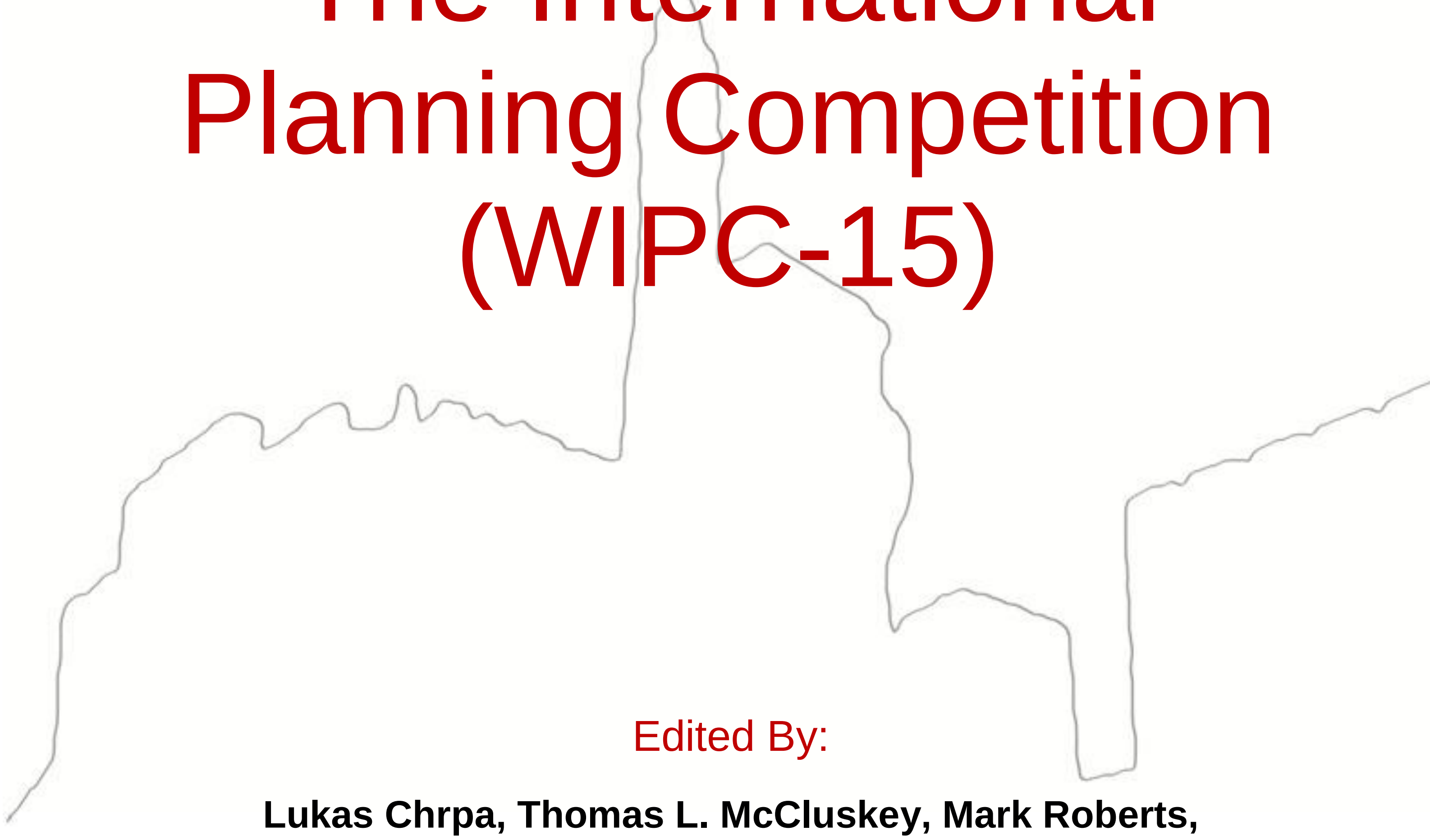
Research Workshop of the
Israel Science Foundation

Proceedings of the 4th Workshop on

The International Planning Competition (WIPC-15)



Edited By:

Lukas Chrpa, Thomas L. McCluskey, Mark Roberts,

Scott Sanner, Mauro Vallati

Jerusalem, Israel 8/6/2015

Organizing Commitee

Lukas Chrpa
University of Huddersfield, UK

Thomas L. McCluskey
University of Huddersfield, UK

Mark Roberts
Naval Research Laboratory, USA

Scott Sanner
Oregon State University, USA

Mauro Vallati
University of Huddersfield, UK

Program committee

Daniel Borrajo, Universidad Carlos III Madrid, Spain
Amanda Coles, King’s College London, UK
Andrew Coles, King’s College London, UK
Alfonso E. Gerevini, University of Brescia, Italy
Carlos Linares Lopez, Universidad Carlos III Madrid, Spain
Sergio Jiménez, Universitat Pompeu Fabra, Spain
Erez Karpas, MIT, USA
Héctor Geffner, Universitat Pompeu Fabra, Spain

Table of Contents

About Benchmarking and Competitions of Solvers in Constraint Programming

Frédéric Boussemart[1] and **Christophe Lecoutre**[1] and **Arnaud Malapert**[2] and **Cédric Piette**[1]
[1] Université d'artois, CNRS, CRIL, UMR 8188, rue de l'université, 62307 Lens cedex, France
[2] Université Nice Sophia Antipolis, CNRS, I3S, UMR 7271, 06900 Sophia Antipolis, France

Abstract

CSP (Constraint Satisfaction Problem) is the problem of finding an assignment of values to a set of variables such that a set of constraints is satisfied. In this short paper, we briefly describe how past CSP competitions were conducted, present the main features of the new format XCSP3, targeted to become a CP (Constraint Programming) standard, and discuss about classification of instances as well as ranking of solvers. In the context of benchmarking and solver competitions, this position paper aims at encouraging the cross-fertilization of ideas and methods in both planning and constraint reasoning domains.

Constraint programming (CP) is a general technology providing simple, general and efficient models and algorithms for solving combinatorial constrained problems. At the heart of CP, we find the framework CSP (Constraint Satisfaction Problem) that consists in solving problem instances represented by Constraint Networks (CNs). A solution to a CN is obtained by instantiating its set of variables so that its set of constraints is satisfied. This framework has many derivatives, mainly extensions, as indicated in Figure 1: temporal CSP (TCSP), weighted CSP (WCSP), valued CSP (VCSP), quantified CSP (QCSP), constraint optimization problem (COP), Max-CSP, distributed CSP (DisCSP), and so on.

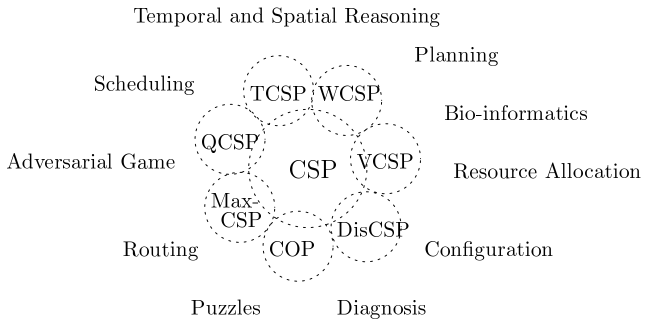


Figure 1: The framework CSP and some of its extensions.

Systems for solving CP instances are called constraint solvers. Several competitions have been organized over the years to assess the different solvers or algorithms developed in the CP community. A first series of competitions has focused on generic, black-box solvers that do not require human intervention to perform well. These competitions have been organized in this spirit, using XCSP 2.1 as input format, in 2005, 2006, 2008 and 2009. They have contributed to identify interesting techniques and to draw a fair picture of state-of-the-art algorithms or heuristics (Lecoutre, Roussel, and van Dongen 2010). On the other hand, every year since 2008, another series of competitions has been run: the MiniZinc challenge. Different solvers are compared on common Minizinc/Flatzinc benchmarks, in particular so that solver implementers can examine the detailed results and determine the strengths and weaknesses of their system, and problem modelers can judge which system might be preferable for their problem (Stuckey, Becket, and Fischer 2010). In the rest of the paper, we briefly describe how past CP competitions were conducted, present the main features of the new format XCSP3, and discuss about classification of instances as well as ranking of solvers.

Constraint Solver Competitions

In this section, we succinctly present how the fourth international constraint solver competition (CSC), held in 2009, was organized. There were three problems:

- CSP
- Max-CSP
- WCSP

The objective for CSP is finding a solution or proving that no solution exists. The objective for Max-CSP is finding an assignment of variables that violates as few constraints as possible. The objective for WCSP is finding an assignment of minimal violation cost (cost functions, instead of hard constraints, are considered). CSP is a decision problem whereas Max-CSP and WCSP are optimization ones.

There were two solver categories:

- complete
- incomplete

Complete solvers can determine if an instance is satisfiable or not (or can find and prove optimality for Max-CSP and WCSP) whereas incomplete solvers cannot prove the unsatisfiability or the optimum.

The format used to represent problem instances was XCSP 2.1; it is described in (Roussel and Lecoutre 2009). Some tools were also provided:

- C and C++ parsers for XCSP 2.1
- A Java parser for XCSP 2.1
- A program to check the validity of problem instances in format XCSP 2.1
- A program to check solutions

Execution Environment

Solvers were run on a cluster of computers using the Linux operating system. They were run under the control of another program that enforces some limits on memory and CPU time. Solvers were run inside a sandbox that prevents unauthorized use of the system (network connections, file creation outside the allowed directory, among others). It was required that two executions of a solver with the same parameters and system resources should output the same result in approximately the same time (so that the experiments can be repeated).

Of course, contestants were asked to provide the organizers with a command line for running their solvers. In this command line, the following placeholders were replaced by the actual information given by the evaluation environment:

- BENCHNAME replaced by the name of the file containing the instance to solve.
- RANDOMSEED replaced by a random seed which is a number between 0 and $4,294,967,295$. This parameter had to be used for initializing the random number generator (when the solver uses random numbers). It is recorded by the evaluation environment, which allows to solve a given instance under the same conditions if necessary.
- TIMEOUT represents the total CPU time (in seconds) that the solver may use before being killed. May be used to adapt the solver strategy.
- MEMLIMIT represents the total amount of memory (in MiB) that the solver may use before being killed. May be used to adapt the solver strategy.
- TMPDIR is the name of the only directory where the solver is allowed to read/write temporary files.
- DIR is the name of the directory where the solver files are stored.

After TIMEOUT seconds have elapsed, the solver first receives a SIGTERM to give it a chance to output the best solution found so far (in the case of an optimizing solver). One second later, the program receives a SIGKILL signal from the controlling program to terminate the solver. Similarly, a solver that used more memory than the limit defined by MEMLIMIT was sent a SIGTERM followed one second later by a SIGKILL.

The solver could not write to any file except standard output, standard error and files in the TMPDIR directory. A solver was not allowed to open any network connection or launch external commands which are not related to the solving process.

Output Rules

The evaluation environment recorded everything that is output by a solver on stdout/stderr (up to a limit of 1MiB) and put a timestamp on each line. This can be very informative to check how solvers behave on some instances.

Of course, solvers had to output special messages to the standard output in order to check results. The output format was inspired by the DIMACS output specification of the SAT competitions. Lines output by the solver had to be prefixed by "c ","s ","v ", "d " or "o ". Other lines were ignored.

- solution ("s " line) These lines are mandatory, start with lower case s followed by space (ASCII code 32), and are followed by one of the following answers:
  - UNSUPPORTED
  - SATISFIABLE
  - UNSATISFIABLE (only used for CSP)
  - UNKNOWN
  - OPTIMUM FOUND

  Note that UNSUPPORTED is used when the solver recognizes a constraint that is not implemented.
- values ("v " line) These lines are mandatory and contain a solution, i.e., a sequence of numbers, with whitespace as separator.
- diagnostic ("d " line) These lines are optional. A keyword followed by a value must be given on such lines. For example, a possible keyword is ASSIGNMENTS.
- comment ("c " line) Such lines are optional and may appear anywhere in the solver output. They contain any information that authors want to output. They are recorded by the evaluation environment for later viewing.
- objective cost ("o " line) (for Max-CSP and WCSP, only) These lines contain an integer value.

On the one hand, a CSP solver had to output exactly one "s " line and in addition, when the instance is found SATISFIABLE, exactly one "v " line. On the other hand, since a MAX-CSP or WCSP solver does not stop as soon as it finds a solution but instead tries to find a better solution, it must be given a way to output the best found solution even when it reaches the time limit. There are two options depending on the ability of the solver to intercept signals. The first option can be used if the solver is able to catch the signal SIGTERM : when interrupted, it must output the best found solution. The second option must be used otherwise: a certificate "v " line is output each time the solver finds a solution which is better than the previous ones.

Ranking

For each problem (CSP, Max-CSP and WCSP) and solver category, a ranking was computed for different categories of instances, based on constraint arity (binary and non-binary constraints) and constraint representation (extensional, intensional and global constraints). Solvers were ranked as follows.

CSP: Solvers claiming incorrect results in a given category were disqualified from this category. Of the remaining solvers, the solver solving the most problems were declared the winner. Ties were broken by considering the minimum total solution time.

Max-CSP/WCSP: Solvers with no incorrect results in a given category were ranked as follows:

Incomplete solvers were ranked in increasing order of their average relative error (i.e. the average normalized difference between the cost of solutions found by the solver and the cost of the best known solutions).

Complete solvers were ranked in decreasing order of the number of instances for which a solution was proved to be OPTIMUM (using average relative error as tie breaker).

Minizinc Competitions

In this section, we succinctly present how the last Minizinc competition, held in 2014, was organized. The language used for representing problem instances in that competition was MiniZinc (Nethercote et al. 2007) (version 1.6), and its related low-level format FlatZinc. Entrants to the challenge provide a FlatZinc solver as well as global constraint definitions specialized for their solver. A translator, called mzn2fzn, is run on MiniZinc models using the provided global constraint definitions to create FlatZinc files, which are given as input to the contestant solvers.

An entrant in the challenge is a constraint solver that is installed in a virtual machine (VM) provided by the organizers. The provided VM with an installed solver is run by an executable file called fzn-exec that invokes a FlatZinc solver handling FlatZinc version 1.6. The syntax is:

```
fzn-exec [<options>] file.fzn
```

where:

- the argument file.fzn is the name of a FlatZinc 1.6 model instance to evaluate.
- the options are:

-a For satisfaction problems, the solver must output all solutions, and for optimization problems, the solver must output the first optimal solution and all found intermediate solutions.

-f The solver is free to ignore any specified search strategy.

-p <n> The solver is free to use multiple threads and/or cores during search. The argument n specifies the number of cores that are available.

There have been four competition CLASSES:

- FD search: solvers must follow the search strategy given in each input file, and are disqualified when they do not follow it.
- Free search: solvers are free to ignore the search strategy.
- Parallel search: solvers are free to use multiple threads or cores.
- Open class: this class allows the usage of portfolio solvers. Solvers are free to use multiple threads or cores to solve the problem.

Output of solvers must conform to the FlatZinc 1.6 specification. For optimization problems, when the time limit is exceeded before the final solution is printed then the last complete approximate solution printed is considered to be the solution for that instance. Each solver s is run on problem instance p and the following information is collected:

- $wck(p, s)$ - the wall clock time used by s.
- $solved(p, s)$ - true if p is solved by s
- $quality(p, s)$ - the objective value of the best solution found by s.
- $optimal(p, s)$ - true if the objective value is proved optimal by s.

Scoring Procedure The scoring procedure is based on the Borda count voting system. Each benchmark instance is treated like a voter who ranks the solvers. Each solver scores points related to the number of solvers that it beats. More precisely, a solver s scores points on problem p by comparing its performance with each other solver s' as follows:

- if s gives a better answer than s' it scores 1 point,
- else if $\neg solved(p, s)$ or s gives a worse answer than s' it scores 0 point.
- else (s and s' gives indistinguishable answers) scoring is based on execution time comparison: s scores $wck(p, s')$ / $(wck(p, s') + wck(p, s))$, or 0.5 if both finished in 0s.

A solver s is said to be better than a solver s' iff:

- for satisfaction problems, $solved(p, s) \wedge \neg solved(p, s')$
- for optimization problems,

$$\begin{array}{l} solved(p, s) \wedge \neg solved(p, s') \\ \vee optimal(p, s) \wedge \neg optimal(p, s') \\ \vee quality(p, s) > quality(p, s') \end{array}$$

From XCSP 2.1 to XCSP3

Although significant efforts were performed these recent years, in the context of competitions of solvers, as shown in previous sections, the Constraint Programming (CP) community still suffers from the lack of a standard low-level format for representing various forms of combinatorial problems subject to constraints and optimization. It is important to note that we specifically refer here to the possibility of generating and exchanging files containing precise descriptions of problem instances (no model/data separation), so that fair comparisons of problem solving approaches can be made in good conditions, and experiments can be reproduced easily. The two current proposals, XCSP 2.1 and FlatZinc, have some drawbacks that certainly prevent them from becoming such a “universal” format. For example, it was not possible to deal with objective functions in XCSP

2.1, and not possible to deal with weighted constraint networks in MiniZinc (and so, in FlatZinc), although a proposal was made in (Ansotegui et al. 2011), but to the best of our knowledge, never incorporated in official specifications.

In (Boussemart et al. 2015), the specifications for a major revision/extension of the basic format XCSP 2.1 are introduced. This new format, XCSP3, is a rather light language that allows us to get integrated representations of combinatorial constrained problems, by enumerating variables, constraints and objectives in a very simple and unambiguous way.

Actually, here are the main advantages of XCSP3:

- **Generality**. XCSP3 allows us to represent many forms of combinatorial constrained problems since it can deal with constraint satisfaction, mono and multi-objective optimization, preferences through soft constraints, variable quantification, qualitative reasoning, continuous constraint solving, distributed constraint reasoning, and probabilistic constraint reasoning.
- **Completeness**. A very large range of constraints is available, including rarely used variants and encompassing practically (i.e., to a very large extent) all constraints that can be found in major constraint solvers such as, e.g., Choco and Gecode.
- **Understanding**. We pay attention to control the number of concepts and basic constraint forms, advisedly exploiting automatic variations of these forms through lifting, restriction, sliding, combination and relaxation mechanisms, so as to facilitate global understanding.
- **Readability**. The new format is more compact, and less redundant than XCSP 2.1, making it very easy to read and understand, especially as variables and constraints can be handled under the form of arrays/groups.
- **Flexibility**. It will be very easy to extend the format, if necessary, in the future, for example by adding new kind of global constraints, or by adding a few XML attributes in order to handle new concepts.
- **Easiness of Parsing**. Thanks to the XML architecture of the format, basically, it is easy to parse instance files at a coarse-grain level. Besides, parsers written in Java and C++ are made available.
- **Dedicated Website**. A website, companion of XCSP3, will be open soon, with many models/series/instances made available. This website will allow the user to make sophisticated queries in order to select and download the instances that he finds relevant.

The main novelties of XCSP3 with respect to XCSP 2.1 are:

- **Optimization**. XCSP3 can manage both mono-objective and multi-objective optimization.
- **New Types of Variables**. It is possible to define 0/1, integer, symbolic, float, qualitative, set, and graph variables, in XCSP3.
- **Lifted and Restricted forms of Constraints**. It is natural to extend basic forms of constraints over lists (tuples), sets and multi-sets. It is simple to build restricted forms of constraints by considering some properties of lists.
- **Meta-constraints**. It is possible to exploit sliding and logical mechanisms over variables and constraints.
- **Relaxed constraints**. Relaxed cost-based constraints can be defined easily.
- **Reification**. Half and full reification is easy, and made possible by letting the user associate a 0/1 variable with any constraint of the problem through a dedicated XML attribute.
- **Views**. In XCSP3, it is possible to post constraints with arguments that are not limited to simple variables or constants, thus, avoiding in some situations the necessity of introducing auxiliary variables and constraints, and permitting solvers that can handle variable views to do it.
- **Structure**. It is possible to post variables under the form of arrays (of any dimension) and to post constraints in (semantic or syntaxic) groups, thereby, partly preserving the structure of the models.
- **Annotations**. It is possible to add annotations to the instances, for indicating search guidance and filtering preferences.

```
<instance format="XCSP3" type="CSP">
  <variables>
    <array id="M" size="[3][3]">
      1..9
    </array>
  </variables>
  <constraints>
    <allDifferent> M[][]</allDifferent>
    <group>
      <sum>
        <list> %... </list>
        <condition> (eq,15) </condition
          >
      </sum>
      <args> M[0][] </args>
      <args> M[1][] </args>
      <args> M[2][] </args>
      <args> M[][0] </args>
      <args> M[][1] </args>
      <args> M[][2] </args>
      <args>
        M[0][0] M[1][1] M[2][2]
      </args>
      <args>
        M[2][0] M[1][1] M[0][2]
      </args>
    </group>
  </constraints>
</instance>
```

As an illustration, let us consider the XCSP3 formulation for the 3-order instance of the Magic Square problem, given above. A magic square of order 3 is an arrangement of numbers $1, 2, \ldots, 9$ in a square grid, where the numbers in each row, and in each column, and the numbers in the main and secondary diagonals, all add up to the same number (15).

Classification

A recurrent issue in benchmarking is the classification of instances. We can distinguish two common practices which are not exclusive: qualitative classification and quantitative classification. A *qualitative* classification gathers instances according to some features (nature of the problems, random generation, ...). A *quantitative* classification gathers instances according to their difficulty. The latter is frequently used in other communities, but to our knowledge, only qualitative classification are publicly available for MiniZinc and XCSP.

Quantitative classification We propose a simple classification method based on the results of various solvers. For example, these results may correspond to those obtained at successive competitions.

The Virtual Best Solver (VBS) is a theoretical construction which returns the best answer provided by one of the solvers. Similarly, the Virtual Worst Solver (VWS) returns the worst answer provided by one of the solvers.

Easy instances are classified using the **VWS**: **training** (less than 10 seconds); or **easy** (less than 1000 seconds). **Hard** instances are classified using the **VBS**: **medium** (less than 1000 seconds); **challenging** (no timeout); or **hard** (timeout or memory exception).

