Pruned prerequisite lattices

Figure 3 shows how two lattices with the same nodes can produce two significantly different instances, based on the different edge prunings. A goal state can be specified by giving a list of courses that are required. This can be done either with an explicit set of courses that must be taken plus a total number, or by specifying the number of courses to be taken from each set of courses (i.e., "take 2 courses from the third level of the lattice").

Generating the Domain

The domains and instances can be generated from the specifications of the lattices described in the previous section. The code examples in this paper use the RDDL domain definition language (Sanner). The code describing these lattices will also contain the conditional probabilities of state transitions. This will include depressed probabilities when prerequisites are untaken. The following table shows possible conditional probabilities for the outcome of taking a course.

CS 32	CS 42 outcomes		
grades	H	L	F
H	0.7	0.2	0.1
L	0.3	0.4	0.3
F	0.1	0.15	0.75
NT	0.05	0.1	0.85

A reward function will be specified in the instance problem based upon the definition of the goal state. This function rewards success in required classes and creates penalties for being further from the goal state. Therefore, reaching the goal state is the way to maximize reward. For instance, consider the following reward description for a domain with four possible courses.

```
reward = 3 * [(CS11 == @High)
              + (CS21 == @High)
              + (CS22 == @High)
              + (CS31 == @High)]
         + 1 * [(CS11 == @Low)
              + (CS21 == @Low)
              + (CS22 == @Low)
              + (CS31 == @Low)]
         + 0 * [(CS11 == @Fail)
              + (CS21 == @Fail)
              + (CS22 == @Fail)
              + (CS31 == @Fail)]
         -5 * [(CS11 == @NotTaken)
              + (CS21 == @NotTaken)
              + (CS22 == @NotTaken)
              + (CS31 ==@NotTaken)].
```

Concurrency can be enforced as the means of reaching a solution by specifying a horizon that requires multiple courses taken at a time to reach the goal. The below example specifies a horizon of 8 and allows up to 5 concurrent actions. If the goal state for this domain required 35 courses to be taken, then concurrency is required to reach the goal.

```
instance advising {
        domain = advising;
        init-state {
            CS11 = @NotTaken;
            CS21 = @NotTaken;
            . . .
        };
        max-nondef-actions = 5;
        horizon = 8;
        discount = 0.99;
}
```

Conclusion

This paper has proposed a novel domain, the academic advising domain, for probabilistic planning competitions. This domain is interesting because it involves concurrent actions, true stochasticity, and multi-valued domain variables. It enables the use of constraints, both in terms of prerequisites and potential mutual exclusion constraints. In addition, it opens the possibility of explicitly considering the planning problem as a multi-criteria optimization problem. Introducing these elements to planning competitions will raise new challenges in the IPPC. It is our hope that this domain will build upon the success of previous competition domains to push the state of the art in planning research.

Acknowledgements This work is partially supported by by NSF EAGER grant CCF-1049360. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Dodson, T.; Mattei, N.; and Goldsmith, J. 2011. Natural language argumentation interface for explanation generation in Markov decision processes. In *Proc. Algorithmic Decision Theory*. also appeared in the EXaCT workshop at IJCAI 2011.

Ehrgott, M. 2000. *Multicriteria Optimization*. Berlin: Springer.

Guerin, J. T., and Goldsmith, J. 2011. Constructing a dynamic Bayes net model of academic advising. In *Proc. Bayesian Modelling Applications Workshop, UAI.*

Khan, O. Z.; Poupart, P.; and Black, J. 2009. Minimal sufficient explanations for factored Markov decision processes. In *International Conference on Automated Planning and Scheduling (ICAPS).*

Little, I., and Thiébaux, S. 2007. Probabilistic planning vs. replanning. In *ICAPS Workshop on IPC: Past, Present and Future*.

Mausam, and Weld, D. S. 2004. Solving concurrent Markov decision processes. In *National Conference on Artificial Intelligence*. AAAI.

Perny, P., and Weng, P. 2010. On finding compromise solutions in multiobjective markov decision processes. In *European Conference on Artificial Intelligence Multidisciplinary Workshop on Advances in Preference Handling*, 55–60.

Sanner, S. Relational dynamic influence diagram language (rddl): Language description.

Sanner, S. 2008. How to spice up your planning under uncertainty research life. In *Workshop on a Reality Check for Planning and Scheduling Under Uncertainty (ICAPS-08).*

Position Paper — Leveraging Classical Planners through Translations

Ronen I. Brafman
Department of Computer Science
Ben-Gurion University of the Negev
brafman@cs.bgu.ac.il

Guy Shani
Department of Information Systems Engineering
Ben-Gurion University of the Negev
shanigu@bgu.ac.il

Ran Taig
Department of Computer Science
Ben-Gurion University of the Negev
taig@cs.bgu.ac.il

Abstract

Classical planners are rapidly becoming sufficiently fast and efficient to be used as black boxes in many applications. In other planning research areas, such as conformant planning, contingent planning, and even probabilistic planning, translation based approaches which solve these difficult tasks by translating them into classical planning problems have become popular. These translations typically create a rather sophisticated classical planning problem, typically with many more variables and actions, and often with special characteristics, such as large number of conditional effects. They also create variants of classical planning problems, such as resource-bounded planning problems. Researchers in these area rely on modern classical planners to provide solutions to these problems in a heartbeat. Yet, these features raise serious difficulties for modern classical planners. We suggest that research in classical planning can no longer afford to ignore these limitations, and believe that the planning competition has an important role in stimulating planning researchers to seek planners that better handle them.

Introduction

Once SAT-solvers reached a certain level of maturity, solving huge problems rapidly, it became common practice in many communities to translate their problems into SAT, and then run some generic SAT-solver to obtain a solution (see, e.g. (Hoffmann and Brafman 2006)). That is, for many practical problems, it became advantageous to make the effort to translate a problem, run a generic solver, and translate the solution back, rather than write a specific algorithm for the problem of interest.

Classical planning is on route to playing a similar role as SAT-solvers for diverse, richer planning models. Classical planning algorithms have scaled in the past decade to solve huge problems with millions of variables and thousands of actions very rapidly. Recently, research in conformant planning(Palacios and Geffner 2009), and partially observable contingent planning(Albore *et al.* 2009), has developed smart translation schemes, taking as input a conformant or continent planning problem, and outputing a classical problem, whose solution can be leveraged to solve the original problem. More recently, we have also seen translations from planning problems with soft-goals (Keyder and Geffner 2009) and conformant probabilistic planning (Brafman and Taig 2011) into classical planning problems and its variants, such as resource-bounded planning.

Even when the resulting problems are pure classical planning problems, their properties are different from those commonly found in most classical planning benchmarks. The most problematic limitations being the inability of modern planners to solve domains with conditional effects, and to accept multi-valued parameters in a human-readable format. In many cases, because theoretical research has discussed and offered solutions to these limitations, and translations exist, e.g., for compiling conditional effects to multiple actions (Nebel 2000; Keller and Eyerich 2011), modern planners consider these issues as solved, expecting users to leverage existing research to write translations that would fit the requirements of classical planners. We believe, however, that requiring users of classical planners to understand such research, and implement compilations and workarounds themselves, narrows the number of classical planning users. The competition can play an important role in encouraging the development of planners that handle problems with diverse features. Furthermore, as competition benchmarks play an important role in challenging existing technology and pushing its boundaries, it is important to include this new class of problems among existing benchmarks.

But translation-based methods require more than plain vanilla classical planners. To capture probabilities and preferences, one needs numeric variants of classical planning, including metric planning (Hoffmann 2003) and resource bounded planning (Smith 2004). These difficult problems receive much less attention than classical planning problems. Here, too, the competition can try to encourage research in this area by enriching the set of benchmark problems and encouraging appropriate tracks that were absent from recent IPCs.

Finaly, to be widely used as a black-box, classical planners must offer users the ability to rapidly encode their domains in a human-readable format. This requires the use of multi-value variables, as in SAS, but with a much more readable syntax, possibly as in PDDL 3.1. Although the transition in this case is not difficult, most modern planners stick to SAS or simple variants of PDDL.

The goal of this paper is to convince the IPC community, and not the general planning research community, in the importance of adding translated domains as an obligatory part of the satisfying track of the IPC, and to add more tracks for resource bounded planning and metric planning. We believe that once features such as conditional effects will become standard in the IPC benchmarks, classical planning researchers will find innovative efficient ideas for solving domains with such features. Furthermore, the IPC offers users a fair comparison and a reasonable testing ground for deciding which planner to use in their application as a black-box, without requiring users to fully understand the algorithms and methods employed by the planner. This would allow users of classical planners to focus on their own research, e.g., on generating more expressive and efficient translations, rather than on implementing required features for classical planners. In the rest of this paper we consider some of the difficulties faced when attempting to use classical planners in solving translation based techniques. We refer the readers to (Palacios and Geffner 2009; Albore *et al.* 2009; Brafman and Taig 2011; Keyder and Geffner 2009; Shani and Brafman 2011) for more details on translation techniques.

The Difficulties of Using Classical Planners on the Translations

We now review a set of difficulties that we encountered in using classical planning algorithms to solve classical planning problems generated by automated compilation-based planners, and in particular, problems generated by contingent planners. Obviously, these difficulties are not unique to translations, and some solutions to these problems exist in theory, but are not integrated into current planners. And in fact, those solution we tried (such as compiling away conditional effects) do not scale up well.

Conditional Effects

One of the key features of translations from conformant and contingent planning is a large increase in the number of conditional effects. Intuitively, the generated classical domains attempt to explicitly model the state of knowledge of the agent. This state depends, conditionally, on the value of observations made. Furthermore, one must also condition the actual effect of actions, whether observed or not, on properties that are unknown to the planner, such as the value of some propositions in the initial state of the world. This require the use of conditional effects. Moreover, conformant and contingent planning problems usually contain many conditional effects in their *original* formulation, as in many cases, operators without conditional effects are ineffective given uncertainty.

While it is arguably difficult to imagine a complete partially observable definition language with no conditional effects, it is often argued that for classical problems, conditional effects are merely a convenience. That is, one can compile an action with a conditional effect into two actions — one where the condition c holds and one where it does not, adding its negation into the precondition of the appropriate action. This omits a single condition from the action, and can be repeated until all conditional effects have been removed from all actions.

Domain	Original actions	Max translated conditions
Localize 3	8	48
Localize 5	19	114
Localize 7	34	204
Localize 9	53	318
Localize 11	76	456
Localize 13	103	618

Table 1: Conditional actions in translated localize domains. In these examples, there are 9 original actions, and 315 translated actions.

This approach results in an exponential growth of actions, as each combination of conditions of an action requires a separate new action. When the number of conditions is not too large, as is typically the case in hand-written problems, this is certainly a viable approach. There are other translations (Nebel 2000) that transform an action with conditional effects into a sequence of actions with no conditional effects, resulting in only a linear growth in the number of actions. The result is, however, that the actual cost of an action computed in the planner heuristic becomes dependent on the original number of conditional effect in the action, which distorts the search heuristic.

Thus, perhaps most modern classical planners do not allow for conditional effects (except, perhaps for FF and its extensions), or impose strange constraints on these conditions. For example, FD does not allow a literal and its negation to appear in two different conditional effects of an action.

However, in our compiled problems, featuring hundreds of conditional effects, the syntactic sugar approach is clearly infeasible, and conditional effects typically contain a literal and its negation, for example, in order to reflect knowledge gain and knowledge loss conditions. As such, conditional effects pose the largest obstacle facing the usage of classical planners for solving current translations.

Specifying Non-Binary Variables

One of the limitations of the widely used PDDL language is its inability to directly express multi-valued variables. The only way to specify such variables is through a set of binary variables, whose obvious dependency is not directly specified. An obvious example is when the problem contains the location of an object upon a grid. This location could naturally be defined through a single *location* multi-valued variable, or through two *x,y* multi-valued variables. However, in PDDL the natural definition of a location on a grid requires $|X| \times |Y|$ binary variables, with no formal method of specifying that only one of these variables is always true.

An alternative can be found in the SAS or SAS+ languages, which are the input languages for many modern successful planners, such as LAMA, or FD. These languages allow us to specify multi-valued variables. However, writing the translations directly in SAS has proven to be very difficult, and the authors of this paper have forsaken a direct SAS output after weeks of failures. This is mainly because SAS has a structure that is difficult for humans to read and understand. Thus, while understanding and hence debugging a PDDL file is relatively straight forward, understanding and

Domain type	Multi-valued variables
logistics	3
ebtcs	1
CB $n-k$	$k, k \in [1,7]$
Doors n	$\frac{n}{2}, n \in [5,17]$
Localize	1
Unix	1
Wumpus n	$1+6n, n \in [5,20]$

Table 2: Number of natural multi-valued parameters in existing PPOS benchmarks.

debugging a SAS file of the translation has proven to be beyond our capabilities.

This was obviously acknowledged previously by other researchers, as LAMA and FD offer a translator from PDDL into SAS. Unfortunately, for our translations, this PDDL-to-SAS translator has proven to be very slow, and in fact becomes the bottleneck of the execution. We believe that a language that would offer the flexibility of SAS joined with the understandability of PDDL would make this translator obsolete.

Of course, such extensions to PDDL were offered many times in the past (see, e.g. (Geffner 2000), and the PDDL 3.1 specification developed for the IPC-2008 by Malte Helmert). Still, modern planners that participate in the IPC support only PDDL with boolean propositions, or the difficult SAS format. Clearly, this is a well-known issue in the community, and making, e.g., functional STRIPS, an obligatory input language in the IPC will force participants to support it.

Execution Speed

The IPC currently focuses on satisfiability (i.e. the ability to provide a solution) through a rather generous time frame (30 minutes). In some cases, although not always, when planners are executed as a black box, it is critical that each problem will be solved very rapidly. A user of such a black box planner may prefer a planner that solves most problems rapidly, but sometimes is unable to solve a problem, to a planner that solves more problems, but does so more slowly. We believe that the execution speed of planners should be evaluated in the IPC directly, and faster planners should receive adequate acknowledgement.

Table 3 demonstrates the amount of time required for scaling up. We report both the number of times that the planner was executed, as well as the total runtime of the entrie contingent online planner. As we can see, planners that take more than a few seconds to compute a solution, become a bottleneck for our approach.

Linux-only Implementations

Many researchers prefer the Linux environment for developing software. To date, all the classical planners that we have explored were originally developed and tested on Linux environments. Still, a vast majority of programming nowadays is done on Windows systems. If the classical planning community wants its planners to be used as black boxes within other applications, the limitation of development under Linux is simply unrealistic.

Domain	Replanning	Total runtime (secs)
elog7	3.3	1.3
ebtcs-70	35.6	15.8
CB $9-7$	310.9	693.3
Doors 17	60	96.9
Localize 17	6.6	75
Unix 4	29.4	12.4
Wumpus 20	17.7	156.1

Table 3: Number of replanning episodes in existing PPOS benchmarks. Reported numbers are averages over 50 executions.

Planner	Compiles	Runs	Solves
FF	Yes	Yes	Yes
FF(h_s)	Yes	No	No
FF(h_{sa})	Yes	No	No
MIPS-XXL	Yes	No	No
Metric-FF	Yes	Yes	No
LPG	Yes	No	No
LAMA (2008)	Yes	No	No
FD	Yes	Yes	Yes

Table 4: The behavior of planners we experimented with on PPOS benchmarks under Cygwin. Planner download pages were accessed from the IPC-6 and IPC-7 webpages.

A workaround that is often used is to compile in Windows under the simulated Linux environment Cygwin. While this is a viable option, and in many cases works well, we have encountered several problems with this approach. First, certain components often used in planners are not working properly in Cygwin. Specifically, Bison and Flex are often used tools for specifying the PDDL syntax, but fail on, e.g, the FF syntax under Cygwin. Workarounds exist, such as precompiling some files in a Linux environment, and then transferring the results to Cygwin, but such solutions put yet another obstacle for the user. Furthermore, several compiled applications do not work the same way under Cygwin as they do under Linux. Specifically, we encountered problems running MIPS-XXL,FF(h_a), and FF(h_{sa}), on Cygwin which did not reproduce on Linux. That being said, some important planners such as FF and FD compile and work well under Cygwin.

While transferring all current Linux *C* and *C++* code into modern *C#* or *Java* is clearly difficult, we urge the community to start writing new planners in languages that can be compiled and executed within a Windows system. One possible low-cost alternative is to allow Windows users to access planners through Cygwin. The IPC should perhaps encourage the submission of planners compiled under Cygwin, and test this compilation to ensure that this compiled version operates as well as the original Linux compilation. Thus, the IPC would ensure that equivalent results will be obtained on both Windows and Linux.

Table 4 shows a part of the planners that we experimented with and the difficulties in using them. Note that although we managed eventually to use FD, this was only achieved after receiving much help by two experts in FD, Erez Karpas and Michael Katz, explaining to us how to translate our conditional effects into a format readable by FD.

Conclusion

To conclude, in this position paper we argue that classical planners may become the new building block of many modern applications. To achieve this ambitious goal, such planners must be able to handle more complicated domains, and allow additional flexibility in the inputs that they process. Based on our experience in using classical planners to solve translated contingent and conformant problems, we point to a number of limitations that, should they be resolved, will make our usage of classical planners much more natural and easy.

We realize that the IPC is run by volunteers and cannot simply dictate to the planning community what research to do and what planners to build. In fact, it is often the case that new tracks are organized because sufficient planners exist. Nevertheless, through awareness to the issues we raise, and to the important trend of using planners as blackboxes, the IPC can play an important role in shaping the direction planning research and making planning technology more usable.

References

Alexandre Albore, Héctor Palacios, and Hector Geffner. A translation-based approach to contingent planning. In *IJCAI*, pages 1623–1628, 2009.

Ronen I. Brafman and Ran Taig. A translation-based approach to conformant probabilistic planning. In *Second Int. Conf. on Algorithmic Decision Theory*, 2011.

Héctor Geffner. Logic-based artificial intelligence. chapter Functional strips: a more flexible language for planning and problem solving, pages 187–209. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

J. Hoffmann and R. I. Brafman. Conformant planning via heuristic forward search: A new approach. *Artif. Intell.*, 170(6-7):507–541, 2006.

Jörg Hoffmann. The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. *J. Artif. Intell. Res. (JAIR)*, 20:291–341, 2003.

Thomas Keller and Patrick Eyerich. A polynomial all outcome determinization for probabilistic planning. In Fahiem Bacchus, Carmel Domshlak, Stefan Edelkamp, and Malte Helmert, editors, *ICAPS*. AAAI, 2011.

Emil Keyder and Hector Geffner. Soft goals can be compiled away. *J. Artif. Intell. Res. (JAIR)*, 36:547–556, 2009.

Bernhard Nebel. On the compilability and expressive power of propositional planning formalisms. *J. Artif. Intell. Res. (JAIR)*, 12:271–315, 2000.

Héctor Palacios and Hector Geffner. Compiling uncertainty away in conformant planning problems with bounded width. *JAIR*, 35:623–675, 2009.

Guy Shani and Ronen I. Brafman. Replanning in domains with partial information and sensing actions. In *IJCAI*, pages 2021–2026, 2011.

David E. Smith. Choosing objectives in over-subscription planning. In *ICAPS*, pages 393–401, 2004.

Advances in BDD Search: Filtering, Partitioning, and Bidirectionally Blind

Stefan Edelkamp and Peter Kissmann
TZI Universität Bremen, Germany
{edelkamp, kissmann}@tzi.de

Álvaro Torralba
Planning and Learning Group
Universidad Carlos III de Madrid, Spain
atorralb@inf.uc3m.es

Abstract

Symbolic search with BDDs often saves huge amounts of memory and computation time, but in the sequential-optimal track of the 2011 International Planning Competition (IPC) explicit-state heuristic search planners performed better.

In this paper we present a set of improvements for blind and heuristic search with BDDs that indicate how to scale better. Besides some basic refinements this paper proposes two general techniques to advance BDD search by refining the image operator to compute the set of successors. First, a transition relation tree selects the set of applicable actions through *filtering* their precondition. Secondly, the state sets to be expanded are *partitioned* in equally-sized state subsets. Experiments on IPC 2011 planning benchmark domains are reported, with surprisingly good results for bidirectional blind search.

Introduction

In the 2011 International Planning Competition (IPC 2011), explicit-state heuristic search planners showed advantages to symbolic planners (that won the preceding one), indicating that the increased quality of search heuristics sometimes exceeds the structural savings for representing and exploring large state sets in advanced data structures.

For the automated construction of search heuristics for planning, symbolic pattern databases have been proposed by Edelkamp (2005) and refined by Kissmann and Edelkamp (2011). Alternatively, Bonet and Geffner (2008) compile the planning heuristic h^+ into a logic program and extract a d-DNNF of it, where d-DNNFs are other succinct representations for Boolean functions for which many operations like model counting are polynomial, but not always as efficient as for decision diagrams.

In this paper we look at improvements for the BDD-based planner GAMER that won the sequential-optimal track of IPC 2008. The improvements for GAMER proposed in (Kissmann and Edelkamp 2011) already went into the competition version of IPC 2011. We will look at further changes to the planner, including changes to the parser, to either always or never applying heuristic symbolic search, and including a list representation of the search frontier (rather than a matrix representation). Then we turn to filtering and lexicographic partitioning, resulting in new image operations for symbolic search.

The paper is structured as follows. First, we reconsider the explicit-state and symbolic heuristic search with pattern database heuristics. Next, we revisit the IPC 2011 competition outcome in the sequential-optimal track and turn to the set of refinements to GAMER that we experimented with. In the experimental results we show that we could advance the state-of-the-art in some of the planning benchmarks of IPC 2011. We close the paper with a discussion and give some concluding remarks.

Explicit-State Pattern Database Search

A *planning task* consists of variables of finite domain, so that states are assignments to the variables, an initial state, the goal, and a finite set of actions, each being a pair of preconditions and effects. In *cost-based planning*, actions are associated with action cost values, which often are integers. The task is to find a path, the *plan*, from the initial state to the goal. The plan is *optimal* if its cost is smallest among all possible plans. A *heuristic* is a mapping from states to natural numbers, and admissible if for all possible states the value is not greater than the cost of an optimal plan. A planning task *abstraction* is a planning task based on a mapping for the initial state, goal state as well as the actions.

The word *pattern* in the term pattern database (PDB) coined by Culberson and Schaeffer (1998) was inspired by a selection of tiles in the sliding-tile puzzle, and has been extended to the selection of state variables in other domains. More general definitions have been applied, shifting the focus from the mere selection of care variables to different state-space abstractions that are computed prior to the search. Following the definition in (Edelkamp and Schrödl 2012), a pattern database is characterized by memorizing an abstract state space, storing the shortest path distance from each abstract state to the set of abstract goal states.

Symbolic Pattern Database Search

The main limitation for applying pattern databases in search practice is the restricted amount of RAM. For the exploration of large state spaces, symbolic search using decision diagrams can save huge amounts of memory and computation time. State sets (Pang and Holte 2011) are represented and modified by accessing their characteristic functions.

Perfect hash functions (PHFs) to efficiently rank and unrank states have been very successful in traversing single-player problems like Rubik's Cube or the Pancake Problem

(Korf 2008) or two-player games like Awari (Romein and Bal 2002). They are also used for creating PDBs (Breyer and Korf 2010). The downside of the construction of PHFs for traversal algorithms like two-bit BFS is that they are problem-dependent.

