Automating the Evaluation of Planning Systems

Carlos Linares López

Computer Science Department
Universidad Carlos III de Madrid (Spain)
E-mail: carlos.linares@uc3m.es

Sergio Jiménez

Computer Science Department
Universidad Carlos III de Madrid (Spain)
E-mail: sergio.jimenez@uc3m.es

Malte Helmert

Department of Mathematics and Computer Science
Universität Basel (Switzerland)
E-mail: malte.helmert@unibas.ch

Research in automated planning is getting more and more focused on empirical evaluation. Likewise the need for methodologies and benchmarks to build solid evaluations of planners is increasing. In 1998 the planning community made a move to address this need and initiated the International Planning Competition – or IPC for short. This competition has typically been conducted every two years in the context of the International Conference on Automated Planning and Scheduling (ICAPS) and tries to define standard metrics and benchmarks to reliably evaluate planners. In the sixth edition of the competition, IPC 2008, there was an attempt to automate the evaluation of all entries in the competition which was imitated to a large extent and extended in several ways in the seventh edition, IPC 2011. As a result, a software for automatically running planning experiments and inspecting the results is available, encouraging researchers to use it for their own research interests. The software allows researchers to reproduce and inspect the results of IPC 2011, but also to generate and analyze new experiments with private sets of planners and problems. In this paper we provide a gentle introduction to this software and examine the main difficulties, both from a scientific and engineering point of view, in assessing the performance of automated planners.

Keywords: Automated planning, evaluation, competition

1. Introduction

Automated Planning is an area of Artificial Intelligence (AI) that studies, in its most general form, the automatic selection of actions for achieving a number of goals from an initial state. Blind methods for planning do not scale up because the search space of planning problems grows exponentially. Hence, research on automated planning focuses on developing general methods capable of effectively exploring the search space in different classes of planning problems.

In practice, the number of planners implementing different ideas trying to improve the existing ones has increased significantly and also, a large number of problems in a wide variety of planning domains have been proposed over the last years. Similarly to other areas of Artificial Intelligence, such as Satisfiability Testing (SAT), Answer Set Programming (ASP) or Machine Learning (ML), the emphasis on performance improvement is leading researchers to pay particular attention to the empirical evaluation, increasing the need for reliable methodologies and benchmarks.

The International Planning Competition (IPC) has taken place roughly every two years in the context of the International Conference on Automated Planning and Scheduling (ICAPS) and is targeted at the empirical evaluation of state-of-the-art planners. Encouraged by this competition, researchers have been constantly improving their planners and proposing new interesting benchmarks to evaluate them. Many of the metrics and problems used at the various editions of the competition have become a reference to the vast majority of practitioners, and the common practice to prove progress with respect to a particular planner is to show the differences in performance on the set of problems from the IPC.

In the Sixth International Planning Competition (IPC 2008)[1] there was a serious attempt to automate much of the process required to run a large number of experiments with an arbitrarily large collection of

[1]See `http://ipc.informatik.uni-freiburg.de`.

planners and problems. The design and some key parts of the code developed at that time were mimicked to a large extent, and extended in several ways, in the last International Planning Competition, IPC 2011 [2]. All the software developed during the IPC 2011 is now available[3] along with technical documentation[4] under the terms of the GNU General Public License version 3. To exemplify its usage, researchers can now: run tests on their own planners and make comparisons with the competitors of the last competition; store a private collection of benchmarking problems; inspect the results of the execution of any planner on any selected subset of problems; perform statistical tests on the data retrieved, etc. The results of all the experiments are permanently stored in a repository, and the results of the last competition are publicly available in the same form, thus improving the accessibility of the results and easing their inspection.

In this paper we present a gentle introduction to this software and examine the main difficulties, both from a scientific and engineering point of view, in assessing the performance of automated planners. The paper is structured as follows: Section 2 introduces basic concepts of Automated Planning and the International Planning Competition. Section 3 introduces the diverse evaluation schemas followed at the IPC series and explains the evaluation decisions that motivate the design of the software. Section 4 explains the structure of the software developed for running IPC 2011. Section 5 shows practical uses of this software illustrating how to use it for both the evaluation of planners and the analysis of results. Finally Sections 6, 7 and 8 discuss, respectively, related work, future work and conclusions.

2. Background

This section first introduces the definition of *automated planning tasks*; then, it addresses important issues when evaluating planners, and finally, it presents the International Planning Competition, the main forum for planning evaluation.

[2]See http://www.plg.inf.uc3m.es/ipc2011-deterministic.

[3]See http://www.plg.inf.uc3m.es/ipc2011-deterministic/FrontPage/Software.

[4]See http://www.plg.inf.uc3m.es/sw-ipc2011/.

2.1. Automated Planning

Automated Planning studies the selection of actions in a dynamic system to reach a state of the system that satisfies a number of goals. The input to the planning process is typically separated into two elements, the domain and the problem. The domain contains the set of predicates used for describing the states and actions of the dynamic system and a schematic description of the actions of the dynamic system. The problem contains the objects to ground the schematic actions of the dynamic system, the specification of its initial state and the goals to achieve. Figure 1 illustrates a typical Automated Planning process.

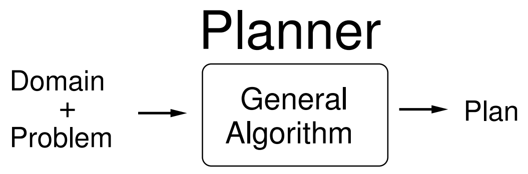

Fig. 1. Overview of the planning process.

There are different approaches to Automated Planning that range from the decomposition of hierarchical tasks [Nau et al., 2003] to the use of timelines [Morris and Muscettola, 2005]. In this paper we focus on *classical planning*, the simplest and most common planning model addressed at the IPC.

Classical planning assumes that the dynamic system can be described by a *state transition* system defined by a tuple $\sum = (S, A, C)$ where S is a finite set of states, A is a finite set of actions and $C(s, a)$ is a function of the cost of applying action $a \in A$ in state $s \in S$. Classical planning is essentially a graph search problem in a weighted and directed graph whose vertices are the states of the problem and whose arcs correspond to state transitions. The difficulty comes from the fact that the graphs are generally too large to be traversed exhaustively. In the formalism we describe here, the number of (grounded) state variables can grow exponentially in the number of objects, and the number of states is again exponential in the number of grounded state variables.

Figure 2 shows the action `unstack(X,Y)` of the *Blocksworld* planning domain represented in PDDL, the representation language of the IPC. The *Blocksworld* is a classic domain in automated planning which consists of a set of blocks, a table and a gripper: blocks can be on top of other blocks or on the table, a block that has nothing on it is clear, and the gripper can hold one block or be empty. This representation indicates the *preconditions* of the action, facts required to be true for the application of the action; the *add effects*, facts made true by the application of the action; the *delete effects*, facts made false by the application of the action and written as a negated fact e.g., `(not (clear ?x))` in Figure 2; and the cost associated with its execution.

```
(:action unstack
  :parameters
    (?x - block ?y - block)
  :precondition
    (and (on ?x ?y)
         (clear ?x)
         (handempty))
  :effect
    (and (holding ?x)
         (clear ?y)
         (not (clear ?x))
         (not (handempty))
         (not (on ?x ?y))
         (increase (total-cost) 1)))
```

Fig. 2. Action `unstack` from the *Blocksworld* domain.

Following the previous notation, a planning problem can be defined as a tuple $P = (\sum, s_0, G)$ where $s_0 \in S$ is the initial state of the dynamic system and $G \subseteq S$ is the set of states where the goals are satisfied (described compactly, for example as a logical formula). Figure 3 shows the problem representation of the *Sussman anomaly* in the *Blocksworld* domain [Sussman, 1975] using PDDL. This problem comprises three blocks labeled A, B, and C. The problem starts with blocks B and A on the table and block C on top of A, and consists of stacking the blocks such that A is on top of B and B is on top of C.

Solutions to planning problems are expressed in different forms such as a sequence of actions, a policy, or a partially ordered set of actions. In the case of classical planning a solution to a planning problem P is a sequence of ground actions $(a_1, a_2, ..., a_n)$ corresponding to the sequence of states $(s_0, s_1, ..., s_n)$ such that: action a_i is applicable in state s_{i-1}, the state s_i is the result of executing a_i in s_{i-1} and s_n is a state where all goals are satisfied, $s_n \in G$. The cost of the sequential plan is the sum of its action costs, formally $\sum_{i=1}^{n} C(s_{i-1}, a_i)$, and therefore, an optimal solution to a planning task is one that minimizes the sum. Figure 4 shows an example of a sequential plan solving the Sussman anomaly. This solution is a plan with six actions and a total cost of six units, the sum of the cost of the actions applied shown in brackets.

```
(define (problem Sussman)
 (:domain blocksworld)
 (:objects blockA blockB blockC  - block)
 (:init (handempty)
        (on blockC blockA)
        (ontable blockA)
        (ontable blockB)
        (clear blockB)
        (clear blockC))
 (:goal
   (and (on blockA blockB)
        (on blockB blockC)
        (ontable blockC))))
```

Fig. 3. The Sussman anomaly in the *Blocksworld* domain.

```
0: (UNSTACK BLOCKC BLOCKA)  [1.000]
1: (PUT-DOWN BLOCKC) [1.000]
2: (PICK-UP BLOCKB) [1.000]
3: (STACK BLOCKB BLOCKC) [1.000]
4: (PICK-UP BLOCKA) [1.000]
5: (STACK BLOCKA BLOCKB) [1.000]
```

Fig. 4. Example of a solution plan for the *Sussman anomaly* problem.

The IPC comprises different kinds of planning tasks that correspond to planning models of different expressiveness. For example, *classical planning* studies the generation of sequential plans in deterministic and fully observable environments. *Temporal planning* studies planning problems where actions are not instantaneous but have different durations and their preconditions and effects are temporally annotated (e.g., effects occurring at the end of the action and preconditions having to hold over the complete duration of applying the action). Typically, the objective of *temporal planning* is generating a concurrent plan with minimal makespan, i.e., the temporal duration of the plan [Cushing et al., 2007, Rintanen, 2007]. *Planning with soft goals* studies how to generate plans when goals express requirements of different strengths, for example, motivated by user preferences [Keyder and Geffner, 2009]. *Planning under uncertainty* studies how to tackle planning problems when states are not fully observable and when the effects of actions are not deterministic [Bonet and Geffner, 2003].

Each of these planning models has its own language extensions for representing the corresponding dynamic system, initial state, goals and solution plans. Likewise, each model has its own algorithms for effectively solving the corresponding planning problems. On the whole, one can say that state-of-the-art planners often rely on heuristic search to generate the solutions [Bonet and Geffner, 2001] but compilations to other forms of general problem solving have also been effectively used. A prominent example is SAT planning [Kautz and Selman, 1996, Kautz and Selman, 1999]. It encodes the planning task as a SAT problem which is then solved by a SAT solver. A number of SAT encodings are known to be effective for solving specific planning problems, such as for conformant planning [Palacios and Geffner, 2007] or contingent planning [Albore et al., 2009]. Other examples of compilations transform the planning task into a *Constraint Satisfaction* [van Beek and Chen, 1999] or a *Model Checking* [Cimatti et al., 1997] problem.