These categories form a partition of the instances. Indeed, an instance belongs to the first category above whose acceptance condition is met. For example, a medium instance is solved in less than 1000 seconds by the VBS, but more than 1000 seconds by the VWS (otherwise, it would be an easy instance).

Of course, such a classification attempt should also take into account the experimental protocols followed for obtaining the results (e.g., computing infrastructures)

Scoring Procedure

The main drawback of the scoring procedure in the CSC competition is that solving times are only considered for tie breaking for CSP and not considered at all for optimization. On the other hand, the scoring procedure in the Minizinc competition is based on the Borda count voting system, as explained earlier. Unfortunately, the way points are distributed in case of indistinguishable answers does not capture user's preferences very well. Indeed, if the solver s solves the first n problems in 0.1 seconds and the n last problems in 1000 seconds whereas the solver s' solves the first n problems in 0.2 seconds and the n last problems in 500 seconds, then both solvers obtain the same score (n) whereas most users would certainly prefer s'.

To partially address this issue, we propose a scoring variant. Let t and t' respectively denote $wck(p, s)$ and $wck(p, s')$, i.e., the wall clock times of solvers s and s' when solving the problem p. In case of indistinguishable answers, s scores $f(t, t') = t' \div (t+t')$ according to the Borda system. Here, we define $g(t, t') = g(t) + (1 - g(t) - g(t')) \times f(t, t')$ in which some points are given by contract $g(t) = 1 \div 2 \times (\log_a(t+1)+1)$ $(a = 10)$ where $g(t)$ is a strictly decreasing function from 0.5 toward 0. The remaining points are shared between the two solvers using Function f. Using Function g in the previous example, solvers s and s' are respectively scored $0.81 \times n$ and $1.19 \times n$ points: s' is therefore preferred to s.

This idea arose because in competitions there is usually a significant number of easy problems. Only using Function f introduces a bias toward solvers that solve easy problems very quickly. Besides, one of the main drawback of the Borda count is that it does not satisfy the Condorcet criterion, i.e., a solver who wins a duel against each other candidate is not always the winner of the competition.

In future competitions, it would certainly be relevant to modify the scoring procedure, while being careful that it remains consistent with the competition rules. For instance, the Borda count does not satisfy the independence of clones criterion, stating that the winner must not change when a non-winning solver is duplicated (considered twice). So, rules could impose that each contestant submit at most one solver (contrary to previous competitions).

Conclusion

In this position paper, we have provided CP feedback on benchmarking and solver competitions. We have also presented some innovative developments in terms of problem representation format and competition rules. We hope that this material may be useful to the organizers of the international planning competitions.

Acknowledgments

This work has been supported by both CNRS and OSEO (BPI France) within the ISI project 'Pajero'

References

[Ansotegui et al. 2011] Ansotegui, C.; Bofill, M.; Palah, M.; Suy, J.; and Villaret, M. 2011. W-MiniZinc: A proposal for modeling weighted CSPs with MiniZinc. In *Proceedings of 1st International Workshop on MiniZinc*.

[Boussemart et al. 2015] Boussemart, F.; Lecoutre, C.; Perradin, V.; and Piette, C. 2015. XCSP3: An integrated format for benchmarking combinatorial constrained problems. Technical Report (preliminary), CRIL.

[Lecoutre, Roussel, and van Dongen 2010] Lecoutre, C.; Roussel, O.; and van Dongen, M. 2010. Promoting robust black-box solvers through competitions. *Constraints* 15(3):317–326.

[Nethercote et al. 2007] Nethercote, N.; Stuckey, P.; Becket, R.; Brand, S.; Duck, G.; and Tack, G. 2007. MiniZinc: Towards a standard CP modelling language. In *Proceedings of CP'07*, 529–543.

[Roussel and Lecoutre 2009] Roussel, O., and Lecoutre, C. 2009. XML representation of constraint networks: Format XCSP 2.1. Technical Report arXiv:0902.2362, CoRR.

[Stuckey, Becket, and Fischer 2010] Stuckey, P.; Becket, R.; and Fischer, J. 2010. Philosophy of the MiniZinc challenge. *Constraints* 15(3):307–316.

Better Be Lucky Than Good: Exceeding Expectations in MDP Evaluation

Thomas Keller and **Florian Geißer**
University of Freiburg
Freiburg, Germany
{tkeller,geisserf}@informatik.uni-freiburg.de

Abstract

We introduce the MDP-Evaluation Stopping Problem, the optimization problem faced by participants of the International Probabilistic Planning Competition 2014 that focus on their own performance. It can be constructed as a meta-MDP where actions correspond to the application of a policy on a base-MDP, which is intractable in practice. Our theoretical analysis reveals that there are tractable special cases where the problem can be reduced to an optimal stopping problem. We derive approximate strategies of high quality by relaxing the general problem to an optimal stopping problem, and show both theoretically and experimentally that it not only pays off to pursue luck in the execution of the optimal policy, but that there are even cases where it is better to be lucky than good as the execution of a suboptimal base policy is part of an optimal strategy in the meta-MDP.

Introduction

Markov Decision Processes (MDPs) offer a general framework to describe probabilistic planning problems of varying complexity. The development of algorithms that act successfully in MDPs is important to many AI applications. Since it is often impossible or intractable to evaluate MDP algorithms based on a theoretical analysis alone, the International Probabilistic Planning Competition (IPPC) was introduced to allow a comparison based on experimental evaluation. The idea is to approximate the quality of an MDP solver by performing a sequence of runs on a problem instance, and by using the average of the obtained results as an approximation of the expected reward. Following the optimal policy (i.e., the policy that maximizes the expected reward) leads to the best result in such a setting.

The work on this paper started with our preparation for IPPC 2014, where each solver had to perform *at least* 30 runs within a given time limit, while only *the last* 30 runs were used for evaluation. The decision when to stop the sequence of runs could be taken at any point of the evaluation with knowledge of the rewards that were collected in all previous runs. We describe the MDP-Evaluation Stopping Problem (MDP-ESP) as the optimization problem faced by IPPC participants that focus on their own performance, and show how it can be constructed as a meta-MDP with actions that correspond to the application of a policy on the base-MDP. Interestingly, the computation of the optimal policy is no longer the only objective of participating planners, and the fact that the execution of other policies on the base-MDP can be part of an optimal strategy for the MDP-ESP leads to a problem that is intractable in practice.

However, there are special cases where the MDP-ESP can be reduced to an instance of an optimal stopping problem (OSP). Two functions that depend only on the number of remaining runs – one that specifies the target reward that is necessary to stop, and one that gives the policy that is applied otherwise – suffice to describe an optimal policy. Based on these observations, we present four approximate algorithms for the general problem. A strategy that is inspired from a solution to the related Secretary Problem (SecP) (Dynkin 1963; Ferguson 1989) can be applied even when a policy for the base-MDP is computed online and not known in advance. Two other algorithms require the knowledge of the optimal policy and its expected reward. We show that the expected reward of the optimal policy is a lower bound for the expected performance of both strategies.

Our final algorithm switches between the application of the optimal policy and the policy with the highest possible outcome, which can be computed without notable overhead in the Trial-based Heuristic Tree Search (THTS) framework (Keller and Helmert 2013). We show theoretically and empirically that all algorithms outperform the naïve base approach that ignores the potential of optimizing evaluation runs in hindsight, and that it pays off to take suboptimal base policies in addition to the optimal one into account. Finally, we discuss the influence of the MDP-ESP on the results of IPPC 2014, and propose potential applications of our algorithms by discussing them in the context of related work.

Background

Markov Decision Processes. In this paper we consider problems of planning and acting under uncertainty, where an agent interacts with an uncertain environment by performing a sequence of runs. The environment is described in terms of a finite-horizon MDP $M = \langle V, A, T, R, s_0, h\rangle$ (Puterman 1994; Bertsekas and Tsitsiklis 1996) with a factored representation of states (Boutilier, Dearden, and Goldszmidt 2000) that are induced by a set of state variables V as $S = 2^V$. A is a finite set of actions such that $A(s)$ gives the set of

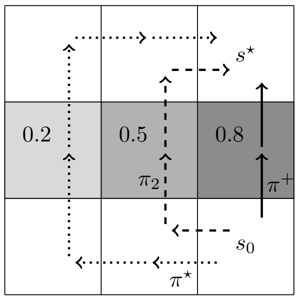


Figure 1: An example instance of the NAVIGATION domain with the policies $\pi^\star$ (dotted), π_2 (dashed) and π^+ (solid).

applicable actions in $s \in S$; $T : S \times A \times S \to [0, 1]$ is the transition function which defines the probability $\mathbb{P}[s' \mid s, a]$ that applying $a \in A(s)$ in $s \in S$ leads to $s' \in S$; $R(s, a)$ is the reward that is gained by applying $a \in A(s)$ in $s \in S$; s_0 is the initial state; and $h \in \mathbb{N}$ is the finite horizon.

The agent has access to the declarative model of the MDP (i. e., transition probabilities and reward function are known) to compute policies $\pi_1, \dots, \pi_n$ that are executed in a sequence of runs $(\phi_1^{\pi_1}, \dots, \phi_n^{\pi_n})$. Each policy π in the set of policies Π maps all states $s \in S$ to an action $a \in A(s)$. When a policy π is applied in a state s, the current state of the environment transitions to successor state $s' \in S$ with probability $\mathbb{P}[s' \mid s, \pi(s)]$. A run is a sequence $\phi^\pi = (s_0, a_0, r_0 \dots, s_{h-1}, a_{h-1}, r_{h-1}, s_h)$ that starts in s_0 and where h actions are applied according to π, i. e., the sequence is such that $a_t := \pi(s_t)$, $s_{t+1} \sim T(s_t, a_t, \cdot)$, and $r_t := R(s_t, a_t)$ for all $0 \le t < h$. We denote the accumulated reward of a run ϕ^π with $R(\phi^\pi) = \sum_{t=0}^{h-1} r_t$.

The expected reward of a policy π can be computed as the solution to a set of equations that describes $V^\pi := V^\pi(s_0)$, which is given in terms of state-value functions $V^\pi(s)$ and action-value functions $Q^\pi(s, a)$, where

$$\begin{aligned} V^\pi(s) &= \begin{cases} 0 & \text{if } s \text{ is terminal} \\ Q^\pi(s, \pi(s)) & \text{otherwise, and} \end{cases} \\ Q^\pi(s, a) &= R(s, a) + \textstyle\sum_{s' \in S} \mathbb{P}[s' \mid s, a] \cdot V^\pi(s'). \end{aligned}$$

The optimal policy $\pi^\star$ can be derived from the related Bellman optimality equation (Bellman 1957; Bertsekas 1995) as the policy with the highest expected reward among all policies, i. e., as $\pi^\star := \arg\max_{\pi \in \Pi} V^\pi$.

Each policy π induces a set of outcomes $\mathcal{O}^\pi$, which consists of each accumulated reward r that can be achieved under application of π paired with the probability of r, i. e., $\mathcal{O}^\pi = \{(r, \mathbb{P}[R(\phi^\pi) = r]) \mid \mathbb{P}[R(\phi^\pi) = r] > 0\}$. We call the highest possible outcome $P^\pi := \max_{(r,p) \in \mathcal{O}^\pi} r$ of a policy π the potential of π. We abbreviate the policy with the highest potential among all policies with π^+. Moreover, we abbreviate the expected reward of $\pi^\star$ (π^+) with $V^\star$ (V^+), its potential with $P^\star$ (P^+), its set of outcomes with $\mathcal{O}^\star$ ($\mathcal{O}^+$) and a run under the policy with $\phi^\star$ (ϕ^+).

An example MDP is depicted in Figure 1. It shows an instance of the NAVIGATION domain of IPPC 2011, where an agent is initially located in grid cell s_0 and aims to reach cell $s^\star$ by moving within the grid. On its way, the agent has to cross the middle row at some point, where it gets stuck with increasing probability from left (20%) to right (80%). The agent has no possibility to break free once it is stuck, and it receives a reward of -1 in each step unless it is located in $s^\star$. If we consider the IPPC horizon of $h = 40$, the agent receives an accumulated reward of $R(\phi^\star) = -6$, $R(\phi^{\pi_2}) = -4$, and $R(\phi^+) = -2$ if it successfully passes the middle row, and of -40 if it gets stuck regardless of the applied policy. The expected reward of $\pi^\star$ is $V^\star = -12.8$, and it induces the set of outcomes $\mathcal{O}^\star = \{(-6, 0.8), (-40, 0.2)\}$ with potential $P^\star = -6$. For π^+, we have $V^+ = -32.4$, $\mathcal{O}^+ = \{(-2, 0.2), (-40, 0.8)\}$, and $P^+ = -2$.

The MDP-Evaluation Stopping Problem. The problem we face in this paper is the MDP-ESP$_k^u$, where a sequence of at least $k > 0$ and at most $u \ge k$ runs is performed on an MDP. A strategy σ assigns policies $\pi_1, \dots, \pi_n$ to runs on the MDP and stops the evaluation after n ($k \le n \le u$) runs. The objective is to find a strategy σ that maximizes the average accumulated reward of the last k runs, i.e., where

$$\mathcal{R}_k^u(\sigma) := \frac{1}{k} \cdot \sum_{i=n-k+1}^{n} R(\phi_i^{\pi_i})$$

is maximal in expectation. The quality of a strategy σ is measured in terms of its expected average reward $\mathbb{E}[\mathcal{R}_k^u(\sigma)]$.

An instance of the MDP-ESP not only optimizes the evaluation of a sequence of policies on a base-MDP, it can be described in terms of a meta-MDP itself. A state in the meta-MDP is given by a sequence of rewards $(r_1, \dots, r_n)$, where $r_i := R(\phi_i^{\pi_i})$ for $i = 1, \dots, n$ is the accumulated reward of the runs that were performed before reaching a state. The meta-MDP provides an action a_π for each policy $\pi \in \Pi$, which encodes the execution of policy π on the base-MDP. Furthermore, there is a single action $a_\otimes$ that encodes the decision to stop the MDP-ESP and evaluate the meta-run under σ based on the result of the last k runs on the base-MDP. We describe the transition function of the meta-MDP in terms of its actions: $a_\otimes$ is not applicable in a state $(r_1, \dots, r_n)$ if $n < k$, and it is the only applicable action in a state $(r_1, \dots, r_u)$. Its application leads deterministically to an absorbing terminal state and yields a reward $R((r_1, \dots, r_n), a_\otimes) = \frac{1}{k} \cdot \sum_{i=n-k+1}^{n} r_i$. The application of an action a_π in state $(r_1, \dots, r_n)$ incurs no reward and leads to a state $(r_1, \dots, r_n, r)$, where r is drawn according to the outcome function $r \sim \mathcal{O}^\pi$ of the executed policy π.

Theoretical Analysis

Upper and Lower Bounds. Most optimization problems on MDPs have in common that the theoretical upper bound of the expected reward of a policy is less than or equal to the expected reward $V^\star$ of the optimal policy $\pi^\star$. Examples include the Multi-Armed Bandit problem and Online Reinforcement Learning (RL), where a logarithmic regret on $V^\star$ must be accepted (Lai and Robbins 1985) since all runs are evaluation runs and $\pi^\star$ must be derived from experience of the interaction with the environment. In Offline RL (or Probabilistic Planning), all runs are evaluation runs as well, but $\pi^\star$ can be computed before evaluation starts and $V^\star$ can hence be achieved if $\pi^\star$ is available (Sutton and Barto 1998).

This is different in the MDP-ESP. Since the agent is allowed to decide in hindsight if the last k runs were good enough to be used for evaluation, there are strategies that allow an expected performance that is at least as good as $V^\star$ for all instances of the MDP-ESP^u_k. Moreover, it is impossible to achieve a result that is higher than P^+.

Theorem 1. *$V^\star \leq \max_\sigma \mathbb{E}[\mathcal{R}^u_k(\sigma)] \leq P^+$ for all $k > 0$ and $u \geq k$.*

Proof sketch: We start with a discussion of the lower bound $V^\star$ by considering the subset of instances where $u = k$. The MDP-ESP^k_k, where each performed run is an evaluation run reduces to Offline RL, and the optimal strategy is hence the strategy that only executes $\pi^\star$. (We denote the strategy that executes $\pi^\star$ in each run and never stops prematurely with $\sigma_{\pi^\star}$ in this proof sketch). Since the expected reward of each run under $\pi^\star$ is $V^\star$, the expected average reward of the whole sequence of k runs is $V^\star$ as well. If we apply $\sigma_{\pi^\star}$ to instances where $u > k$, the additional, prepended runs have no effect as they are not used for evaluation. Therefore, $\mathbb{E}[\mathcal{R}^u_k(\sigma_{\pi^\star})] = V^\star$ for any instance of the MDP-ESP, and the lower bound is as stated in Theorem 1.

P^+ is an upper bound of the MDP-ESP^u_k since it is the highest possible outcome of all policies, and it is therefore impossible to achieve a higher expected reward in a run and a higher expected average reward in a sequence of k runs. Moreover, the bound is tight for the MDP-ESP$^\infty_k$, i. e., the subset of instances with an infinite number of runs and a finite number of evaluation runs. Since any sequence of outcomes will occur eventually in an infinite number of runs, the optimal strategy for the MDP-ESP$^\infty_k$ applies π^+ in every run until a sequence of k runs in a row yields P^+, and the expected average reward of this strategy is P^+.

Optimal Strategies. Even though we have provided tight upper and lower bounds for the MDP-ESP, the expected reward of optimal policies in the space between the discussed extreme cases is not yet clear. It is not hard to show that the expected reward of the MDP-ESP^u_k under an optimal strategy increases strictly from $V^\star$ to P^+ with increasing u for all k (unless $\pi^\star = \pi^+$ and $\pi^\star$ deterministic, in which case $\max_\sigma \mathbb{E}[\mathcal{R}^u_k(\sigma)] = V^\star = P^+$ for all $k > 0$ and $u \geq k$). We omit a formal proof sketch for space reasons, though, and turn our attention to the MDP-ESP$^u_1$ instead. It corresponds to a finite-horizon version of the House-Selling Problem (Karlin 1962), where offers come in sequentially for a house an agent wishes to sell. The offers are drawn from a known probability distribution, and the agent has to accept or decline each offer right after receiving it. The agent aims to sell the house for the highest price among at most u offers. The subset of instances where only a single run is used for evaluation is interesting for our purposes because an optimal strategy can be described with two simple functions: the target reward function $t : \{1, \ldots, u-k\} \to \mathbb{R}$ describes the average reward $t(n)$ of the last k runs that must have been achieved in a state $(r_1, \ldots, r_{u-n})$ to apply $a_\otimes$, and the policy application function app : $\{1, \ldots, u-k\} \to \Pi$ specifies the policy that is taken otherwise.

n	$\pi^\star$	π_2	π^+	app(n)
1	$\mathbf{-12.8}$	-22	-32.4	$\pi^\star$
2	$\mathbf{-7.36}$	-8.4	-10.64	$\pi^\star$
3	-6.272	$\mathbf{-5.68}$	-6.288	π_2
4	-5.936	$\mathbf{-4.84}$	-4.944	π_2
5	-5.768	-4.42	$\mathbf{-4.272}$	π^+
6	-5.654	-4.136	$\mathbf{-3.8176}$	π^+

Table 1: The optimal strategy for the MDP-ESP$^u_1$ on the NAVIGATION instance of Figure 1 applies app(n) if the current result is less than $t(n)$ (in bold) and stops otherwise.

A solution for the MDP-ESP$^u_1$ is to compute these functions by applying backward induction, a popular method to solve full information optimal stopping problems where an increasing number of available runs u is considered (Gilbert and Mosteller 1966). We know that it is optimal to apply $\pi^\star$ in the MDP-ESP$^1_1$, and the expected reward is $V^\star$, i. e., app$(1) = \pi^\star$. Now consider the MDP-ESP$^2_1$: if, after the first run, our current result is higher than $V^\star$, we stop the evaluation, since the remaining problem is exactly the MDP-ESP$^1_1$ with expected reward $V^\star$. Otherwise, we apply app$(1) = \pi^\star$. The target reward function is therefore such that $t(1) = V^\star$. The policy that is applied in the first run of the MDP-ESP$^2_1$, app(2), can be computed as the policy that maximizes the expected reward given $t(1)$, which in turn allows the computation of $t(2)$ and so on.

Take for example the NAVIGATION domain that was presented earlier. We have app$(1) = \pi^\star$ and $t(1) = V^\star = -12.8$. If we apply $\pi^\star$ in the first run of the MDP-ESP$^2_1$, we achieve a reward of -6 with probability 0.8 and of -40 with probability 0.2. Since we prefer not to stop in the latter case, we get $t(2) = (0.8 \cdot (-6)) + (0.2 \cdot t(1)) = -7.36$. Table 1 shows these computations for all three policies of the NAVIGATION instance that are depicted in Figure 1. It reveals that it is optimal to execute π^+ if five or more runs are left, and to stop only if a run successfully crosses the middle row and yields a reward of -2. If three or four runs are left, the strategy proposes the execution of policy π_2, and $\pi^\star$ is executed only in the last two runs. The example shows that restricting to strategies that consider only $\pi^\star$ and π^+ is not sufficient for optimal behavior.

Complexity. It is not hard to see that finding an optimal strategy for the general MDP-ESP^u_k is practically intractable. It corresponds to solving the meta-MDP with a search space of size $(|\Pi| \cdot \mathcal{O}_{\max})^u$ with $\mathcal{O}_{\max} = \max_{\pi \in \Pi} |\mathcal{O}^\pi|$, which is intractable even if $|\Pi|$ were manageable (which is usually not the case). We have discussed three special cases of the MDP-ESP, though, and we have shown that an optimal strategy for two of them – the MDP-ESP^k_k and the MDP-ESP$^\infty_k$ – can be derived in constant time under the assumption that the cost of deriving policies in the base-MDP can be neglected. For the third, we have provided an algorithm that regards all outcomes of all policies $\pi \in \Pi$ in at most $(u-k)$ decisions, and it is hence linear in u, $|\Pi|$, and $\mathcal{O}_{\max}$. Even though the

	$u = k$	$k < u < \infty$	$u = \infty$
$k = 1$	$O(1)$	$O(u \cdot \lvert\Pi\rvert \cdot \mathcal{O}_{\max})$	$O(1)$
$1 < k < \infty$	$O(1)$	$O\big((\lvert\Pi\rvert \cdot \mathcal{O}_{\max})^u\big)$	$O(1)$

Table 2: Complexity results for different instances of the MDP-ESP^u_k given an oracle for the underlying base-MDP.

dependence on $|\Pi|$ is discouraging as the computation of all policies is intractable, it also shows that efficient approximations of good quality are possible if we consider only a subset of Π. The complexity results are summarized in Table 2. The manageable cases all have in common that two simple functions that map the number of remaining runs to a target reward and a policy suffice to describe the strategy. In the next section, we show how these ideas can be used to approximate the general case with strategies of high quality.