Binary decision diagrams (BDDs) (Bryant 1985) are a memory-efficient data structure used to represent Boolean functions as well as to perform set-based search, where the BDD represents all binary state vectors that evaluate to true. More precisely, a BDD is a directed acyclic graph with one root and two terminal nodes (0 and 1), called sinks. Each internal node corresponds to a binary variable of the state vector and has two successors (low and high), one representing that the current variable is false and the other representing that it is true. For any assignment of the variables on a path from the root to a sink the represented function will be evaluated to the value labeling the sink. Moreover, BDDs are unique by applying the two reduction rules of (1) eliminating nodes with the same low and high successors and (2) merging two nodes representing the same variable that share the same low successor as well as the same high successor.

Symbolic search with BDDs takes two sets of variables, one (x) representing the current states and another (x') representing the successor states. To find the successors of a set of states S represented in the current state variables given a BDD T (the transition relation) for the entire set of actions, the *image* is computed, i. e., $image(S, x') = \exists x \ . \ S(x) \wedge T(x, x')$. Similarly, search in backward direction is done by using the *pre-image* operator (i. e., $pre\text{-}image(S, x) = \exists x'.S(x') \wedge T(x, x')$.

For *symbolic backward search* we start with the abstract goal set and iterate the computation of the pre-image until we encounter the initial state. Each state set in a layer is efficiently represented by a corresponding characteristic function. We may assume that the variable ordering is fixed and has been optimized prior to the search.

Symbolic PDBs (Edelkamp 2005) are PDBs that have been constructed symbolically as BDDs for later use either in symbolic or explicit heuristic search. Their construction exploits that the transition relation is defined as a relation. In contrast to the posterior compression of the state set (Ball and Holte 2008), the construction in (Edelkamp 2005) works on compressed representation, allowing much larger PDBs to be constructed. The savings observed by the symbolic representation are substantial for many planning domains.

For symbolic PDB construction in unit-cost graphs, backward symbolic BFS is used. For a given abstraction function the symbolic PDB $Heur(value, x)$ is initialized with the projected goal. As long as there are newly encountered states, we take the current backward search frontier and generate the predecessor list with respect to the abstracted transition relation. Then we attach the current BFS level to the new states, merge them with the set of already reached states, and iterate. When action costs are integers this process can be extended from breadth-first to cost-first levels, and it is possible to combine different symbolic heuristics by taking their maximum or by a controlled combination of their sum.

The variables encoded in *value* are often queried at the bottom or at the top (in which case we obtain the equivalent to a vector of BDDs). For symbolic heuristic search (Jensen, Veloso, and Bryant 2008) it is often more convenient to choose the latter, i. e., where the heuristic relation is partitioned into $Heur[0](x), \ldots Heur[\max_h](x)$, with

$$Heur(value, x) = \bigvee_{i=0}^{\max_h} (value = i) \wedge Heur[i](x).$$

BDDA* (Edelkamp and Reffel 1998) operates on a matrix *Open* of sets represented by BDDs. The successors of the BDD $Open_g[h]$ for a chosen transition with cost c are unified with the BDD $Open_{g+c}[h']$, where $h' \in \{0, \ldots, h_{max}\}$ is the partitioning obtained by the heuristic evaluation of the successor set. As we aim at *cost-optimal symbolic sequential planning* we work on a partitioning of the search space in g- and h-values, where g is the cost of the path traversed so far and h is the heuristic estimate on the cost to reach the goal. To guarantee optimal cost we expand the matrix along the f-diagonals with increasing g-values.

Competition Outcome

The IPC 2011 version of GAMER (Kissmann and Edelkamp 2011) already contained some improvements in the planner's code wrt. the IPC 2008 version, e. g., on handling large action costs (as in the PARC-PRINTER domain). The plan script was improved (it uses 15 minutes real-time for backward search, not CPU time). Moreover, it features the automated calculation of abstractions, and the improvement of the variable ordering. More precisely, the IPC 2011 competition version of GAMER contained two major improvements wrt. the IPC 2008 version.

Variable Ordering Heuristics The problem of finding a good variable ordering in a BDD is co-NP-complete, so that we decided to approximate another optimization problem to find a good variable ordering without BDDs. Following Kissmann and Edelkamp (2011) we incrementally compute the optimization function $\sum_{1 \leq i,j, \leq n, (u_i, v_j) \in D} (\pi(i) - \pi(j))^2$, where π denotes the applied permutation and D denotes the set of the causal dependencies.

Partial and Abstract PDBs A *partial PDB* (Anderson, Holte, and Schaeffer 2007) does not apply abstractions but truncates the exploration when it exhausts its allocated time slot. PDBs based on abstraction usually search for a number of variables on which the exploration is done.

The decision criterion we apply is to check whether backward exploration in the original space is too costly by looking at the CPU times for the backward exploration. If it rises too rapidly, we change from the partial PDB to the automated selection of variables for the *abstraction PDB*.

Our test-cases are benchmarks from IPC 2011. Unfortunately, in that competition GAMER did not score as well as it did in 2008 (the IPC 2011 results are depicted in Table 1). Of the twelve planners it finished ninth with only 148 solved instances, while one of the FAST DOWNWARD STONE SOUP versions (Helmert, Röger, and Karpas 2011) won with a total of 185 solutions. If we compare the number

Problem	FDSS-1	FDSS-2	SelMax	M&S	LM-cut	FD auto	ForkInit	BJOLP	LMFork	Gamer	IForkInit	CPT4
NoMystery	20	20	20	20	15	15	20	20	20	14	14	9
Parking	7	7	4	7	2	2	7	3	5	0	4	0
Tidybot	14	14	14	13	14	14	14	14	14	0	14	0
VisitAll	13	13	10	13	10	10	12	10	12	9	14	10
Total (unit-cost)	54	54	48	53	41	41	53	47	51	23	46	19
Barman	4	4	4	4	4	4	4	4	4	4	4	0
Elevators	18	17	18	11	18	18	16	14	14	19	14	0
Floortile	7	7	7	7	7	7	2	2	2	9	2	0
Openstacks	16	16	14	16	16	16	16	14	12	20	16	0
PARC-Printer	14	13	13	14	13	13	11	11	10	7	10	17
Peg-Solitaire	19	19	17	19	18	17	17	17	17	17	17	1
Scanalyzer	14	14	10	9	12	12	10	6	8	6	6	1
Sokoban	20	20	20	20	20	20	19	20	19	19	20	0
Transport	7	7	6	7	6	6	6	7	6	7	6	0
Woodworking	12	11	12	9	12	12	4	9	5	17	3	6
Total (costs)	131	128	121	116	126	125	105	104	97	125	98	25
Total (all)	185	182	169	169	167	166	158	151	148	148	144	44

Table 1: Number of solved problems for all domains of the sequential optimal track of IPC 2011, competition results.

of solved instances of the domains with and without action costs the results are quite peculiar. For the unit-cost domains Gamer found only 23 solutions; only one participant was worse than that. For those with action costs Gamer found 125 solutions; only three other planners were able to find more (with the maximum being 131).

Improvements to the Competition Version

We now present the improvements to optimal symbolic search in Gamer that were issued posterior to IPC 2011. Recall that this planner allocated 15 minutes for backward search, and the remaining time for forward search.

Solution Reconstruction After the competition we investigated the results in some detail and found several problems with Gamer. First of all, there was a small bug in the solution reconstruction for bidirectional BFS. It supposed that at least one forward and at least one backward step were performed. The two easiest problems of VisitAll require only a single step, so that the solution reconstruction crashed.

Parser Enhancements The parser we used was extremely slow. In some cases, parsing the ground input took more than 15 minutes, so that actually no search whatsoever was performed in the domains with action costs. First, the input was parsed in order to generate a PDB, the calculation of which was killed after 15 minutes, and then the input was parsed again for BDDA*. In the unit-cost domains the parsing sometimes also dominated the overall runtime. Thus, we switched to a parser generator for Java programs, with which the parsing typically takes no more than a few seconds.

One Computation of the Transition Relation In the most complex cases, generating the BDDs for the transition relation takes a long time. So far we had to generate them twice in case of domains with action costs if we did not use abstraction, once for the PDB generation and once for BDDA*. To omit this, we now store the transition relation BDDs, the BDD for the initial state and that for the goal condition on the hard disk; storing and reading them is often a lot faster than generating them from scratch.

Early Fail in Backward Search In two domains, namely Parking and Tidybot, we found that the first backward step takes too long, often even more than 30 minutes, so that the decision whether to use bidirectional or unidirectional BFS could not be finished before the overall time ran out. In these cases, the single images for all the actions were quite fast, but the disjunction took very long. Thus, during the disjunction steps we entered the possibility to check whether too much time, in this case 30 seconds, has passed. If it has we stop the disjunctions and the planner only performs unidirectional BFS (or PDB generation using abstractions). This enabled us to find some solutions in Tidybot, where we failed completely during the competition.

As the computation of the reachability set is done in compact form, invalid (unreachable) states may appear in the backward traversal. Due to the partial description of the goals there are many planning domains where the set of backward reachable states is much larger than the one in forward search. Consider the $n^2 - 1$-Puzzle with $N = n^2 - 1$ being the number of tiles and with the blank position not mentioned in the goal state. The inverse of planning actions that move a tile has the position of the blank and the tile to

Algorithm 1 Bidirectional BDD-Dijkstra: $\mathcal{A}$, $\mathcal{I}$, $\mathcal{G}$, w, T_a

```
$fClosed \leftarrow bClosed \leftarrow 0$
$fReach_0 \leftarrow \mathcal{I}$
$bReach_0 \leftarrow \mathcal{G}$
$g_f \leftarrow g_b \leftarrow 0$
$w_{total} \leftarrow \infty$
while $g_f + g_b < w_{total}$
  if $NextDirection = Forward$
    $\{g_1, \ldots, g_n\} \leftarrow fStep(fReach, g_f, w, fClosed, bClosed)$
    for all $g \in \{g_1, \ldots, g_n\}$
      for all $i \in \{i \mid i < g_b \wedge bReach_i \neq 0 \wedge g + i < w_{total}\}$
        if $fReach_g \wedge bReach_i \neq 0$
          $w_{total} \leftarrow g + i$
          update $\pi$
    $g_f \leftarrow g_f + 1$
  else // same in backward direction
  return $\pi$

Procedure $fStep(fReach, g, fClosed, bClosed)$
  $Ret \leftarrow \{\}$
  $fReach_g \leftarrow fReach_g \wedge \neg fClosed$
  for all $c \in \{1, \ldots, C\}$
    $Succ(x') \leftarrow \bigvee_{a \in \mathcal{A}, w(a)=c} \exists x. fReach_g \wedge T_a(x, x')$
    if $Succ \wedge bClosed \neq 0$
      $Ret \leftarrow Ret \cup \{g + c\}$
    $fReach_{g+c} \leftarrow fReach_{g+c} \vee Succ$
  $fClosed \leftarrow fClosed \vee fReach_g$
  return $Ret$
```

be moved in the precondition, and the exchange of tile and blank in the effects. Since the blank position is not known to the planner, backward exploration will generate states with tiles on top of each other, so that with N^N the set of backward reachable states is exponentially larger than the number of $N!/2$ forward reachable states.

Matrix BDDA* Search In the competition version we used bidirectional BDD-BFS for unit-cost domains. Afterward, we tried running BDDA* in all cases, no matter if we are confronted with costs or not. We call the resulting approach *Matrix BDDA**.

Bidirectional Shortest Path BDD Search As another extreme, we implemented bidirectional shortest path search on domains with action costs (cf. Algorithm 1). The motivation for this was that for PDB generation the domains are often not abstracted, so that the heuristic corresponds to a perimeter around the goal. In those cases bidirectional blind search is more flexible, since the A* version uses a fixed timeout for the backward exploration to generate the PDBs, and the remainder of the available time in the forward exploration, while the bidirectional search is able to select whether to perform a backward or a forward step at any time.

The algorithm takes a set of actions $\mathcal{A}$, the initial state $\mathcal{I}$, the goal states $\mathcal{G}$, the action costs w and the transition relation T_a in form of a BDD for each action $a \in \mathcal{A}$ as input. The forward search starts at $\mathcal{I}$, the backward search at $\mathcal{G}$. Procedure $fStep$ perform a forward step, which removes the already expanded states from the bucket to be expanded of the open list $fReach$. Then it computes the image to find the set of successor states and inserts them into the correct buckets of $fReach$. In case of a backward search the algorithm looks the same, only that the forward and backward sets are swapped and pre-images instead of images are applied.

The stopping criterion in the bidirectional BDD version of Dijkstra's (1959) algorithm is not immediate, as individual shortest paths for the states cannot be maintained. Fortunately, in the context of external search the following criterion has been established (Goldberg and Werneck 2005): *stop the algorithm when the sum of the minimum g-values of generated states for the forward and backward searches is at least w_{total}, the total cost of the cheapest plan found so far.* They have shown that this stopping condition is correct. Since the g-value for each search is monotone in time, so is their sum.

After the condition is met, every state s removed from a priority queue will be such that the costs of the plans from $\mathcal{I}$ to s and from s to $\mathcal{G}$ will be at least w_{total}, which implies that no plan of cost less than w_{total} exists.

List BDDA* Search When a state set is expanded in BDDA* all the successors have to be classified according to their h-values by applying conjunctions with all the heuristic BDDs. When the number of heuristic values grows, this can be inefficient since some of these conjunctions could be avoided. The representation in the matrix can be simplified to a vector for the states in the *Open* list ordered along the g value (serving as the *Unclassified* list). The reasoning behind this strategy is to defer the heuristic calculation by computing the conjunction of the successor set with the heuristic estimate only when it is needed for expansion in the currently traversed f-diagonal. We call the resulting algorithm *List BDDA**. Additionally, while Matrix BDDA* uses BFS to get the states reachable with 0-cost actions independent of their actual h-values, in the list version we apply a conjunction with the heuristic value to get only those states in the current f-diagonal.

Filtering

In many explicit-state planners there are speed-up techniques for filtering the operators that match, such as the successor generators used in FAST DOWNWARD (Helmert 2006). There, operators are organized in a tree (see Figure 1). Leaf nodes of the transition tree contain a set of operators with the same preconditions. Every internal node is associated with a variable v and splits the operators with respect to their precondition for v, one child for every possible value of v and an additional one for the cases that are independent of v.

We found that the approach carries over to BDD search as follows. As all the variables are binary, each internal node has three children, dividing the operators depending if they have v or $\neg v$ as precondition are have a precondition inde-

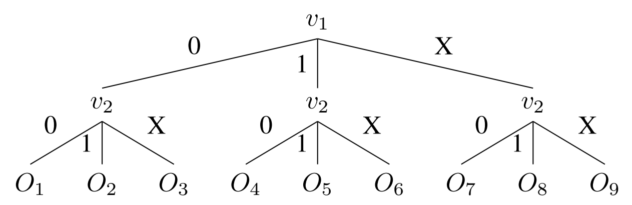


Figure 1: Improving image computation in domains with many operators using a transition tree.

Algorithm 2 Image using the transition tree

```
if node is leaf:
    return $\bigvee_{a \in node.A} \exists x \,.\, S(x) \wedge T_a(x, x')$
else
    $v \leftarrow$ node.variable
    $r_0 \leftarrow image(node.c0, bdd \wedge \neg v)$
    $r_1 \leftarrow image(node.c1, bdd \wedge v)$
    $r_x \leftarrow image(node.cx, bdd)$
    return $r_0 \vee r_1 \vee r_x$
```

pendent of v. When computing the successor states, instead of applying the operators over a single state we need to compute the successors of a state set represented as a BDD. Algorithm 2 shows the algorithm to compute the image of a BDD using the transition tree structure. For each internal node three recursive calls are applied, one for each of the child nodes. In the leaf nodes we compute the image using the transition relation of the operators in the node.

Using this kind of transition tree brings two advantages over the previous approach of just computing the disjunction of the result of the images for every transition. On the one hand, all the operators regarding branches without any state are ignored, reducing the total number of images. On the other hand, if several operators have the same precondition, the conjunction of the state set with that precondition is only computed once, reducing the size of the BDDs for which we are computing the images.

Partitioning

BDD-BFS is widely known to improve the memory profile of the search in many planning problems. If the BDDs are still growing too quickly we apply a partitioning option which we call lex-partitioning. Given a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$, defined over n inputs $X_n = \{x_1, \ldots, x_n\}$, the *lex-partitioned BDD* representation of f is a set of k assignments $a_1 \ldots, a_k \in \{0,1\}^n$ and k functions $f_1, \ldots, f_k : \{0,1\}^n \rightarrow \{0,1\}$ that are also defined over X_n and satisfy the following conditions.

1. f_i are represented as BDDs respecting the same variable ordering as f, for $1 \leq i \leq k$.
2. $a_k = (1, \ldots, 1)$ and, for all $i < k$, we have $a_i <_{lex} a_{i+1}$.
3. $f_1 \vee \ldots \vee f_k = f$.
4. $f_i \wedge f_j = 0$ for all $i \neq j$.
5. $f_1 = f \wedge \bigvee_{a \leq_{lex} a_1} a$ and $f_i = f \wedge \bigvee_{a_{i-1} <_{lex} a \leq_{lex} a_i} a$, for all $1 < i \leq k$.

The advantage is that by the lexicographical ordering we obtain control over the state set size (*satcount*) resulting from a split. More formally, given the BDD G_f and any assignment $a \in \{0,1\}^n$, the binary lexicographic *split* function computes the BDDs G_g and G_h with the satisfying sets $S_g = \{b \in \{0,1\}^n \mid b \leq_{lex} a\}$ and $S_h = \{b \in \{0,1\}^n \mid b >_{lex} a\}$. The recursive procedure partitions the original BDD into two by splitting nodes on the path defined by s into a left and right representative and setting links to 0. If nodes are missing they are allocated and split accordingly.

Theorem 1 (Time Complexity Split Function)**.** *In a shared BDD representation, given BDD G_f that has been annotated with* satcount*-values at each node and provided an assignment $a \in \{0,1\}^n$ the lexicographic split function computes the BDDs G_g and G_h in $O(n)$ time. Moreover, at most $2n$ nodes are created.*

Proof. As at most $O(n)$ nodes are processed in post-order, the time complexity follows. All original nodes remain valid in the shared representation and each new node that is created in the bottom-up traversal is checked for applying BDD reduction rules (by issuing a look-up in the unique table). In this process at most $2n$ new nodes are created. □

It is not difficult to extend the splitting strategy to a k-fold split that produces k equally-sized sets represented as BDDs and runs in $O(kn)$ time.

Algorithm 3 Fold-BDD-DFS: $\mathcal{A}, \mathcal{I}, \mathcal{G}, T_a, M$

```
$Open_0 \leftarrow \mathcal{I}$
for all $g = 0, 1 \ldots$
    if ($explore(g, Open)$) return

Procedure $explore(g, Open)$
    if ($S \wedge \mathcal{G} \neq 0$) return true
    if ($satcount(Open_g) > M$)
        $Open_g[1], \ldots, Open_g[p] \leftarrow split(Open_g)$
    else
        $Open_g[1] \leftarrow Open_g$
    for all $p \in Open_g$
        $S \leftarrow Open_g[p]$
        for all $i = 1, \ldots, C$
            $Succ_i(x') \leftarrow \bigvee_{a \in \mathcal{A}, w(a)=i} \exists x \,.\, S(x) \wedge T_a(x, x')$
            $Open_{g+i} \leftarrow Open_{g+i} \vee Succ_i$
        if $explore(g+1, Open)$
            return true
    return false
```

Fold BDD-DFS To participate in the partition of state sets that exceeds some predefined memory threshold M, we devise a generic symbolic search algorithm that is shown in Algorithm 3. It refers to an *Open*-list that is already partitioned along the g-value. We assume the initial and goal states represented in form of characteristic functions $\mathcal{I}$ and $\mathcal{G}$. For the sake of simplicity, we assume unit-cost transition relations T_a, one for each action $a \in \mathcal{A}$. If the maximum p in $Open_g[p]$ is 1, then we are in the unpartitioned case, otherwise $Open_g[p]$, $1 \leq p \leq k$, denotes its partition.

The algorithm has interesting special cases. For $M = \infty$, Fold BDD-DFS resorts to BDD-BFS. For $M = 1$, Fold BDD-DFS resorts to explicit-state DFS.

Note that deleting the sets in the partitions may hinder efficient solution reconstruction, so that we save the expanded state sets. This also allows full duplicate detection.

Theorem 2 (Fold BDD-DFS)**.** *Let V be the induced state space of a planning problem. When applying the lexicographic folding with full duplicate detection, Fold BDD-DFS finds a solution if it exists and requires at most $O(|V|/M)$ splits.*

Proof. Full duplicate detection implies the completeness of Fold BDD-DFS. We have $|V|$ states in total and each split reduces this set by at least M states, so that we have at most $O(|V|/M)$ splits. □

It is not difficult to extend the algorithm to support iterative deepening and finally return an optimal plan.

Fold BDDA* When combining the partitioning approach with List BDDA* we call the resulting procedure *Fold BDDA** (see Algorithm 4). Fold BDDA* applies lex-partitioning on the f-diagonal. Due to its depth-first strategy it has the advantage to find solution paths on the final f-diagonal faster. Of course, all state sets with f-value smaller than the optimal one have to be expanded completely.

To preserve optimal solutions we apply partitioning only on states with common f-value, i. e., that are located on the same f-diagonal. If a set $Open_g[h]$ is partitioned we switch from breadth-first minimum g-wise ordering to depth-first maximum g-wise ordering, so that successor partitions have all been worked upon when the next partition of a BDD is looked at. If all partitions have been handled, the next possible g-value on the current f-diagonal is processed.

Theorem 3 (Optimality Fold BDDA*)**.** *Fold BDDA* is optimal, i. e., it returns the optimal solution cost when encountering its first goal state.*

An example of the algorithm is provided in Figure 2. The state sets in buckets 7 and 8 are too large and split into two. The tree is traversed in depth-first order and the exploration may terminate at a goal each time bucket 10 is reached.

The algorithm relates to Beam-Stack Search (Zhou and Hansen 2005). However, instead of monitoring different f-values that lead to larger plateaus, here we use the lexicographic ordering with sharp boundaries.

Algorithm 4 Fold-BDDA*: $\mathcal{A}, \mathcal{I}, \mathcal{G}, w, Heur_h, T_a, M$

```
$Unexplored_0 \leftarrow \mathcal{I}$
**for all** $f = 0, \ldots$
  **for all** $g = 0, \ldots, f$
    $Open_g \leftarrow Unexplored_g \wedge Heur_{f-g}$
  **if** ($exploreDiagonal(f, g, Open)$) **return**

**Procedure** $exploreDiagonal(f, g, Open)$
  **if** ($g > f$) **return false**
  $h \leftarrow f - g$
  **if** ($h = 0$) **and** ($S \wedge \mathcal{G} \neq 0$) **return true**
  **if** ($satcount(Open_g) > M$)
    $Open_g[1], \ldots, Open_g[p] \leftarrow split(Open_g)$
  **else**
    $Open_g[1] \leftarrow Open_g$
  **for all** $p \in Open_g$
    $S \leftarrow Open_g[p]$
    **for all** $i = 1, \ldots, C$
      $Succ_i(x') \leftarrow \bigvee_{a \in \mathcal{A}, w(a)=i} \exists x \,.\, S(x) \wedge T_a(x, x')$
      $Open_{g+i} \leftarrow Open_{g+i} \vee (Succ_i \wedge Heur_h)$
      $Unexplored_{g+i} \leftarrow Unexplored_{g+i} \vee (Succ_i \wedge \neg Heur_h)$
    **if** $exploreDiagonal(f, g+1, Open)$
      **return true**
  **return false**
```

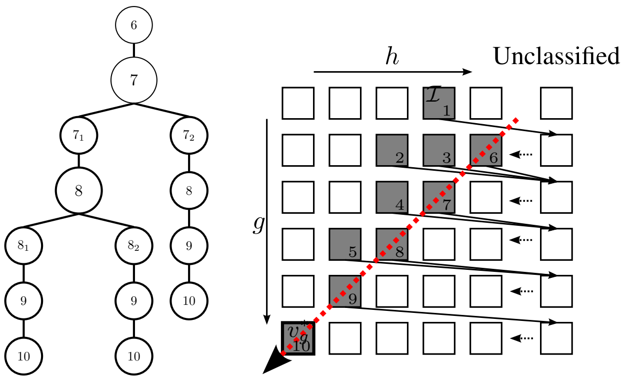
Experiments

We implemented the refinements in GAMER (Kissmann and Edelkamp 2011) using the CUDD library of Fabio Somenzi (compiled for 64-bit Linux using the GNU gcc compiler, optimization option -O3). For the experiments we used our own machine (Intel Core $i7$ 920 CPU with 2.67 GHz and 24 GB RAM) with the same settings concerning timeout (30 minutes) and maximal memory usage (6 GB) as in the competition. For the experiments we used the software infrastructure from the resources of IPC 2011.