2.2. Evaluation of planners

The experimental evaluation of planners usually consists of running the planners over a set of different problems and comparing their performance by looking at some features of the generated solutions. To ensure that the obtained evaluation results are meaningful, the evaluation must be compliant, at least, with the following characterization:

- *Objectivity*. The evaluation setting must be objective and it must be the same for all evaluated planners. Important aspects of the setting include:
    - *Representation language*. Planning problems can be represented using different formalisms and even using the same formalism, they can be represented in different ways affecting the performance of planners. Indeed, it is a well-known fact that the performance of a planner can be altered by modifying the order of the definition of some elements in either the domain or problem description [Howe and Dahlman, 2002].
    - *Evaluation metric*. Planners can be evaluated with regard to different metrics. Examples of such metrics include the number of solutions found, their lengths and costs, and the overall time spent for finding them. Previous IPCs have shown that the selection of a particular metric can affect the selection of the planner with the best performance. Different metrics might favor different planning approaches. For example, SAT-based planners have been quite successful for finding optimal temporal plans in the setting where all actions have the same duration, but have achieved less success for finding optimal sequential plans (minimizing total action cost).
    - *Validation mechanism*. A reliable mechanism must validate the solutions output by the planners. This mechanism must verify that the solution plans are executable, reach the goals starting from the given initial state, and that the reported metric values are correct. In the case of tracks that mandate optimal solutions, ideally the optimality of the reported solutions should also be verified. However, this is generally challenging, as verifying the optimality of a solution requires an exhaustive proof that no better solutions exist.
- *Exhaustiveness*. The evaluation should support general conclusions. This means that planners must be evaluated over problems of a different nature. Currently there is no generally accepted measure for the diversity of planning problems. In addition, the selected problems must cover different difficulty ranges. Problem generators can help to synthesize initial states and goals of different difficulty keeping the same problem structure, but again there is no general measure for predicting the difficulty of the generated planning problems.
- *Comparability*. The evaluation should quantify progress with respect to previous evaluations. Including well-studied algorithms as baselines or reusing problems in the evaluation makes it easier to tie new results to the literature.
- *Reproducibility*. The evaluation must be reproducible ensuring that the same results can be obtained when using the same experimental conditions. Therefore the evaluation process must report all the environmental conditions used during the experimentation. These conditions comprise, at least:
    - *Hardware details*. The CPU model, the available RAM, the size of the cache memory, etc. affect the performance of planners.
    - *Software details*. The version of the operating system, the compiler and the process that executes the planner can also affect the performance of planners.

  * *Experimental setup*. Planners must be evaluated with the same CPU time, memory and disk usage limits.
  * *Random behavior*. Planners with (pseudo-) random behavior must be able to reproduce their performance using the same random seed.

– *Reliability*. It is highly desirable that the used software is widely usable by the community while ensuring the aforementioned principles. This can be achieved by using portable programming languages across platforms which do not pose complex installation requirements. This way, more researchers can access the same planners and definitions of planning tasks so that bugs and errors of any nature can be more easily spotted.

2.3. The International Planning Competition

The International Planning Competition (IPC) was created in 1998 to set a common ground for comparing different planning techniques. Nowadays the IPC is considered a reference source when building a planner and most new planning techniques are evaluated regarding the languages, benchmarks and metrics defined in the previous competitions.

The first IPC was organized in 1998 by Drew McDermott. Since then, participation has increased dramatically over the years and a growing number of tracks has formed, representing the broadening community. The most recent competition, the seventh edition, took place in 2011 and achieved a record number of entrants, almost eight times more than the first competition.

The competition is run by the organizers over a period of several months, with participants submitting their planning systems electronically. Typically, entrants in the competition come from academia, though some industrial colleagues have been involved, and industrial sponsorship has been secured. The results of each edition of the competition are presented in a special session of the International Conference on Automated Planning and Scheduling, ICAPS. The organization of the competition series is overseen by the Competition Liaison, one of the officers of the ICAPS executive council.

While the competition usually declares both a winner and a runner-up per track, an equally important role is to gather data and to disseminate it to all researchers. It has also been useful to promote and review standards in research in Automated Planning.

One of the major contributions of the IPC is the establishment of a common standard language for defining planning problems – the Planning Domain Definition Language (PDDL) [McDermott, 1998, Fox and Long, 2003,Hoffmann and Edelkamp, 2005, Gerevini et al., 2009]. PDDL has been developed and extended throughout the competition series and is a key in allowing fair benchmarking of planners. Today PDDL has five levels with different features, corresponding to different planning formalisms being PDDL 3.1 the last and most complete level. However, despite PDDL3.1 covers the representation needs of the new planning challenges most of the existing planners do not implement them; in fact, the majority of participants of the IPC-2011 implemented the first PDDL level only supporting the STRIPS feature besides typing and the equality predicate. Table 1 shows the different levels of the PDDL language with their set of features.

PDDL Level	**Features**
1.2	:strips, :typing, :equality, :negative-preconditions, :disjunctive-preconditions, :existential-preconditions :universal-preconditions, :quantified-preconditions :conditional-effects
2.1	:fluents :durative-actions
2.2	:derived-predicates :timed-initial-literals
3.0	:preferences :constraints
3.1	:object-fluents, :numeric-fluents, :action-costs
3.1	accumulated quality

Table 1

The levels of the PDDL language and its corresponding sets of features

Another important contribution of the competition is the spread of the plan validation tool VAL [Howey et al., 2004]. VAL was introduced in the third IPC and since then it has allowed reliable validation of the several thousand plans produced by the competitors, as well as helping researchers in the development and debugging of their planners.

Three parts of the competition have formed over the years: the deterministic part, the learning part and the uncertainty part. We will focus on the deterministic and the learning part of IPC 2011 since their evaluations were carried out using the software described in this paper. The uncertainty part used different software since it required an evaluation framework able to cope with stochastic or non-deterministic effects of actions.

The deterministic part of the competition is the longest-running part and evaluates the ability of classical planners to solve problems across a wide range of unseen domains. At IPC 2011, the deterministic part includes the sequential tracks with a track for optimal planners, i.e., planners that are guaranteed to produce solutions with the minimum plan cost, a track for satisficing planners, i.e., planners that attempt to find solutions of low cost but need not guarantee optimality, and a track for satisficing multi-core planners, planners that are able to take advantage of multiple processing units. It also includes a track for temporal planners, which try to find solutions that minimize the plan makespan in a setting with durative actions that may temporally overlap.

The learning part of the competition, first held at IPC 2008 [Fern et al., 2011], evaluates planners able to learn and exploit domain-specific knowledge (DSK). Unlike the deterministic part, it provides participants with the competition domains, so that planners can automatically generate knowledge that improves the performance of their planners in these domains. Planners are then evaluated on unseen problems from the same domains.

The seventh competition followed the successful IPC 2008 and was run in a very similar way. The deterministic and learning parts of IPC 2011 continued with the same representation language, without introducing extensions, as planners still need to catch up with the currently available features of PDDL.

IPC 2011 also maintains the evaluation metrics introduced in IPC 2008, favoring quality and coverage over problem-solving speed. Briefly, each planner is allowed 30 minutes (15 minutes for the learning part), up to 6 GB of RAM and 750 GB of hard disk memory on each planning task. For every solved task, a planner receives a score between 0 and 1, computed as the ratio between the cost of the optimal solution for the task and the cost of the solution reported by the planner. (In the sequential tracks the cost of a plan is the sum of action costs; in the temporal track, it is makespan.) For tracks requiring optimal solutions, this means that the score is 0 for unsolved tasks and 1 for solved tasks, so the scoring mechanism simplifies to counting the number of solved tasks.

For tasks in the satisficing tracks for which optimal solutions are not known, the quality of the best known solution is used instead, including solutions found by the participating planners. Using best known solutions as proxies for optimal solutions in computing the quality is problematic and should be avoided if possible; we discuss this further in Section 3.3.

To compute an overall score for a planner, the scores for each task in the competition are summed, and the winner and runner-up are the planners with the highest aggregate score. Scores are not combined between tracks since each track is an independent competition.

The learning part uses the same metric as the sequential tracks but additionally includes another metric to quantify the benefit that resulted from the learned DSK (see Section 3.5).

3. Evaluation metrics at the International Planning Competition

Since its first edition, the organizers of the IPC have aimed to provide the community with standard mechanisms for evaluating the performance of planning systems. In particular there has been a chase for a single evaluation metric that could be used as a guideline to judge which is the best current planner.

After seven editions of the competition this chase has proved to be a difficult challenge. The difficulty lies in the nature of planning, which lends itself to a number of very diverse and often conflicting evaluation criteria, from straight-forward binary success measures (finding a solution/proving that there is none) to optimization along many different dimensions. In addition, an evaluation metric for a planner has to measure the domain-independence of planners, quantifying their versatility for different classes of problems across different domains. Accordingly, the performance of planners can be analyzed from different views such as the number of problems solved, the CPU time used, the length of the plans, their cost, makespan or further features of plans like diversity or flexibility.

Because of this wide variety of possible objectives, the selection of the best planner necessarily depends on the context in which it will be used. Despite this difficulty the IPC series has kept pushing for the development of mechanisms that allow fairer and more significant planning evaluations. This section reviews the evaluation metrics used throughout the IPCs and analyzes the decisions taken for the design of the software for evaluating planners at IPC 2008 and IPC 2011.

3.1. Historical perspective

Throughout the competition series different evaluation schemes have been proposed to score and rank planners. The first edition of the competition in 1998 [Mcdermott, 2000] attempted to define a single scoring function to evaluate the planners. The proposed function regarded three criteria: number of problems solved, time used and length of the solutions found. However, the way in which these features were combined by the scoring function was found lacking, as it diverged significantly from the expectations of organizers and participants. In the end the scoring function was abandoned and winners were selected by a judgment call of the competition organizer.

The second edition of the competition, IPC 2000, [Bacchus, 2001] followed the same approach of selecting winners by a judgment call taking into account number of problems solved, runtime, plan lengths, but also scientific novelty of the planning approach.

IPC 2002 [Long and Fox, 2003a] for the first time allowed specifying optimization metrics for each planning problem, so solutions could be compared by domain-adequate metric values rather than just by looking at the number of actions. However, there was still no clearly defined way of how to combine the metric values for different planning problems or how to trade them off against runtime and number of solved problems. Because of these complications, the winners were again selected by a judgment call of the competition organizers. The post-competition analysis of IPC 2002 also introduced statistical methods for quantifying if intuitive interpretations of the raw results were adequately supported by the data. This statistical analysis resulted in partial orders which showed which planners dominated which others with regard to a specific level of significance in terms of the number of problems solved.

IPC 2004 [Hoffmann and Edelkamp, 2005] for the first time separated the evaluations of optimal and satisficing planners (where optimal planners could choose between two different optimization metrics, sequential or parallel plan length) and introduced a new way to rank planners by assigning 1^{st} or 2^{nd} *position* tags to the groups of planners that scaled best (and roughly similarly within each of the two groups) in a given planning domain. Winners of IPC 2004 were then selected according to the number of 1^{st} and 2^{nd} positions achieved by the planners. The decision of how exactly to allocate 1^{st} and 2^{nd} places were again made as a judgment call of the competition organizers.

IPC 2006 [Gerevini et al., 2009] used essentially the same evaluation criteria as IPC 2004 based on 1^{st} and 2^{nd} positions per domain but gave more attention to plan quality. In addition IPC 2006 introduced new mechanisms to express user requirements about the quality of plans such as *soft goals* and *state trajectory constraints*. The IPC-2006 also postulated an additional requirement that a planner could not be declared a winner in a certain track unless it improved over the previous state of the art, i.e., over the performance of the 2004 winners. Post-competition, the organizers performed similar statistical analyses as the IPC 2002 organizers with a strong emphasis on number of problems solved.

In all planning competitions before IPC 2008, the decision how the winners would be determined was made by the organizers only after the competition was run. Participants were informed ahead of time that number of problems solved, runtime and plan length would be important considerations, but evaluation details such as the "$1^{st}/2^{nd}$ rank by domain" procedure, the tradeoff between different quality measures, or (in the case of IPC 2006) the requirement to outperform the 2004 winners were not communicated in advance.

There was also no clear division into tracks apart from the separation starting in 2004 between satisficing and optimal planners: a planner could be judged a winner if it performed particularly well for one class of planning problems (such as sequential planning problems), if it covered a wide range of different planning problems (e.g., sequential ones, temporal ones and ones involving soft constraints), or if it offered a good tradeoff between planner performance and support of rich domain models.

IPC 2008 was the first international planning competition to introduce a clear separation into different tracks where each track used a single scoring function, defined in advance, to evaluate the performance of planners. The same evaluation scheme was used for IPC 2011. In this evaluation scheme, each planner is given 30 minutes to solve each problem and receives a score in the range 0–1 for each solved problem depending only on the quality of the found solution, without regard to runtime. Apparently this new evaluation scheme puts less emphasis on the number of problems solved and the time used for generating the solutions, although number of problems solved of course remains important because unsolved problems are scored as 0. Arguably, runtime remains relevant to some extent since planners able to quickly find solutions can dedicate more time to improve the quality of the initially produced plans.

All through the years, this evolution of evaluation schemes has been accompanied by fruitful and sometimes heated discussions about the most suitable way to evaluate planner performance. One major aspect of this discussion has been a continuous tension between researchers in favor of pushing the expressiveness of planners, aiming at bringing planning systems closer to the expressiveness requirements of many realistic problems, and researchers in favor of working with more limited planning models, promoting increased performance of planners over expressiveness. In some cases, the latter group has argued that classical planning techniques can also be capable of efficiently dealing with more expressive problems through compilations from richer planning models (e. g., see [Keyder and Geffner, 2009, Bonet and Geffner, 2011]).