Strategies for the MDP-ESP

We consider three possible states of a-priori information: first, we look at the case where $\pi^\star$ and $V^\star$ are unknown, and assume that the computation of a policy and its execution are interleaved. We continue with MDPs where $\pi^\star$ and $V^\star$ can be computed, and present two strategies, one that aims at avoiding bad luck and one that pushes its luck under execution of $\pi^\star$. In the last part of this section, we present a strategy that mixes $\pi^\star$ and π^+ and prove that it is theoretically superior to the other considered strategies.

Secretary Problem. While most instances of the IPPC are such that they cannot be solved in the given time, it is always possible to perform more than k runs. Even if the available time is distributed equally among all planning steps beforehand, there are reasons for spare time: the PROST planner (Keller and Eyerich 2012) that is used for our experiments detects reward locks; it recognizes states with only one reasonable action; it is able to solve an encountered state even in larger MDPs if the remaining horizon is small; and it reuses decisions if it encounters a state more than once.

If the optimal policy is not available, the MDP-ESP^u_k is similar to the SecP, which is a variant of the finite-horizon House-Selling Problem where the underlying probability distribution is not revealed to the agent. It involves a single secretarial position and u applicants which are interviewed sequentially in a uniformly random order. Applicants can be ranked unambiguously, and the decision to hire a candidate has to be made right after the interview and is irrevocable. The objective is to have the highest probability of selecting the best applicant of the whole group, and it can be shown that an optimal solution is to reject the first $\lfloor \frac{u}{e} \rfloor$ applicants ($\approx 36.8\%$) and select the first subsequent candidate that is ranked higher than all candidates before (e.g., Ferguson 1989; Bruss 2000).

To apply the SecP strategy to the MDP-ESP^u_k, we pretend that all sequences of k consecutive runs are independent, identically distributed data points. We perform $\lfloor \frac{u-k+1}{e} \rfloor$ runs and stop as soon as the last k runs yield a higher average reward than all data points before. It is important to note that the data points are of course not independent and identically distributed in our setting – each data point depends on the previous one(s) unless $k = 1$, since two consecutive samples differ only in a single reward. Our empirical evaluation (where only $\pi^\star$ is executed) shows that the SecP is a strategy that improves over $V^\star$ significantly nonetheless.

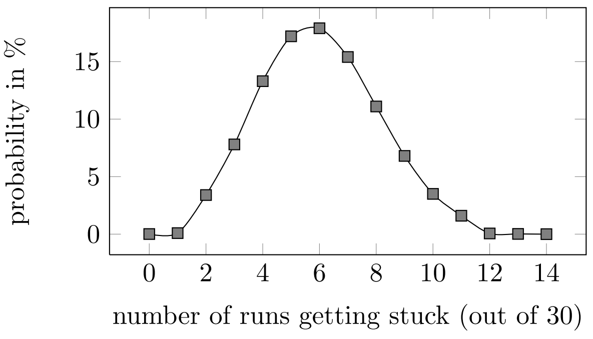


Figure 2: Probability of getting stuck x times in 30 runs of the example NAVIGATION instance.

Meet-The-Expectations. The IPPC benchmarks offer MDPs of varying complexity, including some instances where $\pi^\star$ can be computed. However, it is always possible that the execution of a policy is unfortunate. Take, for example, the NAVIGATION instance from Figure 1. If we execute $\pi^\star$ for $k = 30$ runs, the expected reward $V^\star$ is achieved if the agent ends up stuck exactly six times. Figure 2, which depicts how likely it is that the agent gets stuck, reveals that the probability that it gets stuck more than six times is roughly 40%. A strategy that avoids bad luck if more than k runs are available is a first step in the right direction. We call the strategy with $t_{\sigma_{\text{MTE}}}(n) = V^\star$ and $\text{app}_{\sigma_{\text{MTE}}}(n) = \pi^\star$ for all n the Meet-The-Expectations (MTE) strategy.

Theorem 2. *$V^\star \leq \mathbb{E}[\mathcal{R}^u_k(\sigma_{\text{MTE}})] \leq P^\star$ for all $k > 0$ and $u \geq k$.*

Proof sketch: If $\pi^\star$ is deterministic, all inequalities are trivially equalities. Otherwise, both inequalities hold since only $\pi^\star$ is applied. The first is strict for $u > k$ since we accept lucky results and improve unlucky ones, and the second is strict even for most instances of the MDP-ESP$^\infty_1$ since MTE stops with a result between $V^\star$ and P^+.

Pure Strategy. We have presented a strategy that avoids bad luck while applying $\pi^\star$, so the question naturally arises how to push the envelope and aim for good luck. After all, Figure 2 shows that the probability of getting stuck less than six times is also approximately 40%. Since an optimal target reward function is intractable in practice even if $\text{app}_{\sigma_{\text{PS}}} = \pi^\star$ for all n, we use a simulation approach in the Pure Strategy (PS) to estimate $t_{\sigma_{\text{PS}}}$. PS performs a sequence of m simulations $(\oslash_1, \dots, \oslash_m)$ (m is a parameter of the algorithm), where each $\oslash_i$ consists of u runs $(\phi^\star_{i1}, \dots, \phi^\star_{iu})$. We use the simulations to compute the target reward function as $t_{\sigma_{\text{PS}}}(n) = \text{median}(\mathcal{R}^n_{\max}(\oslash_1), \dots, \mathcal{R}^n_{\max}(\oslash_m))$, where $\mathcal{R}^n_{\max}(\oslash_i) = \max_{l \in \{1,\dots,n\}}(\frac{1}{k}\sum_{s=l}^{l+k-1} R(\phi^\star_{is}))$.

Algorithm 1: Mixed Strategy for MDP-ESP^u_k with $u > k$

```
compute_mixed_strategy($u, k, m$):
    let all $t(n) \leftarrow -\infty$, app$(n) \leftarrow \pi^\star$ and $n_0 \leftarrow 1$
    for $i = 0, \dots, k$ do
        sample_run_sequences($u, k, m, i$)
        update_strategy($u, k, i$)
sample_run_sequences($u, k, m, i$):
    for $j = 1, \dots, m$ do
        for $n = 1, \dots, u$ do
            if ($n$ mod $k$) $< i$ then $r_n \leftarrow$ sample($\pi^+$)
            else $r_n \leftarrow$ sample($\pi^\star$)
            if $n > k$ then
                $t_{ij}(n–k) \leftarrow \max_{l\in\{1,\dots,n\}}(\frac{1}{k}\sum_{s=l}^{l+k-1} r_s)$
    for $n = 1, \dots, u - k$ do
        $t_i(n) \leftarrow$ median($t_{i1}(n), \dots, t_{im}(n)$)
update_strategy($u, k, i$):
    for $n = u - k, \dots, n_0$ do
        if $t_i(n) > t(n)$ then $t(n) \leftarrow t_i(n)$
        else
            for $l = n_0, \dots, n$ do
                if ($l$ mod $k$) $< i$ then app($n$) $\leftarrow \pi^+$
            $n_0 \leftarrow n$ and return
```

Theorem 3. *$V^\star \leq \mathbb{E}[\mathcal{R}^u_k(\sigma_{\text{PS}})] \leq P^\star$ for all $k > 0$ and $u \geq k$. For all finite $k > 0$, $\mathbb{E}[\mathcal{R}^\infty_k(\sigma_{\text{PS}})] = P^\star$ and $\mathbb{E}[\mathcal{R}^u_k(\sigma_{\text{PS}})]$ is monotonically increasing in u and converges to $P^\star$.*

Proof sketch: $V^\star$ and $P^\star$ are bounds since only $\pi^\star$ is applied and $\mathbb{E}[t_{\sigma_{\text{PS}}}(n)] \geq V^\star$ for a sufficiently large m. $\mathbb{E}[t_{\sigma_{\text{PS}}}(n)]$ increases monotonically in n from $V^\star$ to a value $\leq P^\star$ for $u > k$ since the number of considered data points in the simulations grows, and it reaches $P^\star$ for $u = \infty$ and therefore $\mathbb{E}[\mathcal{R}^\infty_k(\sigma_{\text{PS}})] = P^\star$. $\mathbb{E}[\mathcal{R}^u_k(\sigma_{\text{PS}})]$ is monotonically increasing since the expected reward is bounded from below by a probability weighted sum of all target rewards.

Mixed Strategy. π^+ can not only be derived when $\mathcal{O}^\pi$ is available for all π, but it can be computed as $\pi^+ := \max_{\pi\in\Pi} P^\pi(s_0)$, which is described by the set of equations

$$\begin{aligned} P^\pi(s) &= \begin{cases} 0 & \text{if } s \text{ is terminal} \\ W^\pi(s, \pi(s)) & \text{otherwise, and} \end{cases} \\ W^\pi(s,a) &= R(s,a) + \max_{s' \in \text{succ}(s,a)} P^\pi(s'), \end{aligned}$$

with succ$(s,a) := \{s' \mid \mathbb{P}[s' \mid s,a] > 0\}$. Note that the only difference to the Bellman optimality function is that $W^\pi(s,a)$ only cares about the best outcome while $Q^\pi(s,a)$ uses the weighted average of all outcomes. By turning these equations into assignment operators that extend the backup function of a THTS algorithm, we can describe algorithms – in our case a UCT* variant – that derive π^+ as a side-effect of the $\pi^\star$ computation and without notable overhead.

While it is possible to use π^+ as the base-policy of PS, it turns out that u has to be prohibitively large to outperform PS based on $\pi^\star$. Instead, we generate a policy that is inspired by our analysis of the MDP-ESP$^u_1$, where a function app$_{\sigma_{\text{MS}}}(n)$ is used to describe which policy is executed solely in terms of the number of remaining runs. We restrict ourselves to the policies $\pi^\star$ and π^+ in our version of a Mixed Strategy (MS), but adding more policies is an interesting (and certainly non-trivial) topic for future work. Initially, MS computes a function t_0 (the index stands for the number of runs under π^+ in each data point) which is equivalent to $t_{\sigma_{\text{PS}}}$. MS, which is depicted in Algorithm 1, continues by performing simulations where π^+ is executed in i out of k runs. The functions $t_{\sigma_{\text{MS}}}$ and app$_{\sigma_{\text{MS}}}$ are updated after the ith iteration by finding the largest n where $t(n) \geq t_i(n)$, i. e., by finding the element in the sequence where the number of runs is small enough that an additional execution of π^+ does not pay off anymore. Note that our implementation stops the computation prematurely when a t_i does not alter $t_{\sigma_{\text{MS}}}$ in the update procedure (unlike the depicted Algorithm 1).

Theorem 4. *$\mathbb{E}[\mathcal{R}^u_k(\sigma_{\text{PS}})] \leq \mathbb{E}[\mathcal{R}^u_k(\sigma_{\text{MS}})]$ for all $k > 0$ and $u \geq k$ and $\mathbb{E}[\mathcal{R}^\infty_k(\sigma_{\text{MS}})] = P^+$ for all finite $k > 0$.*

Proof sketch: If we assume that the number of simulations is sufficiently high, then it is either not beneficial to apply π^+ and MS reduces to PS, or it is beneficial and $\mathbb{E}[\mathcal{R}^u_k(\sigma_{\text{MS}})] > \mathbb{E}[\mathcal{R}^u_k(\sigma_{\text{PS}})]$. MS converges towards P^+ with an increasing number of u since at some point it pays off to only apply π^+ with an expected reward of P^+ in the limit.

Experimental Evaluation

To evaluate our algorithms empirically, we perform experiments on the domains of IPPC 2011 and 2014. We use the UCT* algorithm (Keller and Helmert 2013) to solve the base-MDP, an algorithm that is specifically designed to find high-quality policies in finite-horizon MDPs even of larger size when a declarative model of the MDP is provided. It is a THTS algorithm that can be described in terms of four ingredients: the action selection is based on the UCB1 formula (Auer, Cesa-Bianchi, and Fischer 2002), Monte-Carlo sampling is used to simulate stochastic outcomes, and a partial Bellman backup function is used to propagate collected information in the search tree. Since we use the implementation of UCT* that comes with the probabilistic planning system PROST, the used heuristic function is the default heuristic of the planner. It performs a lookahead based on a sequence of iterative deepening searches on the most-likely determinization of the MDP (Keller and Eyerich 2012).

We have altered the THTS framework to perform a sequence of searches with an increasing horizon, a change that is inspired by the Reverse Iterative Deepening approach that is used in GLUTTON (Kolobov et al. 2012). A higher number of instances can be solved since state-values of solved states are reused, which occurs more often if the horizon is increased iteratively (the possibly weaker anytime performance is not important here). The resulting algorithm is able to solve 34 instances of the 120 existing IPPC benchmarks: four of CROSSING TRAFFIC, five of ELEVATORS, three of GAME OF LIFE, all NAVIGATION instances, and six both of SKILL TEACHING and TRIANGLE TIREWORLD. Apart from the ELEVATORS domain, where $\pi^\star$ can be derived for the instances 1, 2, 4, 7, and 10, the instances with the lowest indices are solved. The number of evaluation runs k is set to 30 in all experiments, which corresponds to the number of

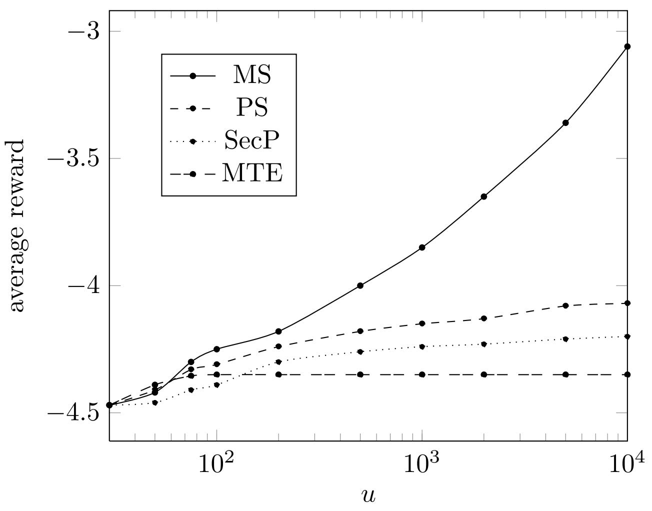


Figure 3: Results for an increasing number of available runs u on the first instance of the CROSSING TRAFFIC domain with $V^\star \approx -4.43$, $P^\star = -4$, and $P^+ = -2$.

evaluation runs at both IPPC 2011 and 2014, and the values for u are increased from 30 to 10000. Each experiment is conducted 20 times and average results are reported.

Figure 3 shows the average reward of the experiments on the first instance of CROSSING TRAFFIC with increasing u. We have selected the instance since comparably small values of u showcase our theoretical results. Nevertheless, if u is large enough, any instance could have been selected. Table 3 shows normalized IPPC scores which are computed over the average of the results of the experiment sets for $u = 200$ and $u = 1000$. A score of 1.0 is assigned to the best performing strategy, and a score of 0.0 is assigned to an artificial baseline solver with an average reward of $V^\star$ in all instances. All other scores are normalized with the procedure that is used at IPPC with the described minimal and maximal values.

The expected reward in the depicted CROSSING TRAFFIC instance is $V^\star \approx -4.43$. The simple MTE strategy is already an improvement over the baseline solver. It reliably avoids bad luck already with only a few extra runs. However, it has converged to -4.35, a value that is only little above $V^\star$, in the CROSSING TRAFFIC instance already with $u = 100$ and does not improve any further. The same can be observed in Table 3, where the result does not improve when u is increased from 200 to 1000. In that experiment set, merely 35 to 40 runs suffice to avoid bad luck reliably over all instances, and in between 100 and 200 runs suffice for convergence. Except for special cases like an MDP-ESP$^u_1$ with $|\{(r, p) \in \mathcal{O}^\star \mid r \geq V^\star\}| = 1$, MTE converges to a value that is greater than $V^\star$ yet less than $P^\star$.

The SecP strategy does not suffer from this problem – the larger u, the better the result in our experiments. It quickly outperforms MTE even though the availability of the optimal policy is no condition for the applicability of the strategy. It should nevertheless be noted that the presented SecP results are based on an implementation that executes the optimal policy in all experiments to allow a better comparison of the strategies. In this setting, the SecP converges to $P^\star$ with growing u: since only $\pi^\star$ is applied, it cannot improve over $P^\star$, and since $\lfloor \frac{u-k+1}{e} \rfloor$ grows with u, the target reward and in turn the expected average reward approach $P^\star$.

The two sampling-based strategies yield comparable results in the experiment on all solvable IPPC instances that is given in Table 3, and both outperform the other considered algorithms significantly and in all domains. Obviously, simulation based approaches are well-suited to create strategies of high quality. It is not surprising that PS outperforms MTE with increasing u since PS converges to $P^\star$ according to Theorem 3 while MTE usually does not, and since PS reduces to MTE if it ever were reasonable. The theoretical relation between the performance of PS and SecP is an open question, but it appears that the latter converges to $P^\star$ with a slower pace. PS often has an edge over MS when u and k are close, since applying π^+ is rarely reasonable in these cases and MS can hence only be misled by its additional possibilities. Increasing the number of simulations m will neglect the slight advantage PS has in some instances. The larger u compared to k, the larger the advantage of MS over PS. Figure 3, which depicts one of the smaller instances of the used benchmarks, shows this clearly: MS quickly outperforms all other strategies (as soon as it starts to mix in runs under π^+) and converges to $P^+ = -2$. Table 3 also supports this claim since MS outperforms PS in all domains with $u = 1000$. The only exception is the ELEVATORS domain, where $P^\pi(s_0) = 0$ for all policies π since there is a small chance that no passenger shows up. If ties were broken in favor of better action-values in our implementation of π^+ (instead of uniformly at random), MS and PS would perform equally in the ELEVATORS domain.

Discussion

We started with the work at hand due to the evaluation schema that was used at IPPC 2014. Only three IPPC solvers made use of the rule that more than 30 runs are allowed: both versions of PROST and G-PACK, a variant of the GOURMAND planner (Kolobov, Mausam, and Weld 2012). The latter does not reason over the MDP-ESP, though. It simplifies the original MDP by considering at most N outcomes for all actions, and computes and executes the policy with highest expected reward in the simplified MDP. If time allows, this process is repeated with a larger N, which is the only reason that more than k evaluation runs are performed.

Therefore, only our submissions actually considered the MDP-ESP as the relevant optimization problem. However, most of the work described in this paper was done *after* the competition – only the SecP strategy and a PS variant with a target reward that is independent from the number of remaining runs were applied at IPPC 2014. Note that both are strategies that aim at optimizing the evaluation of $\pi^\star$ or a near-optimal policy. PROST 2011 applied the SecP strategy in 33 out of 80 instances, while PROST 2014 used it in 28 and PS in another six instances. Even though we were able to improve the average reward in 22 (19) instances with the SecP strategy and in five with the PS variant in the 2014 (2011) version, the total IPPC scores are mostly unaffected: had we stopped evaluation after 30 runs in all instances with

	Crossing		Elevators		Game		Navigation		Skill		Tireworld		Total	
u	200	1000	200	1000	200	1000	200	1000	200	1000	200	1000	200	1000
MTE	0.21	0.21	0.26	0.26	0.28	0.28	0.37	0.37	0.23	0.23	0.27	0.27	0.27	0.27
SecP	0.34	0.45	0.34	0.59	0.27	0.52	0.57	0.85	0.37	0.63	0.38	0.68	0.38	0.62
Pure	**0.59**	0.84	**0.59**	**0.98**	0.63	**0.98**	**0.74**	0.93	0.57	**0.97**	**0.62**	0.96	0.62	0.94
Mixed	0.57	**1.00**	0.55	0.93	**0.68**	**0.98**	0.73	**1.00**	**0.60**	0.96	**0.62**	**0.99**	**0.63**	**0.98**

Table 3: IPPC scores of the proposed algorithms for the MDP-ESP$^u_{30}$ on instances of IPPC 2011 and 2014 that can be solved with PROST. The expected reward $V^\star$ of the optimal policy $\pi^\star$ is used as the minimum for normalization.

both solvers, the final result would differ only slightly with total IPPC scores of 0.816 (-0.09) and 0.773 (+0.04) for the PROST versions and of 0.739 (+0.05) for G-PACK. Even though the differences are small (we believe this is due to the poor minimum policy at IPPC, the individual results indicate that considering the MDP-ESP does pay off), we would like to emphasize that the PROST competition results should not be used for comparison. Otherwise, this includes instances where PROST achieved a result that is far above $V^\star$.

There are many applications for OSPs, including sponsored search (e.g., Babaioff et al. 2007; Zhou and Naroditskiy 2008), online auctions (e.g., Hajiaghayi, Kleinberg, and Parkes 2004), or optimal stock market behavior (e.g., Griffeath and Snell 1974; Shiryaev, Xu, and Zhou 2008). Most applications are based on variants of the SecP that differ in the number of applicants that must be selected as in the Multiple-Choice SecP (MCSP) (Freeman 1983), that have full information as in the House-Selling Problem or where the selected values must be maximized under constraints on each element as in the Online Knapsack Problem (Marchetti-Spaccamela and Vercellis 1995). The MDP-ESP differs from the MCSP in two details: first, the selected applicants must show up consecutively, and second, the probability distribution that gives the next sample must be selected from a known set of probability distributions.