The results are depicted in Table 2. The different versions are the competition version (IPC11), the one with all improvements except for A* search in unit-cost domains enabled (Post-IPC11), and the final Matrix A*, List A*, and Fold A* versions. Those versions using the transition tree extension are denoted by suffix (TT). The last version is the bidirectional BDD-Dijkstra.

We see that on our computer the competition version solves two fewer problems than in IPC 2011, highlighting that our computer is not stronger than the one used in the competition. All the small improvements helped mainly in the unit-cost domains. There we are now able to find the two trivial solutions in VISITALL, as well as six solutions in TIDYBOT. In the domains with action costs the new parser helped us to find three additional solutions in the SCANALYZER domain.

Switching from BFS to BDDA* in case of unit-cost domains we see only a small improvement: for PARKING we find one solution. Overall, the Matrix A* solves 158 problems, which is 12 problems more than with the competition version. List A* and Fold A* are performing well in the domains with action cost, finding two more solutions than Matrix A*, while they are slightly worse in the unicost domains, losing two solutions. Overall, all three approaches find the same number of solutions. When comparing the runtimes of these three approaches (Figure 3) we see



(a) Partition (b) Last diagonal

Figure 2: Example of the Fold A* Algorithm (with DFS tree for last diagonal and corresponding bucket layout).

that overall surprisingly Matrix A* is the fastest, especially in NoMystery, Tidybot, VisitAll and Scanalyzer, while List A* is slightly faster than Fold A*.

Concerning the use of the transition tree, the overall outcome of Matrix A* and Fold A* is decreased to 154 resp. 155 solutions, while that of List A* is increased to 159. When comparing the influence of the transition tree to the runtime the results suggest that it is highly domain dependent (Figure 4). For some domains, such as Elevators, Floortile, and Sokoban, the runtime using the transition tree is higher in most cases than when not using it, but for others (e. g.,, in Scanalyzer, Transport, and Woodworking) the runtime decreases.

Finally, the results of bidirectional BDD-Dijkstra are rather astonishing. In the unit-cost domains it performs exactly the same as bidirectional BDD-BFS, which is to be expected. In those with action costs it finds the most solutions in the Barman, Floortile, PARC-Printer, and Transport domains, and in the others it finds the same number of solutions as Matrix A* without the transition tree, resulting in 164 solutions in total, 133 of those in the domains with action costs. Compared to the other planners of the actual competition in Table 1 we see that with 133 solved problems in the domains with action costs this approach is best among all participants. Moreover, if we were to exclude the PARC-Printer domain, which is special due to the very diverse and extremely high action costs, the picture would be even more fortunate for bidirectional BDD-Dijkstra. Thus, it seems that blind symbolic search works better than the much more informed explicit state heuristic search planners in the domains with action costs, while in the unit-cost domains the latter are still ahead.

Conclusion

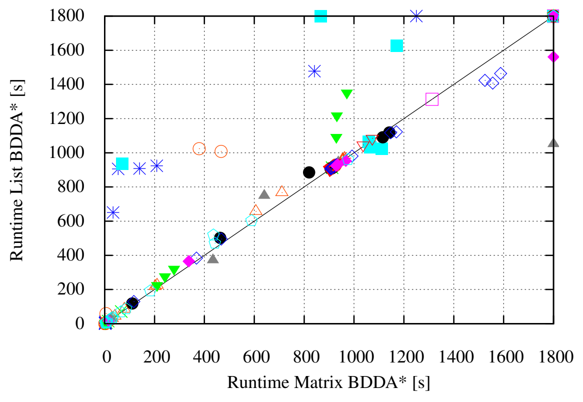
According to the outcome of the last two IPCs, heuristic and symbolic search are two leading methods for sequential optimal planning. Symbolic search is effective in exploring large state sets, while explicit-state heuristics are often more informed. The outcome of IPC 2011 suggested a clear advantage for heuristic explicit-state search. We have shown that symbolic search planners can be competitive with state-of-the-art planners, at least in cost-based domains.

	IPC11	Post-IPC11	Matrix A*	List A*	Fold A*	Matrix A* (TT)	List A* (TT)	Fold A* (TT)	B-Dijkstra
NoMystery (✳)	14	14	14	13	13	14	14	14	14
Parking (⊡)	0	0	1	1	1	1	1	1	0
Tidybot (■)	0	6	6	5	5	4	4	4	6
VisitAll (⊙)	9	11	11	11	11	11	11	11	11
Total (unit-cost)	23	31	32	30	30	30	30	30	31
Barman (+)	4	5	4	4	4	4	7	4	8
Elevators (●)	19	19	19	19	19	19	19	19	19
Floortile (×)	8	9	8	8	8	8	8	8	9
Openstacks (△)	20	20	20	20	20	20	20	20	20
PARC-Printer (▲)	7	7	7	8	8	7	7	8	8
Peg-Solitaire (▽)	17	17	17	17	17	17	17	17	17
Scanalyzer (▼)	6	9	9	9	9	9	9	9	9
Sokoban (◇)	19	19	19	19	19	16	17	16	19
Transport (◆)	7	7	7	8	8	8	8	8	8
Woodworking (⬠)	16	16	16	16	16	16	17	16	16
Total (costs)	123	128	126	128	128	124	129	125	133
Total (all)	146	159	158	158	158	154	159	155	164

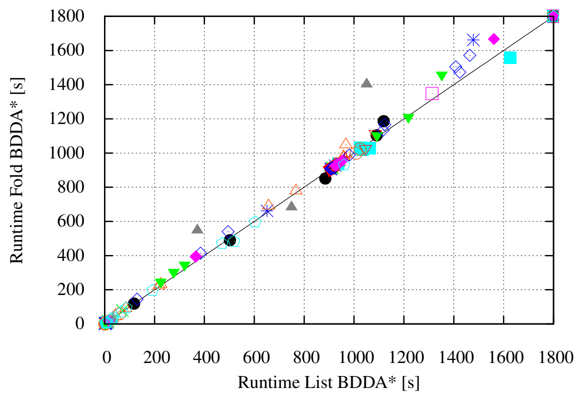
Table 2: Number of problems solved by Gamer for the sequential optimal track of IPC 2011, own results.

While the competition version of Gamer used bidirectional blind search for unit-cost domains and BDDA* for the rest, we have compared the performance of both approaches separately. This comparison shows advantages for the bidirectional blind search. Surprisingly, after many years of research on finding refined heuristics in the AI planning domain this form of blind search still outperforms all existing planners, at least on the IPC 2011 domains with action costs. We also tried some improvements in the A* implementation, such as the List and Fold versions of BDDA* and the transition tree for the successor generation, but experimental results show small impact on the number of solved problems.

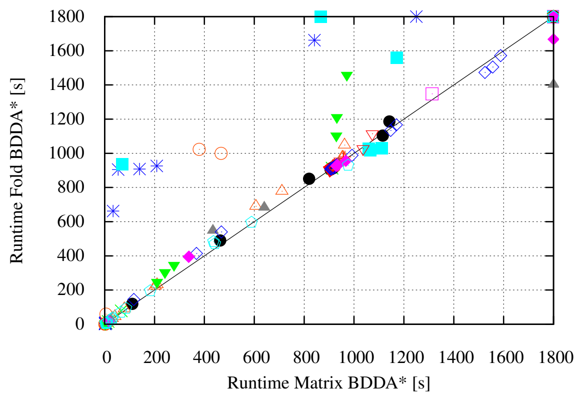
The motivation of the partitioning pointing towards future work is that explicit search can be more space-efficient if perfect hash functions are available. With ranking and unranking as proposed by Dietzfelbinger and Edelkamp (2009) we can eventually connect a symbolic state space representation with BDDs and an explicit bitvector based exploration. The BDDs can serve as a basis for a linear-time ranking and unranking scheme. As we have control over the number of states in a BDD we can switch between symbolic and explicit state space generation when the main memory available is sufficient to cover the partitioned state sets. In other words, for the *explicification* of the search we provide a combination of the two methods, where the BDDs are used to define hash functions for addressing states in the bitvector representation of the state space. This may have implications to other AI applications. For example, in game playing we think of a layered approach to perform forward search with a BDD, and retrograde analysis that changes from symbolic to explicit state representation find strong solutions.



(a) List BDDA* vs. Matrix BDDA*.

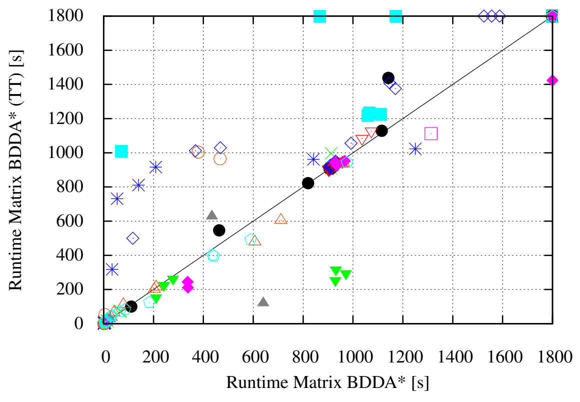


(b) Fold BDDA* vs. List BDDA*.

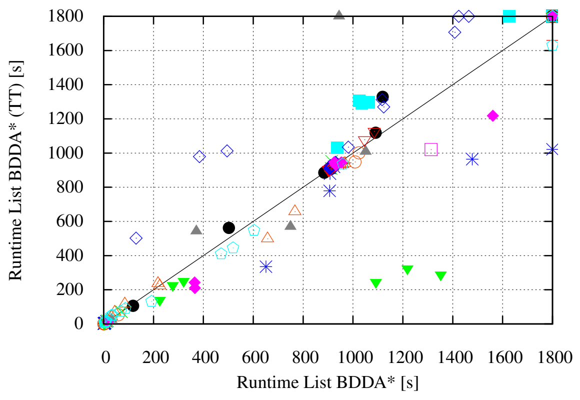


(c) Fold BDDA* vs. Matrix BDDA*.

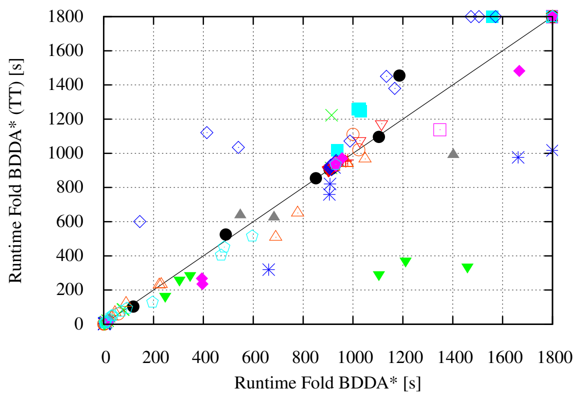
Figure 3: Comparing the runtime results of different BDDA* versions without the transition tree. The keys are omitted for better readability. The symbols correspond the domains with the same symbol in Table 2.



(a) Matrix BDDA*.



(b) List BDDA*.



(c) Fold BDDA*.

Figure 4: Comparing the runtime results of different BDDA* versions with and without the transition tree. The keys are omitted for better readability. The symbols correspond the domains with the same symbol in Table 2.

Acknowledgments

Thanks to Deutsche Forschungsgesellschaft (DFG) for support in the project ED 74/11-1. Thanks to the Spanish Government for support in the MICINN projects TIN2011-27652-C03-02, TIN2008-06701-C03-03 and to the Comunidad de Madrid for support in the project CCG10-UC3M/TIC-5597.

References

Anderson, K.; Holte, R.; and Schaeffer, J. 2007. Partial pattern databases. In *SARA*, volume 4612 of *LNCS*, 20–34. Springer.

Ball, M., and Holte, R. C. 2008. The compression power of symbolic pattern databases. In *ICAPS*, 2–11.

Bonet, B., and Geffner, H. 2008. Heuristics for planning with penalties and rewards formulated in logic and computed through circuits. *Artif. Intell.* 172(12–13):1579–1604.

Breyer, T. M., and Korf, R. E. 2010. 1.6-bit pattern databases. In *AAAI*, 39–44.

Bryant, R. E. 1985. Symbolic manipulation of boolean functions using a graphical representation. In *DAC*, 688–694.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Dietzfelbinger, M., and Edelkamp, S. 2009. Perfect hashing for state spaces in BDD representation. In *KI*, 33–40.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271.

Edelkamp, S., and Reffel, F. 1998. OBDDs in heuristic search. In Herzog, O., and Günter, A., eds., *KI*, volume 1504 of *LNCS*, 81–92. Springer.

Edelkamp, S., and Schrödl, S. 2012. *Heuristic Search – Theory and Applications*. Academic Press.

Edelkamp, S. 2005. External symbolic heuristic search with pattern databases. In *ICAPS*, 51–60.

Goldberg, A. V., and Werneck, R. F. F. 2005. Computing point-to-point shortest paths from external memory. In *Workshop on Algorithm Engineering and Experiments and Workshop on Analytic Algorithmics and Combinatorics, ALENEX /ANALCO*, 26–40.

Helmert, M.; Röger, G.; and Karpas, E. 2011. Fast downward stone soup: A baseline for building planner portfolios. In *ICAPS-Workshop on Planning and Learning (PAL)*.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Jensen, R. M.; Veloso, M. M.; and Bryant, R. E. 2008. State-set branching: Leveraging BDDs for heuristic search. *Artif. Intell.* 172(2–3):103–139.

Kissmann, P., and Edelkamp, S. 2011. Improving cost-optimal domain-independent symbolic planning. In *AAAI*, 992–997.

Korf, R. E. 2008. Minimizing disk I/O in two-bit breadth-first search. In *AAAI*, 317–324.

Pang, B., and Holte, R. C. 2011. State-set search. In *SOCS*.

Romein, J. W., and Bal, H. E. 2002. Awari is solved. *International Computer Games Association (ICGA) Journal* 25(3):162–165.

Zhou, R., and Hansen, E. A. 2005. Beam-stack search: Integrating backtracking with beam search. In *ICAPS*, 90–98.

A Multi-Agent Extension of PDDL3.1

Daniel L. Kovacs
Budapest University of Technology and Economics
Budapest, HUNGARY
dkovacs@mit.bme.hu

Abstract

Despite a recent increase of research activity in the field of multi-agent planning there is still no de-facto standard for the description of multi-agent planning problems similarly to the Planning Domain Definition Language (PDDL) in case of deterministic single-agent planning. For this reason, in this paper a multi-agent extension of the currently latest official version of PDDL (3.1) is proposed together with a corresponding multi-agent planning track for the International Planning Competition (IPC). Our aim is to allow for a more direct comparison of planning systems and approaches, a greater reuse of research, and a more coordinated development in the field. Multi-agent planning is fundamentally different from the single-agent case with a broad range of applications (e.g. multi-robot domains). Not only is it inherently harder because of an exponential increase of the number of actions in general, but among others also constructive/destructive synergies of concurrent actions, and agents' different abilities and goals may need to be considered. The proposed multi-agent extension copes with these issues and allows planning both for and by agents even in temporal, numeric domains. It implies minimal changes to the syntax of PDDL3.1 and the related parsers.

1. Introduction

Multi-agent planning (de Weerdt and Clement 2009) is about planning by N planning agents for M executing agents (or actors, actuators, bodies) situated in a multi-agent environment, with a broad range of applications. Planning and executing agents may be the same or separate entities. Executing agents are always situated in the environment, while planning agents may be external to it. However the most typical scenario is either when an external agent is planning for a group of situated agents ($N = 1, M > 1$), or when there is a group of autonomous, situated planning-and-executing agents ($N = M > 1$). In general four cases can be distinguished (cf. Table 1).

In cases where $M > 1$ the control of multiple executing agents may be *centralized* or *decentralized*. A typical case of *distributed* planning is when $N > 1$ agents plan for $M = 1$ agent. Planning can be done *on-line* or *off-line*, and agents and environments can correspond to types mentioned in Chapter 2 in (Russell and Norvig 2010).

N \ M	=1	>1
=1	single-agent planning for a single agent	single-agent planning for multiple agents
>1	multi-agent planning for a single agent	multi-agent planning for multiple agents

Table 1: a general categorization of multi-agent planning

Multi-agent planning is inherently harder than single-agent planning because agents may act independently and thus the number of possible actions in general is exponential (combinations of individual actions need to be considered). Moreover agents may be heterogeneous; they may have different abilities, contradicting goals or asymmetric beliefs; they may require coordination of plan execution, communication or synchronization of concurrent actions; constructive/destructive interference of joint actions may arise (joint actions may produce different effects from the union of effects of their parts); cooperation and self-interest, goals of teams and individuals may need to be conciliated; and the level of coupling between agents is also important. Thus single-agent planning can't be directly applied to multi-agent planning problems.

Research in the field of multi-agent planning was focusing recently mainly on the following topics.

- *scaling up the performance* of planners, e.g. (Shah, Conrad, and Williams 2009; Stefanovitch et al. 2011; Jonsson and Rovatsos 2011; Kumar, Zilberstein, and Toussaint 2011; Spaan, Oliehoek, and Amato 2011);
- coping with *more realistic domains*, e.g. (Beaudry, Kabanza, and Michaud 2010; Pajarinen and Peltonen 2011; Zhuo and Li 2011; Wang and Botea 2011; Fox, Long, and Magazzeni 2011);

- *improving solution quality*, e.g. (Yabu, Yokoo, and Iwasaki 2009; Marecki and Tambe 2009);
- *exploiting problem structure* (Brafman and Domshlak 2008; Nissim, Brafman, and Domshlak 2010);
- utilizing *learning*, e.g. (Martins and Demiris 2010; Zhuo, Muñoz-Avila, and Yang 2011);
- reasoning about *agents' knowledge*, e.g. (Baral et al. 2010; Baral and Gelfond 2011);
- and addressing *agents' self-interest*, e.g. (Brafman et al. 2009; Crosby and Rovatsos 2011).

The problem is that despite all this progress there is still no standard description language for multi-agent planning problems allowing a more direct comparison of systems and approaches and a greater reuse of research similarly to the Planning Domain Definition Language (PDDL) (McDermott et al. 1998) in single-agent planning, a base language of the International Planning Competition (IPC).

Naturally there were some previous approaches (cf. Section 2.2), but none of these languages became de-facto standards probably partly because of their limitations. On the other hand PDDL is not enough to describe multi-agent planning problems in general (e.g. possibly different goals and utilities of different agents, synergy of joint-actions).

To address these issues, in this paper a *multi-agent extension of PDDL3.1* is proposed, which is currently the latest official version of PDDL (Helmert 2008), and based on this extension, ideas for a corresponding *multi-agent planning track* are also proposed for the upcoming IPCs.

The structure of the paper is as follows: after Section 1 discusses the motivation behind the proposed approach, Section 2 examines its background; Section 3 presents the main result of the paper, the formal syntax and informal semantics of the proposed multi-agent extension of PDDL3.1 and an example; Section 4 discusses ideas for a multi-agent planning track at the upcoming IPCs based on the proposed extension; finally Section 5 concludes the work and outlines some directions for future research.

2. Background

This section examines some of the considerations and decisions behind the proposed multi-agent extension of PDDL3.1. Namely it discusses **(1)** some minor corrections of PDDL3.1's syntax, **(2)** previously published multi-agent planning problem description languages, and **(3)** requirements of an appropriate multi-agent extension.

2.1. Corrections of PDDL3.1's syntax definition

Since PDDL3.1 was chosen as the basis of the multi-agent extension, a complete and correct BNF (Backus-Naur Form) definition of its syntax becomes necessary, which was made available in (Kovacs 2011). It makes mainly the following minor corrections to previously published BNF.

- The default type (of objects) in PDDL is `object`, but until now this was not made explicit in the grammar. Accordingly the next rule should be **added** to the BNF.

```
<primitive-type> ::= object
```

- Similarly the definition of the built-in 2-ary = predicate in case of the `:equality` requirement was also left out from previous definitions of PDDL. To correct this, the following rule needs to be **added** to the BNF.

<atomic formula(*t*)> ::=$^{\text{:equality}}$ (= *t* *t*)

- Since PDDL2.1 (Fox and Long 2003) function-expressions in the domain description allowed only 2-argument numeric operators, although in the problem description, in the definition of `metric` they could be also multi-argument. To fix this, the following two production rules should be **added** to the grammar.

<f-exp> ::=$^{\text{:numeric-fluents}}$
        (<multi-op> <f-exp> <f-exp>$^{+}$)
<f-exp-da> ::=$^{\text{:numeric-fluents}}$
        (<multi-op> <f-exp-da> <f-exp-da>$^{+}$)

- The definition of non-terminals `<name>` and `<number>` was underspecified until now, so it is suggested to **define** them more precisely, for example as shown in the Appendix in (Teichteil-Königsbuch 2008).
- The following rule would allow durative actions to have *non temporally annotated* numeric effects, which would contradict the specification of durative actions in (Fox and Long 2003). Thus it needs to be **deleted**.

<da-effect> ::=$^{\text{:numeric-fluents}}$
        (<assign-op> <f-head> <f-exp-da>)

- The following rule is present in the BNF of PDDL2.1 and PDDL3.0 (Gerevini and Long 2005). The problem with it is that `<a-effect>` is not defined anywhere, so `<a-effect>` should be **changed** to `<cond-effect>`. Otherwise `<p-effect>` or `<effect>` may also be considered, but the former would not allow conjunctions of propositions, while the latter would overly complicate the syntax and allow semantically ambiguous constructs (e.g. nested conditional effects).

