Mach Learn (2011) 84:81–107
DOI 10.1007/s10994-011-5234-y

The first learning track of the international planning competition

Alan Fern · Roni Khardon · Prasad Tadepalli

Received: 29 May 2010 / Revised: 13 December 2010 / Accepted: 30 December 2010 /
Published online: 31 January 2011

Abstract The International Planning Competition is a biennial event organized in the context of the International Conference on Automated Planning and Scheduling. The 2008 competition included, for the first time, a learning track for comparing approaches for improving automated planners via learning. In this paper, we describe the structure of the learning track, the planning domains used for evaluation, the participating systems, the results, and our observations. Towards supporting the goal of domain-independent learning, one of the key features of the competition was to disallow any code changes or parameter tweaks after the training domains were revealed to the participants. The competition results show that at this stage no learning for planning system outperforms state-of-the-art planners in a domain independent manner across a wide range of domains. However, they appear to be close to providing such performance. Evaluating learning for planning systems in a blind competition raises important questions concerning criteria that should be taken into account in future competitions.

Keywords Planning · Machine learning · Competitions · Reinforcement learning

1 Introduction

Starting in 1998 the *International Planning Competition (IPC)* has been held in conjunction with the *International Conference on Automated Planning and Scheduling (ICAPS)* on

Editors: S. Whiteson and M. Littman.

A. Fern (✉) · P. Tadepalli
School of Electrical Engineering and Computer Science, Oregon State University, Kelley Engineering Center, Corvallis, OR, USA
e-mail: afern@eecs.oregonstate.edu

P. Tadepalli
e-mail: tadepall@eecs.oregonstate.edu

R. Khardon
Department of Computer Science, Tufts University, Medford, MA, USA
e-mail: roni@cs.tufts.edu

a biannual basis. The competitions are typically organized around a number of planning domains, where a planning domain (see Sect. 2) is simply a class of problems that share a common action schema—e.g. Blocksworld is a well-known planning domain that contains a problem instance for each possible initial tower configuration and goal configuration. For each domain, the IPCs typically include a number of problems (40–50) and the planners are run on each problem with a time limit. The planners are then evaluated based on a combination of how many problems they solve and the quality of the solutions. While there is always debate about the utility of this and other competitions, there is no question that the IPC has dramatically improved the rigor of planner evaluation, and that there has been a dramatic improvement in the state-of-the-art in automated planning since its inception.

Given that the planners in the IPC are asked to solve many problems from individual domains, and that problems within a domain generally have common solution structures, it makes sense to consider learning from previous problem-solving experience in a domain to better solve future problems in the same domain. Here better solve could mean to solve the problems faster, to find higher quality solutions, or to solve problems that were not solvable without prior experience. However, no IPC participant prior to 2008 included learning capabilities. Rather, these prior IPC planners treat each problem as completely novel and learn nothing from previous problem-solving experience.

Perhaps one reason that learning-based planners have not appeared in previous competitions is that the structure of the IPC did not provide a good venue for planners that learn. This is because most approaches for learning-to-plan require a substantial learning period, which would be counted against them in the standard IPC format. This would reduce the potential observed benefit from learning. However, there are good arguments that learning time should not be treated the same as planning time, since learning time can be amortized over all future planning experiences in a domain.

Based on this motivation, we decided to run a learning track for IPC 2008. The learning track was designed so that learning time would not be counted against planners during their evaluation, but rather the track would include an explicit learning phase, where planners would be free to learn in a domain for a significant but bounded amount of time before the formal evaluation. This provided a venue for addressing the following question: *Do we have planners that can leverage a learning period to outperform state-of-the-art non-learning planners across a wide range of domains?*

As briefly reviewed in Sect. 3, there is a long history of research in learning-based planners, with many demonstrations of successful learning. However, this work has not produced solid evidence that the answer to the above question is "yes". Perhaps one reason for this is that learning-based planners have not been evaluated very well in most studies, typically focusing on a very small number of domains and/or problems per domain. Accordingly, the issue of robustness to different domains and problems has typically not been seriously addressed and evaluated. One goal of introducing the learning track was to facilitate a more thorough evaluation of learning-based planners and to force the issue of robustness to be taken seriously. The hope is that this will lead to the same type of rapid progress in learning-based planning systems as was observed for non-learning systems after the introduction of the IPC.

The results from our competition suggest that at this stage no learning for planning system outperforms state-of-the-art planners in a domain independent manner across a wide range of domains. However, they appear to be close to providing such performance, in the sense that several learning for planning systems outperformed all but one system from the non-learning track of the IPC when evaluated on our benchmarks. Finally we note that evaluating learning for planning systems in a blind competition raises important questions con-

cerning criteria that should be taken into account in future competitions. These observations are discussed in greater detail in the rest of the paper.

The paper is organized as follows. The next two sections define the problem by describing the STRIPS language for planning (which was the official language of the competition) and the problem of learning-to-plan. We also review previous work on learning-to-plan and relate it to work in reinforcement learning. Sections 4, 5 and 6 describe the structure of the competition, details of the domains used in the competition and the systems participating in the competition. The competition results are presented and evaluated in Sect. 7. The concluding section provides a discussion of the results and competition with a view to future competitions of learning for planning systems.

2 STRIPS planning domains and problems

An important consequence of the IPC has been the introduction of a standard language for describing planning domains and problems. The *planning domain definition language (PDDL)* (McDermott 1998) has become the standard language for the deterministic planning track of the IPC, and more recently *probabilistic PDDL (PPDDL)* (Younes 2003) has been used for the probabilistic track. The availability of these languages and planners that support them has greatly simplified the process of comparing different planners on a wide variety of domains. The languages also make it easier for IPC organizers to specify the scope of a competition, or individual tracks in a competition, by specifying the fragment of PDDL that will be used.

The first major decision in creating the learning track of IPC 2008 was to select a planning domain language. Our primary goal was to select a language that would maintain a relatively low entry bar, particularly for researchers who were already working on learning-based planners. For this reason, we decided to focus on deterministic planning domains described in the STRIPS fragment of PDDL. The STRIPS language has been widely used since the earliest work in automated planning (Fikes and Nilsson 1971) and has been arguably the most commonly studied language in research on automated planning and learning-based planners. Further since STRIPS is just a subset of the full PDDL, participants in the learning track could easily leverage the existing state-of-the-art planners, all of which support the STRIPS fragment and more. In the remainder of this section we introduce the concepts of planning domains and problems and illustrate how they are represented in the STRIPS language. The description in this section is largely illustrative. Appendix A gives example definitions for the Blocksworld, and the formal syntax and semantics can be found at the competition web-site:

http://eecs.oregonstate.edu/ipc-learn/.

Planning domains. The IPC is organized around the concepts of planning domains and problems. The main purpose of a planning domain specification is to define a vocabulary for specifying properties of the world and for defining the dynamics of actions that can be taken by the planner. Specifically, a *planning domain* $\mathcal{D}$ defines a set of possible *action schemas* $\mathcal{A}$, a set of *predicate symbols* $\mathcal{P}$ for describing facts about the world, and a possibly empty set of *domain constants* $\mathcal{C}_d$. The set of predicates can be used to describe properties of objects in the world as well as relationships among those objects. For example, in the Blocksworld, the predicate (on-table x) indicates that block x is on the table and (on x y) indicates that block x is on y. Note that here we are using a LISP-style syntax for predicates rather than for example a more typical syntax such as on-table(x) and on(x, y). The action schemas

describe how the world state changes after applying ground instances of the schemas. For example, in the Blocksworld the STRIPS action schema for the *pickup* action, that picks up a block from the table, might look like:

```
(:action pickup
  :parameters (?h ?b)
  :precondition (and (hand ?h) (block ?b) (clear ?b) (on-table ?b)
                     (empty ?h))
  :effect (and (holding ?h ?b) (not (clear ?b)) (not (on-table ?b))
               (not (empty ?h))))
```

where ?h and ?b are variables representing generic object parameters. In addition to the parameters, each schema specifies the *preconditions*, which state the properties of the world that must be true in order for the action to be executed. For the *pickup* action the hand ?h must be empty and the block ?b must be clear and on the table. The action schema also specifies the effects of the action that occur when the preconditions are satisfied. The effects specify the facts to be added to the world (*add effects*) when the action is taken and the facts to be deleted from the world (*delete effects*). In the above example, the only add effect is (holding ?h ?b) indicating that the hand is holding the block after the pickup. The delete effects are those that are preceded by the token "not". In the above example, there are three delete effects, including (not (empty ?h)), indicating that the fact (empty ?h) will be deleted from the state after executing the action.

Planning problems. Given a planning domain it is possible to define *planning problems* which specify a set of problem constants $\mathcal{C}_p$, an initial state s_0, and a goal condition g. The initial state is a set of *state facts*, where each state fact is simply a predicate in $\mathcal{P}$ applied to the appropriate number of constants in $\mathcal{C}_d \cup \mathcal{C}_p$. For example, an initial state involving three blocks might be:

```
((hand H) (block A) (block B) (block C)
 (clear A) (on-table A) (on B C) (clear B) (on-table C) (empty H))
```

where H, A, B, C are problem constants. Similarly the goal can be any set of state facts, but need not fully specify a state. For example, a goal might simply be (on-table B) indicating that any state in which block B is on the table is considered to be a goal.