The former difference does not alter the applicability of our algorithms, since scenarios where k consecutive data points must be selected exist, e. g., with resources that decay with time, goods on an assembly line, or in traffic control. Moreover, our algorithms can also be applied to a variant of the MDP-ESP where k results can be selected in arbitrary order. An exemplary application is, as in the MCSP, the plan to hire k employees. In our scenario, all applicants have provided application documents, which allow the estimation of a probability distribution over the candidate's aptitude (e. g., grades or experience influence the expectation and the rest of the CV the variance and hence the potential). We invite at most u of the applicants (more than u applicants are necessary since we do not restrict the number of times a "type of applicant" is selected), and we have to decide right after the interview if we hire the candidate with knowledge of the aptitude. This problem differs from the MDP-ESP only in the way the k applicants are selected. However, it is easy to see that Theorem 1 also holds, that the MTE algorithm can be applied with the same bounds on the expected reward (i. e., Theorem 2 holds) and that Theorems 3 and 4 hold for versions of PS and MS where line 12 of Algorithm 1 is replaced with an equation that sums the k largest rewards independently from their position. Since this is a simple and useful generalization of the popular MCSP, it is well-suited to describe existing OSP applications more realistically.

Conclusion

We have shown how the MDP-ESP is constructed as a meta-MDP where actions encode the execution of a policy in an underlying base-MDP. The expected reward of the optimal policy of the base-MDP is a lower bound for optimal strategies in the MDP-ESP. We have derived a procedure from the Bellman optimality equation to compute the policy that maximizes its potential, and have presented an upper bound for MDP-ESP strategies that corresponds to the potential of π^+. While the general MDP-ESP is intractable in practice, we have shown that there are special cases – the MDP-ESP^k_k, the MDP-ESP$^\infty_k$, and the MDP-ESP$^u_1$ – where the knowledge of $\pi^\star$ or π^+ suffices to compute an optimal strategy.

We have introduced four different strategies for the MDP-ESP based on our theoretical analysis. A strategy that is derived from the related SecP not only allows us to treat MDPs as MDP-ESP instances even though the optimal strategy cannot be computed , but is furthermore a strategy of high quality that allows to exceed the average expected reward of the underlying policy both in the experiments presented in this paper and at IPPC 2014. If $\pi^\star$ is available, we show that avoiding bad luck is already an improvement over a base policy that stops after k runs of $\pi^\star$. However, by pushing the luck under application of the optimal policy, we derive a strategy that converges towards $P^\star$ and hence to a result that can only be achieved in a very lucky set of evaluation runs. We showed empirically that the corresponding strategy PS is of high quality if u and k are similar. The use of MS, which switches between executing $\pi^\star$ and π^+, becomes more appealing with an increasing difference between u and k. It outperforms all other approaches significantly in our empirical evaluation and demonstrates that it is indeed sometimes better to be lucky than good.

Acknowledgments. We would like to thank Scott Sanner from ANU Canberra and Malte Helmert from the University of Basel for pointing us to the Secretary Problem, and Andrey Kolobov from Microsoft Research, Redmond for valuable information about GLUTTON and GOURMAND.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time Analysis of the Multiarmed Bandit Problem. *Journal of Machine Learning Research* 47:235–256.

Babaioff, M.; Immorlica, N.; Kempe, D.; and Kleinberg, R. 2007. A Knapsack Secretary Problem with Applications. In *Proceedings of the 10th International Workshop on Approximation (APPROX 2007)*, 16–28. Berlin, Heidelberg: Springer-Verlag.

Bellman, R. 1957. *Dynamic Programming*. Princeton University Press.

Bertsekas, D., and Tsitsiklis, J. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

Bertsekas, D. 1995. *Dynamic Programming and Optimal Control*. Athena Scientific.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 2000. Stochastic Dynamic Programming with Factored Representations. *Artificial Intelligence (AIJ)* 121(1–2):49–107.

Bruss, F. T. 2000. Sum the Odds to One and Stop. *The Annals of Probability* 28(3):1384–1391.

Dynkin, E. B. 1963. The Optimum Choice of the Instant for Stopping a Markov Process. *Soviet Mathematics Doklady* 4.

Ferguson, T. S. 1989. Who Solved the Secretary Problem? *Statistical Science* 4(3):282–289.

Freeman, P. R. 1983. The Secretary Problem and its Extensions: A Review. *International Statistical Review* 51(2):189–206.

Gilbert, J. P., and Mosteller, F. 1966. Recognizing the Maximum of a Sequence. *Journal of the American Statistical Association* 61(313):35–73.

Griffeath, D., and Snell, J. L. 1974. Optimal Stopping in the Stock Market. *The Annals of Probability* 2(1):1–13.

Hajiaghayi, M. T.; Kleinberg, R.; and Parkes, D. C. 2004. Adaptive Limited-supply Online Auctions. In *Proceedings of the 5th ACM Conference on Electronic Commerce*, 71–80. New York, NY, USA: ACM.

Karlin, S. 1962. Stochastic Models and Optimal Policy for Selling an Asset. *Studies in Applied Probability and Management Science* 148–158.

Keller, T., and Eyerich, P. 2012. PROST: Probabilistic Planning Based on UCT. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 119–127. AAAI Press.

Keller, T., and Helmert, M. 2013. Trial-based Heuristic Tree Search for Finite Horizon MDPs. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 101–105.

Kolobov, A.; Dai, P.; Mausam; and Weld, D. 2012. Reverse Iterative Deepening for Finite-Horizon MDPs with Large Branching Factors. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 146–154.

Kolobov, A.; Mausam; and Weld, D. 2012. LRTDP vs. UCT for Online Probabilistic Planning. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012)*, 1786–1792.

Lai, T. L., and Robbins, H. 1985. Asymptotically Efficient Adaptive Allocation Rules. *Advances in Applied Mathematics* 6(1):4–22.

Marchetti-Spaccamela, A., and Vercellis, C. 1995. Stochastic On-line Knapsack Problems. *Mathematical Programming* 68(1):73–104.

Puterman, M. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley.

Shiryaev, A.; Xu, Z.; and Zhou, X. Y. 2008. Thou Shalt Buy and Hold. *Quantitative Finance* 8(8):765–776.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press.

Zhou, Y., and Naroditskiy, V. 2008. An Algorithm for Stochastic Multiple-Choice Knapsack Problem and Keywords Bidding. In *Proceedings of the 17th International World Wide Web Conference (WWW 2008)*, 1175–1176.

Unplannability IPC Track

Christian Muise **Nir Lipovetzky**

Department of Computing and Information Systems,
University of Melbourne, Australia
{christian.muise,nir.lipovetzky}@unimelb.edu.au

Abstract

The majority of research in the field of automated planning focuses on the synthesis of plans for problems that are solvable. We propose an IPC track to focus on the important and understudied area of *unplannibility*: proving that a planning problem is unsolvable. We will focus on classical planning problems, as methods for determining whether or not unplannability can have wider applications for classical planning problems (e.g., recognizing and avoiding deadends in the state space) as well as solving planning problems with uncertainty (e.g., identifying when a deterministic approximation of the problem is unsolvable). The unplannability track follows similar contests in other fields; for example, the UNSAT track for the field of Boolean Satisfiability. In a similar vein, we hope that the introduction of an unplannability track will foster new innovation for techniques dedicated to identifying planning problems that cannot be solved.

Motivation

The International Planning Competition (IPC) has had a long history of evaluating the state-of-the-art planners on their ability to synthesize plans for solvable planning problems. However, the need for effective detection of *unsolvable* instances often is overlooked. We propose an IPC track that focuses primarily on the task of *unplannability*: identifying when there is no solution to a given problem.

It is common for competitions in other academic fields to include a combination of both solvable and unsolvable instances in the benchmark domains: see for example the SAT (`http://www.satcompetition.org/`) and constraint programming competitions (`http://www.minizinc.org/challenge.html`). Conversely, the IPC historically has included only solvable instances for the planning benchmarks.[1] In order to focus on and showcase the techniques for detecting unplannability, we feel that an IPC track unique from the satisficing track is warranted.

While largely unstudied, the task of unplannability is an important and difficult problem. As identified by Bäckström et al. (2013), the source of many planning problems naturally have an unsolvable counterpart that is as important, if not more-so, than synthesizing solutions. Examples where it is useful include identifying human error in manually generated plans (Goldman, Kuter, and Schneider 2012), penetration detection of software systems (Boddy et al. 2005), and system verification (Edelkamp, Leue, and Visser 2007) among others. As another example, Hoffmann et al. prove a result for cellular automata by enumerating many initial states for the same goal and identifying which ones have no solution (Hoffmann, Fatès, and Palacios 2010). For this work, they use the sound and complete version of FF (Hoffmann and Nebel 2001).

In the remainder of this proposal, we survey some of the existing techniques that may form a basis for competitors in an unplannability IPC track, outline the specifics of the proposed track such as the timeline and evaluation, and conclude with a discussion.

Existing Approaches

Perhaps the most obvious possible solution to unplannability is any sound and complete planner. This is the approach used by (Hoffmann, Fatès, and Palacios 2010), and coupled with an efficient means of expanding states, this approach can be quite successful. However, more direct methods have been considered in the literature recently that aim to target unplannability in particular.

The first clear attempt to solve problems of unplannability was by Bäckström, Jonsson, and Ståhlberg (2013). In this work, Bäckström et al. construct abstractions of the problem in a systematic way, bounded by a constant k, such that the unplannability of many planning problems can be detected efficiently. Their approach is sound, but incomplete unless k is set to be sufficiently high.

The other approach that aims to tackle unplannability directly, was by Hoffmann, Kissmann, and Torralba (2014). Hoffmann et al. adjust the strategies for merge and shrink so that instead of preserving admissibility, the existence (or lack thereof) of a solution is preserved. This approach extended the capability of unplannability detection presented in (Bäckström, Jonsson, and Ståhlberg 2013), and represents the state of the art in unplannability detection.

Two additional techniques recently were introduced that may be amenable to unplannability detection. Keyder, Hoffmann, and Haslum (2014) introduce an improved mechanism for merging fluents in classical planning problems, and indicate that their approach may assist in the detection

[1]Exceptions to this generalization typically are viewed as "bugs" in the benchmark set.

of deadends (cf. footnote 7 on page 508). Finally, Suda (2014) demonstrates how the new planning technique of property directed reachability can be leveraged to determine unplannability (cf. Section 5.8 on page 306).

The existing techniques offer a variety of options for detecting unplannability, though these are limited in number. An IPC track that focuses on unplannability will help to distinguish which techniques are viable, and pave the way for new methods in detecting unplannability.

Track Details

Time-line

Following similar tracks in the past, the proposed time-line for the unplannability track is as follows:

- **Jun, 2015**: Announcement of the track
- **Jul, 2015**: Call for domains / expression of interest
- **Oct, 2015**: Registration deadline
- **Nov, 2015**: Demo problems provided
- **Jan, 2016**: (optional) Initial feedback on buggy output
- **Feb, 2016**: Domain submission deadline
- **Mar, 2016**: Final planner submission deadline
- **Apr, 2016**: Paper submission deadline
- **Mar - May, 2016**: Contest run
- **Jun, 2016**: Results announced in London

Note the emphasis on a January deadline for the initial planner feedback phase. While any sound and complete planner can be used to check for unplannability, existing software may be prone to errors when run to completion. Consequently, we hope to iterate on the planners submitted to mitigate against submissions containing bugs.

Another important note is that we do not necessarily coincide with the next deterministic IPC track. Because the techniques for satisficing planning and unplannability detection are similar, running the two tracks at alternate times may allow for greater participation.

Scoring

Since there is no standard certificate of unplannability currently, and we do not wish to restrict the type of planners submitted to the inaugural unplannability IPC track, we will focus on the *coverage of problems correctly identified as being unsolvable*. Similar to the optimal IPC track, we will disqualify a solver for a particular domain if it identifies a solvable instance incorrectly as having no solution or vice versa. This increased penalty aims to deter approximate and unsound solutions.

Orthogonal to the coverage score, we will evaluate the speed of each technique following the standard IPC runtime score: sum over all problems T^*/T, where T^* is runtime of the fastest planner. Further, the time score will serve as a means for breaking ties in terms of planner coverage. With regards to computational resources, we will follow the same requirements as the last IPC-8 competition, 6GB RAM and 30 min computational limits.

Benchmark Domains

Bäckström et al. 2013 introduce a set of unsolvable planning instances that was extended subsequently by Hoffmann et al. 2014. We will use these domains with newly generated problem instances as a seed for the domains in the IPC track. Although we cannot guarantee that all (or even most) domains will be ideal unplannability benchmarks, we will strive to find domains with the following properties: (1) the problems are a combination of both satisfiable and unsatisfiable instances; (2) there is no obvious syntactic difference between the satisfiable and unsatisfiable instances; and (3) (at least some of) the satisfiable instances will be difficult for existing satisficing planners to solve. Point (1) is important to avoid submissions of the form "`return True`" and points (2)+(3) are meant to deter submissions of the form "`if FF fails to solve in under k seconds, return True`".

Ultimately, we will aim to use domains that exhibit interesting and useful properties with respect to unplannability and deadend detection; relying in part on the existing problems where deadends and unsolvable instances play a role. Potential sources for existing benchmarks include oversubscribed problems that encode resources, optimal planning benchmarks with the added constraint that a plan should have a cost cheaper than the optimal solution, unsolvable translations of different planning formalisms (e.g., encodings of conformant or contingent problems), etc.

Discussion

We hope that this track will attract attention and foster research on unsolvable tasks, which to date remains an insular topic within the ICAPS community. Our motivation in submitting this proposal to the Workshop on the International Planning Competition is to promote a dialogue for improving the endeavour of an unplannability IPC track. To that end, we have identified the following open questions that we believe are worth discussing with members of the ICAPS community interested in shaping the future IPC tracks:

1. What aspects of unsolvable problems should the benchmarks focus on? E.g., problems where most abstractions fail to recognize unplannibility, or problems with a hidden unplannable core.
2. How do we avoid submissions that "game the system"? The three properties of an ideal benchmark that we listed earlier offer one step towards achieving this. Another is the disqualification of a solver for an incorrect answer. Should the solver source be released for verification? Should a solver be disqualified entirely for an incorrect answer (i.e., false positive or false negative Etc.)?
3. What is an appropriate level of feedback to ensure that all submissions are reasonably free of errors? An initial suite of problems will be constructed to verify that the systems are producing reasonable output for simple problems. Should more than one such iteration exist?
4. Is it reasonable to schedule an unplannability track at a time other than the deterministic track? We argued earlier that this would promote a higher participation in the

unplannability track, but should this track eventually be co-run or even merged with the classical planning IPC? E.g., using a mix of solvable and unsolvable instances in the benchmark domains similar to the satisfiability and constraint programming contests.

Acknowledgements This research is partially funded by the Australian Research Council Discovery and Linkage Grants DP130102825 and LP11010015.

References

Bäckström, C.; Jonsson, P.; and Ståhlberg, S. 2013. Fast detection of unsolvable planning instances using local consistency. In *Proceedings of the Sixth Annual Symposium on Combinatorial Search, SOCS 2013, Leavenworth, Washington, USA, July 11-13, 2013.*

Boddy, M. S.; Gohde, J.; Haigh, T.; and Harp, S. A. 2005. Course of action generation for cyber security using classical planning. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005)*, 12–21.

Edelkamp, S.; Leue, S.; and Visser, W. 2007. Summary of dagstuhl seminar 06172 on directed model checking. In *Directed Model Checking*. Dagstuhl Seminar Proceedings. Dagstuhl, Germany.

Goldman, R. P.; Kuter, U.; and Schneider, T. 2012. Using classical planners for plan verification and counterexample generation. In *AAAI Workshop on Problem Solving Using Classical Planners*.

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

Hoffmann, J.; Fatès, N.; and Palacios, H. 2010. Brothers in arms? on AI planning and cellular automata. In *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, 223–228.

Hoffmann, J.; Kissmann, P.; and Torralba, Á. 2014. ”Distance”? Who Cares? Tailoring Merge-and-Shrink Heuristics to Detect Unsolvability. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, 441–446.

Keyder, E. R.; Hoffmann, J.; and Haslum, P. 2014. Improving delete relaxation heuristics through explicitly represented conjunctions. *Journal of Artificial Intelligence Research (JAIR)* 50:487–533.

Suda, M. 2014. Property directed reachability for automated planning. *Journal of Artificial Intelligence Research (JAIR)* 50:265–319.

Bagged Representations in PDDL

Pat Riddle, Mike Barley, Santiago Franco and Jordan Douglas
Department of Computer Science, University of Auckland
Private Bag 92019, Auckland, 1142 NZ

Abstract

In this paper, we demonstrate that many problems used in the IPC are more naturally represented as bags instead of sets. These tend to be problems with lots of objects. In the standard PDDL representations the objects are all denoted individually by unique names and this causes the problem solvers to encounter combinatorial explosions on trying to solve them. Frequently we don't care about individual objects, we want something to happen to a whole set of objects, but currently we are forced to identify them individually anyway. These bags of objects are similar to resources in scheduling. We propose a new formulation for these types of problems in the current PDDL STRIPS representation. We analyze the new and original representations in an attempt to determine when each representation performs better. In this paper's experiments, the best results were always obtained on the new representations. All new representations in this paper were generated manually, but we are currently developing a system that automatically generates these types of representations from the original PDDL representation, so throughout this paper the new representations are called the transformed representations.

Introduction

The PDDL based language is based on sets, so each individual object is uniquely identified which leads to combinatorical explosions. A more natural representation for problems with a large set of objects is a bag based representation (like resources in scheduling). In this paper we explore problems from 6 domains. We analyze results on two representations for each problem. We refer to these representations as the original and transformed representation, although all the transformed representations discussed in this paper were created manually. We are currently developing a transformation system which creates the new representations automatically and translates the resulting solution back into the original representation. (Riddle et al. 2015b) The goal of this paper is to determine whether bagged-based representations have an advantage over the normal set-based PDDL representations for domains with a large number of objects of the same type.

PDDL (McDermott et al. 1998) is fundamentally based on sets. In a domain like sokoban-strips-opt-08, each of the stones are individually identified. For instance, in problem p20 there are 5 individually identified stones. The goal state specifies that each stone is at some goal location ($at\text{-}goal\ stone$-05), but it doesn't matter which goal location a stone is at. This allows us to call each stone "stoneX". In the PDDL problem file we can rename all the stones in all the predicates to stoneX. We must also alter the goal state, otherwise PDDL's set semantics mean once one stone is put in any goal position the problem is solved. In this instance the goal state becomes $(and\ (at\ stoneX\ pos$-03-03) ($at\ stoneX\ pos$-03-04) ($at\ stoneX\ pos$-03-05) ($at\ stoneX\ pos$-03-06) ($at\ stoneX\ pos$-03-07)), which is equivalent to the original goal description. For sokoban-strips-opt-08, the domain file does not have to be altered to use this new representation.

We solved p20 in both representations with Fast Downward (Helmert 2006). Using the blind heuristic in the new representation it found a 31 cost solution (with 87 steps) in a search time of 3.07 seconds which expanded 752,651 states until last jump. Fast Downward's "until last jump" values return the number of states for the last fully expanded f-level. We use these counts throughout this paper to avoid any stochastic effects caused by search tree ordering. The problem solver on the original representation ran out of memory during f-level 31. These representations will have the same cost optimal solution, so we can look at the last f-level they both completed. In the original representation, at the end of f-level 30, Fast Downward expanded 30,006,650 states in 105.7 seconds. At the same f-level in the transformed representation Fast Downward expanded 752,651 states in 3.07 seconds. We use blind search to compare the actual size of the state space. This simple change, ignoring the names of identical objects, reduces the search tree size by a factor of 40, as can be seen in Table 3. The reduction factor is calculated by dividing the nodes expanded in the original representation by the nodes expanded in the transformed representation. If both problems are not solved we use the expanded nodes at the last common f-level in this calculation. This works for the sokoban-strips-opt-08 domain because no two stones can occupy the same location, so it never tries to enter ($at\ stoneX\ pos$-4-3) twice within the same state.

We would like to use the same type of representations in other domains. For example, in the gripper domain we would like to call all the balls "ballX". The problem is two different balls can be in the same location within a state. If

Table 1: Transformed gripper problem representation

```
(define (problem strips-gripper-x-1)
(:domain gripper-strips)
(:objects n4 n3 n2 n1 n0 rooma roomb ballX left right)
(:init(room rooma)(room roomb)(ball ballX)(free left)
 (free right)(at-robby rooma)(more n0 n1)(more n1 n2)
 (more n2 n3)(more n3 n4)(gripper left)(gripper right)
 (count ballX rooma n4)
 (count ballX roomb n0))
(:goal(and(count ballX roomb n4))))
```

we enter $(at\ ballX\ rooma)$ more than once in a state, then when one of them is removed all of them will be removed because of PDDL's basic set representation. To solve this problem we create a bag representation that can be used in PDDL instead.

A Bagged Representation

In order to avoid having multiple copies of the same predicate in a state (e.g., $(at\ ballX\ rooma)(at\ ballX\ rooma)$) every predicate that refers to the "objects to be merged" will need to be replaced by a count predicate. So for instance in the gripper domain $(at\ ball1\ rooma)$ $(at\ ball2\ rooma)$ $(at\ ball3\ rooma)$ $(at\ ball4\ rooma)$ will be replaced by $(count\ ballX\ rooma\ 4)$. This is the equivalent of multiple identical predicates within the same state. To implement the counts, we need to do simple arithmetic. Unfortunately only a handful of optimal planners currently handle PDDL numerical fluents. We want to create PDDL that can be used by any PDDL planner, so we add simple counting predicates $(more\ n1\ n2)$ which allow us to alter the domain actions to increment or decrement the count predicates. The transformed representation for gripper is shown in Tables 1 and 2.

We created a gripper problem instance p250 with 250 balls. We solve this problem with both representations with Fast Downward using the blind heuristic. Note that the representation used in these experiments merged the grippers together as well as the balls, as opposed to the PDDL shown in the tables above. This more complex transformed representation is given at (Riddle 2015). With the transformed representation it found a 749 step solution in a search time of 1.22 seconds which expanded 1,495 states until last jump. Whereas with the original representation, Fast Downward runs out of memory during f-level 6. These representations will have the same length optimal solution, so we can look at the last f-level they both completed. In the original representation, at the end of f-level 5 Fast Downward expanded 250,502 states in 10.98 seconds. At the end of f-level 5 in the transformed representation Fast Downward expanded 9 nodes in 1.02 seconds. This change reduces the search tree size by a factor of 27,834, as can be seen in Table 3.