<timed-effect> ::=
        (at <time-specifier> **<a-effect>**)

- Production rules for non-terminals `<assign-op-t>` and `<f-exp-t>` are referenced, but missing from the BNF since PDDL3.0 in the form they were given in the BNF of PDDL2.1. They should be **included** again.
- The following rule defines the syntax of derived predicates since PDDL2.2 (Edelkamp and Hoffmann 2004a). The problem with it is that there is no mention of the name of the derived `<predicate>`. Thus instead of `<typed list (variable)>` (Edelkamp and Hoffmann 2004b) would suggest `<atomic formula(term)>`, which is better, since it includes the

name of the derived predicate, but then there are no argument-types in the head of the derived-rule as one might expect in case of `:typing`. To include both the name of the predicate and the type of its arguments `<atomic formula skeleton>` should be **used instead** of `<typed list (variable)>` below.

```
<derived-def> ::=
     (:derived <typed list (variable)> <GD>)
```

- The following rule is present in the initial conditions part of the problem description since PDDL2.1, but it is incorrect, since `<f-head>` may be lifted, although it should be grounded. To fix this `<basic-function-term>` can be **used instead** of `<f-head>` below.

```
<init-el> ::=:numeric-fluents
                   (= <f-head> <number>)
```

- PDDL3.0 introduced plan constraints via modal operators at the 5th IPC in 2006, but they were not allowed to be nested at the time of the competition. Nonetheless this restriction could be lifted by using production rules provided in Section 3 in (Gerevini and Long 2005). The problem with those rules, which are still part of the BNF, is that they do not allow a normal end to the recursive nesting of modal operators. This needs to be **corrected**, e.g. as given in (Kovacs 2011).

2.2. Previous approaches

Previous multi-agent planning problem description languages provided valuable experience and ideas for the design of the proposed multi-agent extension of PDDL3.1. In the following an overview of these languages is given.

2.2.1. Non-deterministic Agent Domain Language

The Non-deterministic Agent Domain Language (NADL) introduced in (Jensen and Veloso 2000) is suitable for describing multi-agent planning domains to a limited extent. It could be seen a predecessor of numeric fluents of PDDL2.1, but its syntax differs significantly from PDDL.

In NADL each explicitly given agent is a collection of actions that have preconditions and effects (numeric and/or propositional formulas). Actions can also refer to state variables they constrain. These constraints are then used in planning time to avoid joint actions that have destructive synergetic effects, i.e. which constrain an overlapping set of state variables (e.g. actions that assign different values to a numeric fluent). Constructive interferences on the other hand are not modeled. This makes for a relatively simple model of interactions among concurrent actions.

However NADL allows a distinction between system and environment agents, the latter being non-controllable and thus responsible for possible non-deterministic effects.

NADL's model of time is discrete. Actions have equal duration and each agent can execute only one action at a time. All agents share the same goal. Later in (Bowling, Jensen, and Veloso 2002) this was extended to multiple agents having possibly different goals, but no accompanying description language was provided, and the model was applicable only to propositional domains.

2.2.2. Concurrent interacting actions in STRIPS

This multi-agent extension of STRIPS (Boutilier and Brafman 2001) provides a more elaborate way to model interactions of concurrent actions than NADL based on the idea of *concurrent action lists*. Essentially the same (but a bit simplified) idea is presented in Section 11.4.1 in (Russell and Norvig 2010). Concurrent action lists refer to state variables and concurrently executed actions in form of separate lists attached to actions' preconditions or to conditional effects' conditions. In Section 2.2 in (Boutilier and Brafman 2001) they are described precisely as follows.

> If an action schemata A' appears in the concurrent action list of an action A then an instance of schema A' must be performed concurrently with action A in order to have the intended effect. If an action schema A' appears negated in the concurrent action list of an action A then no instance of schema A' can be performed concurrently with action A if A is to have the prescribed effect.

This is a generic and intuitive way to model interference of concurrent actions, however the implicit quantifiers over actions are a bit restrictive and the scope of quantification (the whole list) is also a bit broad. This could be improved by having explicit quantification, and including reference to concurrent actions directly in (pre)conditions.

Agents responsible for the execution of actions are referred to in form of variables (always the first parameter of an action). The only issue with this is that there is no typing, and thus no distinction between agents and objects. Effectively every object can be considered an agent (e.g. a planner may try to instantiate a `table` object in the first variable of a `pickup` action, which wouldn't make much sense). Otherwise the language is just like STRIPS: time is discrete, states are propositional. Moreover, each agent can execute only one action at a time (no parallel or partially ordered actions are allowed for one agent), and all share the same goal, i.e. only cooperative agents are modeled.

Despite all these limitations this language shows that a proper multi-agent extension can be achieved with minimal changes to a single-agent base language (STRIPS), and that the changes implied to planners may also be limited.

2.2.3. Multiagent Planning Language

Multiagent Planning Language (MAPL) was presented in (Brenner 2003a; 2003b) after interest in such an extension was coined in the Call for Contributions of the Workshop on PDDL at the ICAPS-03 conference. However there was no multi-agent planning track at any IPC ever since or before. Understanding all the reasons is beyond our scope, but some observations can still be made regarding MAPL.

MAPL builds upon PDDL2.1 and thus it includes PDDL2.1's main features (typing, numeric fluents, and durative actions), but at the same time it also makes quite drastic changes to the base language, which may be partly responsible for MAPL's limited success. Among others it abandons the closed-world assumption and instead of predicates it introduces *n*-ary state variables (which may be even `unknown`). This is done partly to cope with partial observability arising from multiple agents operating in the environment, but it also gives rise to the question, if e.g. actions' preconditions reflect an agent's knowledge necessary to execute the action, or *states of the "physical" environment* in which the action can be executed? In our interpretation the latter is closer to the design philosophy of PDDL, since "PDDL is intended to express the 'physics' of a domain" (McDermott et al. 1998). Moreover object-fluents added in PDDL3.1 allow for a very similar functionality without significant changes (to PDDL3.0).

MAPL also introduces a qualitative model of time, which was introduced in PDDL3.0 in a more concise form (of modal operators). This was necessary to coordinate multiple agents' behavior via *speech acts: fixed meta-actions*, whose definition is not part of the description.

Such coordination was necessary to synchronize actions or events in general with initially *uncertain duration* which is again a novelty of MAPL intended to allow greater realism and flexibility. An additional *control function* (for each agent) decides whether this duration is controlled by the environment or by the agent. Similarly there is also a *responsibility function*, which maps state variables to agents to represent which agent is responsible over a state variable. The mentioned additional functions are not part of the MAPL description, yet the definition of planning problems includes the control function for example, which may be confusing. These additions are effectively *advices* to the planner, which contradicts original intentions again: "We have endeavored to provide no advice at all as part of the PDDL notation" (McDermott et al. 1998).

Despite the above additions MAPL still handles the interaction of simultaneous actions similarly to NADL. "Two events are mutually exclusive (mutex) if one affects a state variable assignment that the other relies on or affects" (Brenner 2003a). This model avoids destructive synergetic effects, but isn't considering constructive ones.

In MAPL every agent may face a different planning problem, but all of them eventually share the same goals. In actions, agents are represented with variables like in Section 2.2.2, but they are handled just like any other parameter, which implies further questions, e.g.: *Can an action have more/no agent-parameters? Should the actor always be the first? Can an agent inherit an action defined for a parent-type? Can actions be redefined for children?*

Such questions may become important when a planner is being implemented and so they should be addressed together with a complete syntax definition at least. The reception of MAPL may have also been influenced by that it is not just a new requirement (as derived predicates or numeric fluents), but it is a new language, which is again not in accordance with some intentions behind PDDL, e.g. as stated in the Preface of the Proceedings of the Workshop on PDDL at ICAPS-03: "how the...development of PDDL can be managed within the community to ensure that it does not...fork into multiple incompatible directions...".

2.2.4. Concurrent STRIPS

Concurrent STRIPS (CSTRIPS) was proposed in (Oglietti and Cesta 2004). It is classic STRIPS with the addition of *concurrent threads*, which are explicitly declared, fixed subsets of action schemas. However they are defined only at model level without exact syntax (not even in examples).

Each agent and controllable environmental process can have a separate thread. The planner should find a sequence of fully instantiated actions for each thread based on their respective action-subsets to produce a joint-plan that achieves a common set of goals. While the simplicity of this approach may be tempting, it is not enough to describe challenging multiagent problems (e.g. action interactions).

2.2.5. MA-STRIPS

MA-STRIPS (Brafman and Domshlak 2008) is a multi-agent extension of STRIPS. Its idea is similar to CSTRIPS: partition different agents' grounded actions into disjoint subsets (corresponding to threads in Section 2.2.4). This however may raise implementation-level questions like: *Is it possible that different instantiations of the same operator-schema belong to different agents? If yes, then how should we represent this exactly, syntactically?*

Because of the similarity with CSTRIPS, eventually the same conclusions hold here too, but it must be noted, that the work of (Brafman and Domshlak 2008) focused mainly not on the subtleties of describing multi-agent planning problems, but given MA-STRIPS, a simple description language, they rather set out to formalize and efficiently exploit *loosely coupled agents*. They provided formal results to quantify the notion of agents' coupling and a centralized multi-agent planning algorithm that was shown to be polynomial in the size of the planning problem for fixed coupling levels. Their notions of *internal/public atoms/actions* and the *agent interaction digraph* could be extended to more complex descriptions straightforwardly, but the extension of their planning algorithm and the implied complexity results relying on these notions could be less trivial (e.g. extending them to actions with interacting effects, continuous time or competing goals).

2.3. Requirements of a multi-agent extension

Based on the observations made in previous sections the requirements of a multi-agent extension of PDDL3.1 can be summarized as follows. **In general** it should be...

- *Additional*: a new, additional, optional extension (a PDDL-requirement), not a completely new language;
- *Minimalistic*: introduce only minimal changes to the base language and try to minimize the modifications implied to existing planning systems and approaches;
- *Backward compatible*: compatible with every existing extension (PDDL-requirement) in the official language;
- *Forward compatible*: designed to be easily integrated with anticipated future extensions (e.g. partial observability, stochastic effects, events, processes);
- *General*: useful in all four general categories of multi-agent planning shown in Table 1;
- *Conforming*: in accordance with the design philosophy of the language, i.e. neutrally expressing the "physics" of the domain and including no advice for planners;
- *Compact*: the extended problem- and domain-description should include every model-level detail;
- *Well-defined*: has a complete and accessible definition of formal syntax and at least informal semantics;

In particular the multi-agent extension should allow...

- Modeling concurrent actions with interacting effects;
- Modeling competitive, cooperative or mixed domains;
- Agents having possibly different actions/goals/utilities;
- Straightforward association of agents and actions;
- Distinction between agents and non-agent objects;
- Inheritance/polymorphism of actions/goals/utilities;
- Different agents in different problems of a domain;
- Modeling full- and/or partial-observability;
- Optional use of any combination of PDDL3.1 features.

Optional communication of agents can be modeled by PDDL (by defining communicative actions in the domain), so communication can be realized in execution time. Similarly agents' knowledge can be represented with domain-specific predicates and/or by using a PDDL-extension allowing for partial observability, if necessary. The proposed multi-agent extension should be *modular*, i.e. useable with such other extensions, e.g. partial-observability or probabilistic-effects. But in this paper now *we focus only on* extending PDDL3.1 to *multiple agents*.

3. A multi-agent extension of PDDL3.1

In the following the main result of this paper, a multi-agent extension of PDDL3.1 (MA-PDDL) is presented based on the requirements gathered in Section 2.3. First the syntax and semantics are given, then an example and a discussion of solutions. The section should be best read in conjunction with the BNF of PDDL3.1, e.g. in (Kovacs 2011).

3.1. Domain description

A new `:multi-agent` PDDL-requirement is introduced to indicate the presence of multiple agents in the domain. Agents are considered objects (or constants) that may have associated actions, goals and utilities (metric definitions). I.e. the idea is to associate actions, goals and utilities directly to objects and/or types (say, sets of objects).

The necessary changes implied to the grammar first include the following slight **modification** of the production rule defining *non-durative actions* in the BNF of PDDL3.1.

```
<action-def> ::=
        (:action <action-symbol>
             [:agent <agent-def>]:multi-agent
             [:parameters (<typed list (variable)>)]
             <action-def body>)
```

The only difference is the addition of an optional part for the associated agent(s). It can be used only if the `:multi-agent` requirement is declared, and also implies **addition** of the following rules for `<agent-def>` to the grammar.

```
<agent-def>    ::= <name>
<agent-def>    ::= <variable>
<agent-def>    ::=:typing <type>
<agent-def>    ::=:typing <variable> - <type>
```

This means that agents can be associated to an action in form of constants or variables. If `:typing` is declared, then they can be given in form of types or typed variables too. If a type or a variable is given, then the action-schema is associated to every object whose type *is a subset* of the given type or the type of the variable (since in PDDL types correspond to sets of objects). Without an explicitly declared type the agent-variable is assumed to have type `object` by default (corresponding to the set of every object). In this case every object is eventually considered an agent, since those and only those objects are considered *agents*, which have at least one associated action-schema. If agents are referred to with variables, we suggest to use types to enable distinction between agents and non-agent objects. Furthermore it is required that the names of objects and primitive types are unique and not overlapping, and that every object has only one directly associated type.

If the agent is referenced with a variable in the action-schema, then this variable may appear in the body of the action-schema (in conditions and effects) just like any other action-parameter, and thus the name of the agent variable is required to be different from parameter-names.

If the `:multi-agent` requirement is declared, but the agent-reference is not given in the action-schema, then the schema is associated to the type `object` by default.

Now in case of `:typing`, given a type-hierarchy, an agent-object with a given type is associated with actions that are either associated to it directly, or directly to its type, or directly to an ancestor-type (superset) of its type. This is called *inheritance* of actions. *Polymorphism* on the other hand works as follows: an action with the same name and arity (number of arguments) can be redefined for descendants, i.e. an action directly associated to a type redefines any action directly associated to its ancestor type (superset), if the name and arity of the actions are equal.

An action associated directly to an object (constant) redefines any actions with the same name and arity directly associated to the type of the object, or which are inherited. An agent-object or type cannot have two or more directly associated action-schemas with the same name and arity. Therefore association of actions to agents is unambiguous.

Beyond typing an even more important aspect of a proper multi-agent extension is how interaction of concurrent (joint) actions works. For the proposed extension a *generalization of concurrent action lists* presented in Section 2.2 in (Boutilier and Brafman 2001) is proposed. The idea is simply to allow references to concurrent actions not only in a special construct, such as the concurrent action list, but also among the preconditions of actions and the conditions of actions' conditional effects. Interweaving the content of concurrent action lists with conditions this way allows for a more convenient and compact design. The proposed idea is similar to "progressive predicates" suggested in Section 2.1 in (Bacchus 2003), except that we now refer directly to ongoing actions, and not to facts. This also implies that the name-arity pairs of fluents (predicates, numeric and object fluents) need to be unique, and cannot overlap with the name-arity pairs of actions (durative or non-durative).

The above considerations imply the **addition** of the following 4 new production rules to the grammar.

```
<GD>     ::=:multi-agent <action formula(term)>
<GD>     ::=:multi-agent + :negative-preconditions
                         (not <action formula(term)>)
```

This means that if the `:multi-agent` requirement is declared, a goal description, `<GD>` is allowed to refer also to ongoing actions (similarly to state fluents). The exact form of reference to concurrent actions is the following.

```
<action formula(t)>::= (<action-symbol> t t*)
<action formula(t)>::=:durative-actions (<da-symbol> t t*)
```

The first argument (term) should be always the agent executing the referenced action, while further arguments should be the actual parameters of that action in their respective order. If during execution a grounded reference to an action *A* needs to be positive for the conditions of a grounded action *B* to hold, then this means that *A* needs to be executed in parallel with *B* for *B*'s respective effects to take place. Otherwise, if the grounded reference to *A* needs to be negative for conditions of *B* to hold at a given time during execution, then *A* should not be executed in parallel with *B* for *B*'s respective effects to take place at that time.

The exact time(interval) when *A* should or should not be under parallel execution with *B* is the same time(interval), when a unique "progressive" proposition P_A corresponding to *A* should or should not be true respectively for *B*'s conditions to hold were all occurrences of *A* replaced with P_A in the grounded PDDL-description and thus in the plan. Now this depends only on the semantics of PDDL3.1.

It must be noted though that the *consistency of a joint-action* (where all member actions either refer to other members in their conditions, or they are referred to by at least one of them) should be more relaxed than the traditional definition of mutex actions. For *non-durative actions*, similarly to Definition 2 and 3 in (Boutilier and Brafman 2001) we only require that the members of the joint-action have consistent joint-(pre)conditions and joint-effects in a given state, i.e. interference among effects and (pre)conditions is not considered. In case of *durative actions* the consistency check should focus on the exact time instants and intervals when (pre)conditions need to hold, and when effects take place during scheduled execution. That is in this case it may happen that the (pre)conditions of a member of a durative joint-action are inconsistent with the effects of another member, which may imply a re-scheduling of these actions by the planner. The way this is achieved is beyond the scope of this paper. *A consistent joint-plan consists of consistent joint-actions.*

Durative actions of multiple agents are defined similarly to non-durative actions. The rule for `<durative-action-def>` needs to be slightly **modified** as follows.

```
<durative-action-def>::=
        (:durative-action <da-symbol>
             [:agent <agent-def>]:multi-agent
             [:parameters (<typed list (variable)>)]
             <da-def body>)
```

This means that we can now associate agents to durative actions just like we did to non-durative actions.

Many further **additions** could be made to the grammar of the domain. A minimal, but useful one is the following.

```
<f-exp> ::=:numeric-fluents + :multi-agent
           (num (<typed list (variable)>)
                 <emptyOr (GD)>
                 <action formula(term)>)
```

The built-in function `num` calculates the number of actions under execution (during execution) for every instantiation of a list of variables where given conditions hold. This small extension adds great expressivity.

3.2. Problem description

The problem description needs to be extended slightly to cope with possibly different goals and utilities of agents. This requires **modification** of the problem definition first.

```
<problem>    ::= (define (problem <name>)
                     (:domain <name>)
                     [<require-def>]
                     [<object declaration>]
                     <init>
                     <goal>+
                     [<constraints>]:constraints
                     <metric-spec>* :numeric-fluents
                     [<length-spec>])
```

The only change here is that now at least one `<goal>` and zero or more `<metric-spec>` structures are required.

Goals can be empty (always true), while utilities don't need to be present when `:numeric-fluents` is declared.

In case of multiple agents, *goals* can be captured by the **addition** of the following production rule to the grammar.

```
<goal> ::=:multi-agent (:goal
                         [:agent <agent-def>]
                         :condition <emptyOr (pre-GD)>)
```

The only essential change here compared to PDDL3.1 is the addition of the agent-reference. If it refers to the agent with a variable, then the variable may appear in the goal formula. Goal conditions are prefixed with `:condition` to emphasize them more. *Utilities* need a similar **addition**.

```
<metric-spec> ::=:multi-agent + :numeric-fluents
           (:metric
              [:agent <agent-def>]
              :utility <optimization> <metric-f-exp>)
```

The declaration of the `:multi-agent` requirement is necessary for the use of the above two structures, but we can also use default PDDL3.1 goals and metric structures, which would mean – similarly to the case of actions – that goals and utilities are associated with the `object` type, i.e. with every agent-object. Above we see that goal and utility schemas can be associated directly to objects (or constants) or types similarly to actions, although one object or type can have only one directly associated goal or utility schema in contrary to actions. But inheritance and polymorphism are the same as in case of actions. Therefore the assignment of goals/utilities to agents is unambiguous.

One last **addition** is necessary to the grammar to allow agent-variables in hitherto grounded metric expressions.

```
<metric-f-exp>::=:multi-agent (<function-symbol> <term>*)
```

This way now we can include *fluents* in connection with agents in the definition of their utility, but naturally the value of metric needs to remain numeric. Not all agents have to have a utility though, but all of them need to have (at least an inherited) goal, which may be the same for all of them or different depending on the problem at hand.

An important topic is still left untouched: *For which agents is a planner planning? Which object(s) represent(s) the planner in the description? Should it be represented?*

The answer depends on how the MA-PDDL description is used: whether the planner is external or situated; whether planning is centralized or decentralized, whether it is distributed; or whether planners share the same MA-PDDL description. The association of the planner and agent(s) can vary from run-to-run (similarly to how an agent may assume different players' role during different plays of the same game (von Neumann and Morgenstern 1944)). This meta-information, the connection of planners and agent-objects is therefore not included in the description. It is the responsibility of the planner to know *for whom* it plans, and possibly *which object(s) represent(s) it* in the problem. So it is suggested to planner applications to have 1-2 more inputs carrying this information beside other parameters.

3.3. Example

The following simple example aims to give a basic idea of how the proposed multi-agent extension works. A minimal set of PDDL features is used to illustrate important aspects, such as cooperation, joint-actions, constructive synergy.

The only action-schema in the domain (`lift`) is associated with type `agent`, which is a direct descendant (subset) of `object`; `lift` allows an agent to lift the `table` (the only domain-constant of type `object`), but only if it is not yet lifted, if the agent is at the `table`, and if there is at least one other agent at the `table` lifting it simultaneously.

```
(define (domain ma-lift-table)
(:requirements:equality :negative-preconditions
              :existential-preconditions :typing :multi-agent)
(:types agent)(:constants table)
(:predicates  (lifted ?x - object) (at ?a - agent ?o - object))
(:action lift :agent ?a - agent :parameters ()
:precondition (and (not (lifted table)) (at ?a table)
                (exists  (?b - agent)
                 (and (not (= ?a ?b)) (at ?b table) (lift ?b))))
:effect (lifted table)))
```

The related problem description defines 2 agents: `a` and `b`, both being at the `table`, and having the same goal: the `table` being lifted. Their goal is defined for type `agent`.

```
(define (problem ma-lift-table-1)
(:domain ma-lift-table)
(:objects a b - agent)
(:init (at a table) (at b table))
(:goal :agent agent :condition (lifted table)))
```

The solution requires cooperation from `a` and `b`, since they have the same goal and the only way for them to achieve it is to coordinate their actions. The only, trivially simple solution is when both `lift` the `table` starting at time 0: `[0:(lift a) 0:(lift b)]`. Because of the lack of options the same plan should arise in case one or more external *rational* planners plan for `a` and `b`.

When a planner chooses a grounded `lift` action for execution in a given state, it can assert a corresponding unique `(lift ·)` fact to the state, and check what implications this has on the applicability of other chosen actions. If they remain executable, it may continue, otherwise it may retract `(lift ·)` from the state, and choose different actions. This should work also in general.

In case of decentralized planning, i.e. when different planners plan for different agent-objects, but all planners share the same MA-PDDL description (which is *common knowledge* among them), then solution-plans should not be fully ordered sequences of temporally annotated actions anymore. They should be rather *strategies* (for each agent) that prescribe actions to observation-histories of the agent. A joint plan in this case is a *combination of such strategies*.

In our case, since partial observability isn't introduced yet as another extension, the planning environment is *fully observable*, i.e. observations are complete descriptions of new states and action-combinations that produced them.

We should also note that though an MA-PDDL description may be converted to *an extensive- or normal-form game*, it *would be a much less detailed description*.

Two issues arise in the decentralized case: **(1)** coping with durative strategies; and **(2)** both in durative and non-durative case it is not trivial how to compactly represent strategies, especially in case of large state-spaces.

However both issues can be solved **(1)** by *reasonably restricting the scheduling* of durative actions (e.g. an agent could schedule its next actions only when an other action starts/ends); and **(2)** by using a *client-server architecture* with planners as clients. Planners could receive new observations for relevant time-instants (see previous issue) from the Server and answer with their actual actions.

4. A multi-agent planning track at the IPC

In this section a short proposal is made for a multi-agent planning track at the forthcoming IPCs based on the multi-agent extension of PDDL3.1 presented in Section 3.

There are 3 organizational steps (similarly to current IPCs): **(1)** preparation; **(2)** competition; and **(3)** evaluation. During the *preparation* phase the following should be made public: a Call for Submissions; detailed rules of the competition/evaluation; any source-code and additional applications with documentation; a detailed manual/article about MA-PDDL; and domains/problems for participants.