This tension between expressiveness and efficiency is still present nowadays. As an example, the last version of the description language PDDL3.1 covers many functionalities for representing temporal or numeric aspects of real problems, or for representing much more general kinds of conditions and effects than the precondition, add and delete lists introduced in Section 2 of this paper. However, most of the planners competing at the last IPC only implement very limited subsets of these features, rarely going significantly beyond the "core" classical planning subset of PDDL defined for IPC 2000.

3.2. *Evaluation of optimal planners*

The currently used metric at the IPC for the evaluation of optimal planners is *coverage*, i.e., the number of problems optimally solved with respect to a fixed set of benchmark problems and fixed time and memory settings. Comparing the quality of the plans generated by optimal planners is fruitless because they necessarily have to generate optimal solutions.

The main difficulty when using coverage for evaluating optimal planners comes from the fact that obviously, there is no domain-independent solver that can be used to check the optimality of plans. The organizers of IPC 2008 developed several domain-specific solvers to validate the optimality of the solutions provided by participants, although they did not achieve complete coverage. All optimal planners evaluated at the time had at least one bug at some point that made them produce suboptimal plans sometimes, which is often much easier than proving optimality. At IPC 2011, no optimal domain-specific solvers were developed for any domain. However, the competition results showed no cases where two planners disagreed on the quality of optimal solutions reported for a given planning problem, giving some confidence that the reported solutions were indeed optimal.

A general observation that also applies to other competition tracks is that the coverage metric weighs all problems equally, so it can only fairly evaluate planners when the selected benchmark problems are unbiased with respect to the evaluated planners. For example, if the benchmark is (perhaps unintentionally) biased towards a specific planning paradigm it could favor one kind of planner over others. To investigate this issue more deeply, there has been research on creating random generators of synthetic unbiased problems for restricted planning models [Rintanen, 2004], but further research has to be done for the general case. All in all, there is precious little research on the generation of problems for planning evaluation, considering its dramatic influence on the results of the evaluation [Howe and Dahlman, 2002].

Some related competitions use more structured mechanisms to select test problems. For example, the organizers of the SAT competition select problem instances from a public pool of problems and distinguish different tracks for *random*, *industrial* and *handcrafted* problem instances. The organizers of the CSP competition MINIZINC CHALLENGE only select problems that are solvable by at least one of the participants in a sensible time frame. Finally, the organizers of the Answer Set Programming Competition aim to provide a balance between problem categories such as *search*, *query* and *optimization* and covering different computational complexity classes such as *polynomial*, *NP* and *Beyond NP*.

3.3. *Evaluation of satisficing planners*

Unlike optimal planners, satisficing planners can produce valid solutions of diverse quality. Within the planning community, it is a largely (though not universally) supported notion that plan quality is an important consideration in most application domains. If we follow this argument, coverage – which only considers the number of problems solved without solution quality considerations – is an inadequate evaluation metric.

Apart from coverage, the CPU time invested by satisficing planners to generate their solutions has also been extensively used to evaluate their performance. The minimization of CPU time is a desirable feature for many planning applications that require fast response, such as controlling the actions of an autonomous physical system. However, emphasizing the role of CPU time in the evaluation of satisficing planners at the IPC is not without problems:

– Making CPU time a primary consideration can push competitors to implement low-level optimizations. Currently there are many aspects in most planners that are not time-critical. Because these aspects are typically completed in few seconds, they do not have to be heavily optimized. With runtime factoring heavily into the scoring function, every aspect would become critical for the score.
  While this could be beneficial for improving the engineering quality of planning systems, it can also dramatically increase the barrier of entry for the competition, especially for novel planning approaches that cannot readily reuse heavily optimized components of existing planners. In the context of a scientific competition, this has the undesirable effect of hampering the exploration of truly novel ideas.
– Making a planner substantially faster does not necessarily imply making it capable of solving substantially harder problems. Results of previous IPCs show that making a planner twice as fast hardly increases the number of problems solved at all, since there are few problems that can be solved with a 60-minute timeout, but not a 30-minute timeout. Hence, if the ultimate objective is to scale planners to harder problems, the difference between solving a planning problem in, say, 10 seconds vs. 20 seconds is less significant than it might intuitively appear.

For these reasons, the evaluation scheme introduced for IPC 2008 and reused for IPC 2011 emphasizes plan quality over other considerations. In detail, this scheme fixes the runtime per problem instance for all competitors and only considers the cost of the solutions found. An optimally solved problem contributes a score of 1 (as in the optimal planning tracks described above), while suboptimal plans contribute lower scores scaled inversely to the cost of the solution. For example, a solution with four times the optimal cost would lead to a score of 0.25. The overall score is obtained by summing the scores for all solved problems.

Even though this score is computed regarding only the quality of the solutions, coverage and runtime are implicit to some extent, as unsolved problems do not contribute to the score, and slow planners are less likely to meet the runtime bound on difficult problem instances, leading to more unsolved problems. Also, fast planners can rely on anytime planning strategies to improve the quality of the generated plans.

An important question for this quality-based scoring mechanism is how to compute the score when the optimal solution quality is not known. A simple remedy, which was used at IPC 2008 and 2011, is to compare the quality of the generated solutions to the *best known* solution quality rather than the *optimal* quality. A subtle issue here is that the best known solution could be a solution found by one of the competition participants. From a fair evaluation perspective, this is problematic, since it can violate the decision-theoretic axiom of *independence of irrelevant alternatives*.

Assume that the outcome of a planning competition is such that P_1 wins by narrowly outscoring P_2. Assume further that the same competition is repeated, but this time with an additional planner P_3 which is clearly worse overall than P_1 or P_2, but improves over the best known solutions in a few problems. This can affect the balance between P_1 and P_2 in such a way that P_2 becomes the winner. In other words, adding the additional participant P_3 influences which of P_1 and P_2 is considered the better planner.

This kind of situation is highly undesirable since it can introduce strategic considerations into the competition that run counter to its scientific goals. For example, a team of researchers might refrain from entering planner P into the competition because it might adversely affect the performance of another planner Q by the same team of researchers.

A good way to avoid or reduce such quandaries is to use strong domain-dependent solvers to find high-quality (or even optimal) reference plans, so that the best known solutions will not be ones (uniquely) found by the participating planners. Many such specially developed domain-dependent solvers, and in some cases human-generated solutions, were used at IPC 2008 for this reason.

3.4. *Evaluation of temporal planners*

In temporal planning, *makespan* (the temporal duration from the the start of the first action execution to the end of the last action execution) is generally accepted as a useful quality criterion for plans. Hence, the scoring function used at IPC 2008 and 2011 for temporal planning is the same as for the sequential planning tracks, except that it considers makespan instead of total action cost as the quality measure of a plan.

(Consequently, there is no notion of action cost in the temporal planning tracks.)

VAL is also able to validate the correctness of satisficing temporal plans. Like with sequential optimal planners, there is no domain-independent validator for the minimality of makespan of temporal plans. This has not been an issue at recent IPCs because no evaluation of optimal temporal planning has taken place due to the lack of participating planners.

When evaluating temporal satisficing planners, the choice of evaluation domains is of particular importance. In recent years, the notion of planning domains with *required concurrency* – temporal planning domains in which every feasible plan must involve overlapping actions – has attracted significant attention [Cushing et al., 2007]. Perhaps surprisingly, the overwhelming majority of existing temporal PDDL planning domains do *not* exhibit required concurrency, even though concurrency is usually required to come up with solutions of low makespan. Maybe as a consequence of this, most existing temporal planning systems are not *temporally expressive*, i.e., are not complete for problems with required concurrency.

This state of affairs means that it is hard to find a benchmark set that is not biased towards particular temporal planning approaches. Certainly, a benchmark set with required concurrency is biased towards temporally expressive planners. Also, a benchmark set *without* required concurrency can be considered as being biased *against* temporally expressive planners, since temporal expressiveness often comes at a cost in performance compared to simpler planners that cannot handle required concurrency. Since entering a planning competition involves a lot of effort, it thus appears advisable to clearly communicate before the competition if and to what extent required concurrency will feature in the evaluation domains, in order to avoid disappointed expectations.

3.5. Evaluation of planners exploiting domain-specific knowledge

When looking at the performance of planners able to benefit from domain-specific knowledge (DSK) two aspects can be evaluated:

- The final performance of planners exploiting the DSK.
  It is common knowledge that planners using hand-coded DSK can perform many orders of magnitude better than ones that do not in most planning domains. After all, at the extreme end of the spectrum, a hand-written domain-specific planner can be seen as an instance of a (very) general planner with "domain-specific knowledge" (its source code), and indeed the most successful domain-specific planners are closer to programming languages than to their domain-independent counterparts.
  Therefore, it is not useful to run domain-specific and domain-independent planners in the same competition. Instead, IPC 2000 and IPC 2002 have featured special competitions for planners using DSK ("hand-tailored planners"). However, these competitions have not been repeated as part of the IPC since 2002, partly due to the concern that it is not clear how to measure the quality of the *planner* as opposed to the abilities of the person providing the DSK.
  However, DSK has reemerged in the International Planning Competitions with the advent of the learning part of the IPC since 2008. Here, the final performance of planners exploiting the DSK is the primary consideration for deciding the winner of the competition. In difference to the hand-tailored planners of IPC 2000 and 2002, here the DSK has to be found *automatically* by the planner.
- The benefit of exploiting DSK.
  The learning part of the International Planning Competition was created in 2008 to evaluate the performance of planners that are able to automatically acquire and exploit DSK. In addition to looking at the final performance with the learned DSK, the track also tries to identify the planners that *benefited* the most from the learned DSK, by comparing their performance with and without it. There is, however, a clear problem with this issue. Participants could simply submit a planner that performs deliberately poorly without DSK. The learning track of the IPC 2011 introduced a new mechanism for ranking planners based on *Pareto* domination. The planner with the greatest benefit from using the DSK would be the planner that was *Pareto* dominated by the fewest number of planners in terms of two dimensions: (1) the performance metric under consideration, for example plan quality; and (2) the improvement in performance due to the DSK.

4. The software of the Seventh International Planning Competition

The software of the Seventh International Planning Competition was created with all the previous scientific considerations in mind. On the other hand, the organization and automation of the whole process for evaluating planners in an automated fashion poses additional engineering challenges. Specifically, the purpose of the software of the Seventh International Planning Competition is twofold:

1. *Automated evaluation of planners*. It provides specific means to automate the whole process running a particular set of planners on a particular collection of domains. Finally, it also eases the task of automatically validating all the results of the experiments. It has been designed mainly to address all the engineering challenges that result from automating an arbitrary number of experiments.
2. *Analysis of results*. The software provides specific means to analyze the performance of the planners with a myriad of variables to inspect without the need of repeating experiments over and over. It provides both a wide range of metrics and different nonparametric and parametric statistical tests. It has been devised fundamentally to deal with all the scientific considerations presented in the previous sections.

Note that this software can be used to reproduce the experiments of the IPC 2011 and that it provides tools for carrying out further analysis of the reported results. On the other hand, the software can be used to run and analyze the results of private experiments over planners and problem sets different from the ones used at the last International Planning Competition.

The idea of automating the tasks necessary for evaluating planners was fully considered in the Sixth International Planning Competition. Indeed, a specific design was devised with this goal in mind. The software of the Sixth International Planning Competition was built around three different components: a wiki page for exchanging information among the organizers and with the competitors; a svn repository used, mainly, to store the source code of all entries in the different tracks and the problems selected for the competition as well as the results of the executions; thirdly, the computer premises. The software built at that time had interfaces with all these components though its main contribution was a number of scripts used to automate the whole experimentation: building the test sets, compiling the planners from scratch and executing the planners on a number of selected planning tasks. On the other hand, there were tools to automatically retrieve the results which were immediately validated with VAL [Howey et al., 2004] and even to generate informative pdf documents with an overview of the overall performance per track. While a significant body of work was general, some parts were specific to the particular arrangements made at the IPC 2008. Finally, the software was publicly available on demand.