More Complex Domains

So far the transformations have been fairly easy because the objects that we wanted to rename only occurred in a single predicate. Things become more complex when we have multiple predicates. For instance in the Barman-opt11-strips domain, let us assume we do not care which shots contain which drinks, but only care that there "exists" a shot with

Table 2: Transformed gripper domain representation

```
( define ( domain gripper-strips )
(:predicates (room ?r)(ball ?b)(gripper ?g)
  (at-robby ?r)(count ?b ?r ?n)(free ?g)(carry ?o ?g)
  (more ?n1 ?n2))

(:action move :parameters ( ?from ?to )
 :precondition (and (room ?from)(room ?to)
  (at-robby ?from))
 :effect (and (at-robby ?to)(not (at-robby ?from))))

(:action pick :parameters (?n1 ?n0 ?obj ?room ?gripper)
 :precondition (and (ball ?obj)(room ?room)
   (gripper ?gripper)(at-robby ?room)(free ?gripper)
   (more ?n1 ?n0)(count ?obj ?room ?n0))
 :effect (and (carry ?obj ?gripper)
   (not (count ?obj ?room ?n0))
   (count ?obj ?room ?n1)(not (free ?gripper))))

(:action drop :parameters (?n1 ?n0 ?obj ?room ?gripper)
 :precondition (and (ball ?obj)(room ?room)
   (gripper ?gripper)(carry ?obj ?gripper)(more ?n0 ?n1)
   (at-robby ?room)(count ?obj ?room ?n0))
 :effect (and (not (count ?obj ?room ?n0))
   (count ?obj ?room ?n1)(free ?gripper)
   (not (carry ?obj ?gripper)))))
```

each specified drink. Similarly, we have two hands "right" and "left" and do not care which hand picks up and holds something, only that we don't pick up 3 things! We want to create the same type of abstracted representation for the barman domain. Unfortunately now we have multiple predicates that refer to shots. To overcome this problem, we create macro-predicates.

The transformed barman-opt11-strips problem for pfile01-001.pddl is given at (Riddle 2015). In the original representation, there are a number of predicates that take a shot as an argument. These are: $(ontable\ ?c$ - $container)$, $(holding\ ?h$ - $hand\ ?c$ - $container)$, $(clean\ ?c$ - $container)$, $(empty\ ?c$ - $container)$, $(contains\ ?c$ - $container\ ?b$ - $beverage)$, $(used\ ?c$ - $container\ ?b$ - $beverage)$. This kind of distributed representation is very common in PDDL. It can be used because the unique identifiers ("shot1" "shot2" etc.) allow it to associate the distributed predicates with each other. Unfortunately, the unique identifiers are the cause of the combinatorial explosion! We combine the predicates into a single macro-predicate. The initial state in the original representation had $\{(ontable\ shot1)$, $(ontable\ shot2)$, $(ontable\ shot3)$, $(ontable\ shot4)$, $(clean\ shot1)$, $(clean\ shot2)$, $(clean\ shot3)$, $(clean\ shot4)$, $(empty\ shot1)$, $(empty\ shot2)$, $(empty\ shot3)$, $(empty\ shot4)\}$. This is represented in the new representation as $(count1\ shotX\ empty\ clean\ ontable\ 4)$, which states that in the initial state there is: (1) a $(clean\ shotX)$, (2) an $(empty\ shotX)$, and (3) an $(ontable\ shotX)$ for the same 4 unique shots. This solves the problem of distributed representations, but it causes problems with representing the goal state.

The original representation's partial goal description was $(and\ (contains\ shot1\ cocktail3)$, $(contains\ shot2$

$cocktail1$), ($contains\ shot3\ cocktail2$)). It contains some information from our macro-predicate but not all the information. We could use (1) variables or (2) existential quantifiers, or (3) an OR construct in our final goal description. Unfortunately many of the current IPC planners do not allow any of these options in a goal description. Alternatively we create a goal-predicate that only contains the information required in the goal description. For this problem that is: (and($count$-$goal\ shotX\ cocktail3$ 1) ($count$-$goal\ shotX\ cocktail1$ 1) ($count$-$goal\ shotX\ cocktail2$ 1)). Of course the domain actions must be designed to deal with the count macro-predicates and the newly created goal-predicates, these can be seen in the Barman domain file at (Riddle 2015).

We solve problem pfile01-001.pddl in these two representations with Fast Downward using the blind heuristic. In the new representation it returned a 90 cost solution with 36 steps in a search time of 4.41 seconds which expanded 289,946 states until last jump. In the original representation, it returned a 90 cost solution with 36 steps in a search time of 38.91 seconds which expanded 5,967,050 states until last jump. This reduces the search tree size by a factor of 21, as is shown in Table 3.

Domains with Lots of Objects

The advantage of the "bagged representation" is that it scales much better than the standard representation, for instance you can solve the 250 ball gripper problem shown above. We present three additional domains to emphasize this point. The first is the Spanner domain from the IPC 2011 Learning track, created by Amanda Coles, Andrew Coles, Maria Fox and Derek Long. The second is the ChildSnack domain from the sequential track in IPC-2014, created by Raquel Fuentetaja and Tomás de la Rosa Turbides. The third is the Pizza domain also created by Raquel Fuentetaja and Tomás de la Rosa Turbides.

In the Spanner domain, we solve a smaller version of problem pfile01-001.pddl (which has 30 nuts to tighten and 30 spanners). In the transformed representation we merge all the nuts and spanners. We keep track of the counts of spanners at each location and the number of loose and tightened nuts.

In the ChildSnack domain, we solve problem child-snack_pfile01.pddl (which has 6 children, 6 breads, 6 contents, and 8 sandwiches). In the transformed representation we merge the bread, contents, and sandwiches. We keep track of the counts of each item.

In the Pizza domain, we solve problem rnd-goal_pizza_base_p02.pddl (which has 5 guests and 16 slices of pizza). In the transformed representation we merge the slices. We keep track of the counts for each item.

We present a number of experiments on these domains (as well as the 3 domains presented earlier), but first we discuss the related research.

Related Research

There has been considerable work on problem reformulation, starting with George Polya's *How to Solve It* (1957). Due to lack of space we will focus on planning-specific reformulation research. The Fast Downward system (Helmert 2006) transforms the PDDL representation into a multi-valued planning task, similar in spirit to SAS^+. Using this representation, the system generates four internal data structures, which it uses to search for a plan. Helmert (Helmert 2009), extending this work, focused on turning PDDL into a concise grounded representation of SAS^+. Additional work in this area transforms between PDDL and binary decision diagrams (Haslum 2007), transforms between PDDL and causal graphs (Helmert 2006), and identifies and removes irrelevant parts from a problem representation (Haslum, Helmert, and Jonsson 2013).

To the best of our knowledge, little research has focused on transforming a PDDL representation into another PDDL representation. Two notable exceptions are (Areces et al. 2014; de la Rosa and Fuentetaja 2015), the latter of which we describe at the end of this section. Instead, it has almost exclusively focused on either reformulating PDDL to a planner's internal representation or transforming one internal representation to another. Although these approaches lead to more knowledge about the connectivity of the search space, and the power to alter the representation in these internal forms, there are advantages to altering PDDL. First, the new representations can be used by any PDDL planner. Second, in some domains, creating SAS^+ or the domain transition graphs take a lot of time; this might be avoided by first transforming to a different PDDL representation and transforming that into SAS+ or an internal representation.

Our system has much in common with symmetry reduction systems. Fox and Long (1999) group symmetric objects together in TIM. They require objects to be indistinguishable in both the initial state and the goal description. They keep track of the symmetry groups during planning but only with respect to the goal description, so they cannot remove all the symmetries in gripper.

Pochter et. al. (2011) generalize the work by Fox and Long, by using generators to create automorphic groups. These groups are based on SAS^+ and so are more general than objects. They still require the symmetric groups to be indistinguishable in both the initial and goal description.

Domshlak et. al. (2012) extended this work to only require symmetric groups to be indistinguishable in the goal description in the DKS system. They compared their work to Pochter's system, where they solved 8 more problems over 30 domains.

Metis (Alkhazraji et al. 2014), uses orbit search to do symmetry breaking. It is an improvement on DKS, since it does not store extra information with each state. Metis also includes an incremental LM-cut heuristic and partial order reduction with strong stubborn sets.

The closest work to our automated system for creating these transformations is the system by de la Rosa et. al. (2015). They reformulate PDDL into PDDL and they merge objects in a similar way. The main differences are 1) we merge objects if they are the same in the initial state or the same in the goal description whereas their system merges objects if they do not appear in the goal description 2) they explicitly use numeric fluents in their modeling, restricting

Table 3: Analysis of both representations across 6 domains using blind search. X signifies the planner was killed because it ran out of memory under the 2014 IPC constraints. Solution state expansions are "until last jump" to normalize for different tree orderings on the last level. Reduction Faction of expanded states shows by what factor it has been reduced; we use the last common f-bound in this calculation if either representation is not solved.

domain	problem	heuristic	representation	# SAS+ vars	# SAS+ actions	search time	states expanded	states exp. last common f-bound	common f-bound	solution cost	solution length	Reduction Factor of Expanded States
sokoban	p20	blind	transformed	25	120	3.07s	752,651	752,651	30	31	87	40
			original	35	312	X	X	30,006,650	30	X	X	
gripper	p250	blind	transformed	5	500,002	1.22s	1,495	9	5	749	749	27,834
			original	253	2,002	X	X	250,502	5	X	X	
barman-opt11	pfile01-001	blind	transformed	147	4,501	4.41s	289,946			90	36	21
			original	62	358	38.91s	5,967,050			90	36	
spanner	pfile01-001-small	blind	transformed	16	1,851	18.38s	8,699,505	126,611	53	111	111	71
			original	91	981	X	X	9,038,188	53	X	X	
ChildSnack	child-snack_pfile01	blind	transformed	74	1,656	1.8s	90,162	272	5	20	20	5,553
			original	36	456	X	X	1,510,534	5	X	X	
pizza	base_p02	blind	transformed	28	2,304	1.05s	10,837	205	6	15	15	3,400
			original	37	14,668	X	X	705,106	6	X	X	

them to planners that support them such as metric-FF (Hoffmann 2003) 3) both our systems translate the solution back to the original representation but our system can generate plans which have different explicit values specified in the goal state.

Experimental Results

In this section we explore 6 domains using a number of different problem solving configurations. First we analyze the representations using blind search. Next we focus on the gripper domain and look at the effect that small changes to the PDDL representation have on the SAS+ variables and actions and therefore on the problem solving ability. Following this we summarize some of the lessons learned from these experiments.

Blind Search

We discussed the sokoban, gripper and barman domains earlier. Table 3 shows the reduction factor of expanded states for each problem in these 3 domains is 40, 27,834 and 21 respectively.

In the Spanner domain, we solve a smaller version of problem pfile01-001.pddl (which has 30 nuts to tighten and 30 spanners) in both representations with Fast Downward using the blind heuristic. In the transformed representation it returns a 111 step solution in a search time of 18.38 seconds which expanded 8,699,505 states until last jump. Whereas in the original representation it runs out of memory during f-level 54. At the end of f-level 53 it expanded 9,038,188 states in 35.08 seconds. At the end of f-level 53 in the transformed representation it expanded 126,611 nodes in 1.24 seconds. The new representation for spanner can be seen at (Riddle 2015). The reduction factor for this problem is 71.

In the ChildSnack domain, we solve problem child-snack_pfile01.pddl in both representations with Fast Downward using the blind heuristic. In the transformed representation it found a 20 step solution in a search time of 1.8 seconds which expanded 90,162 states until last jump. Whereas in the original representation it runs out of memory during f-level 6. At the end of f-level 5 it expanded 1,510,534 states in 82.06 seconds. At the end of f-level 5 in the transformed representation it expanded 272 nodes in 1.01 seconds. The new representation for ChildSnack can be seen at (Riddle 2015). The reduction factor for this problem is 5,553.

In the Pizza domain, we solve problem rnd-goal_pizza_base_p02.pddl in both representations with Fast Downward using the blind heuristic. In the transformed representation it found a 15 step solution in a search time of 1.05 seconds which expanded 10,837 states until last jump. Whereas in the original representation it runs out of memory during f-level 7. At the end of f-level 6 it expanded 705,106 states in 34.2 seconds. At the end of f-level 6 in the transformed representation it evaluated 205 nodes in 1.02 second. The new representation for pizza can be seen at (Riddle 2015). The reduction factor for this problem is 3,400.

Alternative PDDL Representations Effects

In the gripper example above we used the predicate (count ballX rooma n4) to represent that there are 4 ballXs in rooma in the initial state. When you merge all objects of a single type into an equivalence class, you could just remove that type from the predicate altogether and make it (count rooma n4) since the ballX can be assumed implicitly. We tried these two alternative representations (using both the ball and gripper equivalence classes). We wanted to see what effect they would have on the SAS+ variables and actions created and also what effect that had on Fast Downward's problem solving.

The first two rows of Table 4 show the results on the

Table 4: Analysis of both representations across both representations using iPDB. X signifies the planner ran out of memory under the 2014 IPC constraints. Solution state expansions are "until last jump" to normalize for different tree orderings on the last level.

domain	prob	heuristic	SAS+ representation	SAS+ vars	actions	translate time	preprocess time	search time	total time	last-jump states expanded	total states expanded	initial heuristic estimate	solution cost	solution length
gripper	p250	iPDB	with-objects	5	500,002	107.17s	640.50s	1.05s	23.26s	0	750	749	749	749
			without-objects	755	500,002	111.81s	738.41s	X	X	X	X	X	X	X
gripper	p250	iPDB/symba	with-objects	5	2,002	108.22s	74.88s	1.03s	2.62s	0	750	749	749	749
			without-objects	507	2,002	119.99s	127.04s	1.21s	3077.64s	1491	1494	36	749	749

Table 5: Analysis of both representations across both representations using LM_cut. Solution state expansions are ”until last jump” to normalize for different tree orderings on the last level.

domain	prob	heuristic	SAS+ representation	SAS+ vars	actions	translate time	preprocess time	search time	total time	last-jump states expand	total states expand	initial heuristic estimate	solution cost	solution length
gripper	p250	LM_cut	with-objects	5	500,002	107.79s	665.85s	9777.18s	9790.72s	1492	1497	251	749	749
			without-objects	755	500,002	113.25s	729.72s	11820.7s	11843.3s	1492	1497	251	749	749
gripper	p250	LM_cut/symba	with-objects	5	2,002	108.47s	75.15s	11.01s	11.07s	1492	1497	251	749	749
			without-objects	507	2,002	120.78s	131.33s	14.45s	14.55s	1492	1497	251	749	749

gripper problem with-objects and without-objects (when we have removed ballX and gripperX from the predicates) using the iPDB heuristic (Haslum et al. 2007). The number of SAS+ variables explodes in the without-objects representation, going from 5 to 755. Fast Downward's ability to find mutexes is severally hampered by this representation. As a consequence of this, Fast Downward runs out of memory while it is still looking for PDBs. While in the with-objects representation the problem is solved in 23 seconds total time and goes straight to the solution expanding 750 nodes with a solution length of 749.

The second two rows of Table 4 show the same two experiments with the only difference being that the translator and preprocessor software is replaced by that used in SymBA* (Torralba et al. 2014) which has been altered to work with Fast Downward. The translator and preprocessor software are described in (Alcázar and Torralba 2015). These two rows show that many of the actions are found to be spurious and removed. In addition, the number of SAS+ variables is also reduced when using the SymBA* preprocessor and the without-objects representation. Also the preprocessing time is significantly reduced in both representations. The with-objects representation performs about the same, except the total time is smaller because the spurious actions are removed. The without-objects representation is now solved without running out of memory because of the SAS+ reduction in size.

The question is whether this behavior is specific to iPDB or whether the same trends will occur with other heuristics. To test this we ran the same 4 experiments using LM-cut (Helmert and Domshlak 2009), the results are shown in Table 5. All 4 configurations can solve the problem. In addition, they all expand exactly the same number of nodes. But the failure of the original Fast Downward preprocessor to remove spurious states, means that the first two experiments take about 10,000 seconds total time. With the SymBA* preprocessors the last two experiments are solved in 15 seconds total time. Using LM-cut there is a slight benefit to using the with-objects representation no matter which preprocessor you use.

There is a clear trend: 1) improvement when using the with-object representation and 2) improvement when using the SymBA* preprocessor. We will look at one last set of experiments where we use the Blind heuristic. We ran the same 4 experiments using the blind heuristic, the results are shown in Table 6. The preprocessor time using SymBA* is still lower, but all four experiments expand the same number of states. Because there are fewer SAS+ variables and actions, the total time for Fast Downward is an order of magnitude less when using the SymBA* preprocessor. Another very interesting fact is that the number of nodes expanded using the blind heuristic is the same as those expanded using LM_cut. LM_cut does not seem to exclude any of the nodes expanded with the blind heuristic, but it does take 3 orders of magnitude more time. iPDB on the other hand, takes more time than blind search, but at least does expand fewer nodes which might be useful in larger problems.

Lessons Learned

Unfortunately the transformed representation is not always better than the old representation. For instance even the transformed Sokoban representation has some drawbacks with the LM-cut and iPDB heuristics, see (Riddle et al. 2015a). This further supports earlier results by (Riddle, Holte, and Barley 2011) that there is no “best representation” for all problems within a domain. The main drawbacks concern: how the SAS+ variables and actions are made, and how the heuristics are affected. We will discuss each of these drawbacks in turn.

SAS+ Variables This paper showed that the transformed representation sometimes gave a larger number of SAS+ variables and actions and sometimes made them smaller. In some domains such as sokoban, gripper, spanner, and pizza, the new representation actually has fewer variables and actions generated. This is a boon to the planners because they use less memory to represent each state. In other domains such as barman-opt11 and ChildSnack, more variables and actions are created. This means the planner will be using more memory, not less! One solution is to use a better SAS+ preprocessor, such as (Alcázar and Torralba 2015), this is an area we are currently exploring. As shown in Tables 4, 5, and 6, the SymBA* preprocessor frequently returned fewer variables and actions. These experiments also highlight how sensitive all the SAS+ preprocessors are to small changes in the PDDL description. In with-objects we

Table 6: Analysis of both representations across both representations using the Blind heuristic. Solution state expansions are ”until last jump” to normalize for different tree orderings on the last level.

domain	prob	heuristic	SAS+ representation	SAS+ vars	actions	translate time	preprocess time	search time	total time	last-jump states expanded	total states expanded	initial heuristic estimate	solution cost	solution length
gripper	p250	Blind	with-objects	5	500,002	107.21s	635.97s	1.19s	14.27s	1495	1498	1	749	749
			without-objects	755	500,002	113.0s	710.46s	1.30s	23.89s	1495	1498	1	749	749
gripper	p250	Blind/symba	with-objects	5	2,002	108.19s	75.19s	1.18s	1.24s	1495	1498	1	749	749
			without-objects	507	2,002	120.63s	128.91s	1.22s	1.32s	1495	1498	1	749	749

had predicates which always had only one choice of value ”ballX” and ”gripperX”. We removed these two values since they were implicitly defined, but this caused both preprocessors to create more variables. In addition, this caused the original Fast Downward preprocessor to create more SAS+ actions as well.

Additionally we have found that the number of states expanded in the transformed representation is reduced to such an extent, that the extra costs of the additional SAS+ variables are not a problem. This is not always the case. In problems where you only bagged 2 or 3 objects together and got an increase in SAS+ variables, the new representation would likely take longer to solve.

We learned two main lessons concerning SAS+ variables. The first is that small changes to PDDL representations can have large impacts in SAS+ representations. This is very important if you are automatically generating PDDL representations because you cannot normally try a number of similar representations and pick the best one. But it is also important in the main creation of PDDL representations. A designer needs to carefully try a series of representations to find one that works well with a problem solver and heuristic combination.

The second lesson learned is altering the PDDL representation can greatly reduce the size of the search space and problem solving time even when heuristics are involved. Because the search space is reduced to such a large extent, it is still frequently better to solve a problem in the transformed space even if the heuristics are less accurate. This is explored more completely in (Riddle et al. 2015a). So how do we determine which heuristic to use in the transformed domains?

Heuristics Another issue is how the new representations affect the heuristics, this relates to the section above because any change to the SAS+ variables will change the heuristics' predictive power. As we can see in our experiments sometimes the new representation improves the accuracy of the initial heuristic estimate and sometimes it decreases it.[1] This can be seen in Table 5 and is explored further in (Riddle et al. 2015a). We have made the state space so much smaller, that the fact that the heuristics are not as good frequently has little effect. But finding a heuristic which performs well on the transformed domain is one of our major research directions.

Currently our results show that blind search almost always out performs heuristics in the transformed representation. This allows us to solve more problems than other IPC problem solvers, but will not scale up in the long run to really huge problems. The SymBA* problem solver is another direction of exploration. The combination of our reduced search space and their bidirectional blind search provides a very powerful approach. Using this combination we can solve the first 9 barman-opt14 problems within the IPC constraints. This is the only approach I know of which currently solves all these problems. But the transformed representation can only get rid of accidental complexity (Haslum 2007), so blind search (even bidirectional search) will only scale up so far.

[1]We are assuming that the initial heuristic estimate reflects the overall accuracy of the heuristics.

We have currently looked at iPDB and LM_cut. LM_cut performs poorly on the new representations, iPDB on the other hand performs equally well on the original and transformed representations (as long as the number of SAS+ variables does not grow too large). So a combination of iPDB with the transformed representation using the SymBA* preprocessor is our current best approach for larger problems where SymBA* cannot solve them using blind bidirectional search.

Another approach we are exploring is using RIDA* (Barley, Franco, and Riddle 2014) to chose between several different configurations. It uses sampling and its run-time formulas to decide for this problem which combination of representation, heuristics and problem solver will have the greatest chance to solve the problem within the current time constraints.