For the *competition* the participants would need to submit planners (sources, binaries) and papers about them. The competition itself could consist of 2 fully-observable sub-tracks at first: **(2a)** when $N = 1$ external planner plans *for* $M > 1$ situated agents, and **(2b)** when planning is done *by* $N = M > 1$ situated planners. In both cases problems can be categorized according to 3 properties: whether **(i)** all agents' goals/utilities are the same; **(ii)** if there are utilities at all; and **(iii)** if durative actions are allowed. If in **(2b)** we do not allow durative actions, then altogether *12 categories* of multi-agent competition emerge.

When **(i)** holds, problems are *cooperative*. Otherwise they are *competitive*. The latter case can be divided into sub-cases, where each agent has different goals/utilities, and where only agents in *teams* have the same goal/utility.

It should be noted that the easiest category is sub-track **(2a)** when **(i)** holds, but this is still harder than single-agent planning e.g. because of possible constructive synergies.

In each of the 12 categories approx. 12-14 domains could be present each with around 20 related problems. The *evaluation* in case of sub-track **(2a)** could measure normalized **quality** of joint-plans and planning-time per problem, and the number of solved problems per domain for each planner. The sum of these scores could decide the winner of sub-track **(2a)**. But it should be added, that the quality of plans depends mostly on **(i)**. If **(i)** and **(ii)** hold, then quality is defined by utility, but if **(ii)** is not true, then quality can be the makespan of plans. If **(i)** is false, then the number of agents whose goal was achieved, or the sum of achieved sub-goals or of plans' makespan can be used.

The evaluation of planners in sub-track **(2b)** could be similar to evaluation at the probabilistic track at IPC-2011. As mentioned in Section 3.3, a client/server architecture could be used with planner-clients receiving observations from a server and replying to it with their actions. The server could wait for planners' actions at each step for a given time. In case of time-out (e.g. after 30 seconds/step) the `no-op` action could be chosen for late planners.

Initially the server should broadcast the MA-PDDL description, and then for each problem and permutation of planner-agent assignments it could execute e.g. 30 runs to determine planners' average **fitness** for each assignment (since some may make non-deterministic decisions). The sum of these averages over assignments could be the score of a planner for a problem, and thus the sum of scores over problems and domains could determine the winner of sub-track **(2b)**. If **(ii)** holds, then planners' fitness could be the individual utility of their agent. Otherwise it could be the maximum of its simultaneously achieved sub-goals.

5. Conclusions

A *multi-agent extension* of PDDL3.1 was proposed with a corresponding *multi-agent planning track* for the IPC to enable more direct comparison of multi-agent planners and approaches and a greater reuse of research. Planning *by* and *for* agents is both possible. The syntax and semantics of the extension were provided together with an example. A few corrections to the BNF of PDDL3.1 were also listed and an overview of current research in the field was given.

Future research could focus on providing more detailed, possibly formal semantics; planning algorithms; more application domains (e.g. multi-robots, such as RoboCup, or networking problems, such as efficient routing with limited resources). The addition of *partial observability* (in a separate, modular PDDL-requirement) would be primary, but *probabilistic effects* and *events/processes* may also be considered to allow treatment of more realistic problems. The corresponding multi-agent IPC track could also be developed further to narrow the gap between theory and practice and to advance the field of multi-agent planning.

Acknowledgments

This work was partially supported by the ARTEMIS JU and the Hungarian National Development Agency (NFÜ) in frame of the R3-COP (Robust & Safe Mobile Co-operative Systems) project. The author wishes also to thank SUZUKI Foundation Japan, Derek Long, Carlos L. López, Gabriele Röger, Ron Alford, Éric Jacopin, Takashi Watanabe, Naoki Fukuta and Tadeusz P. Dobrowiecki.

References

Bacchus, F. 2003. The Power of Modeling - a Response to PDDL2.1 (Commentary). *J. of AI Res.* 20:125-132.

Baral, C.; Gelfond, G.; Son, T. C.; and Pontelli, E. 2010. Using Answer Set Programming to model multi-agent scenarios involving agents' knowledge about other's knowledge, In *Proc. of AAMAS-2010*, 259-266. IFAAMAS.

Baral, C.; and Gelfond, G. 2011. On representing actions in multi-agent domains, In Engelmore, R., and Morgan, A. eds. *Logic programming, knowledge representation, and nonmonotonic reasoning*. 213-232. Springer.

Beaudry, E.; Kabanza, F.; and Michaud, F. 2010. Planning for Concurrent Action Executions Under Action Duration Uncertainty Using Dynamically Generated Bayesian Networks. In *Proc. of ICAPS-10*, 10-17. AAAI Press.

Boutilier, C.; and Brafman, R. I. 2001. Partial-order planning with concurrent interacting actions. *J. of AI Res.* 14(1):105-136.

Bowling, M.; Jensen, R.; and Veloso, M. 2002. A formalization of equilibria for multiagent planning. In *Proc. of the Workshop on Planning with and for Multiagent Systems, AAAI-02*, 1-6.

Brafman, R. I.; and Domshlak, C. 2008. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *Proc. of ICAPS-08*, 28-35. AAAI Press.

Brafman, R. I.; Domshlak, C.; Engel, Y.; and Tennenholtz, M. 2009. Planning Games, In *Proc. of IJCAI-09*, 73-78. AAAI Press.

Brenner, M. 2003a. A Multiagent Planning Language. In *Proc. of the Workshop on PDDL, ICAPS-03*, 33-38.

Brenner, M. 2003b. Multiagent Planning with Partially Ordered Temporal Plans, Technical Report No. 190, Institut für Informatik, Universität Freiburg, Germany.

Crosby, M.; and Rovatsos, M. 2011. Heuristic Multiagent Planning with Self-Interested Agents, In *Proc. of AAMAS-2011*, 1213-1214. IFAAMAS.

Edelkamp, S.; and Hoffmann, J. 2004a. PDDL2.2: The Language for the Classical Part of the 4th International planning Competition, Technical Report No. 195, Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Germany.

Edelkamp, S.; and Hoffmann, J. 2004b. PDDL2.2: The Language for the Classical Part of IPC-4. In *IPC-4 Booklet, ICAPS-04*, 1-5.

Fikes, R. E.; and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 5(2):189-208.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to pddl for Expressing Temporal Planning Domains. *J. of AI Res.* 20: 61-124.

Fox, M.; and Long, D. 2006. Modelling Mixed Discrete-Continuous Domains for Planning. *J. of AI Res.* 27:235-297.

Fox, M.; Long, D.; and Magazzeni, D. 2011. Automatic Construction of Efficient Multiple Battery Usage Policies. In *Proc. of ICAPS-11*, 2620-2625. AAAI Press.

Gerevini, A.; and Long D. 2005. BNF Description of PDDL3.0. *Unpublished manuscript from the IPC-5 website*. http://cs-www.cs.yale.edu/homes/dvm/papers/pddl-bnf.pdf

Helmert, M. 2008. Changes in PDDL 3.1. *Unpublished summary from the IPC-2008 website*. http://ipc.informatik.uni-freiburg.de/PddlExtension

Jensen, R. M.; and Veloso, M. M. 2000. OBDD-based universal planning for synchronized agents in non-deterministic domains. *J. of AI Res.* 13(1):189-226.

Jonsson, A.; and Rovatsos, M. 2011. Scaling Up Multiagent Planning: A Best-Response Approach. In *Proc. of ICAPS-11*, 114-121. AAAI Press.

Kovacs, D. L. 2011. BNF definition of PDDL 3.1. *Unpublished manuscript from the IPC-2011 website*. http://www.plg.inf.uc3m.es/ipc2011-deterministic/Resources

Kumar, A.; Zilberstein, S.; and Toussaint, M. 2011. Scalable Multiagent Planning Using Probabilistic Inference, In *Proc. of IJCAI-11*, 2140-2146. AAAI Press.

Marecki, J.; and Tambe, M. 2009. Planning with Continuous Resources for Agent Teams, In *Proc. of AAMAS-09*, 1089-1096.

Martins, M. F.; and Demiris, Y. 2010. Learning Multirobot Joint Action Plans from Simultaneous Task Execution Demonstrations, In *Proc. of AAMAS-2010*, 931-938. IFAAMAS.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL---The Planning Domain Definition Language, Technical Report, CVC TR-98-003/DCS TR-1165, Yale Center for CVC, NH, CT.

von Neumann, J.; and Morgenstern, O. 1944. *Theory of games and economic behavior*. Princeton University Press.

Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A General, Fully Distributed Multi-Agent Planning Algorithm, In *Proc. of AAMAS-2010*, 1323-1330. IFAAMAS.

Oglietti, M.; and Cesta, A. 2004. CSTRIPS: Towards Explicit Concurrent Planning. In *Proc. of the 3rd Italian WS on Plan. and Sched., 9th Nat. Symp. of Assoc. Italiana per l'Int. Artif.*, 1-13.

Pajarinen, J.; and Peltonen, J. 2011. Efficient Planning for Factored Infinite-Horizon DEC-POMDPs, In *Proc. of IJCAI-11*, 325-331. AAAI Press.

Russell, S.; and Norvig, P. 2010. *Artificial Intelligence: A Modern Approach.* Prentice Hall.

Shah, J. A.; Conrad, P. R.; and Williams, B. C. 2009. Fast distributed multi-agent plan execution with dynamic task assignment and scheduling. In *Proc. of ICAPS-09*, 289-296.

Spaan, M. T. J.; Oliehoek, F. A.; and Amato, C. 2011. Scaling Up Optimal Heuristic Search in Dec-POMDPs via Incremental Expansion, In *Proc. of IJCAI-11*, 2027-2032. AAAI Press.

Stefanovitch, N.; Farinelli, A.; Rogers, A.; and Jennings, N. R. 2011. Resource-Aware Junction Trees for Efficient Multi-Agent Coordination, In *Proc. of AAMAS-2011*, 363-370. IFAAMAS.

Teichteil-Königsbuch, F. 2008. Extending PPDDL1.0 to Model Hybrid Markov Decision Processes. In *Proc. of the WS on A Reality Check for Plan. and Sched. Under Unc., ICAPS-08*, 1-8.

Wang, K. C.; and Botea, A. 2011. MAPP: a Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees. *J. of AI Res.* 42:55-90.

de Weerdt, M.; and Clement, B. 2009. Introduction to planning in multiagent systems. *Multiagent Grid Systems* 5(4):345-355.

Yabu, Y.; Yokoo, M.; and Iwasaki, A. 2009. Multiagent Planning with Trembling-Hand Perfect Equilibrium in Multiagent POMDPs, In Ghose, A.; Governatori, G.; and Sadananda, R. eds. *Agent Computing and Multi-Agent Systems*. 13-24. Springer.

Zhuo, H. H.; and Li, L. 2011. Multi-Agent Plan Recognition with Partial Team Traces and Plan Libraries, In *Proc. of IJCAI-11*, 484-489. AAAI Press.

Zhuo, H. H.; Muñoz-Avila, H.; and Yang, Q. 2011. Learning Action Models for Multi-Agent Planning, In *Proc. of AAMAS-2011*, 217-224. IFAAMAS.

Mining IPC-2011 Results

Isabel Cenamor, Tomás de la Rosa, and **Fernando Fernández**
Departamento de Informática, Universidad Carlos III de Madrid
Avda. de la Universidad, 30. Leganés (Madrid). Spain
icenamor@inf.uc3m.es,trosa@inf.uc3m.es, ffernand@inf.uc3m.es

Abstract

The International Planning Competition (IPC) offers a wonderful scope to evaluate and compare different planning approaches and specific planner implementations in benchmark domains. In IPC 2011, the software generated for the competition permits to obtain a lot of data about the execution of all the planners in the different tracks in a simple way, which permits its recovery and use. In this work, we propose to analyze such data from a Data Mining (DM) perspective, including additional features which can be useful for the analysis. In such a way, we are able to construct models of the results obtained, allowing us to make additional analysis to the ones performed by the organizers. In this work, we report some initial analysis after constructing classification and regression models for the sequential satisficing and optimal track.

Introduction

Since 1998, the International Planning Competition (IPC) has offered the opportunity to researchers in Automated Planning to evaluate and compare their ideas about how to develop better and faster planners. Each competition produces a big amount of data specially from the execution of the planners in the different tracks, domains and problems. This fact is remarkable in the last two IPC, where the number of participants and the number of evaluated domains have increased considerably. Additionally in IPC 2011, the execution of a planner for a planning problem reported a total of 33 features, including runtime, number of solutions found, the moment where each solution was obtained, or the quality of those solutions. Such an amount of data opens a wide variety of analysis and studies from a Data Mining (DM) perspective. For instance, one would think whether it is possible to generate a model that predicts if a planner will be able to find a solution for a given problem and, if so, with what probability or how long it will take. The results of the prediction can help us to find some insights about the performance of planners or can be used to configure a portfolio of planners that takes into account the particular features of a planning problem.

In this work we perform an initial analysis of the IPC 2011 data. We follow a classical data mining methodology as it is introduced in the following section, which describes the DM process using a data workflow. The following sections explains the main steps deeply: the pre-process of the data, including feature generation, extraction and instance selection; then, the different data models generated, its evaluation and analysis. Later, some related works are summarized, while the last section describes our main conclusions and future research lines.

Data Mining Workflow of the IPC 2011 Results

Data Mining is a process of discovering implicit knowledge from data. Such process may contain different phases depending on the goals, data sources and tools. Figure 1 shows the complete process performed in this work. We have followed the phases described in the CRISP-DM methodology (Chapman et al. 2000): data understanding, data preparation, modeling, evaluation and deployment. [1]

In addition to previous phases, CRISP-DM starts with a business understanding phase, which is very related to business intelligence approaches, where the organization where the data mining process is going to be performed is analyzed to generate the data mining goals. In our case, we have defined the data mining goals as the creation of predictive models that predict, on the one hand, whether a planner will be able to solve a problem, and if so, what will be the time required to compute a plan. The first problem is a classification task, where the predicted attribute has only two possible values: $\{true, false\}$. The second problem is a regression task, where the output belongs to the positive real numbers, but restricted to the time limit given to the planners (i.e., 1800 seconds). The reason why we have chosen these two tasks is two-fold. On the one hand, we want to characterize under which conditions a planner will succeed, so this characterization will support a better knowledge of the planners and their possible improvements. On the other hand, and from a more engineering point of view, we want to obtain predictive models that can be used for the selection of planners when configuring a portfolio-based planner.

Given those goals, the data source of the process is the SVN repository of the IPC 2011. From this repository we downloaded the domain and problem files in PDDL, and the competition results using the IPCReport tool. From the domain and problem files, we get the features of each instance

[1] In this work we do not study in deep methodological aspects, so any other methodology could be used for the description.

and compute additional features we think will serve for a better modeling. Once we have the features, the data is integrated, generating a first training set. This data suffers new transformations (feature generation and selection, and instance selection) depending on the requirements of the models to build. Afterwards, the models are learned using machine learning techniques and then evaluated using a suitable evaluation scheme for estimating the prediction capabilities of the models. In a real world scenario the methodology also includes a deployment phase, but we have not consider it for this work. The next sections describe in depth all these phases.

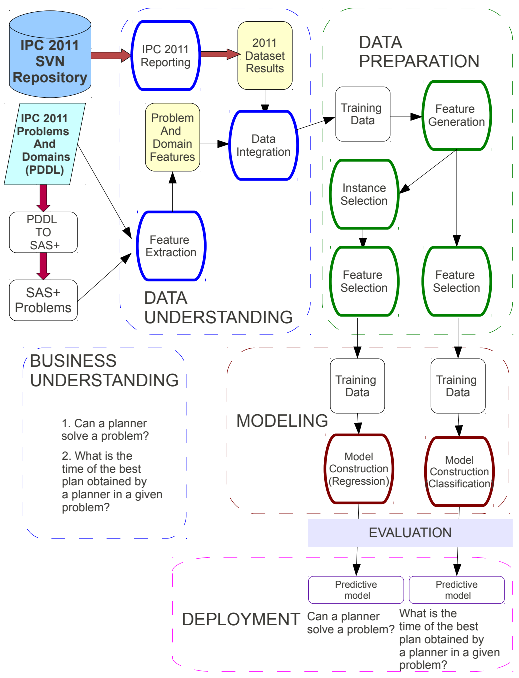


Figure 1: Data workflow of the mining process following CRISP-DM methodology

Data Understanding

This first step consists of recognizing the available sources of data and devising a good way to collect and integrate this data. For our analysis we have focused on the sequential satisficing track and the sequential optimal track of the IPC-2011. In the first one, planners should find a plan within the time bound. Since the solution does not need to be optimal, most planners develop the strategy of continuously reporting a solution that improves a previous one until the time runs out. In the sequential optimal track, planners should report a single optimal solution within the time bound.

For both tracks, the input data to the DM process comes from the competition results and from the domain and problem files. The competition results are in fairly suitable way for applying DM because each result for a planner execution, given a domain and a problem, corresponds to an example in terms of machine learning, i.e., a set of features and the given target attribute. However, this is not the case for PDDL files, therefore we need to extract some useful features from them. In the following sections we describe the data collected and generated for the mining process.

Features from IPC 2011 results

The IPC-2011 software (López 2011) contains several packages, which facilitates to test planners, compare their performance and obtain reports of the results. IPCReport is the package in charge of providing access to the data generated during the competition. The report is able to present 33 raw variables and 23 elaborated variables. The raw variables are the principal observations about the execution of planners for a given problem and domain. The elaborated variables are the ones derived from raw data, for example, a maximum of a set of values in the raw variables. In this work, we use some of the raw variables, which are described next:

1. *Planner*: the name of the planner.
2. *Domain*: the name of the domain.
3. *Problem*: a problem identifier (the problem number for a particular domain).
4. *Oktimesols*: For each validated plan found by the planner, a vector containing the time elapsed (in seconds) since the planner was started until such solution was found. This variable can either be empty (no solution was found), have a single value or an array of values. We will refer to this feature as the *time vector*.
5. *Values*: A vector containing the plan quality for the found solutions. The $i-th$ position of vector *OKtimesols* corresponds to the same plan as $i-th$ position of vector Values. We will refer to this feature as the *quality vector*

From the sequential satisficing track we got 7560 instances, corresponding to the execution of 27 planners in a total of 20 problems for 14 domains. From the the sequential optimal track we got 3360 instances corresponding to the execution of 12 planners in a total of 20 problems for 14 domains.

Features extracted from IPC 2011 domains and problems

The objective of this process is to generate a set of features that characterize a problem instance. In order to achieve a good characterization, we will use basic features that can be extracted from PDDL files and a set of elaborated features generated from the problem translation to the SAS+ formalism and its induced graphs, i.e., the causal graph and the domain transition graphs.

The basic features from a problem instance are:

1. *Objects:* Number of objects defined in the problem.
2. *Literals:* Number of instantiated predicates in the initial state.
3. *Goals:* Number of instantiated predicates that are true in the final state.

These basic features offer information about the complexity of the problems. In fact, most problem generators receives as input the number of each type object and the number of goals, so they can determine the instance size. However, these basic features and many others that can be extracted from the domain definition will not be sufficient for discriminating between instances of the same size. Indeed, any conceivable set of possible features from the domain, such as the number of actions, maximum predicate arity, maximum number of preconditions, in conjunction with the basic features from the problem, will serve to discriminate between problems of different domains or problem of the same domains with different size, but not between problems having the same object and goal distribution. We argue that additional information can be extracted from the graphs induced by the SAS+ formalism in order to partially recognize the differences between problems of similar size.

The SAS+ formalism (Backstrom and Nebel 1995; Helmert 2009) is an alternative representation for STRIPS. It considers a set of state variables, each one associated to a finite set of possible values. Actions have preconditions (i.e., required assignments of some variables to the action become applicable) and effects (i.e., the new values of the affected state variables). Using this formalism, a problem instance can be represented in a structured way using two types of graphs: (1) The ***causal graph*** (CG), which is a graph that captures the causal dependencies between the state variables of a given problem. (2) The ***domain transition graph*** (DTG) which encodes the allowed transitions between different values of a variable. In a problem there is a DTG for each state variable. For more details see (Helmert 2006).

We have used the LAMA planner (Richter and Westphal 2010) pre-process to generate all graphs. We recall that in the causal graph, the *high-level* variables are the variables for which there is defined a value in the goal. Although the common definition of the causal graph does not consider the edges as weighted, LAMA computes the edge weights of the causal graph as the number of instantiated actions that induced each edge. We also consider these weights for computing our features.We have extracted a total of 47 features for each problem, which are summarized next.

Features from the Causal Graph: For the CG we classify the generated features in four categories: (1) general, which includes the direct information from the graph (2) ratios, which represents interesting proportions that may be equal across problems of different size, (3) statistical, such as the average or the standard deviation of particular elements of a graph. (4) high-level statistical, the same as before but only considering the high-level variables.

- **General Features**
  1. *NumberVariablesCG:* The number of variables of the causal graph.
  2. *High-LevelVariablesCG:* The number of high-level variables.
  3. *TotalEdgesCG:* The number of edges.
  4. *TotalWeightCG:* The sum of weights of the edges.
- **CG Ratios**
  1. *VERatio*: The ratio between the total number of variables and the total numbers of edges. This ratio shows the level of connection in the causal graph.
  2. *WERatio*: The ratio between the sum of the weights and the number of edges. This ratio shows the average weight for the edges.
  3. *WVRatio*: The ratio between the sum of the weights and the number of variables.
  4. *HVRatio*: The ratio between the number of high-level variables and the total number of variables. This ratio shows the percentage of variables involved in the problem goals.
- **Statistics of the CG**

  This statistical information is used to characterize the structure of the causal graph. We compute the average, the maximum and the standard deviation of the following values:
  1. *InputEdgeCG*: The number of incoming edges for each variable. Thus, we compute the average of input edges in the CG, the maximum number of input edges for a single variable and standard deviation of input edges, for knowing the variation between variables.
  2. *InputWeightCG*: The sum of the weights of the incoming edges for each variable. Thus, we compute three new features following the same computations as *InputEdgeCG*.
  3. *OutputEdgeCG:* The number of outgoing edges for each variable. Thus, we compute the average of output edges in the CG, the maximum number of output edges for a single variable and standard deviation of output edges.
  4. *OutputWeightCG:* The sum of the weights of the incoming edges for each variable. Thus, we compute three new features following the same computations as *OutputEdgeCG*.
- **Statistics of high-level Variables**

  This information tries to encode the structure for the variables involved in the problem goals. We compute the average, the maximum and the standard deviation for the following values:
  1. *InputEdgeHV*: The number of incoming edges for each high level variables. This value produces three new features following the same computation as *InputEdgeCG*.
  2. *InputWeightHV*: The edge weight sum of the incoming edges for each high level variables. This value produces three new features following the same computation as *InputWeightCG*.

Features from DTGs: Since the number of DTGs in a problem is variable, it is difficult to encode general attributes for each graph. Instead, we can summarize DTGs characteristics aggregating the relevant properties of all graphs. We compute general aggregated features and some statistics over all graphs.

- **General Aggregated Features**

1. *NumberVerticesDTG*: The sum of the number of vertices of all DTG.
2. *TotalEdgesDTG*: The sum of the number of edges of all DTGs.
3. *TotalWeightDTG*: The sum of the weights of the edges of all DTGs. The weight of An edge in DTG corresponds to the cost of applying the action that induced the edge.