The *ground actions* of a planning problem correspond to all possible ways of instantiating the parameters of an action schema with domain and problem constants. Given a ground action a and a state s, which is just a set of state facts, we say that a is *applicable* in s if its preconditions are satisfied in s. Applying an applicable action a to s results in a new state $s' = (s \setminus \mathrm{Del}(a)) \cup \mathrm{Add}(a)$, where Del(a) and Add(a) are the add and delete facts in a. So, for example, the ground action (pickup H A) is applicable in the above state and its application would result in the new state:

```
((hand H) (block A) (block B) (block C)
 (holding H A) (on B C) (clear B) (on-table C))
```

Solution plans. Given a planning problem, the objective is to find a solution plan. A *solution plan* for a planning problem is a sequence of ground actions $\langle a_1, \ldots, a_l \rangle$, where the sequential application of the sequence starting in state s_0 leads to a state s' such that $g \subseteq s'$.

Typed and untyped STRIPS. PDDL allows for explicit typing of constants in a planning domain or problem definition. For example, the above constants A, B, and C could be declared to have a type *block* and the parameters ?h and ?b of the action schema for pickup can be declared to have types *hand* and *block*. Note that the above examples did not include explicit typing, but rather used domain predicates such as (block x) and (hand x) to enforce type constraints. Including explicit type information in the domain and problem definitions

can be exploited by planners in various ways. For example, type information on the action schema parameters can be used to avoid considering ground actions that are not type correct. Also, it is conceivable that a learner might choose to treat "type predicates" differently compared to other predicates, which is facilitated by including type information. In order to support typing, while avoiding the requirement that participants support it, the learning track allowed participants to use either typed or untyped versions of the STRIPS language. To support this we included typed and untyped versions of each domain and problem in the competition. Appendix A gives example definitions of Blocksworld using both of these languages.

3 Learning to plan

It is common for planning systems to be asked to solve many problems from a particular domain. Given that problems from the same domain share significant structure, it is natural to attempt to learn from past experience in a domain in order to solve future problems from the same domain more effectively.

We formalize this learning-to-plan problem as follows. The input is a planning domain and a training set of problems from the domain drawn according to some target distribution. The goal of learning is to take the training set and domain definition and to produce some form of *domain-specific control knowledge* that can be used by a planner in order to optimize its expected performance on problems drawn from the target distribution. Importantly, the learned knowledge should not only help the planner on the training problems, but also generalize to problems outside of the training set.[1] A typical strategy used by learning systems is to solve the problems in the training set and to then extract some form of knowledge from those solutions in order to help the planner find those solutions more quickly, and enable solutions of new problems. However, the specification of the learning-to-plan problem is agnostic about the type of planner that is being enhanced and the specific approach to learning.

3.1 Prior work on learning-to-plan

There has been a long history of work on learning-to-plan, originating at least back to the original STRIPS planner (Fikes et al. 1972), which learned triangle tables or macros that could later be exploited by the planner. For a collection and survey of work on learning in AI planning see Minton (1993) and Zimmerman and Kambhampati (2003).

A number of learning-to-plan systems have been based on the explanation-based learning (EBL) paradigm, for example, Minton et al. (1989) among many others. EBL is a deductive learning approach, in the sense that the learned knowledge is provably correct. Despite the relatively large effort invested in EBL research, the best approaches typically did not consistently lead to significant gains, and even hurt performance in many cases. A primary way that EBL can hurt performance is by learning too many, overly specific control rules, which results in the planner spending too much time simply evaluating the rules at the cost

[1] Notice that we focus here on knowledge that is transferred to new problems in the same domain where it is often the case that a single simple policy can deliver good performance. This is different from recent work on transfer learning (e.g., Taylor and Stone 2009) where the focus is on transfer of knowledge for use in new domains where the connection between the domains is less immediate.

of reducing the number of search nodes considered. This problem is commonly referred to as the EBL utility problem (Minton 1988).

Partly in response to the difficulties associated with EBL-based approaches, there have been a number of systems based on inductive learning, sometimes combined with EBL. The inductive approach involves applying statistical learning mechanisms in order to find common patterns that can distinguish between good and bad search decisions. Unlike EBL, the learned control knowledge typically does not have guarantees of correctness. However, the knowledge is typically more general and hence more effective in practice. Some representative examples of such systems include learning for partial-order planning (Estlin and Mooney 1996), learning for planning as satisfiability (Huang et al. 2000), and learning for the Prodigy means-ends framework (Aler et al. 2002). While these systems typically showed better scalability than their EBL counterparts, the evaluations were typically conducted on only a small number of planning domains and/or small number of test problems. There is no empirical evidence that such systems are robust enough to compete against state-of-the-art non-learning planners across a wide range of domains.

A slightly different approach is taken by several learning-to-plan systems based on the idea of learning reactive policies for planning domains (Khardon 1999; Martin and Geffner 2000; Yoon et al. 2002). These approaches use statistical learning techniques to learn policies, or functions, that map any state-goal pair from a given domain to an appropriate action. Given a good reactive policy for a domain, problems can be solved quickly, without search, by iterative application of the policy. Despite its simplicity, this approach has demonstrated considerable success improving over baseline planners in some cases. However, these approaches have still not demonstrated the robustness necessary to outperform state-of-the-art non-learning planners across a wide range of domains.

More recently there has been work on learning improved heuristic functions for forward-search planners such as FF (Hoffmann and Nebel 2001). This includes a case-based reasoning approach by La Rosa et al. (2007) and an approach based on feature induction and linear regression (Yoon et al. 2006). While these approaches show signs of improving on state-of-the-art non-learning planners in a number of domains, the results clearly show that there is plenty of room for improved robustness.

In summary, learning for planning has been investigated from several perspectives, and in several cases it has been shown to provide performance improvements over a baseline planner in a specific context. But such systems are not widely in use. In addition, there is no conclusive evidence for their wide applicability in a domain independent manner, which could lead to more immediate applicability. The competition therefore serves to test and encourage such wider applicability.

3.2 Relationship to reinforcement learning

It is interesting to consider the relationship between reinforcement learning (RL) (Sutton and Barto 1998) and the above-mentioned learning-to-plan problem, which is the basis for the learning track. Although the competition was restricted to deterministic problems, the definitions and context are clearly applicable to stochastic planning and we will consider this more general case. Both RL and planning deal with agents choosing actions so as to maximize some criterion, the probability of reaching the goal in probabilistic planning and the more general accumulated reward in RL. Typically in RL, the agent does not have a model of the world and has to maximize long term reward, but model-based RL algorithms learn such models and use them to make better decisions. The study of Markov decision processes (MDP) can be seen to unify the two areas in that the model can be used to express

RL problems but also offers a perspective on planning algorithms that use a model to obtain decisions without any learning (Puterman 1994). A second difference between RL and learning for planning concerns the protocol. The basic RL model has a single infinite run of an agent in a world without reset, which is different from the start state to goal path in planning. However, research in RL has considered episodic tasks that are closer in spirit to planning in that episodes end in specific states and the process is restarted in a start state or start-state distribution. Thus, one can see that the core model in the two areas is very similar and that the main difference (for the planning portion) is one of representation and the kind of state and action spaces considered in research.

Not surprisingly, work that bridges the two areas has already been explored. In particular, relational reinforcement learning (Dzeroski et al. 2001; Tadepalli et al. 2004) shows how RL ideas can be applied in the context of relational planning by learning compact representations of value functions and policies. However, this approach has only demonstrated successful results in problems that are much smaller than typical IPC planning problems and is not likely to scale without significant extensions. More recently, RL-based algorithms that do not need a model have been developed (Fern et al. 2006). However, even these improvements have not yet demonstrated enough robustness to reliably compete with state-of-the-art planners. Along the same lines, planning in relational MDPs has been studied with traditional operations research and RL algorithms such as value iteration, policy iteration and RTDP (Bonet and Geffner 2003; Boutilier et al. 2001; Kersting et al. 2004; Sanner and Boutilier 2009; Wang and Khardon 2007; Wang et al. 2008). Some of these algorithms have been implemented and applied successfully to problems from previous IPCs (Sanner and Boutilier 2009; Joshi and Khardon 2008; Joshi et al. 2010) but have yet to show wide domain independent applicability. As this discussion suggests, there is a close relationship between RL for complex domains and learning for planning, and the planning problems can be seen as presenting significant challenges and opportunities for RL algorithms.

4 Competition structure

One of the original motivations for the learning track was to move closer to demonstrating that learning-based planners are capable of outperforming state-of-the-art non-learning planners. As such, the ideal structure for the learning track would have been to design the learning track as an extension to the standard non-learning track in the IPC, where the learning-based planners would be allowed a learning period before the competition began, which would not be counted against them. This would have allowed for learning and non-learning planners to be evaluated on the same domains and problems, providing a clear view of the relative performance of these approaches.

Unfortunately, this structure of the competition was not possible, due to the fact that the non-learning track of IPC 2008 include extended features of PDDL that fell outside of the STRIPS fragment. The inclusion of these features is a reflection of the more advanced state of the non-learning track of the competition and is a necessary step for promoting progress. It was decided early on that including these extended features in the learning track would dramatically limit participation. As such, it was decided to run the learning track independently of the non-learning track, including the design of an independent set of domains and problem sets restricted to the STRIPS fragment of PDDL. These domains are described in Sect. 5.