Conclusion and Further Research

To scale up PDDL representations to large numbers of objects, it is better to use a Bagged representation. This allows much larger problems to be solved with less combinatorial explosion. We are creating a system for automatically generating a bagged representation from the original PDDL representation (Riddle et al. 2015b), although the PDDL representations shown here were created manually. The automated approach allows the use of a bagged representation even when you care which object is used, because it translates the solution back into the original representation. It does require that the objects to be merged are the same either in the initial state or in the goal description.

This paper shows that the transformed representation has a smaller state space. It certainly can solve some problems more efficiently (the transformed representation combined with SymBA* is the only system we are aware of which can solve the first 9 barman-opt14 problems within the IPC constraints).

There are two major problems with Bagged representations, the SAS+ representation is very sensitive to how the PDDL is represented. This is painful when PDDL is created by hand, but it is even more troublesome when the PDDL is automatically generated. We are currently exploring subtler changes in PDDL representations to explore their effect on the resulting SAS+ representation.

The second major problem is that at least some heuristics find it difficult to provide accurate heuristic estimates in bagged representations. This is interesting in itself because the transformed PDDL representations could be (and in the case of this paper were) created manually. So our current set of heuristics are not appropriate for all PDDL representations. We suspect that people have been making PDDL domains which can be solved with the heuristics we have, rather than challenging the field with representations where our heuristics perform poorly. We plan to explore which heuristics perform better on a bagged representation. Some heuristics seem to perform equally across most representations and some do particularly badly on the bagged representation.

The transformed representation is certainly not always better, especially when there are only a few symmetrical objects. Currently we use RIDA*'s (Barley, Franco, and Riddle 2014) runtime formula to choose between representations on a problem by problem basis. We would like to extend it to choose between heuristics, representations, and problem solving approaches (e.g., Fast Downward versus SymBA*).

We should include bagged representations in future IPC competitions. They are an easy way to explore larger spaces, whether automatically generated or created by hand. It would be interesting to see which heuristics and problems solvers work well on these representations and which do not.

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In *ICAPS*.

Alkhazraji, Y.; Katz, M.; Mattmuller, R.; Pommerening, F.; Shleyfman, A.; and Wehrle, M. 2014. Metis: Arming fast downward with pruning and incremental computation. In *In the Eighth International Planning Competition Description of Participant Planners of the Deterministic Track*.

Areces, C.; Bustos, F.; Dominguez, M.; and Hoffmann, J. 2014. Optimizing planning domains by automatic action schema splitting. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*.

Barley, M.; Franco, S.; and Riddle, P. 2014. Overcoming the utility problem in heuristic generation: Why time matters. In *ICAPS*.

de la Rosa, T., and Fuentetaja, R. 2015. Automatic compilation of objects to counters in automatic planning. case of study: Creation planning. http://e-archivo.uc3m.es/bitstream/handle/10016/19707/TR-objects-to-counters-10-2014.pdf. Accessed: 2015-02-20.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2012. Enhanced symmetry breaking in cost-optimal planning as forward search. In *ICAPS*.

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning problems. In *IJCAI*, 956–961. Morgan Kaufmann.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI*, volume 22-2, 1007. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Haslum, P.; Helmert, M.; and Jonsson, A. 2013. Safe, strong and tractable relevance analysis for planning. In *ICAPS*.

Haslum, P. 2007. Reducing accidental complexity in planning problems. In *IJCAI*, 1898–1903.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *ICAPS*.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26(1):191–246.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173(5).

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

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL-the planning domain definition language.

Pochter, N.; Zohar, A.; and Rosenschein, J. S. 2011. Exploiting problem symmetries in state-based planners. In *AAAI*.

Pólya, G. 1957. *How to solve it: A new aspect of mathematical method*. Princeton University Press, second edition.

Riddle, P.; Barley, M.; Franco, S.; and Douglas, J. 2015a. Analysis of bagged representations in pddl. In *Heuristics and Search for Domain-Independent Planning*.

Riddle, P.; Barley, M.; Franco, S.; and Douglas, J. 2015b. Automated transformation of PDDL representations. In *International Symposium on Combinatorial Search*.

Riddle, P.; Holte, R.; and Barley, M. 2011. Does representation matter in the planning competition? In *SARA*.

Riddle, P. 2015. PDDL files for the representations. http://www.cs.auckland.ac.nz/∼pat/socs-2015/. Created: 2015-02-20.

Torralba, Á.; Alcázar, V.; Borrajo, D.; Kissmann, P.; and Edelkamp, S. 2014. SymBA*: A symbolic bidirectional A* planner. In *The 2014 International Planning Competition - Description of Planners*.

Competition of Distributed and Multiagent Planners (CoDMAP)

Michal Štolba and **Antonín Komenda**
{stolba,komenda}@agents.fel.cvut.cz
Department of Computer Science, Faculty of Electrical Engineering,
Czech Technical University in Prague, Czech Republic

Daniel L. Kovacs
dkovacs@mit.bme.hu
Department of Measurement and Information Systems, Faculty of Electrical Engineering and Informatics
Budapest University of Technology and Economics, Hungary

Abstract

As a part of the 2015 DMAP workshop at ICAPS, we are organizing a semi-official experimental competition in multi-agent planning. The main aims of the competition are to consolidate the planners in terms of input format and formalism, so they can be better compared and to provide a proof-of-concept of a potential future IPC track on multi-agent planning. Another aim is to bring closer the classical and multi-agent planning communities in terms of comparable benchmarks and techniques. In this paper we explain our decisions, describe the formalism and language used and propose how the IPC track might look like. The success (or failure) of the competition will show the potential suitability of such a dedicated IPC track.

Introduction

Various forms of multi-agent planning have recently found their way to the ICAPS (International Conference on Automated Planning and Scheduling) community, partially at the main conference, partially at specialized workshops such as DMAP'13-'15 (Distributed and Multi-Agent Planning). Nevertheless, there was no IPC (International Planning Competition) track for multi-agent planning yet. The reason lies most probably in the wide variety of actual problems the term multi-agent planning covers.

One of the main distinctions can be drawn between centralized planning for multiple agents, distributed planning by multiple agents (for a centralized solution) and distributed planning by multiple agents for multiple agents (themselves). Other than that, the multi-agent planning problems can range from fully observable, deterministic planning with cooperative agents to partially observable nondeterministic planning with adversarial agents.

But even if we restrict multi-agent planning to some well defined subset of the problems, existing planners can hardly be compared as they use a multitude of formalisms and input formats. As the organizers of DMAP'15 we have decided to run an experimental competition of multi-agent planners mainly to address this issue and to enable a possible future multi-agent IPC track.

In this paper we describe the decisions we have made, the rules and the language we have designed and the potential for a future multi-agent IPC track.

Aims of CoDMAP

The first decision we had to make was how to restrict the multi-agent planning problems which would be covered by the competition. We chose an approach similar to that of classical planning, that is start with the smallest possible subset of features and possibly extend them in the future. We wanted to take classical STRIPS planning and extend it with the smallest possible feature set to transform it for the multi-agent setting. Such an approach was already taken by Brafman&Domshlak in the case of MA-STRIPS formalism (Brafman and Domshlak 2008).

Before designing the competition, we were aware of about a dozen of multi-agent planners more-or-less compatible with the MA-STRIPS formalism. One of the main focuses of the competition design was to allow as many of them as possible to enter the competition without large-scale modifications. In order to foster our awareness of the existing planners and their possible extensions, we have conducted a public poll.

Out of the poll and other considerations arose three main restrictions of the multi-agent planning model:

STRIPS-like model This means deterministic, nondurative actions and full observability (with respect to privacy, which will be discussed later). This seems to be the simplest model, compatible not only with most of the current multi-agent planners, but also with classical planners and classical planning techniques, which is good for comparison, reuse of the techniques and benchmarks.

cooperative agents This is a very strong assumption, maybe one of the first candidates to be lifted. On the other hand, some competitive problems can be converted to cooperative by automatic transformation of action costs using mechanism design (Nissim and Brafman 2013).

offline planning We have decided to stick to the offline planning paradigm as used in classical planning (input → planning → plan) in contrast to online planning as used in the probabilistic uncertainty IPC track.

In order to make the transition as smooth as possible for most of the planners, we have decided to run two tracks. The Centralized Track serves as a transitional track, where the input is centralized and the planners can be centralized as well, which both contradicts common assumptions of multi-agent planning. Also most of the language and formalism requirements (described later) can be ignored by the planners (but they have to state that in the description). The other, more ideal, Distributed Track forces stricter rules and also forces the planners to consume distributed (factored) input and run on multiple physical machines in a distributed fashion (each planning agent on one machine). In both tracks, the planning systems are evaluated separately (as in classical IPC), different planners are not interacting.

We have excluded the possibility of a decentralized track, where planners of multiple competitors would plan together cooperatively (or non-cooperatively) in order to find a common plan. Such track would require us to define some common protocol and is far beyond the abilities of most current planners.

MA-STRIPS

A crucial point of the competition was to determine a formalism and an input language. For the formalism, we have chosen MA-STRIPS for its simplicity and wide acceptance among existing planners.

MA-STRIPS extends the STRIPS formalism with two concepts, a concept of *factorization* and a concept of *privacy.* A classical planning STRIPS problem is a tuple $\Pi = \langle P, A, I, G\rangle$, where P is a set of propositions, A a set of actions, I the initial state and G the (partial) goal state. In a MA-STRIPS problem $\Pi_{MA} = \langle P, \{A_i\}_{i=1}^n, I, G\rangle$, the set of actions is factored into n sets of actions, each set representing the capabilities of a single agent. That is agent k can use only actions in A_k. Other parts of the problem keep the STRIPS semantics, as well as the actions: $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a)\rangle$.

Privacy follows unambiguously from the factorization. A fact $p \in P$ is public, if it is shared among two actions of different agents, that is if $p \in (\mathsf{pre}(a_i) \cup \mathsf{add}(a_i) \cup \mathsf{del}(a_i)) \cap (\mathsf{pre}(a_j) \cup \mathsf{add}(a_j) \cup \mathsf{del}(a_j))$ and $a_i \in A_i$, $a_j \in A_j$ and $i \neq j$. All facts p' that are not public are private to some agent k s.t. $p \in \mathsf{pre}(a_k) \cup \mathsf{add}(a_k) \cup \mathsf{del}(a_k)$ for some $a_k \in A_k$. Similarly, an action is public if it contains any public fact in its precondition or effect, otherwise it is private.

Note, that the MA-STRIPS formalism does not explicitly define the pragmatics of the public/private separation, that is, what is the interpretation of privacy for the agents and the planner. One common understanding is that the agents should not "know" or "observe" or somehow "use" the other agent's private information, which implies a special case of partial observability. We adopt this interpretation of privacy in the Distributed Track and allow more freedom in the Centralized Track, as it is not the only view of privacy possible and especially if the planner is centralized, interpretations other than restricted observability are meaningful as well.

Obviously, as MA-STRIPS is a minimal multi-agent extension of STRIPS, there is a multitude of reasonable extensions to MA-STRIPS. First of all, the definition of privacy (in terms of what facts and actions are public/private) can be defined much more loosely. Also the requirement that an action belongs to a single agent can be relaxed in order to allow joint actions (that is actions which have to be performed simultaneously by multiple agents). Similarly the requirement that a fact is either public or private to a single agent is rather strict. It is imaginable to have a fact private to a subset of agents. For simplicity, we have decided to exclude all such extensions from the current competition.

MA-PDDL

Having a common formal model of multi-agent planning is a great leap forward, but is not enough for the competition. In order to be able to run the planners on a common set of benchmarks it is necessary to have a common input language as is PDDL for classical planning. As all the planners we are aware of (and which took part in the poll) use as an input some kind of modified PDDL or PDDL with some additional information (typically defining the factorization and/or the privacy), we have decided to use an extension of PDDL.

Our primary aim was to be able to express the MA-STRIPS factorization of actions and public/private separation using the language, but its expressive power did not have to be limited to it. There were two existing candidates, MAPL (Brenner 2003) and MA-PDDL (Kovacs 2012). MAPL was published in 2003 and was rather a drastic modification of PDDL2.1, introducing many features not required by us and missing the factorization and privacy definitions. MA-PDDL was a more consistent extension of PDDL3.1, but still introducing many features not needed by us and not describing privacy. Since MA-PDDL was designed as modular (allowing to use just some of the features) we have decided to extend MA-PDDL with two new modular features, factorization of domains and problems and a definition of privacy of objects and predicates (and thus implicitly of actions).

In MA-PDDL, the agents are objects which can be associated with an action based on its :agent field.

The extension of MA-PDDL[1] comes in two flavors, a *factored* description, which allows the definition of separate domain and problem description for each agent,

[1]The extended BNF can be found at http://agents.fel.cvut.cz/codmap/MA-PDDL-BNF.pdf

and an *unfactored* description, which allows the definition of factorized privacy in a single domain and problem description.

The rules for defining privacy are common to both variants. Syntactically, privacy is defined over predicate (and function) definitions and constants in the domain description and over objects in the problem description. In order to be able to represent MA-STRIPS problems, the privacy is semantically defined over grounded facts, based on the following set of rules:

1. A public predicate definition grounded with public objects/constants is a public fact.
2. A public predicate definition grounded with at least one object/constant private to agent α is a private fact of agent α (grounding a single predicate definition with objects private to different agents is not allowed).
3. A private predicate grounds to a private fact regardless of privacy of the objects used for grounding.

This definition of privacy is enough to be able to define the MA-STRIPS privacy separation, but allows much more, from defining everything private to defining everything public, regardless of the use of predicates in actions (that is a fact used only by one agent can be declared public or fact used by multiple agents private).

A further restriction of the competition is that all goals are public, resulting in that all agents have the same (common) goal. This is mainly for the compatibility reasons as many planners do not support private goals.

Factored MA-PDDL

Factored MA-PDDL descriptions are annotated with the :factored-privacy requirement. Each agent has its separate (single-agent) domain and problem description, each containing only the particular agent's factor of the global problem. This means, that the domain contains only public predicates (functions, constants), private predicates (functions, constants) and actions of the particular agent. There is no need for explicit definition of the agent (it does not have to be any PDDL object, etc.). Privacy is defined by the following construct:

```
(:private ...)
```

Which denotes that the enclosed predicates, functions, constants or objects are private to the particular agent to which the factor belongs to.

Unlike MA-STRIPS, MA-PDDL allows typing. In the competition, we will keep the PDDL type taxonomy the same for all agents, but in general it does not have to be the same, the type taxonomies in factored MA-PDDL descriptions only have to be compatible in the sense that if type_1 is defined as subtype of type_2 in one agent's domain, it has to be so in all agents' factored domains where both types are present (and also including transitive closure).

The PDDL object representing an agent should be private, if it is not, other agents of the same type are able to ground and use the other agent's actions. An alternative solution may be to use constants to represent the agents and include only the proper partially grounded actions in the agent's domain, or not to use a PDDL object representing the agent at all (as it is not required) and having all actions from the perspective of the particular agent. In general, it is necessary to model the domains carefully in order to avoid such anomalies.

Otherwise the factored domain and problem descriptions are all PDDL3.1 (in the competition we use only the STRIPS + action-costs subset of PDDL).

Unfactored MA-PDDL

An unfactored MA-PDDL description, annotated by the :unfactored-privacy and :multi-agent requirements, is little bit more complicated. First of all, the agents (and thus the following factorization) have to be defined.

The MA-STRIPS formalism does not define an agent as a part of the multi-agent planning problem, it defines only its capabilities as a set of actions, but most of the MA-STRIPS planners define agents as some of the PDDL objects in the problem description, which are then used for factorization. It is somehow natural to see the agents to be present in the problem as PDDL objects. Similarly, the MA-PDDL language associates agents to actions with an explicit agent parameter which is a part of the action definition. Any PDDL object/constant appearing as an agent parameter in some action is considered an agent. The agent parameter can be typed in order to restrict the agents only to some types of PDDL objects/constants.

In unfactored MA-PDDL descriptions, privacy is defined using the same construct as in the factored version, only the agent parameter must also be specified:

```
(:private <agent> ...)
```

It may be the case that the same predicate is private to multiple agents (for example if there are multiple agents of the same PDDL type). If such a predicate can be grounded using only public objects/constants, then the resulting fact becomes private for multiple agents. This is one of the extensions mentioned above, which is correct, but goes beyond MA-STRIPS and thus will not be used in the competition. Such situations can always be prevented by including a variable representing the agent into the argument of the private predicate definition, in that case the grounded fact is private to that particular agent.

The CoDMAP Competition

The full rules, used domains and results are published at the competition website[2]. Relation of the centralized and distributed tracks of the CoDMAP competition to the existing IPC tracks is shown in Figure 1. Planners

[2]http://agents.fel.cvut.cz/codmap

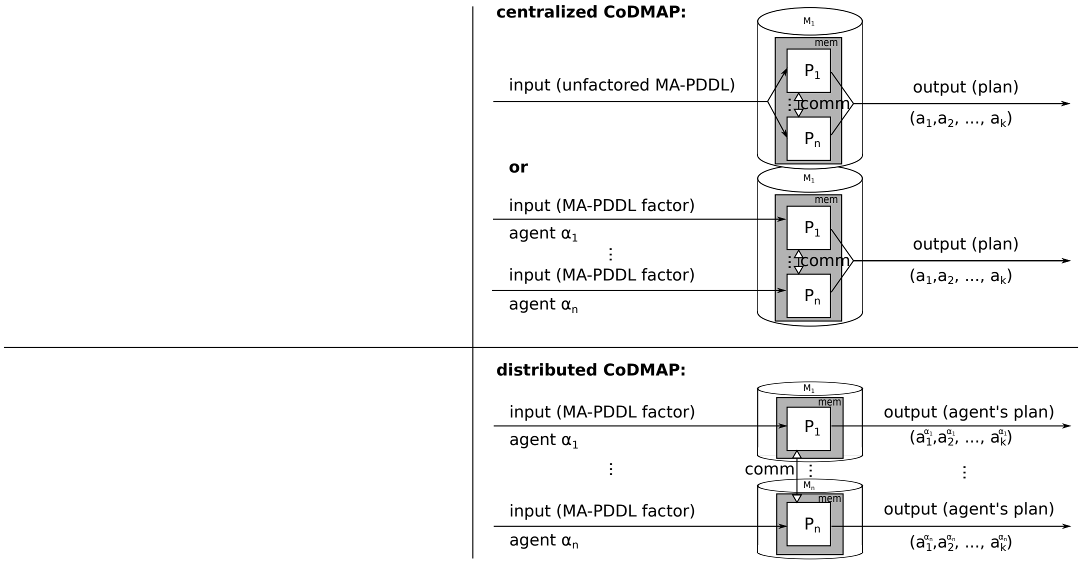


Figure 1: Comparison of IPC and CoDMAP tracks.

in the classical IPC tracks (both opt. and sat.) take a pair of PDDL files (domain, problem) as an input, run on a single machine/single-core and output a sequence of actions as the plan. The multi-core track differs in that the planners may run on multiple cores/threads.

CoDMAP consists of two tracks:

- Centralized Track, aiming for maximal compatibility with classical IPC and existing multi-agent planners
- Distributed Track, aiming for a proper multi-agent setting.

Centralized Track

In the centralized track, the input of a planner is either a single unfactored MA-PDDL domain and problem description, or a separate factored MA-PDDL domain and problem description for each agent in the planning problem. The planner runs on a single machine, with no other restrictions or requirements (the planner may be as well single-core or multi-core, distributed or not, one thread per agent or multiple threads per agents, etc.). The provided input will have factoring and private separation according to MA-STRIPS, but the planners are not required to adhere to it. This is in order to enable planners built on different multi-agent planning models to enter the competition as well. The output of the planner must be a sound linear plan.

We do not restrict any communication between planning agents (if any), nor do we restrict the exchange of private information.

The rules are intentionally weak not to force the MA-STRIPS formalism and our view of communication and privacy on the competing planners. What we request is to accompany the planner with a short paper explaining all the parameters of the factorization, privacy, inter-agent communication (if any), architecture, etc., so they can be included in the final results and each participant can then derive their own conclusions based on the similarity of their planner with other competitors.

The difference between the centralized track and classical multi-core track is mainly in the input format (MA-PDDL) and also the planners are expected to somehow utilize the multi-agent nature of the given problems.

Distributed Track

The distributed track is much more strict in terms of the rules, but it is rather experimental in the sense that there are currently no planners capable of entering it without significant modification. The aim was to provide a track the way we think a multi-agent planning competition should look like.

The planners have to be truly distributed as shown in Figure 1. Each planning agent of such a distributed planner receives its own factor of the factored MA-PDDL domain and problem, runs on its own dedicated machine and outputs its own plan. The MA-PDDL factorization and privacy definition must be adhered to. In most benchmarks the factorization and privacy definition will follow the MA-STRIPS model but does not necessarily have to.

The planning agents of a distributed planner can communicate over TCP-IP (IP addresses of other planning agents will be known up-front), but they should avoid exchanging any private information (all such cases should be clearly explained in the accompanying paper).

The output is a linear plan for each agent, which can all be executed in parallel. The actions of all plans in each time step must not be in mutex (mutual exclusion).

Evaluation

Because of the very weak rules in the centralized track, the evaluation will be a comparison rather than true competition results. Some of the planners will be hardly comparable and the outcome of the comparison will depend on the interpretation of the results (which will be published) and the selection of comparable planners, based on common features.

The planners will be evaluated over a set of 10 benchmark domains converted from the classical IPC and 2 new multi-agent domains. The domains will use only the STRIPS subset of PDDL and some of them action costs. The number of agents in each problem is restricted to at most 10 because of hardware and time limitations. Each run of the planner will be restricted to 30 minutes and 8GB of memory for all agents together in the centralized track and for each agent separately in the distributed track.

The metric used to compare the planners will be coverage over all domains, IPC Score over the plan quality Q (sum over all problems Q*/Q, where Q* is the cost of optimal plan or of the best plan found by any of the planners) and IPC score over the planning time T (sum over all problems T*/T, where T* is runtime of the fastest planner). In the distributed track, the plan quality will be evaluated both in terms of total cost (sum of costs of all used actions) and makespan (the maximum timestep of the plan if executed in parallel). All actions are considered to have a unit duration.