- **Statistics of DTGs**

  Considering all DTGs, we compute the average, the maximum and the standard deviation of the following values:

1. *InputEdgeDTG*: The number of incoming edges for a vertex in a DTG.
2. *InputWeightDTG*: The sum of the weights of the incoming edges of all vertices.
3. *OutputEdgeDTG*: The number of outgoing edges for a vertex in a DTG.
4. *OutputWeightDTG*: The sum of the weights of the outgoing edges of all vertices.

Data Preparation

Data preparation may contain different phases, that can be summarized in four: data cleaning and transformation (like normalizing data), feature generation (generating new features from the available ones), feature selection (eliminate useless features) and instance selection (select a sub-subset of training instances from the total one).

The main tasks of the data cleaning are the management of missing values and the identification of outliers. For the first one, we have not performed any task since we have a strong confidence on the data. The IPCReport tool gives a complete and reliable data set and the feature generation process was computable for all problems used in the competition. For the second one, we decided to eliminate some outliers, mainly data from specific problems in some domains which generated data very far from the average values. Nevertheless initial evaluations demonstrated that results did not vary significantly.

Regarding feature generation, we generated a lot of derived features, as explained in previous sections. Although feature generation is an operation that is typically performed in this step, we performed it in the data collection and understanding phase due to implementation reasons, thus while we extracted the data from the PDDL and SAS+ representations, we computed also the derived features.

The rest of the data preparation processes are explained depending on the data-mining task:

- **Classification**: For this task we create a new attribute *class* representing whether a planner was able to solve a problem. This attribute is set to "yes" if there exists at least one value in the attribute of the *time* vector, otherwise it is set to "no". Both time and quality vectors are removed from the dataset since they are no longer of interest. Additionally, the domain name and problem number are treated as identifiers, therefore they will not be used for modeling. For this task, we did not perform any instance selection process, since data was very clean, and we have a manageable amount of instances. This task will be the same for the satisficing and optimal track.
- **Regression**: In this task we draw a distinction for IPC tracks. For the sequential satisficing track we try to predict three different values: *first-time*, representing the execution time elapsed when a planner found its first solution for the problem; *best-time* representing the execution time elapsed when a planner found its best solution; and *medium-time*, representing the median of the time vector. When predicting each of these values, the others are removed from the training set. The time and quality vectors are also removed. In addition, we have eliminated all the instances where a planner was not able to find a solution, i.e. where time and quality vectors are missing. In the sequential optimal track planners give one single time for finding the optimal solution, therefore the regression task consists of predicting the single value of the time vector. As before, we only consider instances where a solution was found.

Data Modeling and Evaluation

Although Figure 1 presents the data mining process as a cascade, data preparation, modeling and evaluation are typically performed many times iteratively until a satisficing solution is found. In addition, different models and learning algorithms are tested until the "best" one is obtained.

In this work, we have used several algorithms, decision trees (J48) (Quinlan 1993), decision rules (PART) (Frank and Witten 1998), Support Vector Machines (SMO) (Cristianini and Shawe-Taylor 2000), and IBK (Witten and Frank 2005) for different values of k. The implementation of these algorithms is provided by WEKA (Witten and Frank 2005), and they are used with the pre-defined parameters.

Evaluation Set-up

A question that arises when applying machine learning over any dataset is how to perform the evaluation process. Evaluating over the training set typically produces optimistic estimations of performance over future data, since it is easy that the generated models are over-fitted to the training set. Therefore, other evaluation mechanisms must be used. A classical one is *cross validation*, which is a procedure to estimate the performance of a previously learned model over data which has not been used to train the model. It consists on dividing the whole data set, D, in k slides, $D_1, \ldots, D_k$. Then, ten models are learned, $m_1, \ldots, m_k$, each of them using nine slices for training, and a tenth slices for test. Therefore, at the end of the process, we have k evaluations, $e_1, \ldots, e_k$, one for each model. The estimated performance e of the complete model is assumed to be the average of the ten evaluations. The standard deviation over such average usually gives information about how uniform the k slices are. The inductive hypothesis ensures that if the model has been learned over a large amount of representative data, then the cross validation will return an accurate estimation. However, is that true in our case? To answer this question, we divide it in two different ones:

1. is the estimation valid for new problems in the same domains seen in the IPC 2011?
2. is the estimation valid for new problems in domains different to the IPC 2011 ones?

The answer to the first question depends on whether the problems used for learning are a representative set of problems in such domain. Since the IPC goal is to evaluate planners in different situations, problems are selected to cover such objective. Therefore, we can expect that a classical cross-validation is a good estimation. This conclusion would not be strong enough if the distribution of problems (20 instances per domain in IPC-2011) only covers a small space of the possible problems.

In the case of the second question, that is equivalent to ask whether the set of domains is representative of the set of all the possible domains that can be defined in PDDL. To evaluate this issue, we need to modify the evaluation method to a leave-one-out approach. In classical supervised learning, a leave-one-out approach is equivalent to a cross validation where k is set to the size of the data set. This mechanism estimates how the model will behave in the next example received. In planning, we are interested in evaluating how the model will behave in a new domain. Therefore, we can apply a leave-one-out over domains, instead of on the whole dataset. Specifically, if we have data from k domains, we learn k models with the data from $k-1$ domains, and evaluate all them over the $k-th$ domain. The average of the k evaluations is the estimation of the performance of the models in future domains. We call this evaluation process as *leave-one-domain-out*. The metrics used are:

- **Classification Accuracy** is the percentage of instances in a data set that a classification model correctly classifies (the ratio between the correctly classified instances and the total, multiplied by 100). The classification accuracy can be measured over different data sets (training or test sets), or estimated through different processes, like split, cross-validation, or leave-one-out.
- **Relative absolute error** is the ratio between the absolute error of a prediction of a model (difference between the predicted value and the expected one) and the expected output. Since it is a relative value, it is more independent of specific class values than no relative measures. Opposite to classification where maximum accuracy is searched, regression is a minimization problem of the prediction error. It also can be computed over different data sets or estimated through different procedures.

Predicting Planner Success

The goal of this task is to predict whether a planner will solve a given problem. As described previously, the target variable belongs to the domain $\{true, false\}$, meaning whether the input planner was able to solve the input problem in the corresponding domain.

Empirical Results. Tables 1 and 2 show the classification accuracy and the standard deviation obtained in both evaluation processes for the sequential satisficing and sequential optimal track respectively. In each row, we show the results of the different classification algorithms: J48, IBK for different values of k, and SMO.

Dataset	Cross Validation	Leave one Domain Out
J48	**88.75(1.05)**	59.14(12.13)
IBk -K 1	88.67(1.29)	60.83(10.13)
IBk -K 3	87.63(1.07)	60.58(11.76)
IBk -K 5	88.58(1.07)	**61.95(11.10)**
SMO	72.48(1.58)	61.34(10.10)

Table 1: Classification accuracy and standard deviation for predicting planner success in the sequential satisficing track with cross validation and leave one domain out evaluation

Dataset	Cross Validation	Leave one Domain Out
J48	**90.14(1.58)**	60.36 (23.69)
IBk -K 1	86.96(1.57)	60.36 (21.26)
IBk -K 3	87.81(1.81)	58.78 (21.66)
IBk -K 5	83.91(1.90)	60.86 (20.53)
SMO	79.96(2.30)	**67.41 (16.55)**

Table 2: Classification accuracy for predicting planner success in the sequential optimal track with cross validation and leave one domain out evaluation

Regarding the sequential satisficing track, the cross-validation results show that the best result is obtained using decision trees (J48), but IBK obtained fairly similar accuracy. These results reveals that we are able to achieve good classification accuracy in the cases where we are predicting planner success within already seen domains. However, the results for the leave-one-domain-out evaluation worsen considerably, showing that it is very difficult to generalize in the classification task for new unseen domains. Besides, both evaluation schemes show that we are able to create models that are better than a default or random classifier, which would obtain an accuracy of 50.7%. This percentage corresponds to the ratio of successful executions in the track. Concerning the sequential optimal track, algorithms performed similar than in the seq-sat track. As before, the leave-one-domain-out evaluation got worse results than cross validation, though the best result (67.4%) was higher than in seq-sat results.

We also have performed a feature selection process prior the generation of the models. Nevertheless, the default parameters produce an aggressive process. As a result, only one feature was left after the process (i.e., the planner). With only this feature, generalization is not possible, and the classification result was very poor (72.06 ± 1.52) independently of the algorithm). We have not applied other configuration or algorithms for feature selection. We postpone this study to future research.

We performed an additional evaluation to see if we can predict the performance of some planners better than others. For achieving this, we made a separate dataset for each planner (a total of 27). Then, we built the models with J48 (decision trees) and evaluated them using cross validation. Table 3 shows the results for the best, the worst and the average predicted performance. We also include the accuracy for the winner, LAMA. As we can see there is considerable gap in the classification accuracy. Another relevant observation is that it is hard to predict the performance of a planner based on an algorithm-portfolio, which internally could behave as different planners. This is the case of the Fd-autotune-2 which produced the worst model.

planners		accuracy
Minimum	Fd-autotune2	78,2
Maximum	Acoplan, Acoplan2	97,5
Average	–	88,5 ± 5,3
Track Winner	Lama-2011	81,4

Table 3: Different classification accuracies achieved with individual models

Semantic Analysis. Here we give a few examples of the knowledge that we could extract from the models learned. Specifically, we use the PART algorithm implemented in WEKA to obtain a set of decision rules that predict whether a planner will succeed or not. This algorithm firstly creates a decision tree, from which it generates the set of decision rules, which are then pruned (simplified). It also includes a parameter (minNumObj), which sets the minimum number of instances per rule allowed. This is, therefore, a prune parameter. If it is low, it permits models with a large number of rules (maybe sub-optimal due to over-fitting to the training data), which hence, are more complex to understand for a human. However, while this value grows, the models generated contains less rules (maybe sub-optimal due to the lack of expressiveness). In machine learning, balancing expressiveness capability and over-fitting risk is a main issue. Table 4 shows the importance of a correct balance. It shows the prediction capability of the models generated for different pruning values over the training set and estimated by the cross validation. This simple evaluation demonstrates that reducing pruning capabilities increases expressiveness (the generated model has more rules), but it over-fits to training data (difference between training success and cross-validation estimation is larger). A good balance, in this case, is a pruning value of 10, which maintains high accuracy (although significantly worse than the others) without over-fitting, with only 120 rules.

What is the looking of the rule sets generated? Are they really informative? The following is the rule at generated for the pruning parameter set to 1000, which is composed of only four rules.

```
Rule 1: NumberVerticesDTG > 168 AND
  AND STDInputEdgeHV <= 42.9906
  AND AVGInputEdgeDTG > 1.84576
  AND TotalEdgesCG > 699: false  (2160.0/926.0)

Rule 2: NumberVerticesDTG > 168 AND
AVGInputEdgesDTG > 1.84576: true (2349.0/1024.0)

Rule 3: STDInputEdgeHV <= 55.8043: false (1566.0/560.0)

Rule 4: true (1485.0/459.0)
```

Pruning parameter	training success	cross-validation success	number of rules
1	94.9	88.8 ± 1.15	287
2 (default)	94.4	88.6 ± 0.83	233
10	89.9	85.8 ± 1.41	120
100	78.5	75.7 ± 1.44	31
1000	60.72	60.3 ± 1.55	4

Table 4: Different results of PART depending on the pruning parameter (minimum number of instances per leaf) and the evaluation mechanism (over training data or estimated with cross-validation procedure.

The way to read the rules is the following. First, we check the conditions of the first rule: ($NumberVerticesDTG >$ 168 AND $STDInputEdgeHV <=$ 42.9906 AND $AVGInputEdgeDTG >$ 1.84576 AND $TotalEdgesCG >$ 699). If the condition is true, the output is false (i.e. the problem can not be solved). The values $(2160.0/926.0)$ indicates the number of instances that satisfy the condition and that belongs to the class 'false' and 'true' respectively. Therefore, those values provide an idea of the coverage and success of the rule. If an instance does not satisfy Rule 1 condition, Rule 2 is checked, and so on. The last rule returns a default value in case the input instance does not satisfy any previous rule.

We want to highlight that this rule set is so reduced that it does not ask for many of the input features that describe the instances. It is so reduced that it does not ask for the planner. However, it already gives information about the problems, and it seems that to have a number of variables in the DTG higher or lower than 168 is an important feature of the problems, since that condition appears in the the initial rules. In fact, $NumberVerticesDTG$ is a feature that appears in all the decision trees that PART construct to generate the rule sets, independently of the pruning parameter. For instance, when the minimum number of instances per leaf is 2, the first question that the decision tree generated makes is whether $NumberVerticesDTG$ is larger than 106. Then, it asks for the planner.

For higher expressiveness, we need to reduce pruning capabilities, and we can try to understand what are the features that characterize whether a problem will be solved or not. For instance, this is the rule set that characterize the problems that LAMA is not able to solve (for a pruning parameter set to 2):

```
MAXInputWeightDTG > 3: true (166.0/1.0)
HVRatio > 0.970588 AND
  STDInputEdgeCG > 0.026307:true (27.0)
MAXInputWeightHV > 70 AND
  AVGInputWeightHV <= 11520: true (37.0/1.0)
WVRatio > 16896: false (9.0)
MAXInputWeightCG <= 6480: true (20.0/1.0)
goals <= 14: false (4.0)
otherwise: false

literals <= 9801 AND
MAXInputWeightHV > 60: true (164.0/2.0)
```

```
NumberVariablesCG > 58 AND VERatio > 0.00575 AND
MAXInputEdgeHV <= 8: true (51.0)

Goals <= 50 AND
NumberVerticesDTG > 130 AND NumberVariablesCG > 11 AND
MAXInputEdgeDTG <= 1210: false (22.0)

MAXInputEdgeDTG <= 1397: true (36.0)

: false (7.0/1.0)
```

Can this rule set help researches to focus the development or improvement of their planners? That is something that only them can answer but, at least, these rules can give some clues, since they characterize the solved and the unsolved problems. A similar study could be easily performed for any of the planners of the competition, although we omit them due to lack of space.

Predicting Execution Time

Once we know whether the input planner will be able to solve the input problem (using the predictive model learned in the previous phase), the goal of this task is to predict the time a planner needs to find the solutions.

Empirical Results. As explained before, in the sequential satisfying track we have created models for predicting the time invested in finding the first, the median and the best solution. The results obtained for this track are shown in Table 5. We follow the same evaluation scheme of the classification case. The table shows the relative absolute error and standard deviation after the validation process. As in the classification case, results worsen when estimations are performed with the leave-one-domain-out scheme. Besides, IBK with different values of K had less degradation of the estimated errors, showing that this technique could be better when making predictions on new domains.

Regarding the sequential optimal track, if planners report a solution, it should be optimal. Therefore, it only make sense to create a model to predict a time for finding the solution, which is obviously the best one. Table 6 shows the results for both evaluation schemes. Each value corresponds to the relative absolute error and the standard deviation after the evaluation process.

Dataset	Cross Validation	Leave Domain Out
M5Rules	67.08(7.63)	213.87 (231.95)
IBk -K 1	**59.74(8.37)**	141.54 (47.40)
IBk -K 3	59.99(6.32)	**123.37 (11.26)**
IBk -K 5	63.59(6.38)	127.21 (10.96)
SMOreg	66.84(5.71)	15151.04 (54178.83)

Table 6: Relative absolute error (% percentage) and standard deviation of predicting the time planners will need to find a solution in the sequential optimal track

Semantic Analysis. Semantic analysis of regression models is more complex than for classification, because regression models include mathematical models difficult to interpret. In our case, we will analyze a few of them. For this study, we use the algorithm M5Rules, which generate regression rules. Table 7 shows the results of an experimental process equivalent to the one described in Table 4.

Pruning parameter	Training Error	Cross-validation Error	Number of Rules
1	71.26	71.26 ± 6.82	5
4 (default)	71.26	73.38 ± 6.82	5
10	66.68	71.63 ± 3.08	11
100	70.42	71.35 ± 2.30	11
1000	83.86	77.33 ± 2.02	4

Table 7: Different results of M5Rules depending on the pruning parameter and the evaluation mechanism (over training data or estimated with cross-validation procedure.

As in the case of classification, we can see that a value of 10 is a good value for the pruning parameter. However, in this case, the number of rules generated is not so different for different values, as well as the prediction capabilities. Again, we analyze the looking of the rule set generated for LAMA planner In this case, M5Rules generates only 2 rules, each of them with its linear regression model. From a semantic point of view, we can think that the regression models generated could give an idea about the importance of the features in the prediction model. For instance, in this case the output value changes a lot with different values of the $VERatio$ and $MAXInputWeigthDTG$ features (in rule 1), and with $AVGInputEdgeHV$, $STDInputEdgeCG$ and $STDOutputEdgeCG$ (in Rule 2), showing that, for LAMA, the output time is very sensitive to small differences in those values. The relative absolute error of that rule set is 78.74%.

```
Rule: 1
IF STDInputWeigthCG <= 13305.8
MAXInputWeigthDTG <= 4.5
WERatio > 5.52
THEN bestTime =
0.17 * objects
- 0.0764 * goals - 0.0002 * TotalWeigthCG
- 52.7647 * VERatio - 0.0099 * WERatio
- 0.0676 * WVRatio  + 0.7708 * AVGInputEgdeHV
- 0.4982 * STDOutputEdgeCG + 0.0043 * STDInputWeigthCG
- 0.0002 * TotalWeigthDTG - 0.0226 * NumberVerticesDTG
+ 0.0015 * TotalEdgesDTG + 0.0703 * STDInputEdgeDTG
+ 0.0314 * MAXOutputEdgeDTG - 8.4912 * MAXInputWeigthDTG
+ 178.6288 [79/44.805%]

Rule: 2
bestTime = 0.0784 * literals  + 0.0003 * TotalWeightCG
+ 6.8833 * AVGInputEdgeHV - 0.0053 * STDInputEdgeHV
+ 11.8598 * STDInputEdgeCG - 11.4135 * STDOutputEdgeCG
+ 193.2316 [171/85.323%]
```

These analysis are only small examples of how powerful data-mining studies can be for supporting decisions in solving planning problems. For instance, one classical decision in heuristic search based planners is, once a solution is found, to decide whether to wait for a new solution is interesting or not. To make this decision, to predict the required

Dataset	Cross Validation			Leave Domain Out		
	First Time	Medium Time	Best Time	First Time	Medium Time	Best Time
M5Rules	73.81(4.78)	74.02(3.90)	73.66(3.61)	17204.81(60518.16)	1492.24(2798.89)	985.64(2200.93)
IBk -K 1	59.84(5.15)	65.25(5.28)	67.57(4.07)	87.94(30.76)	91.12(29.39)	93.66(23.38)
IBk -K 3	**55.05(3.72)**	**60.02(4.00)**	**62.98(3.12)**	**79.31(28.27)**	89.87(31.70)	85.96(22.26)
IBk -K 5	56.61(3.66)	60.93(3.51)	64.39(3.00)	92.12(29.73)	**89.70(26.57)**	**85.57(19.21)**
SMOreg	60.18(4.06)	64.08(3.65)	69.50(2.87)	835.17(2264.22)	184.10(165.75)	907.32(2620.74)

Table 5: Relative absolute error (% percentage) and standard deviation of predicting the time planners invested in finding the first, median and best solution in the sequential satisficing track.

time to obtain the best plan is required. We can predict that value for LAMA with a mean absolute error of only 217 seconds.

Related Work

The construction of models to predict the performance of planners is not a novel idea. Roberts and Howe (2009) showed that model learned from planners' performance on known benchmarks up to 2008 get high accuracy when predicting whether a planner will succeed or not. They use 19 features extracted from the domain and problem definition. They also proposed to use some features from the causal graph, but they did not find them relevant for the classification task. The main difference with our approach is that we include features also from the domain transition graphs and most of our features come from the ground instantiation of the problem. Additionally, results from both works are not comparable because they do not have common planners or instance sets. Besides, they found that domain features are the more relevant ones, and models are indeed basing their planner success prediction on implicitly predicting the domain. This is a good insight regarding the available data; however, it does not seem a general rule, since one can imagine large planning problems for which any planner will fail to solve it. We think that models may exploit instance features in order to determine the relative hardness between instances of the same domain. To achieve this goal, further investigation is needed and performance data should be collected from instance sets with more diversity than the one that could be derived in 20 instance per domain used in the IPC.

In a more general scope, the prediction of a solver performance is a research topic of interest since it is one of the techniques for creating algorithm portfolios (Xu et al. 2007; Gagliolo and Schmidhuber 2006). The idea consists of learning empirical hardness models based of instance features that can be computed efficiently. Then, whenever the portfolio tries to solve a new instance, the learned models predict the set of solvers that are likely to solve the instance, so the available time is scheduled between the selected solvers using a criteria based on the prediction confidence. This approach can make a per-instance decision of which portfolio configurations are likely to perform the best. However, the successful portfolios in automated planning (Gerevini, Saetti, and Vallati 2009; Fawcett et al. 2011) are only able to use the configuration of the best average performance in a set of training problems. Therefore, given a new set of instances, these portfolios tries to solve them using a fixed configuration (i.e., the best over past benchmarks) or a per-domain configuration, if there is an available example set of a particular domain. This per-domain decision for configuring portfolios is the approach followed by the cited planners in the learning track of the IPC. We argue that one can use the models presented in this work (or other similar ones) to create a per-instance configurable planning portfolio.

Conclusions and Future Work

We have presented an analysis of the IPC-2011 results following a data mining methodology. With this analysis we have given some insights about the performance of planners, in addition to the general results reported by the IPC organizers. We have built classification models for predicting whether a planner will succeed or not in a given problem, and regression models for predicting the time a planner will need to solve the problem.

We have presented the *leave-one-domain-out* evaluation scheme. This kind of evaluation is an alternative to the standard cross-validation if one want to estimate how good the learned models are, in the situations where problems belong to an unknown domain. As we have seen in the results, we can get good classification accuracy when we are dealing with problems of known domains, but it seems that this does not hold in unknown domains. We expect that if we reproduce the same experiment with more domains, the models would generalize better.

We have introduced a set of elaborated features that come from the causal graphs and the domain transition graphs. The results have shown that these features are relevant for partially characterizing the complexity of planning problems. Besides, these features are easy to compute, therefore they can be extracted in a pre-processing stage of a planning process, and then used to query a learned model for deciding the set of planners and the set of times in an instance-based configurable portfolio. In the near future we plan to continue our research in this direction.

Acknowledgments

This work was supported by several Spanish projects: TIN2008-06701-C03-03, TIN2011-27652-C03-02, TIN2010-08861-E and TRA2009_0080.

References

Backstrom, C., and Nebel, B. 1995. Complexity results for SAS+ planning. *Computational Intelligence* 11:625–655.

Chapman, P.; Clinton, J.; Kerber, R.; Khabaza, T.; Reinartz, T.; Shearer, C.; and Wirth, R. 2000. Crisp-dm 1.0 step-by-step data mining guide.

Cristianini, N., and Shawe-Taylor, J. 2000. *An Introduction to Support Vector Machines*. Cambridge University Press.

Fawcett, C.; Helmert, M.; Hoos, H.; Karpas, E.; Roger, G.; and Seipp, J. 2011. Fd-autotune: Domain-specific configuration using fast downward. In *Booklet of the 7th International Planning Competition*.

Frank, E., and Witten, I. H. 1998. Generating accurate rule sets without global optimization. In *Proceedings of the Fifteenth International Conference on Machine Learnin*.

Gagliolo, M., and Schmidhuber, J. 2006. Learning dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence, 47* 3(4):295–328.

Gerevini, A.; Saetti, A.; and Vallati, M. 2009. An automatically configurable portfolio-based planner with macro-actions: PbP. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS-09)*.