During the organization of the Seventh International Planning Competition it was decided to reuse as much code as possible from the previous edition. The main design was mimicked and some relevant parts of the scripts were included in the new software. In particular, the new software enhanced the interface with the svn repository, whereas it offers no support to interface with the wiki page or the computer facilities. Additionally a brand new package was built to inspect the results of the experiments, to generate different sorts of reports in a wide variety of formats and even to automate statistical tests. In the end, all the software has been released for public use under the terms of the GNU General Public License version 3.

In this section, a gentle introduction to the software developed during the last IPC is offered, both for automating the execution of a selection of planners on a particular number of planning tasks, and also to inspect and generate reports with the results of the experiments. For a detailed discussion of the capabilities of software the interested reader is referred to the technical documentation [5].

4.1. The SVN repository

The software is based on the use of an (either public or private) svn repository that stores all the necessary information for running a number of experiments. As discussed in the Introduction, while performing experiments a large collection of planners and problems easily proliferate. Thus, a lot of results are easily generated and cataloging all this information while making it easily accessible is not trivial. The svn repository specifically addresses this issue. Public access is granted to the svn repository used at the IPC 2011[6]. It contains all the entrants of the Seventh International

[5]See `http://www.plg.inf.uc3m.es/sw-ipc2011/`

[6]`svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data`

Planning Competition along with all the domains selected and the obtained results.

One of the main drivers of the software was the requirement to work with any svn repository. This way, any researcher can create her own private repositories so that private experiments can be conducted. For a detailed discussion on this issue see the second practical case discussed in the technical documentation[7].

4.2. *Overview of the software*

The latest version of the software is also available through the official svn repository of the IPC 2011[8]. The software is organized into two main packages: `IPCData` with the services for evaluating planners over a collection of domains and problems and validating the results; `IPCReport` with the services for analyzing the results and generating automatically reports in a variety of formats. The following subsections provide a shallow description of the main services provided by both packages. Figure 5 illustrates the working flow of all the scripts involved in the processes of the automated evaluation of planners and the analysis of results with the software of the Seventh International Planning Competition. First, test sets (see Section 4.2.1) and planners (see Section 4.2.2) are automatically retrieved from the svn repository and their contents properly arranged in preparation to automatically run the experiments (see Section 4.2.3). The results are stored in a *results tree directory* in the local computer which can be automatically validated while generating additional information – see Section 4.2.4. This local structure can be converted into a more compact representation known as *snapshot*. This is indeed a more efficient method to inspect the results (see Section 4.2.5), rank planners (see Section 4.2.6) or perform statistical tests – see Section 4.2.7.

All the scripts discussed in this section acknowledge a help directive that provides on-line assistance. Additionally, a very detailed level of information (also known as *verbose*) is available upon request. This ability is specifically important since some computations can be very lengthy (especially in the scripts included in the first package) so that verbose output keeps the user updated more often. For the same reason, an additional number of services have been included in these scripts, mainly: logging services and automated e-mail notifications.

Logging services can be easily enabled by specifying the name of a log file and a logging level so that only messages of the given level or above are shown. While all services described here are readily available upon installation of the software, automated e-mail requires some additional configuration. Thanks to this feature, the user is automatically notified upon completion of a particular task with an e-mail that contains some useful information along with a copy of the log file generated so far.

4.2.1. *Building test sets*

With the aim of guaranteeing that all planners are evaluated over the same benchmarks, the script `builddomain.py` automatically builds the test sets from the information stored in the svn repository. The script `builddomain.py`, located in the package `IPCData`, checks out all the problem files of a given domain which is qualified by its name and the track/subtrack it belongs to. This script acknowledges the use of regular expressions so that an arbitrary number of domains can be built at the same time. Each test set is built in a separate directory in the local computer.

While domain and problem files can be given any name in the svn repository, they are consistently renamed in each test set as `domain.pddl` and `problem.pddl` when built in the local computer. On the other hand, the ability of this script to generate the test sets in a particular order allows the designer to post problems according to some criteria such as expected difficulty to solve them.

4.2.2. *Building planners*

In order to guarantee the completion of experiments with the same versions of the planners, the `buildplanner.py` script automatically checks out the source code of the selected planners from the svn repository and starts the compilation process from scratch. The `buildplanner.py` script, located in the `IPCData` package, records a verbatim copy of the compilation process to make sure that the building process went fine and that no errors were issued on the way. For example, some planners can pose special requirements in the form of third-party software – e.g., the LPRPGP planner uses ILOG CPlex. Recording the building process is mandatory to ensure that the code has been properly compiled and linked. On the other hand, if anything ever went wrong a whole trace is reported in these files.

To provide compatibility, the author of each planner submitted to the IPC 2011 was requested to provide

[7]See `http://www.plg.inf.uc3m.es/ipc2011-deterministic/FrontPage/Software`

[8]See `svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data/scripts/pycentral`

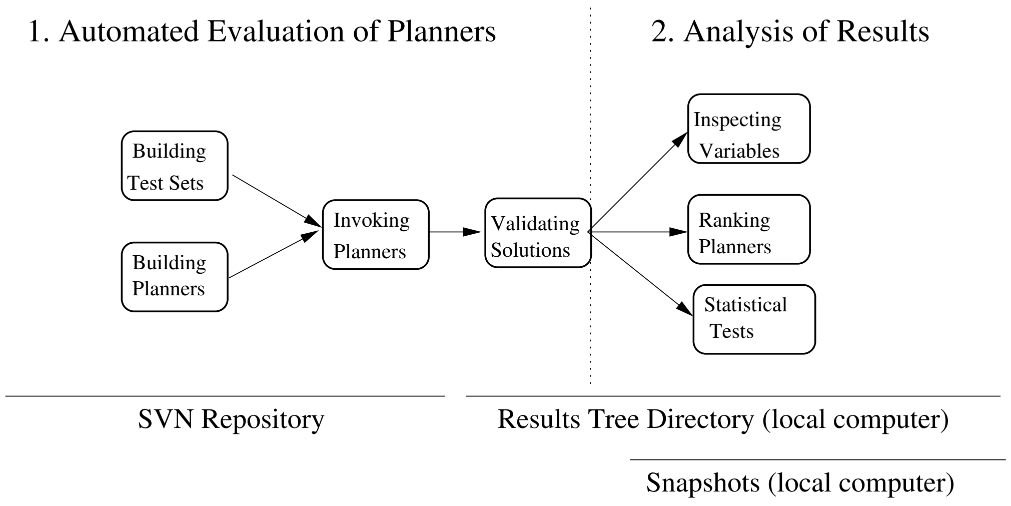

Fig. 5. Overview of the processes for evaluating planners and analyzing results with the software of the Seventh International Planning Competition

a shell script which fully automates the building process. This way, the only responsibility of this software is just to invoke it after checking it out and to record the whole building process in a log file as discussed. The resulting files are located in a directory provided by the user. As with the previous script, this one also acknowledges the use of regular expressions to allow building an arbitrary collection of planners, all in sequence.

4.2.3. Invoking planners

The `invokeplanner.py` script guarantees that the executions of planners are carried out within the same experimental setup. This script, distributed also in the package `IPCData`, runs a particular experiment which involves an arbitrary selection of planners, domains and problems by first, invoking the services of the two previous scripts and then automatically executing every planner with every planning task, all in sequence. The execution can be bounded both in time and memory with arbitrary values measured in seconds and gigabytes, respectively. The limit on memory refers to the maximum area of address space which may be taken by the process including both code and data at any particular instant in time. This is particularly important when running portfolios since the memory limit affects the size and data stored by the portfolio itself but also the size and data used by each planner the portfolio invokes. The time limit refers to CPU time unless the script is invoked on planners and domains of the multicore track. In this case, time is measured as wall clock time. Another important difference is that when being run in multi-core mode, `invokeplanner.py` allows all threads and processes started by the planner to run for the time specified in the time bound so that the overall running time is easily larger; otherwise, the script only allows the planner to run for a time such that the sum of the CPU time of all its threads and processes does not exceed the given bound. If any of these bounds (either time or memory) is exceeded, the script resumes execution by killing the planner and all the subprocesses/threads it launched.

Since every planner has its own arguments, each planner must provide a particular shell script which acknowledges, usually, three parameters: the domain file, the problem file and the prefix to be used in the plan solution files – if an arbitrary number of them can be generated as in the sequential satisficing track, then the given string is suffixed with a number. The script automatically sets-up the necessary parameters to be given to the planner. Since the script `builddomain.py` always generates domain and problem files named `domain.pddl` and `problem.pddl` (see Section 4.2.1) these are the precise names used to invoke this script. The third parameter is always `plan.soln`. An important difference results when the subtrack used is `dck` – which means that the script is running the learning track with domain-specific knowledge, see Section 3.5. In this case, `invokeplanner.py` provides a fourth parameter that has to be accepted by the script provided by the author of the planner: the directory where the domain-specific knowledge resides.

Even if the planners can run in a non-Unix environment, Linux is assumed to be used here. The reason is that this script records, for each execution, a number of additional parameters as they are provided by Linux:

1. The total CPU time the planner is taking for solving the problem.
2. The total memory the planner is taking, also considering the memory taken by all the subprocess and threads launched by the planner.
3. The number of processes currently running.
4. The number of threads currently running.

All of these parameters are sampled at rather regular intervals of five seconds. However, the following are recorded after the whole execution is over or after a particular event (such as finding a solution) takes place:

5. The number of solutions found, if any and zero otherwise.
6. The time elapsed since epoch (i.e., since the planner was started) when each solution was found.
7. The overall running time (in clock wall time) consumed by the planner.
8. The maximum memory consumed by the planner during the whole execution.
9. The memory the planner was taking just before voluntarily resuming the execution or being killed. Note that this is not necessarily equal to the maximum memory though this is usually the case. While most planners take memory incrementally, some others take and release memory during execution – e.g., portfolios equipped with a number of planners each with its own memory profile, see Figure 19 in page 22.

On the other hand, the script also records additional information for each execution about the CPU being used, the operating system and the memory available. All of these parameters are stored in dedicated log files. In addition, `invokeplanner.py` captures both the standard error and the standard output of the planners and stores them in dedicated log files for diagnoses purposes.

All of the resulting files (the domain and problem definition files, the plan solution files generated, all the log files, and the standard output and the error messages generated by each planner) are stored in a particular tree structure known as the *results tree directory*. The scripts discussed in Sections 4.2.4 and 4.2.5 access these directories.

4.2.4. Validating solutions

The final step in the evaluation process is mandatory not only at the International Planning Competition but also at any experiment run at the home laboratory. The script `validate.py` takes care of this step and it is the last script discussed from the package `IPCData`. This script recursively traverses the solutions in the *results tree directory* and process them with VAL 4.2.09[9]. Its output is recorded in a separate log file. In case of error, the corresponding validation log file shows the appropriate data. The validation process does not only makes sure that *all* the plan solution files generated are correct, it also stores other relevant data such as the final value of the metric used and the *step length* of the plan, or number of actions. These parameters, and others, can be inspected in one way or another with the following scripts.

4.2.5. Inspecting the results

To analyze the experimental results, the script `report.py` allows researchers to process queries against validated results. The `report.py` script uses either the *results tree directory* generated by the script `invokeplanner.py` (see Section 4.2.3) or a particular structure known as *snapshot* – discussed below.

This script, distributed in the `IPCReport` package, acknowledges more than 50 different variables that can be queried either as a singleton or combined. Variables are split into two different categories: *raw* variables and *elaborated* data. The first one refers to all the parameters sampled by both `invokeplanner.py` (see Section 4.2.3) and `validate.py` – see Section 4.2.4; the second one refers to data that results from various manipulations with raw data, such as computing the maximum running time for all problems in a given domain for a particular planner or the total number of problems solved in a particular domain. There are, in total, 33 raw variables and 23 elaborated variables.

Moreover, this script generates the output in a wide variety of formats including: table, html, wiki, octave and excel. The first one is just a convenient way to show the information on the console; the next two ones are markup languages that can be used to show the results either in a web page or a wiki. The octave format is also acknowledged by Gnuplot so that it can be used to generate figures or to process data in some way with Octave. Finally, excel pages are also acknowledged.