The learning track was divided into three distinct phases. First, during the *pre-competition phase* the participants were provided with example domains and training sets in

order to help them develop their competition programs, which were submitted to the organizers at the end of the phase. Second, during the *learning phase* the participants ran their learning algorithms on the competition domains and submitted the learned knowledge files to the organizers at the end of the phase. Third, during the *evaluation phase* the organizers evaluated the participant's planners on each of the competition domains, allowing the planners access to the knowledge learned in the learning phase. The remainder of this section describes each of these phases in more detail.

4.1 Pre-competition phase

Each competitor in the learning track was required to submit a pair of programs to the organizers at the start of the competition: a *learner* and a *planner*. The learner was required to accept as input a planning domain description and a set of training problems, and then output a *knowledge file* for that domain. The planner was required to take as input a planning domain, a planning problem, and a knowledge file for that domain (if available), and to then attempt to solve the planning problem and output a solution in a specified format if one was found.

During the pre-competition phase the participants were provided with example planning domains and training sets (details about training sets below) in order to test their programs. Importantly the actual competition domains were not revealed to the participants during this pre-competition phase, which prevented customization of the approaches to the competition domains. The pre-competition phase ended with the submission of the learner and planner programs from each participating team. After this point there was a code freeze and no changes were allowed to the programs for the remainder of the competition, with the exception of clear bug fixes (e.g. parser problems or memory leaks) that were approved by the organizers.

4.2 Learning phase

After the submission of the learner and planner programs the domain definitions and training problems to be used for the competition were made available to the participants and the learning phase began. The desired output of the learning phase was a single knowledge file for each competition domain, resulting from running the learner on each domain.

Training problem sets. For each planning domain it was decided to provide the participants with problems drawn from two distinct distributions over problem instances: the *target distribution* and an easier *bootstrap distribution*. The target distribution corresponded exactly to the distribution used to generate problems for the final evaluation phase of the competition. Thus, problems drawn from this distribution can be viewed as representative of the problems to be used in evaluation. An effort was made by the organizers to design target distributions so that the generated problems were difficult for state-of-the-art non-learning planners.

While designing challenging target distributions provided the opportunity to demonstrate improvements over non-learning planners, it also resulted in difficult training problems, which posed quite a challenge to solve and learn from. Thus, the organizers also provided training problems drawn from a bootstrap distribution, which were significantly easier than problems from the target distribution. In particular, it was verified that the bootstrap problems could be solved by state-of-the-art planners in a short amount of time. The aim of including the bootstrap distribution was to provide participants with a set of problems that were more tractable to solve and learn from, which could be conceivably used to bootstrap

performance in order to learn from problems drawn from the target distribution. It was difficult in general to specify the exact relationship between the target and bootstrap distributions. However, informally, the organizers attempted to scale the number of objects involved in the planning problems to move from the bootstrap to target distributions, but keep other problem characteristics the same. The training sets for each domain contained 60 total problems, with 30 problems drawn from each of the bootstrap and target distributions. These problems were provided to participants in distinct and identified groups. The participants were free to design learners that used problems from either or both distributions. Naturally the set of target problems used in the actual evaluation phase were not made available to the learners during the learning phase.

Running the learners. Previous work suggests that learning control knowledge for planning domains typically requires large amounts of cpu time. This made it impractical for the organizers to run the learning programs on their own machines without significantly restricting the amount of learning time allowed for each domain. The organizers decided that learning time was not the top concern for this first learning track and accordingly decided to allow each participant to run the learners on their own machines. Specifically, the participants were allowed two weeks to run their learners on the training data of each competition domain and to then submit the resulting knowledge files, one per domain, to the organizers for use in the evaluation phase. Since the organizers had copies of the submitted learners, it was possible to perform random checks to help verify that no changes were made to the learning programs after the code freeze. Specifically, for each participant the organizers randomly selected one domain and ran their learner in order to verify that the resulting knowledge file matched that produced by the participant. In the case of algorithms involving randomization, random seeds were taken into account.

4.3 Evaluation phase

At the end of the learning phase, the organizers had knowledge files for each domain from each participating team. In a few cases, teams were not able to produce knowledge files for certain domains, in which case the team's planner was evaluated without the benefit of learned knowledge in those domains.

Running the evaluations. The evaluations were conducted on the organizers' local machines. The evaluation machines were identical in their make and configuration, minimizing the influence of hardware differences on the results. For each domain, the organizers generated 30 new problems from the target distribution to be used for evaluation. Next, for each participant and domain, the organizers ran the participant's planner on the 30 problems, while providing the planner the appropriate knowledge file from the learning phase. A time limit of 15 minutes was enforced for each problem. If a solution to a problem was not found within the time limit, the problem was considered not solved. When planners output a solution within the time limit, the solution plan was checked via an automated plan verification program. This procedure ensured that all returned plans were in fact valid solutions. At the same time, the number of actions in all returned plans were recorded.

In addition to evaluating the planners with the benefit of the learning knowledge files, we also evaluated the planners in the same way as described above, without the knowledge files as input. The aim of this evaluation was to provide data regarding the relative benefit of learning for the individual planners.

Evaluation metrics. There were two evaluation metrics used in the learning track competition: the time to produce a plan and the solution plan quality. For quality, we have chosen to use a simple measure based on the number of actions in the plan. While there are more

elaborate measures, for example those that take into account action costs, or the makespan of a parallel execution of the plan, these seem to focus more on extra capabilities of planners, and therefore of secondary rather than primary interest for this competition.

For both aspects, one has to be careful in evaluation in the context of plan failure. For example, when measuring plan length directly, an unsuccessful planning episode could be potentially handled in two ways. First, we can ignore it when calculating the average plan length. Unfortunately, this artificially makes the average plan length look better than it really is. Alternatively, an unsuccessful planning episode could be penalized with a large plan length which is set arbitrarily, but this again artificially changes the reported average plan length for the system based on the arbitrary penalty parameter. Therefore, we chose to report an alternative *relative* measure where the best system gets a credit of 1 for a problem and a failed planning attempt gets a credit of 0.

More concretely, the first measure, the *planning time metric*, provided a measure for how quickly a planner was able to solve the evaluation problems relative to the other planners. In particular, for a concrete planning problem, let T^* be the minimum time required by any planner in the competition to solve the problem. In cases when no planner solved a problem, the corresponding problem is ignored for purposes of comparative evaluation. A planner that solved the problem in time T received a score of T^*/T for the problem. Those that did not solve the problem got a score of 0. Thus, the planner that solved a problem the fastest got a score of one. The overall planning time metric for a planner was then the sum of the scores received over all evaluation problems.

The second measure, the *plan quality metric*, provided a measure of the quality of solutions found by a planner, where quality was measured in terms of the number of actions in a plan. For a given problem, let N^* be the minimum number of actions in any solution returned by a participant. A planner that returned a solution with N actions received a score of N^*/N for the problem. Those that did not solve the problem got a score of 0. Thus, the planners that found the shortest solution plans for a problem received a score of one for the problem. The overall plan quality metric for a planner was then the sum of scores received over all evaluation problems.

5 Competition domains

This section describes the planning domains used in the learning track of IPC 2008. Overall, the learning track included six domains, none of which had appeared in previous IPCs. As discussed below we have chosen these domains to allow some learning opportunities (chosen by analogy to previously reported successes) as well as hard planning instances that are challenging for current planning systems. All of these domain descriptions along with training problems and evaluation problems used in the competition are available at the competition web site:

http://eecs.oregonstate.edu/ipc-learn/.

The web site also provides problem generators for all of the domains, which include parameters for controlling the complexity of the randomly generated problems. Below we give a general description of each domain.

5.1 Gold miner

A robot is in a mine and has the goal of reaching a location that contains gold. The mine is organized as a grid where each cell has either hard rock or soft rock. There is a special

location where the robot can either pick up an endless supply of bombs or pick up a laser cannon. The laser cannon can shoot through both hard and soft rock, whereas the bomb can only penetrate soft rock. However, the laser cannon will also destroy the gold if used to uncover the gold location. The bomb will not destroy the gold. The problem difficulty is scaled by increasing the size of the grid. Problem instances differ in the contents of each grid cell. The basic actions are to move to an adjacent grid location that is empty, to pick up a bomb, laser, or gold, to put down a bomb or laser, and to fire a laser or detonate a bomb. This domain has a simple optimal strategy (may differ for small grids):

1. Get the laser cannon.
2. Shoot through the rock until reaching a cell bordering the gold.
3. Go and get a bomb.
4. Blast away the rock at the gold location.
5. Pickup the gold.

Thus, a learner that can uncover this general structure can completely avoid search, while maintaining high quality plans. The existence of a compact policy for this domain suggests that there is structure to be captured and thus potential learning opportunities for the competition.

5.2 Matching Blocksworld

This is a simple variant of the classic Blocksworld domain where each block has a polarity of either positive or negative and there are two hands, one of positive polarity and the other of negative polarity. The difference compared to the classic Blocksworld is that if a block is picked up by a hand of opposite polarity then it is damaged such that no other block can be placed on top of it, which can lead to dead ends. The interaction between hands and blocks of the same polarity is just as in the standard blocks world. The initial states and goals correspond to random configurations of block towers. The problem difficulty is scaled by increasing the number of blocks. The basic actions are to pick up, put down, and stack blocks. Blocksworld has simple policies expressible in terms of rules and it has been used before to demonstrate the success of learning for planning. Therefore one might hope that matching Blocksworld would provide learning opportunities for the competition.