Helmert, M. 2006. The fast downward planning system. *JAIR* 26:191–246.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173:503–535.

López, C. L. 2011. The seventh international planning competition documentation. Technical report, Universidad Carlos III de Madrid, Madrid, Spain. http://www.plg.inf.uc3m.es/ipc2011-deterministic/FrontPage/Software.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39:127–177.

Roberts, M., and Howe, A. 2009. Learning from planner performance. *Artificial Intelligence* 173:536–561.

Witten, I. H., and Frank, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*. 2nd Edition, Morgan Kaufmann.

Xu, L.; Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2007. Satzilla-07: The design and analysis of an algorithm portfolio for SAT. In *Proceedings of the 13th CP Conference*.

How Good is the Performance of the Best Portfolio in IPC-2011?

Sergio Núñez and **Daniel Borrajo** and **Carlos Linares López**
Departamento de Informática, Universidad Carlos III de Madrid
Avda. de la Universidad, 30. 28911 Leganes (Madrid). Spain
sergio.nunez@uc3m.es, dborrajo@ia.uc3m.es, clinares@inf.uc3m.es

Abstract

In recent years the concept of sequential portfolio has become an important topic to improve the performance of modern problem solvers, such as SAT engines or planners. The PbP planner and more recently Fast Downward Stone Soup have shown to be successful approaches in automated planning. However neither theoretical analysis nor formal definitions about sequential portfolios have been described. In this paper, we have focused on studying how to evaluate the performance of planners defining a baseline for a set of problems. We present a general method based on Mixed-Integer Programming to define the baseline for a training data set. In addition to prior work, we also present a brief empirical analysis of the utility of training problems to configure sequential portfolios.

Introduction

The AI community constantly designs faster and more efficient heuristics and algorithms for solving automated planning problems. However, it has not been able to develop a single planner that dominates all others for classical automated planning (Roberts and Howe 2009). Also if a planner does not solve a planning problem quickly, it is likely that it will not solve it at all (Howe and Dahlman 2002). Both facts led to a new concept of planner called portfolio. This is based on the following idea: several planners are executed in sequence with shorter timeouts, hoping that at least one of them will find a solution in its allotted time. This technique has been shown to be a successful approach in classical automated planning.

Recently, different approaches to build planner portfolios have been introduced. Fast Downward Stone Soup (Helmert, Röger, and Karpas 2011) and PbP (Gerevini, Saetti, and Vallati 2009) are some interesting examples. The first one is a sequential portfolio of domain-independent planners. There are two main versions of FDSS, one for optimal planning and one for satisficing planning. FDSS was configured using a set of training problems from the IPCs[1] 1998-2008 and a set of planning algorithms. The algorithm uses a hill-climbing search in the space of portfolios. The search starts from an initial portfolio which assigns 0 seconds to each planning algorithm. At each iteration, it generates a set of possible successors (which are new portfolios) to the current portfolio. Each successor is generated by increasing the runtime of one planning algorithm by a small ratio of the total time. The best successor is selected as the current portfolio to the next iteration. Once the total time has been reached, the algorithm finishes the configuration of the sequential portfolio.

The PbP planner is a portfolio-based planner with macro-actions, which automatically configures a sequential portfolio of domain-independent planners. The configuration relies on some knowledge about the performance of the planners in the portfolio for a specific domain and the observed utility of automatically generated sets of macro-actions. Nevertheless, PbP can be used without this additional knowledge. Thus, the portfolio schedules all planners by a round-robin strategy and it assigns the same time slot to the randomly ordered planners. If PbP uses the knowledge computed for the current domain, it only selects a cluster of planners (which is sorted by performance) for configuring the portfolio. The macro-actions sets are not always used by the planners. Only if the set of macro-actions improves the performance of the planner for the current domain, the planner will use the macro-actions set.

Commonly, most of the approaches of planner portfolios are evaluated by comparing the score of the new sequential portfolio on the last IPC with the score reached by the participating planners. We believe this comparison is not a good measure to evaluate the performance of new portfolios because it does not show how far the portfolio is from the optimal configuration in this particular set of planning tasks. Therefore, in this paper we propose a general method based on Mixed-Integer Programming (MIP) to configure the optimal static sequential portfolio for some training data (problems and planners). This optimal portfolio defines a baseline for the training data set. Using this baseline, the performance of any planner can be analyzed. To show an example, we have applied this approach to the IPC 2011 sequential optimization (seq-opt) track.

Additionally, we have performed an empirical analysis of the utility of training problems to configure a sequential portfolio. To carry out this analysis we have used the previous example, which configures the optimal static sequential portfolio for the IPC 2011 seq-opt track.

This document is organized as follows: first in Section 2, we define the concept of the static sequential portfolio.

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

In Section 3 we show the motivations of our work and we describe our proposals. In Section 4 we describe the MIP model. In Section 5 we show the experimental results of our approach over IPC 2011 seq-opt track. Finally, in Section 6 we conclude and introduce future work.

Portfolio Framework

In this work, we focus on static sequential portfolios for classical automated planning. A static sequential portfolio can be defined as a configuration of planners, where each one has been assigned a slice of time. Formally, a static sequential portfolio is a sorted set C where each $c_i \in C$ is a pair $< p, t >$:

- p is a planner, defined as a tuple (A, H, P) where A is a non empty set of search algorithms (A*, hill-climbing, ...), H is a non empty set of heuristics (Merge&Shrink (Helmert, Haslum, and Hoffmann 2007), FF (Hoffmann 2003), ...) and P is the policy to choose which search algorithm $a_i \in A$ and heuristic $h_i \in H$ to run at all times.
- t is the time allotted to p.

Assuming that two or more planners cannot be executed at the same time, the only way to run them is executing every planner p in a given time t, where $< p, t >$ is a pair $c_i \in C$.

Motivation

Currently, the field of sequential portfolios is a hot topic. There are many open issues, like the anytime nature for satisfying planning, the order of the execution sequence, etc. We have focused on studying how to evaluate the performance of planners and subsequently on analyzing the utility[2] of training problems to configure sequential portfolios.

Usually, the performance of new sequential portfolios is evaluated by running them on the last IPC. But this comparison does not show how good a portfolio is. It only shows if the portfolio is better or worse than the participating planners in that particular set of planning tasks. Even if it is better than the winner, the measure does not show the quality of the portfolio. We propose instead to define the best possible sequential portfolio for a set of problems using MIP and evaluate any planner (including portfolios) against it.

Basically, to evaluate a solver (which can be a new single planner, a sequential portfolio of new planners, etc.), we propose to define the best possible sequential portfolio for a set of problems and run them on this particular set of problems. If the solver reaches a higher score than the best possible portfolio, the solver can be said to be an improvement, otherwise the quality of the solver will be said to be low (even though the solver was better than the winner of the last IPC because it can be far from the best possible portfolio).

With the purpose of configuring a sequential portfolio many approaches use two sets of training data: one for planners, and one for training problems. Usually, the first one is a small set of heterogeneous and modern planners (e.g., PbP incorporates seven planners and FDSS for optimal planning used a set of eleven planners). However there are some approaches that build new solvers to configure the portfolio. Commonly, the second one is a large set of training problems from past IPCs (FDSS used a total of 1163 training instances from IPC 1998-2008). The instances of this set are very important to achieve high-performance. However, there is neither an algorithm to find the best training problem set nor a definition of the best problems to build this training set.

The second issue we would like to analyze in this paper is the following: in order to achieve the optimal configuration for the sequential portfolio, it is not necessary to make so many experiments since in many cases the data sets chosen are not very informative. A good training problem set should contain only a few training instances with high utility. We propose a classification of problems into three categories sorted by difficulty. The first one contains problems of high difficulty. These have no utility because no planner is able to solve any of them. The category labeled as medium difficulty is composed of problems that are only solved by a few planners. These problems are the most useful ones. And the last one contains problems with limited utility, because most planners solve them all.

Mixed-Integer Programming model

MIP is the best choice for our purpose for three reasons. First, the search of the optimal static sequential portfolio from a set of training data (planners and problems) can be modeled as a MIP task. Second, this technique ensures optimal solutions. And third, MIP is fast, efficient and it allows us to create and modify constraints quickly, which brings flexibility. Besides, a linear combination of existing planners is a reasonable threshold for judging whether a new portfolio results in an improvement or not.

We have used GLPK[3] for solving the MIP task discussed in this section. This package supports the GNU MathProg modeling language[4], which divides our model into four sections: *Parameters*, *Variables*, *Objective function* and *Constraints*.

The training data set contains a set of n planners S and a set of m training instances I. The *parameters* store the input data which is generated by running every planner $p \in S$ with every instance problem $i \in I$. The following *parameters* have been defined in our MIP model:

- $q(p, i)$. Plan quality found by the planner $p \in S$ for the training problem $i \in I$. If the planner p does not solve the problem i, the value of plan quality will be zero.
- $r_t(p, i)$. Time spent by the planner $p \in S$ to solve the training problem $i \in I$. If the problem i is not solved by the planner p in a given time, the parameter value will be the given time limit to solve the training problem i by the planner p.

[2]We consider the utility of training problems to be the knowledge provided to configure high-performance portfolios as discussed in this paper.

[3]http://www.gnu.org/software/glpk/

[4]www.cs.unb.ca/ bremner/docs/glpk/gmpl.pdf

- $m(p, i)$. Maximum memory used by the planner $p \in S$ to solve the training problem $i \in I$.

The following decision *variables* have been defined in our model:

- $solved\_by_{pi}$. This binary variable indicates whether the plan found by the planner $p \in S$ for the training problem $i \in I$ is considered to configure the sequential portfolio or not.
- $quality_i$. Plan quality found by the sequential portfolio for the training problem $i \in I$.
- $time_p$. It is the output variable, which shows the allocated time to each planner $p \in S$ in the sequential portfolio.
- $memory$. Maximum memory used by the sequential portfolio.

With the purpose of defining the *objective function* of the MIP task, we have considered time, memory and plan quality for every training instance and every planner in the training data set. Time and memory are computed as time spent and memory used by any planner to solve any problem, divided by time and memory limit. Note that using normalized values, the *objective function* has no specific bounds to time and memory. Also, we compute the plan quality in the range [0,1] for each training instance. If the portfolio does not solve an instance, the plan quality of this problem is 0. The plan quality of a solved instance is computed as the lowest solution cost found by any planner in the training data set, divided by the best solution achieved by the portfolio. Hence, the objective function is defined as:

$$\begin{aligned} maximize: \quad & w_1(\textstyle\sum_{i=0}^{m} quality_i) \\ + \quad & w_2(1 - \textstyle\sum_{p=0}^{n} time_p) \\ + \quad & w_3(1 - memory) \end{aligned}$$

The function for evaluating the participating planners in the IPC only focuses on plan quality (equivalent to coverage for optimal planning). We propose an *objective function* that makes it possible to balance maximizing the score achieved by the portfolio (i.e. the sum of the plan quality found for every training problem), the time spent and/or memory used by the portfolio with different weights w_1, w_2 and w_3. Thus, using our *objective function* we are able to maximize the quality of the obtained solutions, as well as the performance of the sequential portfolio measured in time and memory consumption. For instance, selecting $w_1 = 1$, $w_2 = 0$ and $w_3 = 0$, only the overall score is optimized. However, if we increase w_2 and/or w_3, we will obtain a sequential portfolio that achieves the optimal score and optimizes the time spent and/or memory used by the portfolio.

We have defined a set of *constraints* in our MIP model. The *constraints* (1) and (2) define the time and memory bounds to solve each training instance:

$$\sum_{p=0}^{n} time_p <= 1 \tag{1}$$

$$memory <= 1 \tag{2}$$

The MIP task analyzes plan quality, time and memory used by all planners for every training problem. For each one of these problems, the task should select at most one planner to solve it. To avoid selecting two or more planners to solve the same instance, the *constraint* (3) was used. This constraint limits the number of planners that are considered to solve each training problem:

$$\sum_{p=0}^{n} solved\_by_{pi} <= 1, \forall i \tag{3}$$

The following *constraints* compute the sum of all time bounds allotted to all planners in the portfolio (4), the total score achieved by the portfolio (5) and the maximum memory used to solve a training problem (6):

$$time_p >= solved\_by_{pi} \times r_t(p, i) \tag{4}$$

$$quality_i = \sum_{p=0}^{n} solved\_by_{pi} \times q(p, i) \tag{5}$$

$$memory >= solved\_by_{pi} \times m(p, i) \tag{6}$$

Experimental setup and results

In order to define a baseline for the IPC 2011 seq-opt track, we have applied our MIP model over all IPC 2011 seq-opt problems I and a set of planners S. This track contains 14 domains[5] with 20 problems each.

The set of planners S contains all planners from the IPC 2011 seq-opt track, excluding portfolios and adding planners from participating portfolios that were not taking part as single planners: both versions of merge-and-shrink heuristics from FDSS and the A* algorithm with the blind heuristic, as shown in Table 1.

Planner	Authors	Source
Blind	Silvia Richter, et al.	FDSS-2 planner
BJOLP	Erez Karpas, et al.	IPC 2011
CPT4	Vincent Vidal	IPC 2011
FD Autotune	Chris Fawcett, et al.	IPC 2011
Fork Init	Michael Katz, et al.	IPC 2011
Gamer	Peter Kissmann, et al.	IPC 2011
IFork Init	Michael Katz, et al.	IPC 2011
LM-cut	Malte Helmert, et al.	IPC 2011
LMFork	Michael Katz, et al.	IPC 2011
M&S-bisim 1	Raz Nissim, et al.	FDSS-1 planner
M&S-bisim 2	Raz Nissim, et al.	FDSS-1 planner
Selective Max	Erez Karpas, et al.	IPC 2011

Table 1: List of optimal planners used.

To generate the input data to our MIP model, we have executed every planner $p \in S$ with every instance problem $i \in I$ under the same conditions of the IPC 2011 seq-opt

[5]These domains are: barman, elevators, floortile, nomystery, openstacks, parcprinter, parking, pegsol, scanalyzer, sokoban, tidybot, transport, visitall and woodworking.

track, this is, allowing each planner to solve every planning task with a time bound equal to 1800 seconds and a memory bound equal to 6 GB of memory. From all generated data, we have only considered the runtime $r_t(p, i)$, the maximum memory used $m(p, i)$, and whether the problem has been solved or not $q(p, i)$.

The score of the IPC 2011 seq-opt track only measures the number of solved problems. But we also want to optimize the time spent by the sequential portfolio, keeping the highest number of solved problems. However, we cannot know in advance a good configuration of the weights w_1, w_2 and w_3 to this goal. Therefore, we have run a set of systematic tests with different configurations to analyze the results and have picked up the best one.

Good values for our target are $w_1 = 1$, $w_2 = 0.04$ and $w_3 = 0$ as shown in Table 2. This configuration solves 200 instance problems and spends less time than other configuration that also solve 200 problems. Hence, we have selected these weights to configure the optimal static sequential portfolio for the IPC 2011 seq-opt data, which is shown in Table 3.

The execution sequence of this sequential portfolio (and of all sequential portfolios obtained by the MIP model) has not been defined by the MIP task. It only assigns an execution time to each planner, which can be zero. The definition of the execution sequence is simply based on the initial order in which the planners are specified. Given that the optimal track of the IPC only considers coverage, the order does not affect the result.

w_1	w_2	w_3	Solved problems	Time	Memory
1	0	0	200	1800	5696.67
1	0.04	0	200	1705	5696.67
1	0.2	0	186	548	5380.7
0.8	0.4	0	175	302	5305.7
1	0.2	0.2	166	265	778.61
1	0.6	0	163	168	1327.25
1	0.4	0.4	156	173	381.86
0.8	0.4	0.8	151	163	247.56
0.6	0.8	0	151	87	1207.9
0.6	0.4	0.6	147	115	247.56
0.6	0.8	0.8	140	72	243.65
0.6	0.8	1	126	49	99.3

Table 2: Results of the most representative configurations for MIP model. For each configuration, the table shows the number of solved problems, the time spent (seconds) and memory used (MB) by the obtained sequential portfolio.

So far, the reference used to measure the performance of new sequential portfolios for optimal planning was FDSS, the winner of IPC 2011 seq-opt track. FDSS solved 185 problems in the last IPC and this result is very likely to be used in the future to decide whether a new portfolio is better or not. However, it does not show how far the portfolio is from the optimal configuration in this particular set of planning tasks. Instead, we propose to use the number of problems solved by a linear combination of the entrants of the IPC 2011 as the score to beat in practice. This results from the consideration that all planners in the last IPC are state-of-the-art planners and that a linear combination of their performance is a reasonable estimator of the expected performance of state-of-the-art planners.

Planner	Allotted time (s)
CPT4	1
LM-cut	55
M&S-bisim 1	138
M&S-bisim 2	170
IFork Init	104
Gamer	1237
Total Time	1705

Table 3: Configuration of optimal sequential portfolio.

Empirical analysis of the utility of training problems to configure sequential portfolios

Once the optimal static sequential portfolio for the IPC 2011 seq-opt data has been configured, we have analyzed the utility of the training problems. To perform this task we have defined twelve sets of training problems, where each one contains instances from the IPC 2011 seq-opt that were solved by a maximum number of planners. The first set is composed of all problems that were solved by at most one planner. The second set is composed of all problems that were solved by at most two planners, and so on. We have defined twelve sets because we have used a set of twelve planners to configure the optimal static sequential portfolio for the IPC 2011 seq-opt track. Figure 1 shows the problem distribution for the twelve sets of training problems.

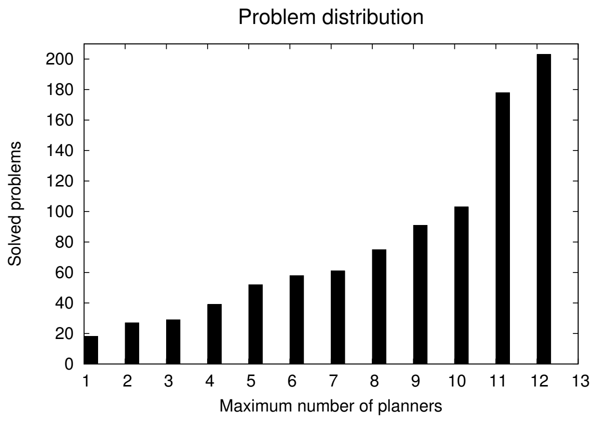


Figure 1: Problems distribution for the twelve sets of training instances.

We have executed our MIP model with each of the twelve training problem sets. The results in Table 4 show the same performance (measured in coverage, time and memory) for the sequential portfolios obtained using all sets of training problems except for the first one (which is composed of all problems that were solved by at most one planner). These sequential portfolios have the same configuration, which is the optimal configuration for the IPC 2011 seq-opt track, as shown in Table 5. Therefore, the minimum set of training problems necessary to configure the optimal portfolio for the competition contains only 27 problems (those which were solved by at most two planners). This fact empirically confirms our initial hypothesis: not all training problems have the same utility and the best set of training problems should contain few instances with a high utility. Based on our initial classification, those instances should belong to the second category: problems with medium difficulty.

Max. Planners	Solved problems	Time	Memory
1	16/18	1499	6144
2	24/27	1705	5696.67
3	26/29	1705	5696.67
4	36/39	1705	5696.67
5	49/52	1705	5696.67
6	55/58	1705	5696.67
7	58/61	1705	5696.67
8	72/75	1705	5696.67
9	88/91	1705	5696.67
10	100/103	1705	5696.67
11	175/178	1705	5696.67
12	200/203	1705	5696.67

Table 4: Results of our model using the twelve sets of training problems. For each set, identified by the maximum number of planners that solve its instances, the table shows the ratio between number of solved problems and size of the problem set, the time spent (seconds) and memory used (MB) by the obtained sequential portfolio.

Planner	Allotted time (s)	
	First training set	Remaining sets
CPT4	1	1
LM-cut	0	55
M&S-bisim 1	0	138
M&S-bisim 2	157	170
IFork Init	104	104
Gamer	1237	1237

Table 5: Sequential portfolios obtained using the twelve sets of training problems.

Conclusions and future work

We have presented a general method based on MIP to define the baseline for a specific set of problems, against which the real performance of planners can be measured. We have applied this method over all IPC 2011 seq-opt problems and we have shown that the real challenge consists of generating portfolios/planners that could solve more than 200 problems when being trained with this data set. The reason is that any portfolio solving less problems falls below the results achievable by learning a portfolio with the solution of a MIP task.

In addition to this prior work, we have performed an empirical analysis to examine if all training problems have the same utility to configure sequential portfolios. The results show that not all problems have the same utility. We have reached exactly the same configuration when running the MIP task with either 27 problems or 280 problems. The sequential portfolio obtained (under the same conditions of the IPC 2011 seq-opt track) is the optimal portfolio for both sets of problems.

In the future we will perform a theoretical analysis based on sensitivity analysis of training problems to determine the accuracy of our hypothesis.

Additionally, we will try to analyze the utility of sequential portfolios for different time limits. Commonly, the sequential portfolios have only been tested using the IPC rules. These rules define a time limit of 1800 seconds to solve every problem. Some sequential portfolios have shown high performance using this time limit. But if these sequential portfolios are executed with a different time limit, their performance is just unknown. In particular, we are interested in finding out whether it is possible or not to derive sequential portfolios that solve at least as many instances as all entrants in the same amount of time or even less.

Finally, we want to study the order of the execution sequence for the optimal static sequential portfolio. We have obtained a randomly sorted sequential portfolio that solves 200 problems of the IPC 2011 seq-opt track. To solve each of these problems, the portfolio spends at most 1705 seconds. This runtime depends on the execution sequence. Therefore, we will analyze whether there is an order of the execution sequence for spending, on average, the minimum execution time.

Acknowledgments

This work has been supported by the INNPACTO program from the Spanish government associated to the MICINN project IPT-370000-2010-8 and different Spanish research departments through projects TIN2008-06701-C03-03, TIN2011-27652-C03-02, TIN2010-08861-E and TRA2009_0080.

References

Gerevini, A.; Saetti, A.; and Vallati, M. 2009. An automatically configurable portfolio-based planner with macro-actions: Pbp. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, (ICAPS 2009)*. AAAI.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, 176–183.

Helmert, M.; Röger, G.; and Karpas, E. 2011. Fast downward stone soup: A baseline for building planner portfolios. *In ICAPS 2011 Workshop on Planning and Learning* 28–35.

Hoffmann, J. 2003. The metric-ff planning system: Translating ”ignoring delete lists” to numeric state variables. *Journal of Artificial Intelligence Research* 20:291–341.

Howe, A. E., and Dahlman, E. 2002. A critical assessment of benchmark comparison in planning. *J. Artif. Intell. Res. (JAIR)* 17:1–3.

Roberts, M., and Howe, A. E. 2009. Learning from planner performance. *Artif. Intell.* 173(5-6):536–561.

"Type Problem in Domain Description!" or, Outsiders' Suggestions for PDDL Improvement

Robert P. Goldman and **Peter Keller**
SIFT, LLC
Minneapolis, MN USA
{rpgoldman,pkeller} at sift.net

Introduction

The International Planning Competition has been tremendously successful in advancing the state of the art in automated planning systems. Planning systems have become far more capable, and are now readily available to interested users. Recent developments in techniques for compiling problems to PDDL (Baier, Fritz, and McIlraith 2007; Alford, Kuter, and Nau 2009; Baier, Bacchus, and McIlraith 2007; Palacios and Geffner 2007, e.g.,) further extend the utility of PDDL planning systems. We feel that the state of the art is approaching that of techniques such as mathematical programming solvers, which can be used to solve a wide variety of problems. We have ourselves been experimenting with such uses of PDDL planners, in areas including cyber attack planning for adversary plan recognition and searching for plan counterexamples (Goldman, Kuter, and Schneider 2012). Our experience in these efforts guides our remarks here.