[9]While VAL is currently maintained at `http://planning.cis.strath.ac.uk/VAL/`, a version slightly modified for its use with this software is distributed in `http://www.plg.inf.uc3m.es/ipc2011-deterministic/FrontPage/Software`

Additionally, `report.py` can create a summary of the results, also called *snapshot*, which is just a binary file with exactly the same data but which is faster to process than the original results tree directory. Since snapshots are usually orders of magnitude smaller than their counterparts, the *results tree directories*, they are a convenient way to exchange data among researchers. As a matter of fact, *snapshots* are the preferred method to rank planners and perform statistical tests as discussed in the following subsections.

4.2.6. Ranking planners

Of particular interest to the organization of the International Planning Competition, but also in private experiments conducted at the home laboratory, it is to derive the score of all entrants of a given track/subtrack. The scripts `score.py` and `tscore.py` address precisely this requirement in various forms. Both scripts are located in the `IPCReport` package and accept a selection of the planners and domains present in either a *results tree directory* or, alternatively, in a *snapshot* and rank all planners according to a particular metric. Since these scripts are not only thought for organizers of an IPC but also to private researchers as stated above, up to six different metrics are acknowledged:

1. *quality*, this is the official metric of both the Sixth and Seventh IPCs. It computes for each problem a score which equals $\frac{C^*}{C}$ where C^* is the cost of the best known plan for a particular problem and C stands for the cost of the plan produced by the planner being evaluated[10]. If the planner found no solution the quality score is set to zero.
2. *coverage*, this score function awards one point to every planner that solves the current task and zero otherwise. This is the function used to rank planners in the sequential optimal track.
3. $time_0$, it computes the score of a planner for a given task as the quotient $\frac{T^*}{T}$ where T^* is the minimum time required by any planner to solve the task, and T is the time spent by the evaluated planner to solve the same planning task. All times below 1 second are considered to be exactly equal to 1 second to reduce emphasis on micro-level optimizations of any part of the planner. This was the official metric used to evaluate time performance at the learning track of the Sixth International Planning Competition.
4. $time_1$, this score function computes the score of a planner for a given task as the quotient $\frac{1}{1+\log\left(\frac{T}{T^*}\right)}$ where T^* is the minimum time required by any planner to solve the task and T is the time it took this particular planner to solve the same planning task. Since differences in time are normally greater than in quality, logarithms are widely used for scaling time scores properly. Again, all times below 1 second are considered to be exactly equal to 1. This is the official metric to evaluate time performance at the learning track of the Seventh International Planning Competition.
5. $time_2$, this score function computes the score of a planner for a given task as the quotient $\frac{\log(1+T^*)}{\log(1+T)}$ where T^* is the minimum time required by any planner to solve the same task and T is the time it took this particular planner to solve the same task. It is provided just as an alternative measure of time and it has never been officially used.
6. *qt*, computes for each planner and task a tuple (Q, T) where Q stands for the quality of the best solution found by the same planner and T is the time (in seconds) it took for the planner to find it. Next, it awards each planner with a score that equals the number of tuples it pareto-dominates[11] for the same task. The main advantage of this metric is that it compares the performance of a planner with the performance of every other planner one by one instead of using the performance of the best one as reference.

Of course, there might be many other metrics and, as a matter of fact, an alternative measure previously suggested for ranking planners according to CPU time consists of assigning a score of 1.0 to a runtime of 1 second and a score of 0.0 for a runtime of 1800 seconds (or, alternatively, to the maximum allotted time), and interpolate logarithmically in between these two (e. g., [**?**]). This results in an *absolute time score* since the score of a planner does not depend on the score of other planners. However, this metric has not been implemented in the software described in this paper.

These metrics are computed per planner and planning task and then summed up for all planners across

[10]This assumes that plan quality can be measured in terms of a cost to minimize, such as total sum of action costs for classical sequential planning or makespan for temporal planning. For the net-benefit planning track of the Sixth International Planning Competition, the formula was changed to $\frac{Q}{Q^*}$, where Q is the net benefit of the generated plan and Q^* is the net benefit of the best known plan.

[11](Q, T) is said to pareto-dominate (Q', T') if and only if $Q \leq Q'$ and $T \leq T'$.

the same track/subtrack. Next, all planners are ranked in decreasing order of score, the first one being the winner. Both scripts generate rank tables for all domains selected and an additional one with all the accumulated results. As it turned out with the script `report.py`, these scripts can generate these tables in the same formats. Additionally, `score.py` can generate the same tables in LaTeX which, when being processed, provide the same information both textually and using color codes which are easier to interpret.

On the other hand, `score.py` acknowledges a last directive that explicitly sets up a time bound on the analysis – if not given, the time bound used for running the competition is used instead. When given, all the results generated after the specified threshold are discarded. This is useful for making analysis of the form: "*What if the competition would have been run in 15 minutes instead of 30?*". This ability is exploited by the script `tscore.py` which computes how the score of every selected planner evolves over time for all the planning tasks chosen. It just takes the particular time instants when any planner solved any task and updates the score (for any of the six metrics defined above) of all planners. Since this computation can be very lengthy it is also allowed to request the script to generate the tables for a particular number of time steps instead of all. In the end, the tables show the score of each planner per time step. This is useful for drawing conclusions about the performance of the entrants with regard to the time it takes them to solve problems and to answer questions such as: "*Do planner A always dominate planner B?*" or "*What planner should be chosen if the goal is to solve problems quickly?*"

These questions and others are considered for specific cases in Section 5.

4.2.7. Statistical tests

One of the most important difficulties when evaluating planners with regard to a particular variable – such as time or plan quality (and, in general, when comparing algorithms which are not guaranteed to terminate in a given horizon of time or to find solutions which are bounded by a factor) is that comparisons become harder if one algorithm does not consistently dominate the other. While the consideration of metrics as those described in the previous section provide an assessment of the global differences among planners, the significance and confidence of this assessment is unknown. In these cases, several approaches based on statistical tests have been suggested in different areas of Artificial Intelligence including Automated Planning.

The script `test.py` implements four different statistical tests. For a thorough discussion of the statistical tests mentioned here, the interested reader is referred to [Corder and Foreman, 2009]. In all cases, the statistical tests report the p-values or the probability that the observed differences are real and not due to chance. When the p-value is less or equal than a given threshold, usually referred to as the *critical value*, the Null or Research Hypothesis is rejected and the Alternate Hypothesis is accepted instead with regard to the specific critical value chosen. Typical values of the level of risk are $\alpha = 0.05, 0.01$ and 0.001 which stand for a probability of 95%, 99% and 99.9% respectively that any observed statistical difference will be real. Since parametrical statistical tests make questionable assumptions about the distribution of data and, also, most series are more likely to be relatively short (e.g., in the Seventh International Planning Competition there were 20 planning tasks per domain so that most series have $n = 20$ samples which is regarded in some texts as being borderline between a small and large set) three of them are nonparametric:

1. *Mann-Whitney U-test.* It compares two samples that are independent, or not related. It assesses the Alternate Hypothesis that one of two samples of independent observations tends to have larger values than the other. The test automatically corrects for ties and by default uses a continuity correction. The reported p-value is for a one-tailed hypothesis, i. e., when information about whether one sample have larger values than the other is provided. To get the two-tailed p-value (i. e., when the Null Hypothesis is rejected if the test statistic is either too small or too large) the returned p-value has to be multiplied by two.
   This statistical test is the alternative of choice when comparing, for example, the performance of a planner with regard to problems in different domains since they are not necessarily related to each other.
2. *Wilcoxon signed rank test.* In contraposition to the previous test, the Wilcoxon signed rank test is a two-tailed nonparametric statistical procedure for comparing two samples that are paired, or related. It tests the Null Hypothesis that both samples come from the same distribution.
   Typical uses of this statistical test include comparing the performance of two different planners with regard to the same set of planning instances since they are paired. These sort of studies serve to provide statistical evidence that

one planner is superior to the other. As a matter of fact, it has been already used to compare the performances of planners with respect to speed and quality in the analysis of results of the Third [Long and Fox, 2003b] and Fifth International Planning Competitions [Gerevini et al., 2009]. It has been also used in a number of experiments to determine whether a planner subsumes others [**?**].

3. *Binomial test.* It is an exact two-tailed sign test used with dichotomous data – that is, when each individual in the sample is classified in one of two categories such as success or failure, or '$+$' and '$-$'. It provides statistical significance of the Null Hypothesis that both categories are equally likely to occur.
   When comparing the performance of two planners, those cases where one planner performs better than the other are marked with '$+$' and with '$-$' otherwise. The p-value returned is the probability of obtaining as many positive marks as observed according to a Binomial distribution with $p = 0.5$.
   This test was selected by Hoffmann and Nebel [Hoffmann and Nebel, 2001] to provide statistical evidence that their planner, FF, performed significantly better with some collections of enhancements than with others.

Also, because its popularity, a parametric test is provided as well:

4. *T-test for the means of two independent samples.* It is the parametric equivalent test of the Wilcoxon signed rank test. This is a two-tailed test for the Null Hypothesis that two independent samples have identical average (expected) values.

One restriction of all of these tests, however, is that they just compare two series of data. Other tests such as, the Kolmogorov-Smirnov one-sample test to determine if a data sample meets acceptable levels of normality or the Friedmann or the Kruskal-Wallis H-tests to compare three or more samples, (either related or unrelated respectively), are not currently implemented. Instead, all these statistical tests perform pairwise comparisons of an arbitrary number of series and provide the p-value of each pair according to the selected statistical procedure.

The script `test.py` analyzes data retrieved by the script `report.py` – see Section 4.2.5. Therefore, it acknowledges the same parameters to select the series of data to work with. Given that some entries might be empty (so that they are invalid, such as the quality of an unsolved problem), two additional parameters are accepted: `filter` and `matcher`. The first directive can be used to select only those entries that match a particular variable – e. g., whether a particular planning task has been successfully solved or not, variable `oksolved`. The second directive is used to select how to create pairwise associations between two series of data. It currently accepts three different values: `and` requires that both samples have to match the filtering variable (e. g., that both samples correspond to planning tasks successfully solved in the previous example)[12]; `or` rejects all entries where no sample matches the filtering variable (so that only those cases where both entries correspond to unsolved problems are rejected); finally, `all` accepts all pairs in spite of the filtering variable. If either `all` or `or` is specified, it is then possible to give a default value to the entries that do not match the given filter with the directive `noentry`. This is customary practice in those cases where one can assume a particular value, e. g., when comparing the performance on time of two planners it is typical to assign infinitely bad speed to those cases that are not solved in a particular timespan.

5. Practical cases

In this section a number of practical cases are introduced to exemplify the usage of the software of the Seventh International Planning Competition. For a step-by-step discussion of some practical cases, the interested reader is referred to chapter "*Practical Cases*" of the technical documentation. Those cases introduce in detail the following scenarios: how to run a particular planner over a particular selection of domains; how to set-up the necessary environment to run private experiments; thirdly, how to compare the performance of a brand new planner with those that took part in the IPC 2011. The following examples only cover the main features offered by the software developed at the IPC 2011 and we encourage practitioners to refer to the software manual to get a full picture of all possibilities.

[12]These cases are typically known as *double hits*

5.1. Installing and setting up the software

As of the time of writing, current version of this software is 1.3. It has two dependencies with third-party software: python and svn. Both svn and python are available in a plethora of operating systems. Regarding svn, there is no particular requirement and the latest version should work. As for python, the current requirement is using python 2.7.

The first practical case discussed in the technical documentation explains how to download and configure all this software in just ten command line strokes. If svn has been properly installed then it should be feasible to examine the contents of the official svn repository of the IPC 2011, located at `svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data`. This can be easily done either from a graphical user interface (such as `eSVN` in Linux, `svnX` in MacOS or `TortoiseSVN` in Windows) or the command line with "`svn list`".

Figure 6 shows how to download the latest version of the software. It retrieves all modules in packages `IPCData` (see Sections 4.2.1–4.2.4) and `IPCReport` – see Sections 4.2.5–4.2.7. It also downloads all the documentation [13] and a couple of additional packages – created only to meet very specific demands whose discussion goes beyond the scope of this paper.

All the following examples are referred to the official svn repository of the IPC 2011 and the reader is encouraged to try them. Nevertheless, they work exactly the same if other (either public or private) svn repositories are used instead. Since all of the following examples are different than those considered in the technical documentation, the particular commands issued in each case are shown.

5.2. Downloading test sets

The first case of interest consists of downloading all or some of the domain and problem files used at a particular svn. The script `builddomain.py` serves for this purpose – see Section 4.2.1.