5.3 Classic N-puzzle

This is the classic $N \times N$ sliding puzzle domain. This has been a common test bed for search algorithms and, in particular, work on macro learning (Iba 1989) for search algorithms. The problem complexity is scaled by increasing N and different problem instances of the same size differ in their initial configuration. This domain has been used before to demonstrate the potential of macro learning and should therefore offer learning opportunities for the competition.

5.4 Parking

This domain involves parking cars on a street with N curb locations where cars can be double parked but not triple parked. The goal is to move from one configuration of parked cars to another configuration by driving cars from one curb location to another. The problems in the competition contain $2 \cdot (N - 1)$ cars, which allows one free curb space and guarantees solvability. The initial state and goal of each problem specify locations for each of the cars.

The problem difficulty is scaled by increasing the number of curb locations N. This domain is isomorphic to a bounded-table blocks world domain where there are N table locations and $2 \cdot (N - 1)$ blocks and towers are only allowed to contain at most two blocks. Again, here, the analogy to Blocksworld suggests that the domain might have learning opportunities for the competition.

5.5 Sokoban

This domain is inspired by the popular Sokoban puzzle game where an agent has the goal of pushing a set of boxes into specified goal locations in a grid with walls. The competition problems are generated in a way that guarantees solvability and generally are easy problem instances for humans. Sokoban has been demonstrated to be a very challenging domain for AI search and planning algorithms, even when significant human domain knowledge is provided (Junghanns and Schaeffer 2001). This domain is more complex than the others; our choice of problem setting aimed at allowing for potential learning by capturing the policy that makes it easy for humans.

5.6 Thoughtful solitaire

This domain models a simplified version of Thoughtful Solitaire, which is a solitaire variant that is played with all of the cards visible. The exact rules follow those described in Bjarnason et al. (2007) but are simplified so that one can turn each card from the talon rather than three cards at a time. The problem difficulty is scaled by varying the number of cards and stacks. The competition problems have all been proved to be solvable using the domain specific solver described in Bjarnason et al. (2007). A variant of the Solitaire domain was used with the API system that performs policy learning as an integral step of the algorithm (Fern et al. 2006). The domain might therefore offer learning opportunities but be more challenging than some of the domains above.

6 Competition participants

The turnout for the competitions exceeded expectations. There were a total of 13 systems entered from 10 different teams. Abstracts describing these systems in more detail are available on the competition web-site. The systems in the competition can be roughly divided into five different categories. Below we give a brief description of each system grouped according to categories.

6.1 Policy learners

Systems in this category learned control knowledge in the form of a reactive policy and used that policy in various ways to guide the search process of a planner. The systems differed substantially according to how the policy was represented and learned and how it was used to guide search, and it was not clear at the outset which approach might be more generally applicable. However they can be viewed as applying similar principles at the higher level of planner architecture. There were five such systems.

– **CABALA**
  Tomas de la Rosa; Daniel Borrajo; Angel Garcia Olaya
  Universidad Carlos III de Madrid

CABALA learns a case base policy. The case base uses "object profiles" called typed sequences capturing the predicates holding for an object during plan execution. The policy is used to bias the search of a look-ahead heuristic search planner.

– **Roller**
Tomas de la Rosa; Sergio Jimenez
Universidad Carlos III de Madrid

Roller learns decision-tree policies using ILP (Blockeel and De Raedt 1997). The policies capture "helpful action schemas" in a planning context, and then rank bindings for the actions. Here too the policy is used to bias the search of a look-ahead heuristic search planner.

– **REPLICA**
Rocio Garcia-Duran; Fernando Fernandez; Daniel Borrajo
Universidad Carlos III de Madrid

REPLICA learns instance-based policies. The policies, capturing actions for state-goal pairs, are pruned from an initial set to avoid the utility problem mentioned above. The policy is used to bias an enforced hill-climbing search in the style of FF (Hoffmann and Nebel 2001).

– **ObtuseWedge**
Sungwook Yoon
Palo Alto Research Center

ObtuseWedge learns reactive policies in the form of decision lists. The policy is incorporated into FF's best-first search.

– **Sayphi-Rules**
Susana Fernndez Arregui; Daniel Borrajo
Universidad Carlos III de Madrid

Sayphi-Rules learns decision tree policies by combining EBL to generate training instance for an ILP learner (Blockeel and De Raedt 1997). The search process chooses randomly among the policy and the original heuristic in an enforced hill-climbing search in the style of FF (Hoffmann and Nebel 2001).

6.2 Macro action learners

Systems in this category learned macro actions in the form of sequences of individual actions. The macro actions are then used directly or indirectly in the planning process. There were three such systems.

– **Wizard+FF**
M.A. Hakim Newton; John Levine; Maria Fox; Derek Long
University of Strathclyde

Wizard+FF learns macro actions (from a rich family of potential macros) via genetic algorithms and then simply encodes the macros as additional actions in the domain. Planning is performed directly by the planner FF (Hoffmann and Nebel 2001) using the extended action set.

– **Wizard+SG**
M.A. Hakim Newton; John Levine; Maria Fox; Derek Long
University of Strathclyde

The same as Wizard+FF except that SGPlan (Chen et al. 2006) is used as the base planner instead of FF.

– **Macro-AltAlt**
Murugeswari I; N. S. Narayanaswamy
Indian Institute of Technology Madras

Macro-AltAlt ranks short macro actions by calculating their frequency in the given solution plans. The macros are used indirectly to alter the heuristic function of the heuristic search planner AltAlt (Nigenda et al. 2000). Thus the use of macros is indirect and in some sense this system can be classified with the policy learners of the previous group.

6.3 Sub-problem decomposition learners

There were two variants of a system based on the idea of learning to decompose a problem into subgoals.

– **Divide-and-Evolve (DAE) versions 1 and 2**
Jacques Bibai
Thales Research & Technology

DEA learned knowledge about selecting subgoal decompositions of planning problems. The subgoals are then solved via the CPT planner (Vidal and Geffner 2006). The systems DAE1 and DAE2 are variants on this idea.

6.4 Portfolio configuration

A single system was based on the idea of configuring a portfolio or ensemble of state-of-the-art planners on a per-domain basis.

– **PbP.s**
Beniamino Galvani; Alfonso E. Gerevini; Alessandro Saetti; Mauro Vallati
Università degli Studi di Brescia

PbP.s learns knowledge to configure a portfolio of successful domain-independent planners. The learning algorithm runs and analyzes performance of the planners on the training problems, when running with and without a set of calculated macros. This is used to produce a schedule for running planners optimized to the domain.

6.5 Value function learning

There were two variants of a system based on learning relational value functions, inspired by reinforcement-learning ideas.

– **Relational Function Approximation (RFA1, RFA2)**
Jia-Hong Wu; Robert Givan
Purdue University

RFA1 and RFA2 are two approaches for learning linear value functions for a planning domain and then using those functions as the heuristic for the planner FF.

7 Competition results

At the end of the evaluation phase, we recorded the time and quality metrics over all 180 evaluation problems (30 from each of 6 domains) for each of the 13 systems, both with and without the learned knowledge files. The results of this evaluation are reported below.

Table 1 Relative time scores (see Sect. 4.3) for each planner on each planning domain. The *last column* gives the overall score summed over all problems of all domains (30 problems for each of 6 domains). The maximum score possible by a planner for a domain is 30, which corresponds to solving all 30 problems in a domain faster than any other planner. Thus, the maximum overall score across all 6 domains is 180

Systems	Domains						Overall
	Gold	M-BW	N-Puzzle	Parking	Sokoban	Solitaire	
Cabala	0.00	0.00	0.00	0.00	0.00	0.00	0.00
DEA1	0.01	0.00	0.00	0.00	0.00	0.00	0.01
DEA2	0.01	0.00	0.00	0.00	0.00	0.00	0.01
MacroAltAlt	6.58	5.28	3.91	0.00	0.00	0.00	15.77
ObtuseWedge	9.38	2.03	**29.33**	**28.08**	4.42	3.42	76.65
PbP.s	4.42	**25.85**	7.10	8.96	10.82	**23.02**	**80.16**
Replica	5.14	0.00	0.00	2.42	0.00	0.00	7.56
RFA1	0.00	0.02	0.63	0.70	0.17	10.21	11.73
RFA2	0.29	0.00	0.00	0.56	0.14	1.58	2.57
Roller	6.37	0.00	0.26	2.27	0.00	0.00	8.91
SayphiRules	3.85	0.00	0.09	0.00	0.01	0.00	3.95
Wizard-FF	**24.40**	0.52	2.76	0.85	8.48	8.24	45.25
Wizard-SG	23.25	0.00	4.42	0.00	**26.99**	4.62	59.29

7.1 Overall results and winners

Table 1 gives the time-score results (see Sect. 4.3 for score definition) for all planners when run with the learned control knowledge. Each row corresponds to one of the planning systems and each column corresponds to a planning domain. The table entries give the time score achieved by each planner on each domain and the last column gives the overall time scores for the systems, which is the sum of the scores across each domain. The maximum score possible for a single planning domain is 30, which corresponds to solving all 30 problems in a domain faster than any other planner. *Based on these results, PbP.s was declared the winner with respect to the time metric, achieving a total score of 80.16*. Close behind, in second place, was ObtuseWedge with a score of 76.65, followed by Wizard-SG with a score of 59.29.