The validity and quality of plans will be evaluated using the VAL[3] tool, which can handle parallel plans and performs the mutex checks.

CoDMAP as a Future IPC Track

Depending on the results and success of the CoDMAP competition, we suggest a new multi-agent IPC track for the next IPC. Ideally, the new IPC track would have the format of the CoDMAP Distributed Track (alternatively, both sub-tracks might be included), possibly with some enhancements and modifications according to the experience with the current competition and feedback from the participants.

In the CoDMAP competition, most of the competition domains will be adapted from the existing classical IPC domains. It would be much more interesting to come up with new, multi-agent specific domains for the prospective IPC track in order to explore the specific features of multiagent planning. It would also be worth considering some extensions of included features of the formalism such as joint actions, at least for some domains. In CoDMAP, we are not explicitly separating satisficing and optimal planners, which might be useful in the prospective new multi-agent IPC track.

Acknowledgments This research was supported by the Czech Science Foundation (grant no. 15-20433Y) and by the Grant Agency of the CTU in Prague (grant no. SGS14/202/OHK3/3T/13).

[3]http://www.inf.kcl.ac.uk/research/groups/planning

References

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, 28–35.

Brenner, M. 2003. A multiagent planning language. *Workshop on PDDL, ICAPS'03, Trento, Italy, 2003.* 33.

Kovacs, D. L. 2012. A multi-agent extension of PDDL3.1. In *Proc. of the 3rd Workshop on the International Planning Competition (IPC)*, 19–27.

Nissim, R., and Brafman, R. I. 2013. Cost-optimal planning by self-interested agents. In *Proc. of DMAP Workshop of ICAPS'13*, 1–7.

An IPC Track on Deliberative Acting: Moving the competition ahead towards more relevant scientific challenges

Paolo Traverso
FBK ICT IRST
Trento, Italy

Malik Ghallab
LAAS/CNRS
University of Toulouse
Toulouse, France

Dana S. Nau
Department of Computer Science
and Institute for Systems Research
University of Maryland
College Park, MD, USA

Abstract

Deliberative Acting consists of all the reasoning required to choose, organize and perform actions. Planning techniques have been mainly used to choose and organize actions, assuming that actions are directly executable. We argue instead that, most often, in real world applications, deliberation is also required to perform actions, since some form of reasoning is required to decide which lower level steps to execute and how and when to execute them, how to monitor their execution, and how to react to the world dynamics in order to achieve the action's intended effects. Planning techniques (among other ones, like simulation, synthesis of partial programs, or automated verification) can be used to perform actions.

An *IPC track on Deliberative Acting* requires a major shift in approach with respect to current tracks. The purpose is not to compare search engines, their scalability or their performance, but to demonstrate deliberation capabilities, including the efficient exploitation of low-level capabilities in performing actions, their ability to deal with different kinds of real world environments, exogenous events, highly dynamic domains, etc. In this position paper, we provide the motivations for an IPC track on deliberative acting, a novel notion of domain for the IPC, some possible examples and an initial sketch for a roadmap to launch the IPC.

Deliberative Acting

Deliberative Acting consists of all the reasoning required to choose, organize and perform actions (Ghallab, Nau, and Traverso 2014; Nau, Ghallab, and Traverso 2015). It includes two kinds of reasoning: the deliberation needed to decide which actions to perform, as well as the deliberation that is needed to perform actions.

Planning techniques have been mainly used to choose and organize actions, assuming that actions are directly executable, or that the deliberation needed for their execution is the concern of some other agent mostly decoupled from planning. We argue instead that, most often, in real world applications, deliberation is also required to perform actions, since some form of reasoning is required to decide which lower level steps to execute and how to execute them in order to achieve the action's intended effects. Planning is indeed a particular case of deliberation, which can be used both to decide which actions to perform and to perform actions. The two deliberation processes of planning and acting are tightly coupled.

For instance, a planner can generate a plan that includes the action "open door" for a mobile robot. In this case, planning is the deliberation activity to chose (among others) an action to perform. When the robot needs to actually perform the action "open door", deliberation may still be needed to actually open the door, e.g., to decide whether to move closer to the door in order to be able to grasp the handle, to understand which kind of door it is, to decide whether to pull or to push the door, whether to grasp it and turn it while monitoring the execution of such activities, or while monitoring the external environment.

Most often, in order to perform an action, you need a specification of how to perform it, which cannot be described with a *descriptive model*, i.e., with a (either deterministic, nondeterministic, or stochastic) preconditions-effects model. We need *operational models* that specify how to perform an action, i.e., which lower level steps to perform/execute and how to perform/execute them in order to achieve the action's intended effects. For instance, an operational model for the action "open door" can specify that, while the door is not close enough to grasp the handle, then the robot should activate the command that moves the robot towards the door, and when the robots reaches the door, it should activate its sensing capabilities to understand whether the door is closed, to grasp the handle and then pull the door while maintaining the handle grasped and monitoring the status of the door. Planning techniques (among other ones, like simulation, synthesis of partial programs, or automated verification) can be used to deal with operational models and to decide how to perform actions.

Figure 1 shows a simplified view of an actor with deliberation capabilities. The actor interacts with the external environment and with other actors, which exchange information with the actor and can set its objectives. The actor has two main components: A *deliberation component* and an *execution platform*. For instance, the robot's sensory-motor functions are part of its execution platform, which transforms *commands* into corresponding actuations and sensing operations that carry out these commands. The execution platform also transforms signals from the sensed world into *percepts*

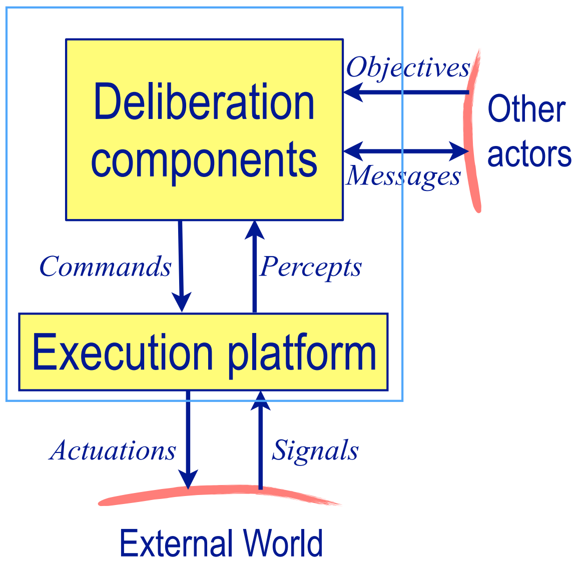


Figure 1: Conceptual view of an actor

and features of the world (e.g., recognition of a physical or virtual object, or information gathered from the Web). The actor's deliberation components will not need a complete and detailed model of the execution platform, but will need enough of a model to enable the actor to use its sensory-motor functions properly.

In order to map its objectives into commands that ultimately achieve these objectives, the actor needs to perform a number of deliberation functions. For example, the actor has to *commit* to goals or tasks meeting its objectives, *plan* for those goals or tasks, *refine* the actions it has planned into commands that the execution platform can execute, *react* to contingent events, and *monitor* its activities to compare the predicted with the observed changes and decide whether recovery actions are needed. All of this is deliberation.

An IPC track on Deliberative Acting

This track should aim at demonstrating the deliberation capabilities of an actor in choosing, organizing, but also performing actions. Such deliberation capabilities should show an intelligent control of a complex behaviour of the actor, able to interact with a dynamic environment in different ways depending on the situation. These capabilities should include:

- *Performing actions* by refining opportunistically and at run-time planned actions into executable commands, e.g. sensory-motor commands, or simulated actions like graphic animations in a video game, or electronic transactions that require interactions with smart objects.
- *Reacting to unexpected exogenous events*, i.e., events that occur because of external factors beyond the actor's control. Even if IPC tracks with nondeterministic or probabilistic models allow theoretically to model the uncertainty caused by exogenous events, taking them into account in such models is often impossible in practice. Most often exogenous events must be ignored because of the impracticality of modelling their possible effects in all situations with every planning operator, and trying to plan in advance for all of the possibilities. An IPC track should test the capability of an actor to respond to respond to events that have not been modelled jointly with planning operators, as soon as they happen, since they can interfere with the execution of an action.
- *Querying and information gathering*, or deliberation on sensing action and focus of attention. Most of the information is not available at planning time, but may become available at run-time. Even in this case, planning with partially observable domains can deal with lack of information. However, most often, the actor needs to plan for and perform actions to make information available.
- *Monitoring the progress of activities*, by recognizing and recovering from failures and adapting to opportunities, by detecting discrepancies between predictions and observations, by assessing and diagnosing these discrepancies, and by deciding about first recovery actions while plan repair is triggered. The actor should show the ability to reason on the required monitoring for its actions and how to do monitoring.
- *Goal reasoning*, i.e., reasoning at the level of the actor's objectives, monitoring the achievement of goals with respect to the evolution of the environment, assessing which goals can be reached after failures, conflicts or opportunities arise, and whether to committ to different (e.g., easier to achieve) goals.
- *Planning at different levels of abstraction*. Actors should be able to perform specialized tasks that have to deal with heterogeneous models. This goes beyond existing hierarchical planning techniques, since tasks at different levels may need different kinds of representations of states and actions (e.g., with discrete or continuous variables). The actor should be able to integrate specialized planning techniques at different levels and have ways to translate among these representations.

A Deliberative Acting IPC Track needs a simulation environment where the actor performs actions. Such environment should have the following characteristics and requirements:

- A dynamic environment with expected as well as unexpected contingent events
- A high variability of the environment with non stationary evolutions
- Large diversity of possible tasks
- Timely actions and reactions
- Partial knowledge of the environment
- Possibly noisy perception actions,
- Dynamic interaction with other actors, including different kinds of actors, e.g., city policy makers and citizens
- High level of autonomy

- Learning capabilities
- Adaptation to the evolution of the environment

The notion of a domain is significantly different here from the classical IPC planning domains, i.e., a detailed PDDL programming of planning operators. The purpose in this track is not to compare search engines but to demonstrate deliberation capabilities. Hence the competition should specify:

- the actors low-level capabilities
- the set of environments and their possible variability and dynamics.
- the set of missions and diversity of behaviours
- the set of interactions
- the competence criteria used by the competition to rank actors

The specification of such capabilities can be partly formal and partly operational with a simulator or even an experimental setup. Hence the quality of the domain modelling and programming should be part of the competition.

The IPC track should rely on simulation platforms, able to integrate rich dynamic environments as well as complex actors. The platforms should be open and suitable for the specification of a large set of scenarios.

In practice, in order to start up the IPC, a strategy is to rely on existing platforms, with possible adaptations. Some examples that need to be analysed in detail and evaluated are the following:

- *Robotics simulation platforms.* For instance, the free Gazebo Robotic Simulator (http://gazebosim.org/), developed by the University of Southern California, provides the ability to simulate populations of robots in complex indoor and outdoor environments. It is based on a simulation physics engine and programmatic and graphical interfaces. Gazebo is free software and has an active community of researchers using it. Another interesting example is MORSE, the Modular OpenRobots Simulation Engine (https://www.openrobots.org/wiki/morse/). MORSE provides realistic 3D simulation of both small and large environments, indoor or outdoor, allowing for multiple autonomous robots. Simulation scenes are generated from simple Python scripts. MORSE includes components simulating a set of standard sensors (cameras, laser scanner, GPS, odometry,...), actuators (speed controllers, high-level waypoints controllers, generic joint controllers) and robotic bases (quadrotors, ATRV, Pioneer3DX, generic 4 wheel vehicle, PR2,...). The CAD system interfaced with MORSE, Blender (http://www.blender.org), is an open source set of tools for modelling physical objects with shape and appearance for rendering, animation and physics simulation (rigid and flexible bodies, fluid, smoke, etc.). See the example in Figure 2.
- *Flight Simulators*, e.g., the NASA Ames Emergency flying simulator (http://ti.arc.nasa.gov/news/emergency-lander-tests/), simulating an environment for an emergency landing system for a damaged aircraft.

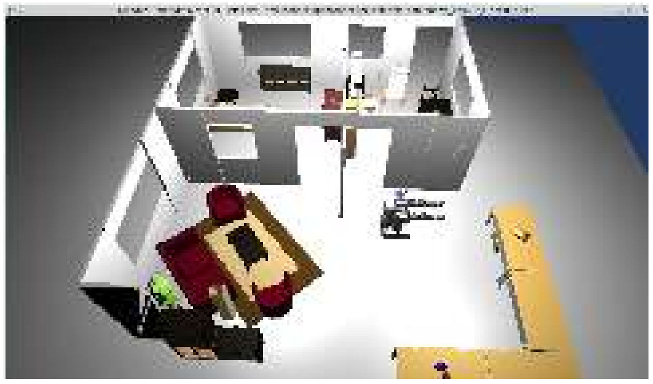


Figure 2: Simulation of a PR2 robot in an apartment with MORSE

- *Video games*, where the competing actor plays the role of the human player interacting with the (more or less intelligent) characters programmed in the game. In multi-player games, the competing actor could play the role of the referee controlling all competing actors. For instance, a video game competition could be adapted from “The General Video Game AI Competition” (http://www.gvgai.net/) which provides a software platform for creating controllers for general video game playing.

- *Serious games*, where the competing actor plays the role of the trainee in some serious training games in the industrial or public sector, e.g., for a medical training program (see. e.g., the Practix solutions for seriuos games for doctors - http://www.practix.net/) or serious games for the training of emergency operations (see, e.g., PRESTO, Plausible Representation of Emergency Scenarios for Training Operations, a system for training in the civil defence sector based on virtual reality - https://shell.fbk.eu/projects/presto).

- *Smart city and community platforms*, i.e., simulation environments for the development of smart city and community services, like the open service platform developed by the High Impact Initiative on Smart Community at FBK (http://www.smartcommunitylab.it/).

In many of the above examples, it is possible to bootstrap the building up of rich domains and scenarios from already existing open source developments. For example, in the robotics case, a number of indoor and outdoor environment models are already available together with models execution platforms and low level commands (e.g., see the ROS command libraries[1] for PR2 robot[2]).

[1]ROS is the Robot Operating System. It provides a set of libraries for the development of robot applications. See - http://wiki.ros.org/.

[2]The PR2 robotics research and development platform. See https://www.willowgarage.com/pages/pr2/overview

The Deliberative Acting Community

In spite of the fact that the vision of deliberative acting described in (Ghallab, Nau, and Traverso 2014; Nau, Ghallab, and Traverso 2015) represents an open research challenge, we believe there is a vibrant community who can be interested in the IPC Track on deliberative acting. Indeed, some of the ideas underlying deliberative acting have been advocated even in pioneering work by many authors, including (Fikes 1971) (Rosenschein and Kaelbling 1986), (Dean and Wellman 1991), and (Pollack and Horty 1999).

The problems addressed in the competition would be of of interest for the robotics community, see e.g., (Despouys and Ingrand 1999; Ingham, Ragno, and Williams 2001; Kortenkamp and Simmons 2007; Effinger, Williams, and Hofmann 2010; Beetz, Mösenlechner, and Tenorth 2010) as well as for the multi-agent community, e.g., (desJardins et al. 1999; Pappachan and Durfee 2000; Brenner and Nebel 2009).

There is also a lot of work in execution control and monitoring that could be relevant for the IPC. The idea to refine actions into lower level commands by proposing different forms of operational models is present in the work based on procedures (e.g., RAP (Firby 1987), PRS (Ingrand et al. 1996), TCA (Simmons 1992), TDL (Simmons and Apfelbaum 1998)), transformation rules (e.g., XFRM (Beetz and McDermott 1994)), situation calculus (e.g., GOLEX (Hähnel, Burgard, and Lakemeyer 1998)), Petri nets and automata (PLEXIL (Verma et al. 2005), SMACH (Bohren et al. 2011), or (Wang et al. 1991; Barbier et al. 2006; Ziparo et al. 2011)). There is also a lot of work in interleaving planning and execution (Koenig 2001; Löhr et al. 2012; Garcia, Prett, and Morari 1989) that could have some relevance for the IPC.

Related Work

The Reinforcement Learning (RL) competition (http://rl-competition.org) tests RL algorithms in problems which can have some similar characteristics to those that we propose. This ithe case of the "Helicopter Domain", where a controller has to interact with a simulator of the helicopter, and to deal with high dimensional, noisy, non-linear dynamics. However, such competition is specific to RL systems, which are very suitable for low level control. Each controllable action is atomic and directly executable by the simulator of the helicopter. No real deliberative acting is involved.

The domain specification language for our competition should allow for, e.g., exogenous and unexpected events, uncertainty and noise in sensing, uncertainty in the result of command executions. Some ideas could be taken by existing IPCs - see, e.g., (Sanner 2010) in the International Probabilistic Planning Competition (IPPC). However, in our competition the problem should not be stated in the usual way: "given the specification of actions and a goal, find a plan that achieves the goal". Such a statement is equivalent to the specification of a search space and reduces the competition to a search problem. The problem should be something like "given a set of low level primitive commands that an actor can perform, e.g., in a simulated environment, and a problem to solve, provide both a modeling of actions (which can, e.g., be refined in some way to the primitive commands) and a set of deliberation techniques that allow the actor to solve the problem". Our competition needs the construction of a model that is used by the actor, as well the use of automated deliberation capabilities of the actors.

We will need to base the domain specification language on experiences from different communities. For instance, in the case of robotics scenarios, we could rely the ROS command libraries. (http://wiki.ros.org/). As a further example, in the case of applications for smart cities and communities, we could use formal languages that have been used for the specification of distributed and pervasive flows, see, e.g., (Bucchiarone et al. 2009).

Our proposal shares some of the difficulties that the International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS) had to address. We should learn from the experience gained in ICKEPS. As suggested by one of the reviewers, we should initially focus on some precise choice, e.g., by choosing a specific simulation architecture. We could focus on a simulator in the field of robotics that allows for a significant number of participants with their minimum effort. We also agree that the scoring criteria for the first run is probably not the major issue compared with the importance of involving the community.

Final Remarks

We realize that we are advocating for a very novel, ambitious, unconventional IPC track. The requirements and domains, including the wide variety of tasks that a competing actor should perform in the IPC track, can be quite challenging. Also the choice about the specification language is a major shift with respect to current IPCs, which use specification language defined (and mainly used only) by our community. This shift could be perceived as one of the major difficulties in involving researchers in the competition. However, we believe that restricting IPCs to specification languages that are not used by other communities for real applications is one of the major barrier to the use in practice of planning techniques.

The set up and development of simulation platforms for the IPC would require a significant effort. We tried to advocate for the re-use of existing simulation engines and to build up from existing environment and execution platform models, which once analysed and evaluated, could help in this task. For this reason we hope that the IPC track that we propose is not impossible to realize.

In the title, with the sentence "Moving the competition ahead towards relevant scientific challenges", we do not mean that previous competitions were irrelevant. They have been extremely useful from the beginning, when with the first IPC at AIPS-98 in Pittsburgh (USA) we understood the potentialities of certain approaches compared to other ones. Moreover, it was important to extend the competition to include different kinds of domains, e.g., nondeterministic and probabilistic domains. However, the risk is that continuing in the restricted view of "plan generation" where the search space is fixed for each domain (given the language) and where actions are directly executable, and planning is not integrated with other forms of deliberation, the research can move apart from useful applications and give raise to more and more artificial and ad hoc, self justified techniques. We believe this is not only the risk of competitions in planning, but it is a general problem for several other fields in AI and more in general in Computer Science. Every competition that may be useful at some point can become quickly a dangerous dead-end for the research community.

For these reasons we hope our proposal will give raise to some effort in a novel form and a novel approach to designing and setting up planning competitions, in spite of the difficulties and the huge effort that such approach requires.

Acknowledgments. This work was supported in part by EU FP7 SAPHARI grant ICT-287513, ARO grant W911NF1210471, and ONR grants N000141210430 and N000141310597. The information in this paper does not necessarily reflect the position or policy of the funders.

References

Barbier, M.; Gabard, J.-F.; Llareus, J. H.; and Tessier, C. 2006. Implementation and flight testing of an onboard architecture for mission supervision. In *International Unmanned Air Vehicle Systems Conference*.

Beetz, M., and McDermott, D. 1994. Improving robot plans during their execution. In *AIPS*, volume 2.

Beetz, M.; Mösenlechner, L.; and Tenorth, M. 2010. CRAM – a cognitive robot abstract machine for everyday manipulation in human environments. In *ICRA*, 1–6.

Bohren, J.; Rusu, R. B.; Jones, E. G.; Marder-Eppstein, E.; Pantofaru, C.; Wise, M.; Mösenlechner, L.; Meeussen, W.; and Holzer, S. 2011. Towards autonomous robotic butlers: Lessons learned with the pr2. In *ICRA*, 5568–5575.

Brenner, M., and Nebel, B. 2009. Continual planning and acting in dynamic multiagent environments. *JAAMAS* 19(3):297–331.

Bucchiarone, A.; Lluch-Lafuente, A.; Marconi, A.; and Pistore, M. 2009. A formalisation of adaptable pervasive flows. In *Web Services and Formal Methods, 6th International Workshop, WS-FM 2009, Bologna, Italy, September 4-5, 2009, Revised Selected Papers*, 61–75.

Dean, T. L., and Wellman, M. 1991. *Planning and Control*. Morgan Kaufmann.

desJardins, M.; Durfee, E. H.; Ortiz, C. L.; and Wolverton, M. 1999. A survey of research in distributed, continual planning. *AI Mag.* 20(4):13–22.

Despouys, O., and Ingrand, F. 1999. Propice-Plan: Toward a unified framework for planning and execution. In *European Workshop on Planning*.

Effinger, R.; Williams, B.; and Hofmann, A. 2010. Dynamic execution of temporally and spatially flexible reactive programs. In *AAAI Workshop on Bridging the Gap between Task and Motion Planning*, 1–8.

Fikes, R. E. 1971. Monitored Execution of Robot Plans Produced by STRIPS. In *IFIP Congress*.