Figure 7 shows how to download all the domains selected for the temporal satisficing track of the IPC 2011. It shows on the standard output the current progress. Since no particular directory has been provided by the user, they are all downloaded to the current working directory.

[13]The html version of the documentation is located beneath `doc/build/html` and the pdf file is under `doc/build/pdf`

This is not the case, however, in Figure 8 where only three specific domains are requested to be downloaded – in particular, those requiring concurrency: MATCHCELLAR, TMS and TURNANDOPEN. In this second case all the test sets will be downloaded to the directory `~/tmp` as specified. Also, since a log file has been provided, the script will proceed silently without showing any messages on the console which are redirected instead to a file prefixed with `test` and suffixed with the current local date and time (so that the same prefix can be used several times). Messages of level `INFO` or above are shown. The log message will be left at the directory specified by the user.

5.3. Downloading planners

Likewise, it is also of interest to download and build planners stored at a particular svn. The script `buildplanner.py` (see Section 4.2.2) has been specifically created to meet this requirement. Intentionally, this script works like `builddomain.py` and uses consistently the same set of directives. Therefore, the example shown in Figure 9 should be easy to understand.

One key difference of the `buildplanner.py` script with respect to the previous one is that it accepts the definition of two different log files, instead of only one. The first one records the output of the execution of the script much the same like `builddomain.py`, see Figure 8; the second one records the building process of the planner – i.e., compiling and linking. This second log file is always left at the same directory where the planner is built (`~/tmp/phsff` in the running example).

Another interesting feature shared by most modules is that when a command-line argument accepts a regular expression, another one is provided to specify those cases that should not match the first expression. While regular expressions are powerful enough to produce the same effect, this mechanism results in simpler definitions. Figure 10 shows an example that requires downloading and building all planners whose name starts by '`a`' but does not end in '`n`'. There were three entrants of the sequential multicore competition that meet the first condition: ACOPLAN, ARVANDHERD and AYALSOPLAN. Thus, the second regular expression excludes the first and third planner, leaving only the winner of that track, ARVANDHERD.

```
   > svn co svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data/scripts/pycentral
```

Fig. 6. Downloading the latest version of the software

```
 > ./builddomain.py --track tempo --subtrack sat --domain .*
                  --bookmark svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data
```

Fig. 7. Downloading all the domain and problem files of the temporal satisficing track of the IPC 2011

```
 > ./builddomain.py --track tempo --subtrack sat --domain 'matchcellar|tms|turnandopen'
                  --bookmark svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data
                  --directory ~/tmp
                  --logfile test --level INFO
```

Fig. 8. Downloading only the domains requiring concurrency at the temporal satisficing track of the IPC 2011

```
 > ./buildplanner.py --track seq --subtrack mco --planner phsff
                  --bookmark svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data
                  --directory ~/tmp
```

Fig. 9. Downloading and building the planner PHSFF of the sequential multicore track of the IPC 2011

```
 > ./buildplanner.py --track seq --subtrack mco --planner 'a.*' --exclude-planner '.*n'
                  --bookmark svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data
                  --directory ~/tmp
```

Fig. 10. Downloading and building a particular selection of planners of the sequential multicore track of the IPC 2011

5.4. *Automating the experimentation*

The core part of the package `IPCData` consists of automatically downloading and building test sets for a particular selection of domains in a track/subtrack and also to download and compile a collection of planners. Next, these planners will be automatically faced up with all the planning tasks and the results will be stored at a results tree directory in the local computer.

The arguments available are, intentionally, consistently named after the same command-line parameters used in the preceding two examples. However, `invokeplanner.py` accepts a couple of additional important parameters: `--time` and `--memory` which bound the time execution and memory available per experiment in seconds and Gigabytes respectively.

The example shown in Figure 11 automates all the experimentation performed at the sequential satisficing track in the IPC 2011. While all the planners and domains checked out are stored at the local directory `~/tmp`, the results of all the experiments will be stored at `~/tmp/results`.

5.5. *Validating the results*

Finally, before starting to inspect the results of a particular experiment, the plan solution files generated have to be validated. The script `validate.py` expects just the location of the results tree directory with the plan solution files to validate.

Figure 12 shows how to validate automatically all the results generated with the command shown in

```
> ./invokeplanner.py --track seq --subtrack sat --planner '.*' --domain '.*'
                     --timeout 1800 --memory 6
                     --bookmark svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data
                     --directory ~/tmp
```

Fig. 11. Execution of the whole sequential satisficing track of the IPC 2011

```
> ./validate.py --directory ~/tmp/results
```

Fig. 12. Automated validation of all the results produced by the command in Figure 11

Figure 11. Upon completion, the script leaves additional information in every directory that is accessible through the reporting scripts discussed next.

5.6. *Inspecting the results*

This section offers various examples of the `report.py` script for retrieving the results of the experimentation. While this script can examine a results tree directory, snapshots are preferred instead. Snapshots can be automatically generated for any results tree directory, however to make it easier to follow the next examples it is recommended just to download the ones generated with the results of the IPC 2011 as shown in Figure 13. In the following it is assumed without loss of generality that these snapshots reside in the same directory where the script `report.py` is located.

As mentioned in Section 4.2.5, this script handles two different types of variables: *raw* and *elaborated*. While raw variables refer to measurements of a particular planning task (such as the time allotted to solve it, `timebound`, or the time a planner spent to solve it or until being killed, `runtime`), elaborated ones refer to the result of some operation performed over raw variables – such as computing either the minimum or maximum runtime, `minruntime` and `maxruntime`, respectively. On the other hand, data is arranged in a hierarchy as follows: track/subtrack, planner, domain and finally, problem. This means, in practice, that raw data can only be reported at the level of problems. However, elaborated data can be computed at any of the other three levels. To distinguish among these when using elaborated variables, the directive `--level` can be used to set the desired level of reporting. To illustrate how elaborated variables can be used, Figure 14 shows how to report the number of problems for which each planner reported at least one plan solution file (variable `numsolved`) and the number of them which are effectively validated by VAL, variable `oknumsolved`, in the sequential satisficing track – to assist private experimentation, the variables `plansoln` and `okplansoln` provide the names of the plan solution files generated and those that are casted as being valid, respectively. Note that the report has been instructed to show the results in descending order of the second variable so that the first planner is the one solving more problems. In general, it is possible to command the script to sort the output either in increasing or decreasing order of the value of any combination of variables.

The result of the command issued in Figure 14 is shown in Figure 15. As it can be seen, it is not so uncommon for some planners to generate invalid plans[14]. This clearly justifies the need to use an automated validator such as VAL. As a matter of fact, the disparity in the number of problems that are claimed to be solved and those that are effectively validated is as high as 13,6% and can be easily computed just by changing the level of the query in Figure 14 to `track` instead of `planner`. Replacing the snapshot used it is possible to derive the same figures for other tracks/subtracks: while this difference was very close to 0% in the sequential optimal track, it is 16,5% in the sequential multi-core and finally as large as 40,1% in the temporal satisficing track.

The script `report.py` shares with all the others the ability to restrict attention to a combination of planners, domains and/or problems that can be specified with regular expressions. For example, Figure 16 shows how to retrieve the times (in seconds) since the

[14]It should be highlighted that in a number of cases plans were considered invalid just because they did not conform to the syntax recognized by VAL, e.g., Sharaabi, see Figure 15

```
    > svn co svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data/results
```

Fig. 13. Checking out the results of the Seventh International Planning Competition

```
  > ./report.py --summary ./seq-sat.snapshot --variable numsolved oknumsolved
                --level planner --descending oknumsolved
```

Fig. 14. Reporting the number of problems successfully solved in the sequential satisficing track

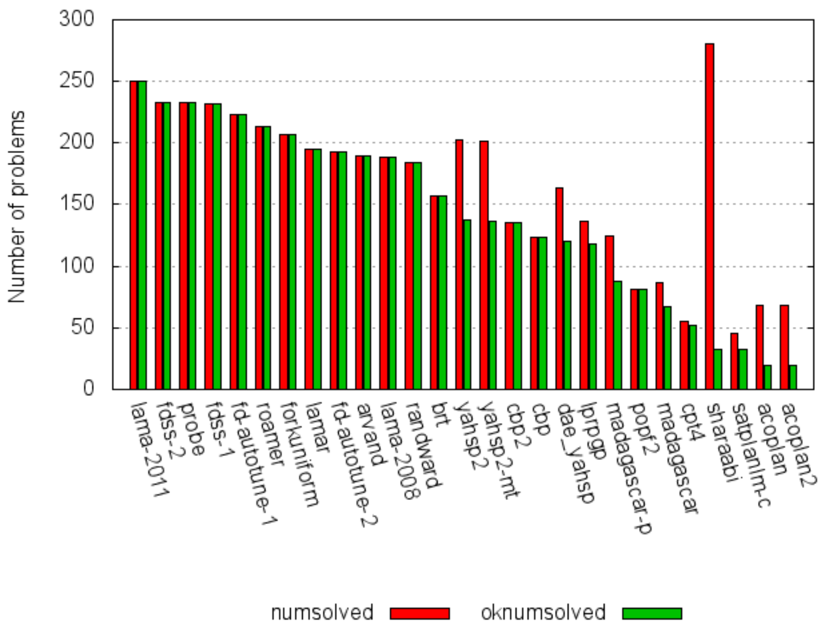

Fig. 15. Number of problems for which each planner of the sequential satisficing track generated at least one plan solution file (numsolved) and the number of them where all the plan solution files generated where valid (oknumsolved)

planner AYALSOPLAN was started when every plan solution file was generated (variable `timesols`), and the value of the metric defined as reported by VAL (variable `values`) for each plan found for problem 010 of the domain OPENSTACKS in the sequential multi-core track. If no combination of planner, domain and problem would have been provided, the script would return the values of these variables for all planners, domains and problems. The ability to fix some (or all) of these levels is very useful to focus on those cases of interest.

In the example shown in Figure 16 the use of the argument `--unroll` is of particular interest. In this case, the variables `timesols` and `values` are arrays that have necessarily the same number of items. Without `unroll`, the script would return just a single line with two arrays in two different columns; however, thanks to this directive it is possible to return pairwise associations of all items in both arrays – where the i-th value of the array `timesols` is attached to the i-th item of the array `values` in the same row. The same query specifically commands the output to be shown in Octave format. Nevertheless, for the output to be directly parseable by Octave it is necessary to suppress the heading that is shown by default by `report.py`. This is achieved with the last directive, `quiet`. In the end, the output can be redirected to a file that can be directly parsed by Octave or, alternatively, by Gnuplot to produce a plot like Figure 17, where the same command shown in Figure 16 was issued for planner AR-

```
> ./report.py --summary ./seq-mco.snapshot --variable timesols values
              --planner ayalsoplan --domain openstacks --problem 010
              --unroll --style octave --quiet
```

Fig. 16. Reporting the time and quality of the plan solution files generated by ayalsoplan in problem 010 of the domain openstacks in the sequential multi-core track

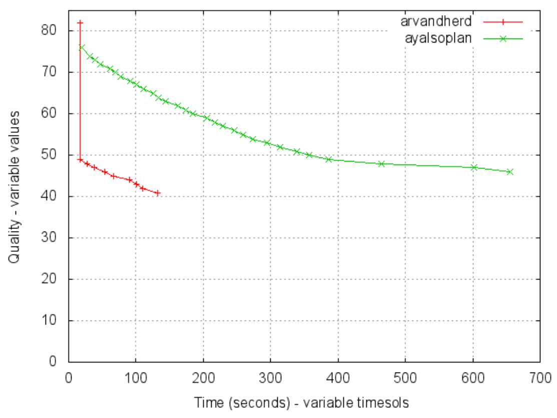

Fig. 17. Time (in seconds) when each solution file was generated and the value of the metric of the plans found by arvandherd and ayalsoplan in problem 010 of the domain openstacks of the sequential multi-core track

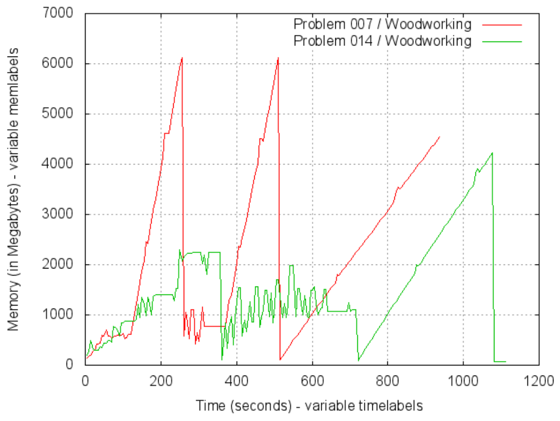

Fig. 19. Memory profile of the portfolio Fast-Downward Stone-Soup 2 in problems 007 and 014 of the domain woodworking of the sequential optimal track

VANDHERD. The resulting plot makes it apparent that, at least in this particular example, ARVANDHERD (the winner of the sequential multi-core track) improved its score faster leading to a better plan than AYALSOPLAN – declared as the runner-up of the same track.