Looking at the results for individual planning domains, it is interesting to note that in each domain there is a dominant system—one that achieved close to the maximum score. Wizard-FF and Wizard-SG dominated in GoldMiner and Sokoban respectively; PbP.s dominated in Matching BW and Thoughtful Solitaire; and ObtuseWedge dominated in N-Puzzle and Parking. The existence of "domain winners" suggests that variation among planners can render a planner very effective in some application but not in others. This finding shows that current systems fall short of our goal of wide domain independent applicability. One might hope that our earlier suggestions as to what domains are suitable for what types of learner-planners will be borne out by these results but unfortunately they are not. Understanding what learner-planner to use in what type of application remains an important open problem for future work.

We also notice that, with respect to the time metric, each system performed quite poorly (score close to zero) in at least one domain. This result suggests that there is a high variance in terms of run times even within the competition bound of 15 minutes, leading to highly skewed numbers in the time metric. In order to better understand the results, Table 2 gives

Table 2 Success rates for each planner on each planning domain. The success rate for a planner is the fraction of problems solved in a planning domain. The *last column* gives the overall success rate, which is the fraction of the total of 180 evaluation problems solved by the planner. The maximum possible success rate is 1.0

Systems	Domains						Overall
	Gold	M-BW	N-Puzzle	Parking	Sokoban	Solitaire	
Cabala	0.00	0.07	0.00	0.03	0.00	0.00	0.02
Dae1	**1.00**	0.07	0.00	0.00	0.03	0.00	0.18
Dae2	**1.00**	0.10	0.00	0.00	0.00	0.00	0.18
MacroAltAlt	0.97	0.87	0.67	0.00	0.00	0.00	0.42
ObtuseWedge	0.67	0.27	**1.00**	**1.00**	0.70	0.27	0.65
PbP.s	**1.00**	**0.93**	0.80	0.87	**1.00**	**0.97**	**0.93**
Replica	**1.00**	0.10	0.07	0.70	0.03	0.00	0.32
RFA1	0.30	0.33	0.57	0.53	0.43	0.67	0.47
RFA2	0.47	0.00	0.00	0.40	0.30	0.40	0.26
Roller	**1.00**	0.07	0.17	0.60	0.00	0.00	0.31
SayphiRules	**1.00**	0.00	0.37	0.00	0.20	0.00	0.26
Wizard-FF	**1.00**	0.50	0.50	0.30	0.63	0.47	0.57
Wizard-SG	**1.00**	0.03	0.47	0.00	**1.00**	0.57	0.51

the success rate for each system in each domain. Here, the success rate is simply the fraction of problems solved in a domain by a planner. The last column of the table gives the success rate over all 180 evaluation problems. Based on these results we see that PbP.s was dominant in terms of overall success rate, solving 93% of the evaluation problems. Behind PbP.s were ObtuseWedge and Wizard-FF solving 65% and 57% of the problems, respectively. This shows that the ranking according to the time metric is highly correlated with the ranking according to success rate and thus strengthens its conclusions. Interestingly, while PbP.s and ObtuseWedge were very close in terms of the time score, PbP.s was dominant with respect to the percentage of problems solved. This result indicates that when ObtuseWedge does solve problems it does so quite quickly, making up for the score reduction due to unsolved problems. On the other hand, PbP.s is able to more consistently solve problems within the 15-minute time limit, but perhaps not as quickly.

Table 3 gives the quality-score results (see Sect. 4.3 for score definition) for all planners when run with the learned control knowledge. The maximum score possible for a single planning domain is 30, which corresponds to solving all 30 problems while also achieving the shortest plan length among the other planners. *Based on these results, PbP.s was declared the winner with respect to the quality metric, achieving a total score of 126.68.* In second place was ObtuseWedge with a score of 95.07, followed by Wizard-FF with a score of 91.21. In this case, PbP.s was much more dominating. Looking at the results for individual planning domains, it is interesting to note that while PbP.s dominated in overall score, it never achieved the best score in an individual domain. Comparing to Table 2 we see that its top ranking is due to its robust performance across all of the domains. Since an unsolved problem results in a score of zero, PbP.s faces this score reduction much less often than the other planners.

Finally, it is interesting to consider coverage across all domains, which is implicit in our goal of wide domain independent applicability. From Table 2, we see that PbP.s achieves the top success rate in all domains except for N-puzzle for which ObtuseWedge solves all problems. The robustness of PbP.s is also demonstrated by the fact that its lowest success

Table 3 Relative quality metric (see Sect. 4.3) for each planner on each planning domain. The *right column* gives the overall results summed over all domains. The maximum score possible by a planner for a domain is 30, which corresponds to solving all 30 problems and finding plans that are no longer than any other planner. Accordingly the maximum overall score for the 6 domains is 180

Systems	Domains						Overall
	Gold	M-BW	N-Puzzle	Parking	Sokoban	Solitaire	
Cabala	0.00	1.89	0.00	0.74	0.00	0.00	2.63
DEA1	**28.69**	2.00	0.00	0.00	0.97	0.00	31.66
DEA2	28.41	2.94	0.00	0.00	0.00	0.00	31.35
MacroAltAlt	27.65	**24.12**	19.16	0.00	0.00	0.00	70.92
ObtuseWedge	17.46	5.69	**24.50**	**25.54**	15.27	6.62	95.07
PbP.s	23.96	20.22	17.77	19.48	27.24	18.01	**126.68**
Replica	7.97	1.59	0.38	14.22	0.88	0.00	25.05
RFA1	7.88	4.63	11.25	10.60	10.30	**19.28**	63.93
RFA2	13.45	0.00	0.00	7.95	4.12	10.19	35.70
Roller	7.85	1.10	0.45	15.61	0.00	0.00	25.02
SayphiRules	16.14	0.00	8.78	0.00	4.40	0.00	29.32
Wizard-FF	25.80	12.70	13.17	8.67	17.11	13.76	91.21
Wizard-SG	25.01	0.59	9.82	0.00	**29.40**	16.30	81.12

rate is 87% in N-Puzzle, whereas all other planners have one or more domains where the success rate is quite low. It is notable that only four of the systems (ObtuseWedge, PbP.s, Wizard-FF, RFA1) were able to achieve non-zero success rates across all domains. For all other systems, there was at least one domain where the success rate was zero.

In summary, PbP.s was the overall winner in terms of both the time and quality metrics, as well as far outperforming the other systems in terms of success rate. ObtuseWedge achieved the second best score in all of the categories, followed by either Wizard-FF or Wizard-SG depending on the category.

7.2 Learning versus non-learning

We now consider the impact of learning on the participating systems. Recall that we ran each system twice on each evaluation problem, once with the learned knowledge file (after learning) as input and once without (before learning). This gave us a total of 26 distinct systems for which we computed the relative time metric, relative quality metric and the success rate. Note that for the relative metrics, the scores are recalculated relative to the best system out of the 26 on a per problem basis and are therefore different from the ones given above. Using these new scores, we can compute for each system the difference in the metrics before and after learning in the combined table. Thus, a system is deemed to have improved if the rank of the learning variant in the combined relative score is higher than the rank of the non-learning variant.

We present these results in Table 4. For conciseness, we only provide these results for the five top performing systems from the above results. The full set of results for all systems before and after learning are provided in Tables 5, 6, and 7 in Appendix B.[2] The second

[2]These tables also include results for planners from the non-learning track of the IPC and hence the relative success and timing results are normalized with respect to results from all of those planners, both learning

Table 4 The difference in absolute scores for the top five systems after and before learning. A positive value indicates that the score improved after learning, while a negative value indicates that the score became worse after learning

Systems	Δ Time score	Δ Quality score	Δ Success rate
PbP.s	−5.26	0.20	0.01
ObtuseWedge	**36.05**	**29.02**	**0.17**
Wizard-FF	21.42	−17.33	−0.06
Wizard-SG	17.90	−3.63	−0.02
MacroAltAlt	1.16	−6.80	−0.04

column shows the difference between the overall time metric after learning and the overall time metric before learning. The third and forth columns are similar but for the quality metric and success rate, respectively. A positive value indicates that the system improved the metric value after learning, while a negative value indicates that learning hurt the metric value overall.

The most striking aspect of these results is that in a number of cases we see negative values indicating that learning hurt performance. In particular, the performance of PbP.s decreased with respect to the time metric after learning. Further, we see that there is relatively small improvement for PbP.s after learning compared to before learning for the quality metric (increase by absolute value of 0.20) and success rate (solves 1% more problems). This result indicates that a large part of the success of PbP.s in the competition was not due to the learning, but rather the underlying planning approach. Recall that PbP.s is a portfolio based approach that combines a number of state-of-the-art planners by scheduling their execution times for a given planning problem. It turns out that the default schedule used by PbP.s (the one before learning) works quite well for the evaluation problems. It is important to note that this result is partly due to the fact that PbP.s was at a preliminary stage of development at the time of the competition. More recent work on the PbP system has shown convincingly that it is able to significantly improve its performance after learning (Galvani et al. 2009). We are hopeful that this finding can be validated in a blind test in a future competition, showing that significant improvement due to learning (our main criterion) can be achieved.