Firby, R. J. 1987. An investigation into reactive planning in complex domains. In *AAAI Conference*. Seattle, WA.

Garcia, C. E.; Prett, D. M.; and Morari, M. 1989. Model predictive control: theory and practice–a survey. *Automatica* 25(3):335–348.

Ghallab, M.; Nau, D.; and Traverso, P. 2014. The actor's view of automated planning and acting: A position paper. *Artif. Intell.* 208:1–17.

Hähnel, D.; Burgard, W.; and Lakemeyer, G. 1998. GOLEX—bridging the gap between logic (GOLOG) and a real robot. In *KI*, 165–176.

Ingham, M. D.; Ragno, R. J.; and Williams, B. C. 2001. A reactive model-based programming language for robotic space explorers. In *i-SAIRAS*.

Ingrand, F.; Chatilla, R.; Alami, R.; and Robert, F. 1996. PRS: A high level supervision and control language for autonomous mobile robots. In *ICRA*, 43–49.

Koenig, S. 2001. Minimax real-time heuristic search. *Artif. Intell.* 129(1-2):165–197.

Kortenkamp, D., and Simmons, R. 2007. Robotics systems architectures and programming. In Khatib, and Siciliano., eds., *Handbook of Robotics*. Springer.

Löhr, J.; Eyerich, P.; Keller, T.; and Nebel, B. 2012. A planning based framework for controlling hybrid systems. In *ICAPS*.

Nau, D. S.; Ghallab, M.; and Traverso, P. 2015. Blended planning and acting: Preliminary approach, research challenges. In *AAAI-15*.

Pappachan, P. M., and Durfee, E. H. 2000. Interleaved plan coordination and execution in dynamic multi-agent domains. In *ICMAS*, 425–426.

Pollack, M. E., and Horty, J. F. 1999. There's more to life than making plans: Plan management in dynamic, multiagent environments. *AI Mag.* 20(4):1–14.

Rosenschein, S., and Kaelbling, L. P. 1986. Integrating planning and reactive control.

Sanner, S. 2010. Relational Dynamic Influence Diagram Language (RDDL): Language Description. Technical report.

Simmons, R., and Apfelbaum, D. 1998. A task description language for robot control. In *IROS*, 1931–1937 vol.3.

Simmons, R. 1992. Concurrent planning and execution for autonomous robots. *Control Systems, IEEE* 12(1):46–50.

Verma, V.; Estlin, T.; Jónsson, A. K.; Pasareanu, C.; Simmons, R.; and Tso, K. 2005. Plan execution interchange language (PLEXIL) for executable plans and command sequences. In *i-SAIRAS*.

Wang, F. Y.; Kyriakopoulos, K. J.; Tsolkas, A.; and Saridis, G. N. 1991. A Petri-net coordination model for an intelligent mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics* 21(4):777–789.

Ziparo, V. A.; Iocchi, L.; Lima, P. U.; Nardi, D.; and Palamara, P. F. 2011. Petri net plans. *AAMAS* 23(3):344–383.

Towards a Protocol for Benchmark Selection in IPC

Mauro Vallati
PARK Research group
University of Huddersfield
m.vallati@hud.ac.uk

Tiago Vaquero
California Institute of Technology
Massachussets Institute of Technology
tvaquero@caltech.edu

Abstract

The planning competition has traditionally played an important role in motivating research and advances in Planning & Scheduling techniques. Despite its pivotal role in the planning community, some aspects of the competition have not been engineered yet. This is the case for the protocol for selecting benchmark instances. Benchmarks are of critical importance, since they can significantly affect competition results.

In this paper we describe desirable properties of a selection protocol, discuss methods exploited in past SAT and planning competitions, and identify challenges that organisers of future competitions have to address in order to improve reliability and usefulness of the insights gained by looking at competitions' results.

Introduction

Competitions are important events to improve a particular research area. Some examples are the International SATisfability Competition (SAT), the Conference on Automated Deduction ATP System Competition (CASC), the Trading Agent Competition (TAC) and many others. This strategy has been used by the AI Planning & Scheduling (P&S) community to develop innovative planning techniques since 1998 through the *International Planning Competition* (IPC) and also to promote the development of knowledge engineering tools and systems since 2005 through the *International Competition on Knowledge Engineering for Planning and Scheduling* (ICKEPS). Both competitions attract researchers in the AI community, especially IPC due to its motivational aspect of developing better and more powerful planning engines to address increasingly large (and hopefully complex) problems. Techniques tested in competitions are then available to be used in real-world applications.

In the IPC, participating planning engines are tested against several benchmark problems, a few of them are inspired by real-world problems. The selection and design of these benchmark domains and problems instances have become one of the main challenges encountered during the organisation of this competition. Given a set of target domains, it is well known that benchmark instances selection can directly impact results (Howe and Dahlman 2002). Moreover, the very small number of theoretical studies on complexity of instances and transition phase (which changes between domains) (Helmert 2006; Rintanen 2004), and the growing number of participants exploiting different planning techniques, make the selection and decision of IPC problem instances a significant challenge for organisers. It is worth noting that the selection of benchmarks is only one of the many difficulties faced by competition organisers. Organising a competition requires a significant amount of work. In fact, organisers of past competitions had to face a lot of pressure for selecting and generating new domains and problems by themselves. For addressing the selection issues, organisers have been using different selection strategies and criteria throughout the years, which have been the target of several post-competition discussions and criticisms. Given the importance of competitions for the community, the responsibility of generating benchmarks should be shared between all the members, rather than delegated to organisers only. Also for this reason, a protocol for selecting benchmarks is highly desirable.

In this paper, we emphasise and highlight the need for a protocol for selecting IPC benchmark problems that: is transparent; reproducible; avoids bias to any particular planning technique; adapts to the state-of-the-art and the existing participating planners; allows and motivates new benchmark domains to be added (e.g., challenging domains from ICKEPS); supports the realisation and exclusion of outdated and uninteresting problems for the participating planners; aims to evaluate and understand the technological progress in the long-term and, possibly, fosters the evaluation of planning techniques in new potential real-world applications.

Desirable Properties

In this section we provide two lists of desirable properties: one for the selection protocol itself, and one for the selected benchmark instances. Although properties of the instances are induced by a proper selection protocol, thus are somehow implicitly guaranteed by the protocol properties, we prefer to divide the lists and make their properties clear for the sake of readability. Desirable properties of a selection protocol are:

- **Transparency**. Others can follow the method and, while considering the same "environmental" conditions, produce the same (sort of) problems.
- **Generality**. It can be applied to any set of planners, on

any target domain.

- **Unbiased**. It does not favour a system against another.
- **History-aware**. This avoids tailored algorithms. It limits the impact of approaches based on learning, which exploits problems and domains from previous competitions.
- **Progress-driven**. It motivates technological progress in new domains and problems.

In order to be useful, a set of benchmark instances must include problems that are:

- **Challenging**. Problems must not be trivially solvable or unsolvable. They must provide information about the performance of participants.
- **Interesting**. They investigate possible exploitation of planning in real-world scenario, or test innovative features.
- **Diverse**. They do not refer to the same kind of problems or models.
- **Finite**. The selected instances must be in a finite number. Moreover, the smaller the set of benchmarks, the easier is to re-run the competition and reproduce results.

It should be noted that the properties we introduce consider also the importance of planning competitions for the community. IPC is a major event of the planning community, therefore it should provide also some guidance about applications, limitations and strengths of the existing solvers, as well as identifying future avenues of research while situating the technological progress.

Existing Protocols

In this section we describe the existing techniques that have been used for selecting benchmark instances in the International SAT competition, and in the deterministic track of International Planning Competitions.

SAT competition

We observed that a very similar selection protocols have been used in SAT competitions since 2012 (Balint et al. 2012; 2013; Belov et al. 2014a). Hereinafter, we will focus on the policies used in the latest edition.

In the 2014 SAT competition, two main sets of benchmarks are considered: (i) uniform random and (ii) application and hard combinatorial. The way in which corresponding instances are selected is different. For the first set, two different sizes – medium and huge – of uniform random formulae are generated by using existing generators (Belov et al. 2014c). The huge benchmarks have millions of clauses and a clause-to-variable ratio ranges from far from the phase-transition ratio to relatively close. On the other hand, medium benchmarks are smaller, but have clause-to-variable ratio equal to the phase-transition ratio. Remarkably, given the theoretical knowledge about complexity of random SAT instances (Rossi, Van Beek, and Walsh 2006), the uniform random benchmark selection does not need to consider the performance of actual solvers; complexity is theoretically assessed.

A different protocol is used for selecting application and hard combinatorial benchmarks (Belov et al. 2014b). In such tracks, it is important to consider the performance of solvers. Firstly, benchmarks collected by previous competitions (either used or unused) and newly submitted benchmarks have been divided into buckets. The assignment to a specific bucket is guided by the combinatorial problem the benchmark originates from, and the submitter. This partition is done in order to limit possible biases deriving from the use of large number of instances that refers to the same problem, or that have been used in previous competitions.

The empirical hardness of benchmarks is evaluated by using five well-performing solvers, per track, from the 2012 SAT challenge. To consider differences in performance due to environment / technological improvements, the CPU runtimes have been scaled. According to solvers performance, benchmarks have been rated as follows:

Easy Benchmarks solved by all the considered solvers in less then 1/10-th of the competition's timeout.

Medium Benchmarks solved by all the solvers within the competition's timeout.

Hard Benchmarks solved by at least one solver within the double of the timeout, and not solved by at least one solver within the competition's timeout.

Too-hard Benchmarks unsolved by any solver within two times the considered cutoff time.

For each track, 300 benchmarks are selected from the medium and easy classes. The selection process, that must provide a 50-50 ratio between satisfiable and unsatisfiable formulae, is controlled by the following constraints:

1. no more than 10% of the instances should come from the same bucket;
2. the percentage of new benchmarks should be as high as possible;
3. the ratio of Medium to Hard benchmarks should be as close to 50-50 as possible. However, this constraint has been relaxed by selecting 20% of the benchmarks from Medium, Hard and Too-hard classes. This for reducing the influence of selected solvers.
4. the performance of the solvers used for the evaluation of the benchmarks should be as uniform as possible, to avoid bias due to a specific technique.

International Planning Competition

In the following we describe the protocols used by the organisers of the deterministic track of the IPC 2011 and 2014. We focus on deterministic track since it is the largest one, in terms of participants, and thus requires clearly defined strategies for benchmark selection.

Before describing the protocols, we would remark that over time, IPC organisers have put more and more effort in studying suitable techniques for selecting benchmark instances. This is due to a number or reasons: firstly, the growing number of participants; secondly, the wide range of problems and domains that can be modelled in PDDL; thirdly, the importance of guaranteeing an unbiased set of instances; and finally, since the IPC has usually been held every 2-3 years, the difficulty of estimating the progress of the state of the art.

2011 edition In the deterministic track of the IPC 2011, organisers adopted two different methods for selecting benchmarks, according to the fact that data on their difficulty were available or not (López, Celorrio, and Olaya 2015).

For newly introduced domains, state-of-the-art planners are used for evaluating the difficulty of reduced test sets of problems, which are generated using randomised generators, with some specific parameters. A cutoff of 300 seconds has been considered for these tests. The easiest problems are those solved in tens of seconds, the most difficult problems are those which are unsolvable by considered planners, in a 300 seconds cutoff. By following a trial-and-error procedure, a suitable set of parameters is found, and can be used for generating 20 benchmark instances.

For domains used in previous competitions, the publicly available data is used for ranking planning tasks according to their expected difficulty, measured by using the Glicko score, and for selecting them.

2014 edition In the deterministic track of the IPC 2014, organisers provided a protocol for selecting, within a specific domain, a set of suitable instances, tailored for the participating planners.[1] They defined as "trivial" instances in which almost all the planners performed very similar – in terms of quality of plans for satisficing subtracks, and runtime for the Agile subtrack – and "too complex" those instances which are not solved by any planner. For each target domain:

1. identify size;
2. given the sizes, generate between 30 and 50 instances per domain, using available generators;
3. anonymise planners;
4. run all the planners on the generated instances;
5. collect results, in terms of solved problems and quality of solutions;
6. order problems by number of planners which solved them;
7. selection of 20 benchmarks.

In the first step, if the domain has been already used in previous IPCs, then the sizes of larger benchmark problems (top half), and also extended them, following the "trend" used by organisers, are taken. Otherwise, some well-known planners, either from literature or from IPC 2011, are used. If no generator is available, all the available instances have been considered.

In step 7, if between (circa) 10 and 20 instances have been solved by some considered planners, then select the top 20 instances accordingly to the order in step 6. If most of the instances are either trivial or too complex, according to planners' performance, then the process is started again from step 1. Otherwise, trivial and too complex instances are removed, in order to obtain a final set of 20 problems.

[1]The protocol can be found at: http://helios.hud.ac.uk/scommv/IPC-14/selection.html

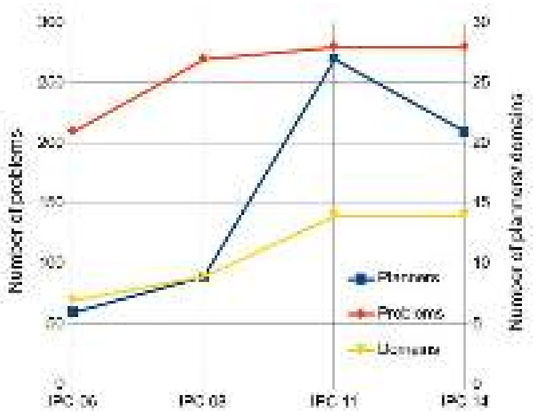


Figure 1: Number of participants, instances and domains considered in IPCs since 2006.

Challenges

Having provided the desirable properties of benchmarks selection protocols, and having introduced protocols used in two major competitions within the artificial intelligence area, in this section we discuss the challenges that should be faced, in order to furtherly improve the importance and significance of competitions.

As a first remark, we observe that the notion of *quality* of benchmarks is missing. In the protocols introduced in the previous section, organisers do not explicitly mention this aspect of benchmarks. Although some properties of useful benchmarks have been identified – and we introduced a few of them – having a formal definition of quality would be extremely helpful, and would also allow the exploitation of knowledge engineering approaches for defining sound selection protocols. Remarkably, a first effort in this direction was done by the IPC 2011 organisers. They tried to measure quality of planning tasks as fitness to a normal distribution (López, Celorrio, and Olaya 2015). In the competitions context, quality of benchmarks is not "static", but it depends on the current state of the art, as well as on the potential applications of the evaluated solvers. In planning, good quality instances should test planning techniques in real-world applications. As a matter of fact, the IPC should investigate pioneering uses of planning.

It is still unclear if there exists a *right* number of benchmarks. Figure 1 shows the variation of number of planners, instances and domains in sequential satisficing subtracks of the IPC over time. It should be noted that last IPCs organisers had a large set of domains to choose and start from, while in earlier IPCs benchmarks had to be developed. Commonly, it is believed that the larger the set of benchmarks, the more accurate the overall evaluation of participants. This leads to computationally expensive competitions, which require significant amount of CPU and human hours to be run (although most of the work can be done automatically through existing IPC software (Linares López, Celorrio, and Helmert 2013)). Moreover, large set of benchmarks can possibly include low quality instances, which introduce noise in the evaluation. It would be interesting to identify a formal way for estimating the number of required benchmarks for assessing the performance of a given number of planning systems, for facilitating planners exploitation outside the community.

It is worthy to note that in the IPC, differently for most of the other contests in AI, domains are explicitly given and are of primary importance. Domains strongly affect the structure of problems, thus having a significant impact on the planners' performance. Therefore, in the IPC both domains and instances have to be wisely selected. Although techniques have been described and exploited for selecting problems, the domain selection process is still some sort of obscure task, which has not been deeply investigated yet. Interestingly, in IPCs, the trend is to increase the number of considered domains, and reducing the number of problems per domain. Also, given the strong impact that different models of the same domain or different model refinements can have on the performance of planners (Riddle, Holte, and Barley 2011; Vaquero et al. 2010), it might be interesting to consider, within a competition, more than one specific PDDL model. Such different models do not have to exploit different PDDL-features; this has been done in previous IPCs. Sadly, state-of-the-art planners support a very limited subset of them. Here we suggest to test different ways of encoding the same domain, with the same set of PDDL features. For instance, using models of blocksworld using 3 or 4 operators, and evaluate planners by considering their average performance. The generation of different models of domains could be for example the scope of future ICKEPS competitions. Moreover, the analysis on the different performance of planners on different models can give useful insights on knowledge engineering aspects of domain modelling (e.g., given a particular model it might be possible to map the planners that would have the better performance). Such evaluation, if put in practice, will significantly increase the already high pressure on organisers. Once again, we would like to emphasise that generation and selection of benchmark should be done collectively by the community, and fostered by a shared protocol. For instance, models can be generated by exploiting crowdsourcing (Zhuo 2015).

All the competitions are using existing techniques for identifying a large set of promising benchmarks. Such techniques should be as various as possible, i.e. exploiting very different planning approaches, in order to avoid biases and identify challenging sets. The actual benchmarks are then usually selected by considering the performance of participants. Even though this reduces the number of useless instances –e.g., trivial or too complex– this can possibly introduce some bias. Specifically, benchmarks might be too focused on the competitors, thus ignoring the larger situation of the state of the art.

Current benchmarks selection protocols are mainly focused on the CPU time needed by a system for solving the given instances. This helps to discriminate between trivial, challenging and too complex instances. In tracks where the runtime is not considered in the metric, like the satisficing subtrack of the deterministic IPC, this approach can be improved by considering also aspects that are accounted for the metric. For instance, the presence of multiple solutions, with different costs, can be useful when evaluating planners that should return high quality plans.

It has been shown that different configurations of hardware and software can differently affect the performance of domain-independent planners (Howe and Dahlman 2002). Given that, it would be interesting to assess the reliability of a competition results, which are collected on a single system, with regards to their generalisation on different infrastructures. Potentially, running the competition few times on different systems and merging results would provide a more accurate evaluation, but of course, will be extremely costly.

Finally, a competition should also provide a clear picture about the progress of the state of the art, mainly with regards to the previous competition. This evaluation is twofold. Firstly, we are interested in evaluating the progress in terms of planning performance; i.e., new planners have to be faster, solve more and more problems, and/or find better plans. Secondly, the progress also involves the languages used for representing knowledge. In particular, are the new languages able to model more problems? Do they positively affect the performance of solvers? While the evaluation of the planners' performance progress seems to be mostly related to the size –as an indicator of complexity– of problems that can be solved, the evaluation of languages is mostly connected with knowledge engineering aspects. On this matter, a cooperation between the IPC and the ICKEPS is strongly suggested.

Conclusion

Selecting benchmark instances is a critical task that every AI competition has to face. It has a dramatic impact on the final results and, given the pivotal role of competitions within AI communities, the selection of benchmarks strongly affects also the future development of the specific area. Given the importance of benchmark selection, and the high pressure organisers have to face on this regards, it would be desirable that the whole community supports organisers in this difficult task. In particular, this can be done also by exploiting a protocol. A proper protocol will lead to more reliable and informative competition results. Even though its central role, desirable characteristics and properties of a selection protocol have not been thoroughly discussed yet.

In this paper, we provide a list of desirable properties of both the selection protocol and the selected instances. We discuss methods used in SAT and Planning competitions for selecting benchmarks in order to gain useful insights on the limitations and strengths of the existing exploited approaches. Such gained knowledge is then used for highlighting challenges and, possibly, providing avenues for improving selection protocols in future planning competitions. In particular, we observe that: (i) a formal technique for selecting domain models is missing; (ii) it is unclear what should be the "right" number of benchmarks; (iii) it might be useful to consider the evaluation metric – used in the competition for evaluating planning systems – also in the selection protocol. Finally, we would remark the importance of the competition for assessing the progress of the state of the art, and for pioneering innovative applications of automated planning.

References

Balint, A.; Belov, A.; Diepold, D.; Gerber, S.; Järvisalo, M.; and Sinz, C. 2012. Sat challenge 2012.

Balint, A.; Belov, A.; Heule, M. J.; and Järvisalo, M. 2013. Sat competition 2013.

Belov, A.; Diepold, D.; Heule, M. J.; and Järvisalo, M. 2014a. Sat competition 2014.

Belov, A.; Heule, M. J.; Diepold, D.; and Järvisalo, M. 2014b. The application and the hard combinatorial benchmarks in sat competition 2014. In *Proceedings of SAT competition 2014*, 81–82.

Belov, A.; Heule, M. J.; Diepold, D.; and Järvisalo, M. 2014c. Generating the uniform random benchmarks. In *Proceedings of SAT competition 2014*, 80.

Helmert, M. 2006. New complexity results for classical planning benchmarks. In *In International Conference on Automated Planning and Scheduling ICAPS*, 52–61.

Howe, A. E., and Dahlman, E. 2002. A critical assessment of benchmark comparison in planning. *Journal of Artificial Intelligence Research* 17(1):1–33.

Linares López, C.; Celorrio, S. J.; and Helmert, M. 2013. Automating the evaluation of planning systems. *AI Commun.* 26(4):331–354.

López, C. L.; Celorrio, S. J.; and Olaya, A. G. 2015. The deterministic part of the seventh international planning competition. *Artificial Intelligence*.

Riddle, P. J.; Holte, R. C.; and Barley, M. W. 2011. Does representation matter in the planning competition? In *Proceedings of the Ninth Symposium on Abstraction, Reformulation, and Approximation, SARA*.

Rintanen, J. 2004. Phase transitions in classical planning: An experimental study. In *In International Conference on Automated Planning and Scheduling ICAPS*, volume 2004, 101–110.

Rossi, F.; Van Beek, P.; and Walsh, T. 2006. *Handbook of constraint programming*. Elsevier.

Vaquero, T. S.; Silva, J. R.; Beck, J. C.; et al. 2010. Improving Planning Performance Through Post-Design Analysis. In *Proceedings of the ICAPS'10 Workshop on Knowledge Engineering for Planning and Scheduling. Toronto, Canada*, 45–52.

Zhuo, H. H. 2015. Crowdsourced action-model acquisition for planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*.