Another utility of the argument `unroll` is to examine the memory profile of any planner. In particular, while most planners take memory incrementally, portfolios are more likely to take it and release it as one planner is killed and the next one is invoked. Figure 18 show how to inspect the memory profile of the portfolio FDSS-2 in problem 007 of the domain WOODWORKING in the sequential optimal track. The variables `timelabels` and `memlabels` are described in items 1 and 2 in Section 4. In short, they report the total CPU time (in seconds) and the total memory (in Megabytes) sampled at regular intervals of 5 seconds. Using `unroll` in Octave format it is possible to plot the memory profile of the portfolio as shown in Figure 19 – where the same query has been issued to draw also the memory profile in problem 014.

In total, `report.py` acknowledges up to 56 different variables. The directive `--variables` list them all along with an explanation of their purpose.

5.7. *Ranking planners*

Undoubtedly, the selection of a metric to compare planners can affect the results in one way or another so that even the best planners according to one particular metric might not be as good when using a different one. In this subsection, it is shown how to rank planners according to different metrics and with different time bounds so that conclusions about their robustness can be drawn. It should be noted however that all entrants of the Seventh International Competition were aware that the official metric to use was *quality* (see Section 4.2.6) and that the time bound was 1800 seconds so that the following conclusions shall be taken with caution: if the adopted metric and time bounds would have been different, a good number of planners would have probably followed a different strategy.

In the first practical case, the script `score.py` is invoked as shown in Figure 20 to compute the score of all planners in the temporal satisficing track. The output shows first, the ranking per domain detailing the score obtained by each planner at every problem. Finally, it shows an overall table with the score of all planners in all domains where planners are sorted in descending order of their score. The same command was exe-

```
    > ./report.py --summary ./seq-opt.snapshot --variable timelabels memlabels
                  --planner fdss-2 --domain woodworking --problem 007
                  --unroll --style octave --quiet
```

Fig. 18. Inspecting the memory profile of the portfolio Fast-Downward Stone-Soup 2 in problem 007 of the domain woodworking of the sequential optimal track

```
    > ./score.py --summary ./tempo-sat.snapshot --metric quality
```

Fig. 20. Ranking the planners of the temporal satisficing track according to the quality metric

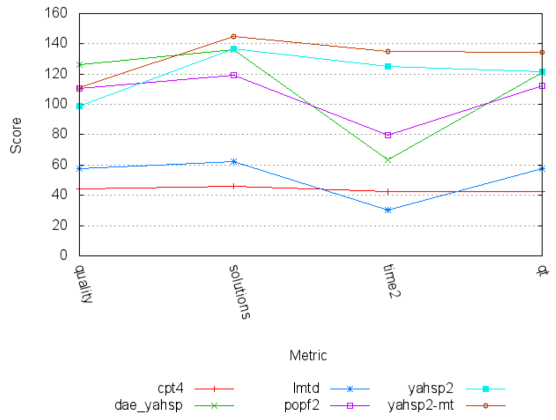

Fig. 21. Differences in ranking of all planners of the temporal satisficing track (but sharaabi and tlp-gp) according to four different metrics: *quality*, *solutions*, $time_2$ and *qt*.

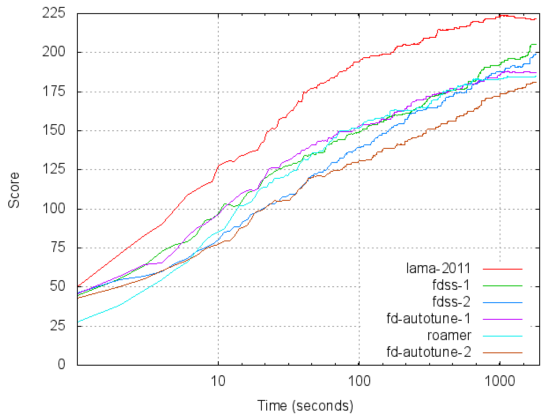

Fig. 23. Evolution of the metric *quality* over time for the first six planners of the sequential satisficing track of the Seventh International Planning Competition

cuted to consider also the following metrics discussed in Section 4.2.6: `solutions`, `time2` and `qt`. Figure 21 shows the result of the differences in score for all planners (but SHARAABI and TLP-GP which solved no problem) with the four metrics considered.

From the preceding figure, a number of conclusions can be drawn. First, all planners seem to be split into two groups: on one hand, DAE_YAHSP, YAHSP2, YAHSP2-MT and POPF2; on the other hand, CPT4 and LMTD. The last two planners perform worse across all metrics. Second, it is clear that the planner DAE_YAHSP was the most benefited from the selection of the metric since that is the only case where it ranks first. On the other hand, YAHSP2 is the worst affected by the selection of the metric *quality* for the Seventh International Planning Competition since in all the others it performs better ranking always second instead of fourth.

In the second practical case, only the metric *quality* is considered but `tscore.py` is used to compute how it evolves over time up to the time bound used in the IPC 2011, 1800 seconds. For the sake of clarity only the first six planners of the sequential satisficing track were considered. Figure 23 shows the evolution of the metric *quality* over time (in log scale) for the planners LAMA-2011, FDSS-1, FDSS-2, FD-AUTOTUNE-1, FD-AUTOTUNE-2 and ROAMER. Note that the score does not necessarily grow monotonically. The reason is that since the scores of all planners are computed again every time a planner finds a solution, the score of all the others decrease if it improved the quality of the best solution found so far. The figure was generated with the data computed with the command shown in Figure 22. The metric *quality* is assumed by default since this was the official metric at the IPC 2011. Likewise, note also the usage of regular expressions to specify a particular subset of planners.

From the previous picture, it is easy to see that LAMA-2011 dominated all the others from the very beginning so that it is not affected at all (for this particular metric) by the time horizon chosen and would

```
> ./tscore.py --summary ./seq-sat.snapshot --planner 'lama-2011|fdss|fd-autotune|roamer'
             --style octave --quiet
```

Fig. 22. Using `tscore.py` to compute how the metric *quality* evolves over time (in log scale) for a selected subset of planners of the sequential satisficing track

have been declared the winner the same for any time span – up to 1800 seconds. In contraposition to this observation, ROAMER and FD-AUTOTUNE-1 are affected by the time horizon chosen: ROAMER progresses faster than the other planners and it is observed that its performance worsens only at the end of the time horizon; on the other hand, FD-AUTOTUNE-1 seems to be as good or better than FDSS-1 and FDSS-2 but at the end where it is surpassed by both variants of FDSS.

5.8. Statistical analysis

All of the previous considerations referred just to a particular collection of metrics but no additional information about whether the differences are meaningful or not is provided. The script `test.py` statistically compares an arbitrary number of series with regard to any variable recognized by `report.py`.

For example, the sequential satisficing planners CBP and CBP2 are two variants of the same idea that performed much alike in most domains. However, there were some differences in a few ones when comparing them with the metric *quality* – see Section 4.2.6. For example, in the domain OPENSTACKS, CBP scored 11.30 points and CBP2 was awarded 13.29 points. On the other hand, in the domain TRANSPORT, the scores were 5.73 and 12.16 respectively. While the differences are apparently prominent (especially in the second domain) it would be good to provide statistical significance of them.

Figure 24 shows how to command `test.py` to retrieve the best quality of all the problems solved by either CBP, CBP2 or YAHSP2 (variable `lowervalue`) and to apply a Wilcoxon signed-rank test on the resulting series. The planner YAHSP2 has been added because it scored right in between a total of 10.92 points and also to show the output of the script when more than two series are created. It is worth noting that thanks to the directive `filter`, only those entries that refer to problems validated (variable `oksolved`) are considered. The matching rule `or` specifically commands `test.py` to discard those entries where both planners did not meet the filtering variable so that only those planning tasks that were solved by at least one planner are taken into account. This raises the question how to deal with those cases where one planner did not solve a particular planning task. The directive `noentry` instructs `test.py` to assign and arbitrarily large cost to those cases.

	CBP	CBP2	YAHSP2
CBP	—	8.85×10^{-5}	1.4×10^{4}
CBP2	8.85×10^{-5}	—	0.0111
YAHSP2	1.4×10^{4}	0.0111	—

Table 2

Results of the Wilcoxon signed-rank test performed with the command shown in Figure 24. A dashed line indicates that the analysis is not applicable.

Table 2 shows the p-values returned by the Wilcoxon signed-rank test. As it can be seen, the results correspond to pairwise associations of all the planners selected. Since this is a two-tailed statistical test, it does not provide any information whether one distribution has a smaller median than the other. However, comparing the medians (retrieved with `--median` in Figure 24) suffices in this particular case to conclude that both CBP2 and YAHSP2 perform better than CBP in the TRANSPORT domain with regard to plan length with a high confidence level. Similarly, CPB2 is found to provide shorter plans than YAHSP2 with a level of risk equal to $\alpha = 1 - 0.0111 = 0.9889$ which is below 99.0% and therefore would be rejected in favour of the Null Hypothesis that they perform much alike.

6. Related work

The organizers of the IPC 2008 already considered the development of software for automating the evaluation of planners. The software of the IPC 2011 followed the architecture devised at that time and extended its functionality in various ways. Additionally, it introduces a brand new component for helping researchers to inspect and analyze the results of the experiments. This component allows researchers to create summaries of the evaluation results, to perform queries on more than fifty different variables, to process them to compute different performance scores, to

```
> ./test.py --summary ./seq-sat.snapshot --variable lowervalue
          --planner 'cbp|yahsp2$' --domain transport --test wx
          --filter oksolved --matcher or --noentry 10000000
          --median
```

Fig. 24. Using `test.py` to perform a Mann-Whitney U-test on the best solutions found by planners CBP and CBP2 in the domain TRANSPORT

explore the evolution of these scores over time, to generate reports for all those data in different formats and even to apply statistical tests to the data generated.

The idea of creating software for automating and standardizing evaluation tasks have also been explored in other areas of Artificial Intelligence. The SAT community created SatEx [Simon, 2001], a web site devoted to SAT experimentation. Given a set of SAT solvers and a set of SAT benchmarks, SatEx generated results synthesis as well as detailed explorations of the experimental results. In addition, SatEx provided web-presentation of results and online answering of user queries. The reinforcement-learning (RL) community developed RL-GLUE [Tanner and White, 2009], a software for standardizing the evaluation in RL experiments. The software implemented a graphic interface that illustrated the progress of the different RL algorithms in different environments and allowed working with several languages and computing platforms using network socket communications. RL-GLUE has been used to run the RL International competition[15].

There are also recent efforts to create abstracted software for assisting in the automatic evaluation of solvers of different nature. STAREXEC [16] is a cross community logic solving service written in Java under development at the University of Iowa. LAB[17] is a Python package for conducting and analyzing experiments that run on a single machine or a computer cluster. LAB can also be used in combination with the DOWNWARD package to run experiments and create custom reports for the Fast Downward planning system [Helmert, 2006].

Finally, there is extensive previous work on defining methodologies and collecting good practices for the experimental evaluation of algorithms, several relevant examples are [Rardin and Uzsoy, 2001, Moret and Shapiro, 2001,Johnson, 2002]. During the design of this software we have adapted the general recommendations that were applicable to the particular evaluation of planners.

[15]See `http://www.rl-competition.org/`
[16]See `http://www.starexec.org/`
[17]See `https://lab.readthedocs.org/`

7. Future work

In a survey conducted among the scientific community of Automated Planning in October 2011, 40% of those polled asserted that the software described here already meets their demands. While the remaining 60% replied that it is okay, they consider that it should be improved. In the same survey, among the people that already tried the software of the IPC 2011 at that time, 42.9% used it to download the benchmarking planning tasks; another 42.9% used it to conduct private experiments and to produce informative reports of the performance of their planners. On the other hand, more than 87,5% considered that it already performs either well or very well for: checking out domains and problems, downloading and building planners, automating all the experimentation process, inspecting the resulting data and generating informative reports of different purposes.