On the positive side, we see that ObtuseWedge was able to significantly improve its performance after learning in every category. This included increasing the success rate from 48% to 65%. *Due to this result, ObtuseWedge was awarded the Best Learner Award for its performance in the competition.*

7.3 Comparison to non-learning track planners

In order to help judge the performance of the learning track planners relative to state-of-the-art non-learning planners, we compared to the planners in the non-learning track of IPC 2008. Recall that the non-learning track included PDDL features that were outside of the scope of the learning track. Thus, with the aid of the non-learning track organizers, we were able to run all of the planners on the evaluation problems from the learning track. Since these planners were run on different machines, we did not compare performances of learning and non-learning planners in terms of the time metric. As in the previous comparison, we recalculated the relative scores for all systems together (26 system variants from our competition

track and non-learning track. The differences given in Table 4 are for results that are normalized with respect to only planners in the learning track. Thus the differences in Table 4 do not exactly correspond to those that would be computed from Tables 5 and 7.

plus 11 systems from the non-learning competition). The complete set of results for all of these systems are in Tables 6 and 7 of Appendix B.

The winner of the non-learning track was the Lama planner (Richter and Westphal 2008) and out of the non-learning track planners Lama was the top performer on the learning track problem set. For the quality metric, Lama achieved an overall score of 132.63 while PbP.s after learning achieved a score of 110.47. This result shows that in terms of the quality metric the top state-of-the-art non-learning planner was able to significantly outperform the top learning-based planner. However, in terms of success rate, the story is reversed. Lama achieved a success rate of 79% on the learning track problem set, while PbP.s after learning achieved a success rate of 93%, which is a significant improvement. This result shows that the top performing learning-based planner was able to outperform the top state-of-the-art non-learning planner. However, it is important to recall that PbP.s was able to achieve a 92% success rate before learning, indicating that most of the performance difference between Lama and the competition version of PbP.s was not due to learning.

The second best planner in the non-learning track as evaluated on the learning track problems was FFHA, which achieved a quality score of 63.73 and a success rate of 42%. This low success rate indicates that the domains and problem sets used for the learning track competition were challenging for state-of-the-art planners. Note that PbP.s significantly outperformed FFHA in both metrics. Furthermore, three additional learning systems were able to significantly outperform FFHA in both metrics. In particular, Wizard-FF achieved a quality of 82.2 and success rate of 57%; ObtuseWedge achieved a quality of 80.4 and success rate of 65%; and Wizard-SG achieved a quality of 75.4 and a success rate of 51%. This shows that there were learning-based systems that were able to significantly outperform the vast majority of non-learning systems with respect to evaluation on the learning track problems. However, in all but one case, these learning systems also outperformed FFHA with learning turned off. The exception is ObtuseWedge for the quality metric, which improved from 55.74, without learning, to 80.4, with learning. Thus, the victory over FFHA for these systems cannot be completely attributed to learning, though the margin of victory is increased with learning turned on.

8 Discussion

The learning track of IPC 2008 was the first time that a blind evaluation has been conducted for learning-based planning systems. In particular, the participants were not shown the evaluation domains until after the code freeze of their systems. One cannot underestimate the difficulty of achieving non-trivial results in such a setting. It is extremely promising that we saw clear signs that learning can positively impact planning performance. Most notably, ObtuseWedge demonstrated significant gains in all metrics after learning. In addition, several learning-for-planning systems outperformed all but one of the planners participating in the non-learning competition.

On the negative side, the competition did not provide evidence that learning significantly improved performance compared to the best non-learning planners. In particular, Lama was able to beat all learning-based systems in terms of the quality metric. Further, while PbP.s was the top performer in terms of success rate, the benefit from learning was minimal, improving from 92% to 93%. However, it is important to note that PbP has been improved since the competition and demonstrated improvements upon learning, albeit not in a competition context (Galvani et al. 2009). We also saw in a number of cases that learning can hurt the performance of a system compared to including no learning at all. This finding illustrates the difficulty of achieving robust learning in the context of automated planning.

One observation about the top performing learning systems (PbP.s, ObtuseWedge, and the Wizard systems), is that they were all built on top of state-of-the-art non-learning planners, and PbP.s combines several such planners. This design provided a number of advantages. First, starting with such planners placed these systems at a higher starting point than learning systems that did not build upon state-of-the-art planners. Second, state-of-the-art planners have been highly engineered in terms of efficiency and data structure selection, which can be helpful if competing against a prototype implementation and can clearly affect the results. This barrier can be seen as a further challenge when developing and evaluating learning-for-planning systems because it could limit learning style and opportunities if one wants to succeed in a competition.

Another interesting result of the learning track was that it highlighted the potential effectiveness of portfolio-based techniques, even without the benefit of learning. In particular, for STRIPS planning, it seems likely that such a portfolio-based approach could be very difficult to beat by any individual planner. One reason is that STRIPS planning has been the main focus of automated planning research for many years and there are a wide variety of powerful STRIPS planners. For a particular domain/problem, it is then likely that at least one of those planners will find a solution relatively quickly. A proper portfolio schedule then will be able to exploit this property and perform robustly over a wide range of problems. The success of portfolio-based approaches in a competition setting has also been observed for satisfiability problems (Xu et al. 2007). Further, in any future competition where there is already a wide range of solutions available for the selected competition problems, it seems likely that portfolio techniques will be strong competitors. This success is real and should not be ignored or discounted. In particular, the problem of learning to optimally tune such portfolios to a class of problems for a particular objective is interesting and non-trivial.

On the other hand, the success of portfolio-based approaches is unsatisfying since they work at the meta-level and do not appear to provide deep insights into learning and planning. This point clearly raises a challenge for future efforts to evaluate learning-for-planning systems, because the domain independence requires wide applicability, which is easier to achieve with portfolios, and can therefore mask learning that is useful in some "domain types" but not all. Focusing solely on the larger criterion will not promote such new ideas. One immediate avenue for adjusting the competition to mitigate this concern might be to enrich the class of domains and problems studied to ones where the number of effective existing planners is low, or where existing planners are quite poor. For example, probabilistic planning and temporal planning are much more difficult problems where the state-of-the-art is much less mature. There is simply more room for learning to provide benefit.

This raises the larger issue of evaluating potential competition designs according to how well they match the competition goals. As a starting point, we believe that the practice of conducting a blind evaluation is crucial for competitions, such as ours, where the primary goal is to advance the generality and robustness of learning algorithms. However, this leaves open the important issue of selecting the class of possible evaluation problems, which will be known to the competitors. Competitors have a strong incentive to fully exploit the details of this class—thus, for narrow classes, it is likely that many solutions will be overly-specific with respect to the competition goals. For example, one track of the 2008 RL competition (Whiteson et al. 2010) involved a problem class based on Tetris, where different problems varied in board dimension and the specific reward function. However, the winning entry relied on specialized heuristics that are not likely to be useful for learning in general, which did not align with the competition goals. This led the organizers to suggest that the Tetris domain should be further generalized if used in future competitions (Whiteson et al. 2010). On the other hand, the RL competition also included the Polyathlon track where the problem class was extremely general and could include virtually any RL problem that followed

the state-action-reward interface. Competitors in this track developed very general learning approaches out of necessity. It remains to be seen whether a portfolio-based approach would be effective in future RL Polyathlons.

Thus, picking a problem class at the proper level of generality is one of the key decisions that competition organizers must make. This must take into account the likelihood that competitors will be able to develop overly-specialized solutions for the class, and also the potential for portfolio-based approaches and whether such approaches would contribute to the goals of the competition.

Finally, let us revisit the question that we started with. *Do we have planners that can leverage a learning period to outperform state-of-the-art non-learning planners across a wide range of domains?* Despite the successes observed in the competition, the answer to this question appears to still be "almost but not yet". This finding suggests that another competition similar to the first learning track is warranted, possibly with the inclusion of other tracks that go beyond STRIPS planning. We believe that such efforts will help push learning for planning systems to wider applicability and drive research toward developing a better classification of problems and the solutions appropriate for them.

Acknowledgements We thank Nathan Blackwell for his work in carrying out the evaluations and also Malte Helmert for conducting the evaluations of the non-learning track planners on the learning track problems. We thank all of the participants for their hard work in making this competition a success. We gratefully acknowledge the support of NSF under grants IIS-0546867, IIS-0964457, and IIS-0964705.

Appendix A: STRIPS examples

Below we provide an example domain and problem definition for the Blocksworld in both the typed and un-typed STRIPS language used in the learning track of IPC 2008. The full syntax and semantics of these languages can be found on the competition web-site.