As with mathematical programming, the gateway to use of planning systems is the modeling language, in this case PDDL (Ghallab et al. 1998; Edelkamp and Hoffmann 2004; Hoffmann and Edelkamp 2005; Haslum 2011; Fox and Long 2003; Long and Fox 2003; Gerevini and Long 2005). To achieve the potential of wide use, the modeling language needs to do more to meet the users. In particular, the modeling language needs to have better expressive power, must be easier to author, and should be better standardized.

In this paper we describe obstacles to planner employment from outside the IPC community, based on our experiences attempting to use existing PDDL planners. We hope that this different perspective will bring to the fore concerns that might otherwise be backgrounded. In particular, as outside users, the ability to design new domains and problems is a central concern to us, whereas in the IPC proper, difficulties in formulating problems, and especially difficulties in modeling pre-existing problems in PDDL, are amortized over the community as a whole. Members of the community whose efforts focus on efficiently solving problems which they are presented may not feel these modeling issues as keenly. An outside perspective may also raise some issues that have fallen into the background for those more accustomed to working in the IPC community.

In the following sections, we discuss some challenges posed by the current state of the PDDL standard, ambiguities in PDDL syntax, issues with the PDDL type system, and the problem of planner-specific PDDL dialects. We make some modest suggestions about how the state of affairs in these areas could be improved. We have purposely kept these suggestions modest and incremental, particularly when discussing expressive limitations of PDDL. We have tried to be sensitive to the resource limitations of the community; interested readers will have to look elsewhere for more utopian suggestions.

PDDL standard issues

A primary challenge with the PDDL standard is that it is only available through a historical reading of the available documents. All versions of the specification since the original (Ghallab et al. 1998) are substantially patches to the original document. The most complete of the successors is the excellent article on PDDL 2.1 (Fox and Long 2003), but even this document appeals to the original specification, and focuses primarily on the extensions in PDDL 2.1, notably the addition of durative actions. The PDDL 3.1 documents (Haslum 2011; Kovacs 2011) are little more than revisions to earlier BNF grammars.

With this structure, the informal algorithm for answering questions about PDDL is to start with the most recent specification (or the specification corresponding to a planner that one wishes to use), and search backwards, chronologically, until an answer is encountered. Not only is this a laborious process, it is not always obvious when a given later document overrides an earlier one. Our experience trying to resolve questions about the type system of PDDL (see elsewhere in this document) shows that even persons intimately involved in the IPC may have differing interpretations.

Two related issues have to do with the nature of the specification itself. One is that, with the possible exception of the original specification, the specifications have not been written as specifications. The 2.1 specification is a journal article primarily concerned with the semantics of durative actions, and the PDDL 3.1 documents are primarily revisions to the grammar. As PDDL grows in acceptance, it deserves a specification that *is* a specification, as with other programming languages, if for no other reason than to provide a specification that is organized and indexed for reference use.

The second issue is that the specification efforts to date have concentrated their efforts in two areas. The first is at the semantic level, in specification of the transition relationship. In addition to specifying the basics, a great deal of effort has gone into specifying the transition relationship when concurrent actions are permitted, both in durative and non-durative frameworks. The issues here can be subtle, and while there have been some critiques of design choices, overall the PDDL 2.1 specification is admirably clear (Fox and Long 2003). In later work, a similar amount of care went into specifying the interaction between the transition relation and the evaluation of preferences (Gerevini and Long 2005). The second area of focus has been the syntactic specification of PDDL, and here we feel that the efforts have been less successful. We discuss this in the following section. We also find that aspects of the language that don't fit neatly into either the transition relationship or the syntax, notably the type system, have been less well-served.

We do understand that improving the standardization and the standard document requires a substantial effort, and that there are challenges to rewarding this effort. In particular, in a community that is primarily motivated by refereed publication, expending effort on a document that will not be so published is challenging. No doubt this is substantially responsible for the way that PDDL standards have been developed as a sequence of "patch files." Since we have argued that the current state of affairs is undesirable, and we are unable to provide novel incentives, we can simply propose that the process of updating the standard be eased. Maintaining a readily available copy of the latest (complete) standard in an easy-to-edit and readily "diff-able" form such as LaTeX should minimize the effort of developing a new full standard, making it much more of a piece with developing a simple "patch file" revision. This might also have the benefit of making the standard revision process "as simple as it can be, but no simpler."[1] We may well find that requiring full standardization will not permit the introduction of new facilities whose semantics are uncertain, as opposed to permitting a mere specification of syntax.

Before we continue, we would like to mention the `VAL` PDDL validation program (Howey, Long, and Fox 2004), which provides an additional resource for understanding the PDDL standard. We have found the `validate` program in the `VAL` suite to be quite useful in debugging our domain designs, notably debugging poorly-modeled operators. However, `validate` has some limitations. First, although it is very helpful in evaluating the correctness of plans, with respect to executability and goal achievement, it does not provide a similar amount of guidance in identifying syntactic problems in domain and problem models. For the latter, we have found no real substitute for running such files through the parsers of `validate` and multiple planners (especially FF (Hoffman 2001) and FD (Helmert 2006)). Unfortunately, as we point out below, while these three programs all support the ADL dialect of PDDL, their interpretations of this dialect diverge in ways that cause them to disagree about the validity of PDDL domains and problems. We eventually found it necessary to patch the `VAL` suite to provide an interpretation of the type system that agreed with that used by FD. We discuss this further below.

[1]Roger Sessions, attributed to Albert Einstein.

Syntax issues

One issue that complicates the understanding of PDDL is the issue of repetition. For example, BNF does not provide a notation powerful enough to specify whether a type can have only a single declaration, or whether it can have multiple declarations. It is clear that a single type declaration may designate only a single supertype, but does PDDL permit multiple inheritance (multiple supertypes) in the case of multiple declarations, as in the following example:

```
:types (subtype - supertype1
        subtype - supertype2
        ...)
```

There are several similar issues, for example, whether predicates permit arity or type overloading (are multiple declarations for the same predicate name permitted?). The question of unordered constructs is a case where BNF is not, or at least may not be, a sufficiently powerful notation to capture some of the syntax of the language. The issue of repetition, in contrast, is one where syntax and semantics come together, and we see the problems of attempting to address questions about the meaning of the language with only a quite limited meta-syntactic notation.

We conjecture that the authors of these specifications have appealed to intuitions about what is "natural," and that as the language is exposed to a wider and wider audience, these intuitions are less likely to be shared among the audience. For example, questions of overloading the same name as operator and predicate, overloading/polymorphism, etc., may arise if PDDL spreads to communities that see different assumptions as "natural."

We suggest that the use of BNF may have led to more parsimony in the formulation of the PDDL grammar than is appropriate. For example, one of the non-terminals in the grammar is `<type>`, and one of its production rules is:

`<type> ::= (either <primitive-type>`$^+$`)`

Such a type definition makes perfect sense for a predicate's argument, as in:

```
(has_id ?proc - process ?id - (either uid gid))
```

for an operating system process that may have both a user-id (`uid`) and a group-id (`gid`), but its semantics are uncertain when it is applied to a constant/object or to a type. At the expense of expanding the grammar, we suggest that its categories could profitably be refined.

We conclude with some minor suggestions for improvements to the PDDL syntax. One is to improve the syntax of type declarations. The current syntax has two undesirable features: First, when revising existing domain and problem descriptions it is all too easy to accidentally have a newly-intoduced type expression "take over" additional entities to their left, because of the strong associativity. Second, the asymmetry which forces the definition of "top" classes to the end of the type declarations, in order to avoid being "grabbed up" by type declarations to their right. We agree that maintaining backward compatibility in syntax is desirable above almost all other considerations, but feel that a less-ambiguous extension could be introduced for those who want to use it. For example, the use of parentheses for grouping could mitigate some of the low precedence of the − type-declaration operator. Another thing that would help (see our discussion of the type system, below), would be restoration of the top `object` type, and issuing warnings about untyped entities. The introduction of an alternative type declaration operator, in addition to the hyphen, and either making it strictly binary, or allowing the use of parentheses for grouping on its left hand side, might also be an improvement.

Another possible improvement would be to provide an easier syntax for cost-based planning. Currently, cost-based planning is specified by the `:action-costs` requirement, which signals limitations on the use of the quite expressive numeric fluents syntax. This adds a great deal of complexity both to parsing (especially for planners that do not wish to support numeric fluents) and to the domain modeling. By contrast, the SHOP2 planner has a notation that simply adds an optional `:cost` keyword to operators. Adding such a facility to PDDL might be helpful.

The PDDL type system

The aspect of PDDL that has been the biggest barrier to us in getting our tests to work has been the PDDL type system. In our experience, there is a wide variation in understanding of how the PDDL type system is supposed to work: what syntactic expressions are permitted, and what are the semantics of the constructs. One obstacle has been difference of opinion about how the type system changed in the substantial PDDL redesign between PDDL 1.x and PDDL 2.1. The discussion in this section draws heavily on extensive correspondence about the PDDL type system – particularly about the issue of type introspection – with Patrik Haslum, Malte Helmert, and Derek Long. We are grateful to them for their guidance and patience in fielding our questions. In this section we discuss problems with the semantics of the PDDL type system in the following areas: (1) the use of the `either` construct for disjunction; (2) the open question of whether or not PDDL permits multiple inheritance; (3) the use of `object` as the top type of the type hierarchy; (4) another open question concerning whether or not PDDL permits type introspection.

`Either` construct

We have previously mentioned the `either` construct as a place where the grammar formalism may have led to a construct's generalization beyond a position where it has a clear semantics. When used in providing type constraints on predicate or operator parameters, and quantified variables, `either` is innocuous. However, if it is used in two other locations where the grammar permits, its semantics becomes problematic. For example, what does it mean to say:

```
(:types airplane - (either vehicle toy)...)
```

or

```
(:objects insider - (either good evil)...)
```

Patrik Haslum[2] suggests that the former should indicate that the extension of `airplane` should be a subset of the union of the extensions of `vehicle` and `toy`. A problem with this is that it, like the second example, permits uncertainty about the "true" type of a constant, which seems incompatible with the grounding semantics of existing PDDL planners. In the presence of such constants, one would have to resort to either an over- or under-approximation of the set of applicable actions, since it would not be clear whether, for example, an `airplane` was an acceptable instantiation for a variable of type `vehicle`. With quantified variables, further uncertainty would be added to interpretation of conditional effects, and to quantified preconditions. Our correspondence with Helmert, Haslum and Long suggests a consensus that `either` should be eliminated from any context other than variable type constraints.

Multiple inheritance

As we mentioned earlier, we have not been able to determine with certainty whether multiple inheritance is permitted in PDDL. We were inclined to the interpretation that repeated definitions of the same name were not acceptable PDDL, which would rule out polymorphism in predicates, actions, functions, and would also rule out multiple inheritance. However, we find a comment in the `validate` source code that seems to suggest that it is permitted to have types with multiple supertypes. Given the very limited forms of inference licensed by the PDDL type system, it is likely that permitting multiple inheritance would be benign. However, we suggest two cautions in this regard. First, in the presence of a grammar that does not distinguish between type designators used to specify supertypes, and type designators used to specify variable types (cf., our discussion of syntax and of the `either` construct), this could lead to permitting objects and variables to have multiple types, and the semantics there are less clear. The programming language Common Lisp, for example, permits classes to have multiple superclasses, but requires objects to have a single class (Bobrow et al. 1988; American National Standards Institute and Information Technology Industry Council 1996). The second caution is that a syntax that requires redundant definition such as

```
(:types subtype - supertype1
        subtype - supertype2 ...)
```

seems far worse for comprehension than a syntax that offers explicit conjunction:

```
(:types subtype - (and supertype1 supertype2)
        ...)
```

Top type

The original PDDL specification says that entities not explicitly typed are assigned the type `object`, "...every physical object is an `object`." (Ghallab et al. 1998, p.6) There seems to be substantial confusion about whether `object` is a type name in PDDL. The PDDL 2.1 specification does not mention it, as far as we can determine, and it

[2]Personal communication.

does not appear in the BNF for PDDL 2.1. The latest version of `VAL` (4.2.08) does not recognize it as the top of the object type hierarchy[3]. On the other hand, the Fast Downward translator specifically indicates that untyped entities (except numerical fluents) are of type `object`, and makes `object` available as a token of the language (rather than using an inaccessible internal symbol for the top type). Further, the PDDL 3.1 grammar by Kovacs (Kovacs 2011) explicitly includes the terminal `object` as an expansion for the nonterminal `<primitive-type>`.

We have found the ability to work with the top type helpful in our work, particularly in programmatically generating PDDL domains and problems, so we urge that its presence be confirmed. It is particularly important to have access to a top type, since (at least according to `validate`) type specifiers for predicates, etc., are no longer optional, when the `:typing` requirement is in place. This positive requirement does not seem fully clear in the grammar, primarily because there does not seem to be a negation construct in the requirements specification notation. So in the PDDL 2.1 grammar we see:

<typed list (x)> ::= x^*
<typed list (x)> ::= $^{:typing}$
      x^+ - <type> <typed list (x)>

and it is not clear whether the use of `:typing` in the second rule *overrides* the first rule, or adds to it. The description of the option in the PDDL 2.1 grammar, "Allow type names in declarations of variables" to us suggests the addition interpretation instead of overriding. Furthermore, in the PDDL 1.2 document, we are told that additional material "...is includable only if the domain being defined has declared a requirement for that flag." (Ghallab et al. 1998, p. 3) The same two rules are preserved in the PDDL 3.1 grammar, as is the gloss of the `:typing` requirement. We prefer the more permissive interpretation, but feel that one way or another, the specification should be clarified. We suggest that a negated form of the superscripting operator, indicating that a production is *forbidden*, in presence of a particular feature, be added to the PDDL grammar formalism. For example, if one wished to forbid non-durative actions from appearing in domains with durative actions, one might modify the grammar from:

<structure-def> ::= <action-def>
<structure-def>
  ::=$^{\text{:durative-actions}}$ <durative-action-def>
...

to

<structure-def>
  ::=$^{\text{(not :durative-actions)}}$ <action-def>
<structure-def>
  ::=$^{\text{:durative-actions}}$ <durative-action-def>
...

[3]`validate` will accept the string `object` as a type name, but does not recognize it as the supertype of all other types in its typechecker.

Type introspection

The original PDDL specification states that "an atomic type name is just a timeless unary predicate, and may be used wherever such a predicate makes sense." (Ghallab et al. 1998, p.6) There is not general agreement on whether this part of the specification has been overridden or not. In our correspondence, Malte Helmert felt that this was still active, and Fast Downward permits the use of type names as unary predicates. Derek Long, on the other hand, felt that the PDDL 2.1 specification had quite clearly eliminated this construct from the language. We feel that this construct, which serves to provide a kind of type introspection, is helpful in domain engineering, and we would like to see it unambiguously supported in the language.

Doing without type introspection can lead to quite laborious rewrites to a domain. Consider an example which we encountered when modeling a cyber attack domain (a variant of the work by Boddy, et al. (Boddy et al. 2005)). We had an operator that captured what happens when a program is started, looking something like this:

```
(:action RUN_PROGRAM
  :parameters (?O - host
               ?H - human
               ?P - program)
  ...)
```

Now we wished to reflect the fact that there are particular attack tools, that are a kind of `program`, that would only be run by a person who is `evil`, where `evil` is a subtype of `human`. With type introspection, we can capture this relatively easily:

```
(:action RUN_PROGRAM
  :parameters (?O - host
               ?H - human
               ?P - program)
  :precondition (and
                  (imply (malware ?P)
                         (evil ?H))
                  ...)
  ...)
```

Similarly, there are cases where conditional effects may rely on the types of certain parameters, or the presence or absence of an object of a particular type. Some of these situations can be modeled by adding additional unary predicates which are "type-like," but this makes it more difficult to formulate consistent initial states, a task which is already challenging.

To capture this same information without type introspection requires extensive modification of the domain model. It is not sufficient to simply split `RUN_PROGRAM` into `RUN_PROGRAM` and `RUN_MALWARE`, since the fact that `malware` is a subtype of `program`, and `evil` is a subclass of `human` means that `RUN_PROGRAM` will apply to malware, as well as normal software. We must now introduce a mutually-exclusive decomposition of `program` into `malware` and `non_malware` in order to capture the same domain constraints. Doing this in such a way that operators with parameters known only to be of type `program` still work, can be very ticklish – careful thought must be given to the typing of every predicate involving `program` type arguments, or such operators will fail to type correctly.

With assistance from Derek Long, we have developed a patch to VAL 4.2.08 that provides the option to accept PDDL that incorporates unary type predicates. We have made this patch available to the VAL developers, and would be happy to provide it to other interested parties.

Adding type introspection of this sort suggests further that the community might want to extend PDDL to support a limited species of type inference. Consider this example:

```
(:action typed-action
  :parameters (?x - supertype)
  :precondition (implies (type ?x) P)
  ...)
```

The question is, in `P`, are we allowed to assume that `?x` is of `type`, or must we continue to assume only that it is of type `supertype`? It is clear that the former inference is sound, and there are readily available type inference algorithms that are more than capable of making such inferences. Further, the usefulness of type introspection might be substantially limited absent this kind of "narrowing" type inference. The concern on the other side is that adding type inference might substantially raise the bar for implementing an IPC planner with more than STRIPS expressive power.

Idiolects

A final concern to outsiders trying to make use of PDDL planners is the proliferation of idiolects[4], usually undocumented sub-dialects of PDDL that correspond to a particular piece of software. In multiple examples above, we have mentioned differences between the three PDDL software suites that we have used most extensively: FF, Fast Downward, and VAL. We had to modify our PDDL-generating programs to honor the FF parser's restrictions on placement of newlines (and it is worth saying that we were well rewarded for doing so, since FF's parser has excellent error messages). Idiolects can be especially difficult for programmers when combined with minimal error reporting.

Related issues are the fact that the IPC has tended to avoid parts of the PDDL language that have been documented, but not widely supported by planners. We certainly understand the motivation for this – it would not be much of a competition otherwise. Nevertheless, when one comes from the outside, it would be very helpful to know which parts of the language, although in the specification, are unlikely to be supported by planners, and if supported, may not function reliably. Annotating such parts of the language would be very helpful. It might be well to prune the set of feature combinations, or at least document which are of primary interest to the competition.

The community could take a number of simple steps to reduce the prevalence of idiolects. One would be to provide a test suite for PDDL parsing, with different settings, corresponding to feature combinations. We hope it would not be a major extra burden for planner builders to provide an interface to simply parse and accept or reject PDDL inputs. Indeed, if there were such a test suite, planner programmers might find it was a substantial help in developing their planners. Outside users could, in a pinch, use such a test suite to determine what languages a planner might accept. This would be particularly helpful when trying to find a planner that accepts some of the more experimental features of PDDL.

Expressive power

We urge planner developers to support the higher ends of the expressive spectrum of PDDL. Our experience has convinced us that simple STRIPS is insufficient for modeling most, if not all, problems of interest. Even when a domain can be captured in STRIPS, the burden of doing so is extreme. We find that the extensions provided by ADL and, to a lesser extent, by derived predicates, are a substantial help in such modeling. There are some techniques for compiling away ADL and derived predicate constructs, but the best are internal to planners. Also, such compilation often involves grounding a domain against a particular problem, which means that STRIPS cannot be used as a primary modeling language.

We have found that in complex domains, with many objects that have multiple properties, it can be quite difficult to generate complete and consistent initial state descriptions. We have developed the rudiments of a macro notation that supports partially-automated generation of the set of propositions that describes the state of an object in a domain. Our work in this area was inspired by the work by Boddy, *et al.* on cyber attack planning (Boddy et al. 2005), which used the `m4` macro preprocessor to assemble domain and problem definitions from multiple, simpler files.[5] Our work has used the Common Lisp programming language, because it allows us to work with the s-expression structure of the PDDL model, rather than requiring more primitive string manipulations. With slightly more effort, similar facilities could be built in dynamic languages such as Python or ECMA Script.

As PDDL moves towards modeling ever more complex domains, it may be useful to incorporate some such facility in the language. Note that such a facility need not be built into the planners proper, since this kind of preprocessing may be done entirely prior to more problem-sensitive operations such as grounding. That means that ease in modeling could be achieved without complicating the work of IPC entrants.

Conclusions

Our experience has shown that, under the impetus provided by the International Planning Competition, planning software has become mature enough to be more and more widely used as a general problem-solving tool. Critical to making planning technology more usable is the modeling

[4]"The speech of an individual, considered as a linguistic pattern unique among speakers of his or her language or dialect," The Free Dictionary, `http://www.thefreedictionary.com/idiolect`.

[5]Sample cyber attack domains, with the `m4` macros, may be found in the 2008 IPC resources.

language, PDDL. We have outlined some challenges the modeler faces when using PDDL, and have made some suggestions for improving its standardization.

References

Alford, R.; Kuter, U.; and Nau, D. S. 2009. Translating HTNs to PDDL: A small amount of domain knowledge can go a long way. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

American National Standards Institute, and Information Technology Industry Council. 1996. *American National Standard for information technology: programming language — Common LISP: ANSI X3.226-1994*. American National Standards Institute.

Baier, J.; Bacchus, F.; and McIlraith, S. 2007. A heuristic search approach to planning with temporally extended preferences. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, 1808–1815.

Baier, J. A.; Fritz, C.; and McIlraith, S. A. 2007. Exploiting procedural domain control knowledge in state-of-the-art planners. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS-07)*, 144–151.

Bobrow, D. G.; DeMichiel, L. G.; Gabriel, R. P.; Keene, S.; Kiczales, G.; and Moon, D. A. 1988. Common Lisp Object System specification. Document 88-003, X3J13 Standards Commitee (ANSI Common Lisp).

Boddy, M. S.; Gohde, J.; Haigh, T.; and Harp, S. A. 2005. Course of action generation for cyber security using classical planning. In Biundo, S.; Myers, K. L.; and Rajan, K., eds., *International Conference on Automated Planning and Scheduling*, 12–21. AAAI.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classical part of the 4th international planning competition. report00195, Institut für Informatik, Universität Freiburg.

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

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. Technical Report RT 2005-08-47, Dept. of Electronics for Automation, University of Brescia, Brescia, Italy.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – the planning domain definition language. Technical Report CVC TR-98-003, Yale Center for Computational Vision and Control, New Haven, CT.

Goldman, R. P.; Kuter, U.; and Schneider, A. 2012. Using classical planners for plan verification and counterexample generation. In *Proceedings of AAAI Workshop on Problem Solving Using Classical Planning*. To appear.

Haslum, P. 2011. Changes in PDDL 3.1. From IPC 2008 website.

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

Hoffman, J. 2001. FF: The fast-forward PLanning system. *The AI Magazine* 22(1):57–62.

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

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *Proceedings ICTAI*, 294–301. IEEE Computer Society.

Kovacs, D. L. 2011. BNF definition of PDDL 3.1. From IPC 2011 web site.

Long, D., and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research* 20:1–59.

Palacios, H., and Geffner, H. 2007. From conformant into classical planning: Efficient translations that may be complete too. In Boddy, M. S.; Fox, M.; and Thiébaux, S., eds., *ICAPS*, 264–271. AAAI.