From this feedback, it is understood that future work should still undergo the current development though the current functionality looks promising to a large number of researchers. In this sense, a number of future developments to extend its functionality are identified:

- Developing a web interface. More than 70% of the survey responses suggested to develop a web site that would allow the access to this software in the backend. This is almost straightforward for a number of functionalities including: the reporting tools (that can be used to generate reports on the fly after fulfilling a form) or showing the details of any planning task stored in a repository accessible from the web site.
- Extending the software to support different planning representation languages. Currently, the software of the Seventh International Planning Competition deals only with planning tasks described in PDDL. However, it has been devised to get beyond the IPC so that supporting other representation languages, such as NDDL [Bedrax-Weiss et al., 2005], would be very welcome as well.

On the other hand, a typical practice in previous IPCs was to provide different definitions of the various planning tasks selected for the evaluation. However, from the Sixth International Planning Competition it was decided to provide only a single definition, mostly based on STRIPS and, optionally, with fluents – both for the sequential and temporal tracks in the deterministic part as well as in the learning track[18]. Nevertheless, in order to ease comparisons it would be good also to allow designers of benchmarks to provide different definitions and to allow them to pick up one when running the script `invokeplanner.py`.
- Investigating how to reuse past evaluation results – so that for example progress among IPCs can be measured more reliably. A potential solution would be using virtual machines. One advantage of this approach is that virtual machines can be frozen (e.g., with all the third-party software needed for the compilation of planners) and shared among researchers so that different experiments can be conducted with regard to the same hardware configuration. However, the performance of virtual machines can be dependent on the performance of the hardware configuration of the native host.
- Providing a well-defined API to develop additional functionality. In its current form, every script described herein is structured through a dispatcher that provides the same functionality offered through the command line so that other scripts can invoke them directly. However, there is currently no documentation and a Programmer's Guide should be elaborated.
- Regarding the statistical tests an obvious extension consists of adding new statistical tests as those mentioned in Section 4.2.7, especially the Kolmogorov-Smirnov one-sample test since providing statistical significance that a particular sample follows a normal distribution allows a wide range of statistical tools to be applied. However, it seems more useful to provide additional information such as *confidence intervals* for the selected samples or to consider a measure of the strength such as the *effect size*.
- Finally, the reporting tools serve only to retrieve data generated by either `invokeplanner.py` or `validate.py`. However, it is desirable to allow programmers to add `plug-ins` to their planners so that additional data such as the number of expansions, the progress of the heuristic function and other information could be recognized by the reporting tools as well. In the same vain, all the metrics discussed in Section 4.2.6 are hand-coded. While modifying them or adding new ones is a rather simple procedure, it requires some programming skills. It is thus desirable to provide means to users to add new definitions of metrics that would be automatically acknowledged by the scripts `score.py` and `tscore.py`.

[18]Still, in the Sixth International Planning Competition, an alternative definition based in ADL was provided in a few domains in some tracks.

An important remark is that the reporting tools are not as flexible as one might desire. A solution to this problem would be using a database so that information other than the variables implemented there would be accessible. Finally, because using this software through a command-line can be cumbersome to some people, a Graphical User Interface could be developed to ease its usage.

8. Conclusions

We have described the general problem of assessing the performance of planners and have examined also the engineering challenges associated to its automation. The presented software sets up a framework for automating the evaluation of planning systems to a large extent while providing tools for addressing most of the scientific considerations that are typically raised in this context. The software binds together other pieces built in past International Planning Competitions, more remarkably the automatic validation tool, VAL, and a large body of software created for automating the Sixth International Planning Competition.

Since November 1^{st}, the software web pages have received more than three hundred visits. In the aforementioned survey conducted among the scientific community it turned out that 80% assured they will give this software a try, and 50% replied that standardizing tools is a good idea with an additional 47,2% considering it to be a must. These facts, when taken altogether, make us believe that releasing the competition software may increase the participation and transparency of future International Planning Competitions while it can significantly enhance private experimentations.

In addition, more than 50% of those polled fully agree with the consideration that this software improves reproducibility, exhaustiveness, and reliability. In the end, it is expected that the software of the Seventh International Planning Competition allows researchers to complete an empirical evaluation of planners with the following benefits:

- *Objectivity*. Different researchers can use the same software to rank planners according to different criteria, already implemented. The software also allows researchers to automatically validate a set of experimental results using the same validation tool, VAL. Extensions to these criteria, the validation process or other parts of the same software are immediately available to the whole community since the latest version is always fully available under the terms of the GNU General Public License version 3.
- *Exhaustiveness*. One can create a private svn repository that extends the set of planners, domains and/or problems used in the IPC 2011. Snapshots with the results of particular experiments can be shared among researchers and even the current svn can be extended with new results. The software also provides additional tools to allow everyone to examine the available data under different perspectives.
- *Comparability*. The source code of all the entrants of the IPC 2011 together with the domains and problems used for the evaluation are permanently stored in a svn repository that is publicly available. Thus, easing the process of running experiments with regard to the same set of planning tasks used in the last International Planning Competition. The software stores all the results (both as a results tree or a snapshot) that can be checked up to compare performance with new planners.
  The same principle applies to private repositories: researchers performing experiments with this software can make their results publicly available allowing others to access even far more data than the amount that would typically fit in a scientific communication such as a conference or journal.
- *Reproducibility*. The software of the IPC 2011 automatically generates a number of logs with detailed information about the environmnental conditions (such as the hardware details), the building process of each planner selected (resulting in a large body of information about software details) and the time and memory bounds used for performing the experiments, among other details.
- *Reliability*. Finally, the software is easy to install and it has no particular requirements allowing more people to examine planners, planning tasks and results so that errors of different nature are easier to catch.

All in all, it is expected that releasing the software of the competition: first, allow future competitors to test their software in an important number of domains and problems before submitting it to other IPCs; second, encourage practitioners to make more extensive experimentation when needed in preparation of their research; finally, allows them to make fairer comparisons among the results of different planning systems.

Acknowledgements

The design and implementation of the ideas behind the software of the Seventh International Planning Competition benefited from the contribution of a number of people and organizations who contributed in different ways.

We do want to explicitly express our gratitude to Derek Long for his assistance and support with the automated validation tool, VAL.

Also, to the people that tried previous versions of the software and provided comments to either extend the functionality of the software or to fix bugs of different nature. Among them, Emil Keyder, Jendrik Seipp and Ángel García Olaya.

Very importantly as well, to the IPC council for providing extensive comments and offering a lot of helpful suggestions.

Finally, we have to acknowledge the sponsorship of Decide Soluciones, iActive, the University Carlos III de Madrid and ICAPS. The hardware platform that was primarily used for developing and testing the software was funded by Spanish Science Ministry under project MICIIN TIN2008-06701-C03-03.

References

[Albore et al., 2009] Albore, A., Palacios, H., and Geffner, H. (2009). A translation-based approach to contingent planning. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 1623–1628, Pasadena, United States.

[Bacchus, 2001] Bacchus, F. (2001). Aips 2000 planning competition: The fifth international conference on artificial intelligence planning and scheduling systems. *AI Magazine*, 22(3).

[Bedrax-Weiss et al., 2005] Bedrax-Weiss, T., McGann, C., Bachmann, A., Edgington, W., and Iatauro, M. (2005). EUROPA2: User and contributor guide. Technical report, NASA AMES RESEARCH CENTER, Moffett Field (California), United States.

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

[Bonet and Geffner, 2003] Bonet, B. and Geffner, H. (2003). Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proc. 18th International Joint Conf. on Artificial Intelligence*, pages 1233–1238.

[Bonet and Geffner, 2011] Bonet, B. and Geffner, H. (2011). Planning under partial observability by classical replanning: Theory and experiments. In *IJCAI*, pages 1936–1941.

[Cimatti et al., 1997] Cimatti, A., Giunchiglia, F., Giunchiglia, E., and Traverso, P. (1997). Planning via model checking: A decision procedure for AR. In *European Conference on Planning*.

[Corder and Foreman, 2009] Corder, G. W. and Foreman, D. I. (2009). *Nonparametric Statistics for Non-Statisticians*. John Wiley & Sons, New Jersey, United States.

[Cushing et al., 2007] Cushing, W., Kambhampati, S., Mausam, and Weld, D. S. (2007). When is temporal planning really temporal? In *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, pages 1852–1859.

[Fern et al., 2011] Fern, A., Khardon, R., and Tadepalli, P. (2011). The first learning track of the international planning competition. *Machine Learning*, 84:81–107.

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

[Gerevini et al., 2009] Gerevini, A. E., Haslum, P., Long, D., Saetti, A., and Dimopoulos, Y. (2009). Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners. *Artif. Intell.*, 173(5-6):619–668.

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

[Hoffmann and Edelkamp, 2005] Hoffmann, J. and Edelkamp, S. (2005). The deterministic part of ipc-4: An overview. *J. Artif. Intell. Res. (JAIR)*, 24:519–579.

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

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

[Howey et al., 2004] Howey, R., Long, D., and Fox, M. (2004). VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *The Sixteenth IEEE International Conference on Tools with Artificial Intelligence (ICTAI-2004)*, pages 294–301, Boca Raton, Florida, United States.

[Johnson, 2002] Johnson, D. S. (2002). A theoretician's guide to the experimental analysis of algorithms. *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, page 215–250.

[Kautz and Selman, 1996] Kautz, H. A. and Selman, B. (1996). Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1194–1201, Portland, Oregon, United States.

[Kautz and Selman, 1999] Kautz, H. A. and Selman, B. (1999). Unifying SAT-based and graph-based planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 318–325, Stockholm, Sweden.

[Keyder and Geffner, 2009] Keyder, E. and Geffner, H. (2009). Soft goals can be compiled away. *J. Artif. Int. Res.*, 36(1):547–556.

[Long and Fox, 2003a] Long, D. and Fox, M. (2003a). The 3rd international planning competition: Results and analysis. *J. Artif. Intell. Res. (JAIR)*, 20:1–59.

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

[McDermott, 1998] McDermott, D. (1998). PDDL the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

[Mcdermott, 2000] Mcdermott, D. (2000). The 1998 ai planning systems competition. *AI Magazine*, 21:35–55.

[Moret and Shapiro, 2001] Moret, B. M. E. and Shapiro, H. D. D. (2001). Algorithms and experiments: The new (and old) methodology. *Journal of Universal Computer Science*, 7(5):434–446.

[Morris and Muscettola, 2005] Morris, P. and Muscettola, N. (2005). Temporal dynamic controllability revisited. In *In Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005*, pages 1193–1198. AAAI Press / The MIT Press.

[Nau et al., 2003] Nau, D., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., and Yaman, F. (2003). Shop2: An htn planning system. *Journal of Artificial Intelligence Research*, 20:379–404.

[Palacios and Geffner, 2007] Palacios, H. and Geffner, H. (2007). From conformant into classical planning: Efficient translations that may be complete too. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS-07)*, pages 264–271, Providence (Rhode Island), United States.

[Rardin and Uzsoy, 2001] Rardin, R. L. and Uzsoy, R. (2001). Experimental evaluation of heuristic optimization algorithms: A tutorial. *Journal of Heuristics*, 7:261–304.

[Rintanen, 2004] Rintanen, J. (2004). Phase transitions in classical planning: an experimental study. In *In Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR 2004*, pages 710–719. AAAI Press.

[Rintanen, 2007] Rintanen, J. (2007). Complexity of concurrent temporal planning. In *In Proc. of the 17th International Conference on Automated Planning and Scheduling (ICAPS07*, pages 287–295.

[Simon, 2001] Simon, L. (2001). SatEx: towards an exhaustive and up-to-date SAT experimentation. In *IJCAI'01 Workshop on Empirical Methods in Artificial Intelligence*.

[Sussman, 1975] Sussman, G. J. (1975). *A Computer Model of Skill Acquisition*. Elsevier Science Inc., New York, NY, USA.

[Tanner and White, 2009] Tanner, B. and White, A. (2009). RL-Glue : Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research*, 10:2133–2136.

[van Beek and Chen, 1999] van Beek, P. and Chen, X. (1999). CPlan: A constraint programming approach to planning. In *National Conference on Artificial Intelligence, AAAI99*, pages 585–590.