Typed STRIPS Blocksworld example

```
(define (domain typed-blocksworld)
(:requirements :typing)
(:typing block hand)
(:predicates (clear ?b - block)
             (on-table ?b - block)
             (empty ?h - hand)
             (holding ?h - hand ?b - block)
             (on ?b1 ?b2 - block))
(:action pickup
  :parameters (?h - hand ?b - block)
  :precondition (and (clear ?b) (on-table ?b) (empty ?h))
  :effect (and (holding ?h ?b) (not (clear ?b)) (not (on-table ?b))
               (not (empty ?h))))
(:action putdown
  :parameters  (?h - hand ?b - block)
  :precondition (holding ?h ?b)
  :effect (and (clear ?b) (empty ?h) (on-table ?b)
               (not (holding ?h ?b))))
(:action stack
  :parameters  (?h - hand ?b ?underb - block)
  :precondition (and (clear ?underb) (holding ?h ?b))
  :effect (and (empty ?h) (clear ?b) (on ?b ?underb)
               (not (clear ?underb)) (not (holding ?h ?b))))
```

```
(:action unstack
  :parameters  (?h - hand ?b ?underb - block)
  :precondition (and (on ?b ?underb) (clear ?b) (empty ?h))
  :effect (and (holding ?h ?b) (clear ?underb)
               (not (on ?b ?underb)) (not (clear ?b)) (not (empty ?h)))))
(define (problem typed-blocks1)
         (:domain typed-blocksworld)
         (:requirements :typing)
         (:objects H - hand A B C - block)
         (:init (clear A) (on A B) (on B C) (on-table C) (empty H))
         (:goal (and (on C B) (on B A))))
```

Un-typed STRIPS Blocksworld example

```
(define (domain untyped-blocksworld)
(:predicates (clear ?b)
             (on-table ?b)
             (empty ?h)
             (holding ?h ?b)
             (on ?b1 ?b2)
             (hand ?h)
             (block ?b))
(:action pickup
  :parameters (?h ?b)
  :precondition (and (hand ?h) (block ?b) (clear ?b) (on-table ?b)
                     (empty ?h))
  :effect (and (holding ?h ?b) (not (clear ?b)) (not (on-table ?b))
               (not (empty ?h))))
(:action putdown
  :parameters  (?h ?b)
  :precondition (and (hand ?h) (block ?b) (holding ?h ?b))
  :effect (and (clear ?b) (empty ?h) (on-table ?b)
               (not (holding ?h ?b))))
(:action stack
  :parameters  (?h ?b ?underb)
  :precondition (and (hand ?h) (block ?b) (block ?underb)
                     (clear ?underb) (holding ?h ?b))
  :effect (and (empty ?h) (clear ?b) (on ?b ?underb)
               (not (clear ?underb)) (not (holding ?h ?b))))
(:action unstack
  :parameters  (?h ?b ?underb)
  :precondition (and (hand ?h) (block ?b) (block ?underb)
                     (on ?b ?underb) (clear ?b) (empty ?h))
  :effect (and (holding ?h ?b) (clear ?underb)
               (not (on ?b ?underb)) (not (clear ?b)) (not (empty ?h)))))

(define (problem untyped-blocks1)
         (:domain untyped-blocksworld)
         (:objects H A B C)
         (:init (hand H) (block A) (block B) (block C)
                (clear A) (on A B) (on B C) (on-table C) (empty H))
         (:goal (and (on C B) (on B A))))
```

Appendix B: Extended results

This appendix presents the full set of results on the learning-track problems, including the performance of all learning-track planners, both before and after learning, and all planners entered in the non-learning track.

Table 5 Relative time scores (see Sect. 4.3) for each planner after learning (Learning) and before learning (No Learning) on each planning domain. The *last column* gives the overall score summed over all domains. The maximum score possible by a planner for a domain is 30, which corresponds to solving all 30 problems in a domain faster than any other planner

Systems		Domains						Overall
		Gold	M-BW	N-Puzzle	Parking	Sokoban	Solitaire	
Cabala	Learning	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	No Learning	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Dae1	Learning	0.01	0.00	0.00	0.00	0.00	0.00	0.01
	No Learning	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Dae2	Learning	0.01	0.00	0.00	0.00	0.00	0.00	0.01
	No Learning	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MacroAltAlt	Learning	6.37	4.54	1.50	0.00	0.00	0.00	12.41
	No Learning	6.49	3.17	1.53	0.06	0.00	0.00	11.25
ObtuseWedge	Learning	9.33	2.03	14.45	28.00	4.32	3.28	61.40
	No Learning	0.00	0.15	21.80	0.25	0.20	2.94	25.35
PbP.s	Learning	4.31	25.34	2.81	8.95	7.11	21.16	69.68
	No Learning	10.15	1.47	12.02	10.74	18.40	22.17	74.94
Replica	Learning	5.00	0.00	0.00	1.47	0.00	0.00	6.47
	No Learning	0.00	0.00	0.00	0.00	0.00	0.00	0.00
RFA1	Learning	0.00	0.02	0.23	0.70	0.15	9.03	10.12
	No Learning	0.00	0.00	0.00	0.00	0.00	0.00	0.00
RFA2	Learning	0.29	0.00	0.00	0.56	0.12	1.56	2.52
	No Learning	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Roller	Learning	6.21	0.00	0.08	2.27	0.00	0.00	8.56
	No Learning	0.00	0.00	0.00	0.00	0.01	0.00	0.01
SayphiRules	Learning	3.68	0.00	0.03	0.00	0.01	0.00	3.72
	No Learning	17.94	0.00	0.04	0.00	0.00	0.00	17.98
Wizard-FF	Learning	23.90	0.29	0.97	0.85	7.66	7.93	41.60
	No Learning	3.74	0.07	0.89	3.84	7.90	3.74	20.19
Wizard-SG	Learning	22.77	0.00	1.72	0.00	17.84	4.40	46.73
	No Learning	3.19	0.00	1.82	0.01	19.85	3.96	28.84

Table 6 The top part of the table gives the success rates for each learning-track planner after learning (Learning) and before learning (No Learning) on each planning domain. The *bottom part* of the table gives the success rates for the planners in the non-learning track of the IPC. The success rate for a planner is the fraction of problems solved in a planning domain. The *last column* gives the overall success rate, which is the fraction of the total of 180 evaluation problems solved by the planner. The maximum possible success rate is 1.0

Systems		Domains						Overall
		Gold	M-BW	N-Puzzle	Parking	Sokoban	Solitaire	
Cabala	Learning	0.00	0.07	0.00	0.03	0.00	0.00	0.02
	No Learning	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Dae1	Learning	1.00	0.07	0.00	0.00	0.03	0.00	0.18
	No Learning	0.00	0.07	0.00	0.00	0.00	0.00	0.01
Dae2	Learning	1.00	0.10	0.00	0.00	0.00	0.00	0.18
	No Learning	0.47	0.07	0.00	0.00	0.03	0.00	0.09
MacroAltAlt	Learning	0.97	0.87	0.67	0.00	0.00	0.00	0.42
	No Learning	0.97	0.93	0.67	0.17	0.00	0.00	0.46
ObtuseWedge	Learning	0.67	0.27	1.00	1.00	0.70	0.27	0.65
	No Learning	0.33	0.40	1.00	0.43	0.43	0.27	0.48
PbP.s	Learning	1.00	0.93	0.80	0.87	1.00	0.97	0.93
	No Learning	1.00	0.77	0.93	0.83	1.00	0.97	0.92
Replica	Learning	1.00	0.10	0.07	0.70	0.03	0.00	0.32
	No Learning	0.00	0.00	0.00	0.00	0.00	0.00	0.00
RFA1	Learning	0.30	0.33	0.57	0.53	0.43	0.67	0.47
	No Learning	0.00	0.00	0.00	0.00	0.00	0.00	0.00
RFA2	Learning	0.47	0.00	0.00	0.40	0.30	0.40	0.26
	No Learning	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Roller	Learning	1.00	0.07	0.17	0.60	0.00	0.00	0.31
	No Learning	0.00	0.00	0.00	0.00	0.10	0.00	0.02
SayphiRules	Learning	1.00	0.00	0.37	0.00	0.20	0.00	0.26
	No Learning	1.00	0.00	0.37	0.07	0.00	0.00	0.24
Wizard-FF	Learning	1.00	0.50	0.50	0.30	0.63	0.47	0.57
	No Learning	1.00	0.37	0.50	0.83	0.63	0.47	0.63
Wizard-SG	Learning	1.00	0.03	0.47	0.00	1.00	0.57	0.51
	No Learning	0.57	0.10	0.63	0.17	1.00	0.70	0.53
Planners from non-learning track								
Base		0.30	0.33	0.60	0.47	0.47	0.33	0.42
C3		0.70	0.70	0.37	0.00	0.77	0.30	0.47
DAE1		0.00	0.00	0.00	0.00	0.03	0.00	0.01
DAE2		0.33	0.07	0.00	0.00	0.00	0.00	0.07
DTG-Plan		0.00	0.20	0.17	0.00	0.00	0.00	0.06
FFHA		0.63	0.10	0.17	0.77	0.30	0.53	0.42
FFSA		0.63	0.07	0.20	0.87	0.37	0.40	0.42
Lama		0.97	0.83	0.83	0.80	0.67	0.67	0.79
Plan-a		0.00	0.33	0.00	0.00	0.00	0.00	0.06
SGPlan6		0.77	0.07	0.63	0.17	0.00	0.70	0.39
Upwards		0.00	0.00	0.53	0.00	0.00	0.00	0.09

Table 7 The top part of the table gives the relative quality metric (see Sect. 4.3) for each learning-track planner after learning (Learning) and before learning (No Learning) on each planning domain. The *bottom part* of the table gives the relative quality metric for planners from the non-learning track of the IPC. The *right column* gives the overall results summed over all domains. The maximum score possible by a planner for a domain is 30, which corresponds to solving all 30 problems and finding plans that are shorter than any other planner

Systems		Domains						Overall
		Gold	M-BW	N-Puzzle	Parking	Sokoban	Solitaire	
Cabala	Learning	0.00	1.85	0.00	0.74	0.00	0.00	2.59
	No Learning	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Dae1	Learning	28.60	2.00	0.00	0.00	0.97	0.00	31.57
	No Learning	0.00	2.00	0.00	0.00	0.00	0.00	2.00
Dae2	Learning	28.32	2.94	0.00	0.00	0.00	0.00	31.26
	No Learning	13.33	2.00	0.00	0.00	1.00	0.00	16.33
MacroAltAlt	Learning	27.57	20.83	11.82	0.00	0.00	0.00	60.22
	No Learning	27.71	23.21	11.82	3.35	0.00	0.00	66.09
ObtuseWedge	Learning	17.39	4.65	17.37	20.15	14.43	6.42	80.42
	No Learning	8.81	7.17	15.97	6.69	9.69	7.42	55.74
PbP.s	Learning	23.88	16.60	11.83	15.98	26.11	16.06	110.47
	No Learning	23.40	16.08	13.71	16.74	24.39	15.51	109.83
Replica	Learning	7.91	1.59	0.27	11.76	0.85	0.00	22.38
	No Learning	0.00	0.00	0.00	0.00	0.00	0.00	0.00
RFA1	Learning	7.81	3.92	7.43	9.18	9.69	18.53	56.56
	No Learning	0.00	0.00	0.00	0.00	0.00	0.00	0.00
RFA2	Learning	13.37	0.00	0.00	6.77	3.83	10.01	33.99
	No Learning	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Roller	Learning	7.81	1.07	0.33	13.72	0.00	0.00	22.93
	No Learning	0.00	0.00	0.00	0.00	1.44	0.00	1.44
SayphiRules	Learning	16.07	0.00	5.22	0.00	4.20	0.00	25.49
	No Learning	26.98	0.00	5.26	1.61	0.00	0.00	33.85
Wizard-FF	Learning	25.71	10.88	8.22	7.57	16.17	13.62	82.17
	No Learning	30.00	8.50	9.40	20.45	16.17	13.62	98.15
Wizard-SG	Learning	24.92	0.56	5.93	0.00	28.19	15.80	75.41
	No Learning	17.00	1.60	9.11	3.04	28.19	19.15	78.09
Planners from non-learning track								
Base		7.81	5.51	8.43	7.31	10.55	8.92	48.54
C3		20.30	13.34	4.83	0.00	15.88	6.92	61.26
DAE1		0.00	0.00	0.00	0.00	1.00	0.00	1.00
DAE2		9.33	1.96	0.00	0.00	0.00	0.00	11.29
DTG-Plan		0.00	4.62	3.21	0.00	0.00	0.00	7.82
FFHA		17.26	2.27	2.21	20.64	7.46	13.88	63.73
FFSA		17.26	1.30	2.73	21.90	9.44	10.70	63.33
Lama		28.20	21.87	23.50	21.00	18.89	19.17	132.63
Plan-a		0.00	9.87	0.00	0.00	0.00	0.00	9.87
SGlan6		20.98	1.12	6.99	3.02	0.00	19.08	51.19
Upwards		0.00	0.00	7.02	0.00	0.00	0.00	7.02

References

Aler, R., Borrajo, D., & Isasi, P. (2002). Using genetic programming to learn and improve control knowledge. *Artificial Intelligence*, *141*(1–2), 29–56.

Bjarnason, R., Tadepalli, P., & Fern, A. (2007). Searching solitaire in real time. *ICGA Journal*, *30*(3), 131–142.

Blockeel, H., & De Raedt, L. (1997). Top-down induction of first order logical decision trees. *Artificial Intelligence*, *101*, 285–297.

Bonet, B., & Geffner, H. (2003). Labeled RTDP: improving the convergence of real-time dynamic programming. In *Proceedings of the international conference on automated planning and scheduling* (pp. 12–31).

Boutilier, C., Reiter, R., & Price, B. (2001). Symbolic dynamic programming for first-order MDPs. In *Proceedings of the international joint conference of artificial intelligence* (pp. 690–700).

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

Dzeroski, S., De Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning Journal*, *43*, 7–52.

Estlin, T. A., & Mooney, R. J. (1996). Multi-strategy learning of search control for partial-order planning. In *Proceedings of the thirteenth national conference on artificial intelligence* (pp. 843–848).

Fern, A., Yoon, S., & Givan, R. (2006). Approximate policy iteration with a policy language bias: solving relational Markov decision processes. *Journal of Artificial Intelligence Research*, *25*, 85–118.

Fikes, R., & Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence Journal*, *2*(3/4), 189–208.

Fikes, R. E., Hart, P. E., & Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence Journal*, *3*(1–3), 251–288.

Galvani, B., Gerevini, A., Saetti, A., & Vallati, M. (2009). A planner based on an automatically configurable portfolio of domain-independent planners with macro-actions: PbP. In *International conference on automated planning and scheduling*.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*, 263–302.

Huang, Y.-C., Selman, B., & Kautz, H. (2000). Learning declarative control rules for constraint-based planning. In *Proceedings of seventeenth international conference on machine learning* (pp. 415–422).

Iba, G. (1989). A heuristic approach to the discovery of macro-operators. *Machine Learning*, *3*(4), 285–317.

Joshi, S., & Khardon, R. (2008). Stochastic planning with first order decision diagrams. In *Proceedings of the international conference on automated planning and scheduling*.

Joshi, S., Kersting, K., & Khardon, R. (2010). Self-Taught decision theoretic planning with first-order decision diagrams. In *Proceedings of the international conference on automated planning and scheduling* (pp. 89–96).

Junghanns, A., & Schaeffer, J. (2001). Sokoban: enhancing general single-agent search methods using domain knowledge. *Artificial Intelligence*, *129*(1–2), 219–251.

Kersting, K., van Otterlo, M., & De Raedt, L. (2004). Bellman goes relational. In *Proceedings of the international conference on machine learning*.

Khardon, R. (1999). Learning action strategies for planning domains. *Artificial Intelligence*, *113*(1–2), 125–148.

La Rosa, T., García Olaya, A., & Borrajo, D. (2007). Using cases utility for heuristic planning improvement. In *Proceedings of the seventh international conference on case based reasoning*.

Martin, M., & Geffner, H. (2000). Learning generalized policies in planning domains using concept languages. In *Proceedings of seventh international conference on principles of knowledge representation and reasoning*.

McDermott, D. (1998). PDDL—the planning domain definition language. In *The 1st international planning competition*.

Minton, S. (1988). Quantitative results concerning the utility of explanation-based learning. In *Proceedings of national conference on artificial intelligence*.

Minton, S. (Ed.) (1993). *Machine learning methods for planning*. San Mateo: Morgan Kaufmann.

Minton, S., Carbonell, J. G., Knoblock, C. A., Kuokka, D., Etzioni, O., & Gil, Y. (1989). Explanation-based learning: a problem solving perspective. *Artificial Intelligence*, *40*, 63–118.

Nigenda, R., Nguyen, X., & Kambhampati, S. (2000). AltAlt: combining the advantages of graphplan and heuristic state search. In *International conference on knowledge-based computer systems*. Citeseer.

Puterman, M. L. (1994). *Markov decision processes: discrete dynamic stochastic programming*. New York: Wiley.

Richter, S., & Westphal, M. (2008). The Lama planner using landmark counting in heuristic search. In *Proceedings of the international planning competition*.

Sanner, S., & Boutilier, C. (2009). Practical solution techniques for first-order MDPs. *Artificial Intelligence*, *173*, 748–788.

Sutton, R., & Barto, A. (1998). *Reinforcement learning: an introduction*. Cambridge: MIT.

Tadepalli, P., Givan, B., & Driessens, K. (2004). Relational reinforcement: an overview. In *Proceedings of the workshop on relational reinforcement learning*, Banff, Canada, International conference on machine learning.

Taylor, M. E., & Stone, P. (2009). Transfer learning for reinforcement learning domains: a survey. *J. Mach. Learn. Res.*, *10*, 1633–1685.

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

Wang, C., & Khardon, R. (2007). Policy iteration for relational MDPs. In *Proceedings of the workshop on uncertainty in artificial intelligence*.

Wang, C., Joshi, S., & Khardon, R. (2008). First-Order decision diagrams for relational MDPs. *Journal of Artificial Intelligence Research*, *31*, 431–472.

Whiteson, S., Tanner, B., & White, A. (2010). *AI Magazine*, *31*(2), 81–94.

Xu, L., Hutter, F., Hoos, H., & Leyton-Brown, K. (2007). SATzilla-07: the design and analysis of an algorithm portfolio for SAT. In *Lecture Notes in Computer Science* (Vol. 4741, p. 712).

Yoon, S., Fern, A., & Givan, R. (2002). Inductive policy selection for first-order MDPs. In *Proceedings of eighteenth conference in uncertainty in artificial intelligence*.

Yoon, S., Fern, A., & Givan, R. (2006). Learning heuristic functions from relaxed plans. In *International conference on automated planning and scheduling (ICAPS)*.

Younes, H. (2003). Extending PDDL to model stochastic decision processes. In *Proceedings of the ICAPS-03 workshop on PDDL*.

Zimmerman, T., & Kambhampati, S. (2003). Learning-assisted automated planning: looking back, taking stock, going forward. *AI Magazine*, *24*(2), 73–96.