International Journal on Artificial Intelligence Tools

ASAP: AN AUTOMATIC ALGORITHM SELECTION APPROACH FOR PLANNING

MAURO VALLATI
LUKÁŠ CHRPA
DIANE KITCHIN
PARK Research Group, School of Computing and Engineering, University of Huddersfield, Queensgate
Huddersfield, HD1 3DH, United Kingdom
{*n.surname*}*@hud.ac.uk*

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Despite the advances made in the last decade in automated planning, no planner outperforms all the others in every known benchmark domain. This observation motivates the idea of selecting different planning algorithms for different domains. Moreover, the planners' performances are affected by the structure of the search space, which depends on the encoding of the considered domain. In many domains, the performance of a planner can be improved by exploiting additional knowledge, for instance, in the form of macro-operators or entanglements.

In this paper we propose ASAP, an automatic Algorithm Selection Approach for Planning that: (i) for a given domain initially learns additional knowledge, in the form of macro-operators and entanglements, which is used for creating different encodings of the given planning domain and problems, and (ii) explores the 2 dimensional space of available algorithms, defined as encodings–planners couples, and then (iii) selects the most promising algorithm for optimising either the runtimes or the quality of the solution plans.

Keywords: Automated Planning; Algorithm Selection; Learning for Planning.

1. Introduction

Although in the last decade the performance of domain-independent planners has significantly improved, there is no planner that outperforms all others in every benchmark domain. The performance of current planning systems is typically affected by the structure of the search space, which depends on the planning domain and its considered encoding. In many domains, the planning performance can be improved by deriving and exploiting knowledge about the domain and problem structure that is not explicitly given in the input formalisation, and that can be used for optimising planners' behaviour. Such knowledge can be then encoded in the domain model which allows using standard planning engines. In other words, the domain (and problem) model is reformulated [13].

These observations motivate the idea of extracting additional knowledge about planning domains and automatically selecting the most promising planning algorithm, exploiting such knowledge, for a given domain. The problem of algorithm selection has been introduced by Rice in 1970s [28] hence this research area is not very new. In planning, algorithm selection can help to overcome situations where some algorithm does not work very well on some planning problems while some other algorithm works well on them. The algorithm selection problem can also be viewed as a decision at the meta-level [11]. Although this paper will present an algorithm selection technique which selects an algorithm for a whole class of problems (in the same domain), we are aware of works that study more specific algorithm selection. One possibility is to select an algorithm for a single problem rather than a class of problems [38]. Another interesting aspect of algorithm selection in planning is that we might not be bounded by selecting one algorithm to solve the whole problem but we might change the algorithm at some point during solving the problem [24].

In this paper we propose ASAP, an automatic algorithm selection approach for planning that: (i) for a given domain initially learns additional knowledge, in the form of macro-operators and entanglements (inner and outer), which is used for creating different encodings of the given planning domain and problems (i.e. planning domain/problem reformulation); (ii) explores the 2 dimensional space encodings (e)–planners (p); and then (iii) selects the best algorithm $\langle e, p\rangle$ for optimising the runtimes (ASAPs) or the quality of the solution plans (ASAPq).

In the proposed approach, each *algorithm* has two dimensions: one dimension is represented by different encodings of a given domain; the other is represented by existing high-performance domain-independent planners. We decided to consider each couple $\langle e, p\rangle$ as a different algorithm because the different knowledge carried in the generated encodings, e, makes even the same planner p perform very differently.

We are not aware of other completely automated planning systems exploiting a *pure* algorithm selection approach, in the sense that they automatically select a single algorithm for solving a specific class of planning problems. If we include the portfolio-based approach for planning, which can be considered as a superset of the algorithm selection one, our approach is related to the work of Roberts and Howe [20,32], PbP2 [15,16] and Fast Downward Stone Soup [35], with some significant differences.

The major difference between all the approaches above and ASAP is that we made a domain-specific selection of a single algorithm, which is defined by a couple encoding–planner. Moreover, the Roberts and Howe approaches select the planners to exploit online, while we select the algorithm offline. Additionally, the knowledge generated by the Roberts and Howe systems is domain-independent, while the knowledge generated and exploited by ASAP is domain-specific.

PbP2 learns a domain-specific portfolio. It incorporates seven planners it can choose from. It lets them learn macro-actions for the given domain, and runs up to three best-performing ones in a round-robin fashion with learnt time slots. What differentiates our approach from PbP2, is that (i) we generate new encodings of given domains by looking for both macro-operators and entanglements, (ii) we explore the two-dimensional algorithm space encodings–planners, and (iii) we select only one algorithm to exploit on a domain.

Fast Downward Stone Soup is a recent approach to selecting and combining a set of forward-state planning techniques included in the well known domain-independent planner Fast Downward [17]. Their approach is domain-independent, thus it does not extract any additional knowledge from the planning domains (e.g. macro-operators or entanglements). It exploits a statical combination of several different planning techniques for solving a single problem.

In the rest of the paper, first we give the necessary background on classical planning, problem reformulations and considered planning systems. Then we describe the ASAP approach; we present and discuss the experimental results and finally; we give conclusions.

2. Classical Planning

Classical planning deals with finding a (partially or totally ordered) sequence of actions transforming the static, deterministic and fully observable environment from some initial state to a desired goal state. In the classical representation *atoms* are predicates. *States* are defined as sets of ground predicates. A *planning operator* $o = (\mathit{name}(o), \mathit{pre}(o), \mathit{eff}^-(o), \mathit{eff}^+(o))$ is specified such that $\mathit{name}(o) = \mathit{op\_name}(x_1, \ldots, x_k)$ ($\mathit{op\_name}$ is an unique operator name and $x_1, \ldots x_k$ are variable symbols (arguments) appearing in the operator), $\mathit{pre}(o)$ is a set of predicates representing operator's precondition, $\mathit{eff}^-(o)$ and $\mathit{eff}^+(o)$ are sets of predicates representing operator's negative and positive effects. *Actions* are ground instances of planning operators. An action $a = (\mathit{pre}(a), \mathit{eff}^-(a), \mathit{eff}^+(a))$ is *applicable* in a state s if and only if $\mathit{pre}(a) \subseteq s$. Application of a in s (if possible) results in a state $(s \setminus \mathit{eff}^-(a)) \cup \mathit{eff}^+(a)$.

A *planning domain* is specified via sets of predicates and planning operators. A *planning problem* is specified via a planning domain, initial state and set of goal atoms. A *solution plan* is a sequence of actions such that a consecutive application of the actions in the plan (starting in the initial state) results in a state that satisfies the goal.

3. Planning Problem Reformulations

Analogously to the possibility that a planning system can be implemented in many different ways, so planning domains and problems can be also encoded in several different ways. Typically, environment and action descriptions correspond with real situations which produces useful outputs for agents (or robots) that they can easily execute. On the other hand, sometimes such an encoding is not very efficient and therefore some additional planner independent knowledge (e.g. macro-operators) is often included to increase the efficiency of planning engines.

As a running example we use the well known BlocksWorld domain. It consists of four operators: pickup(?x) refers to a situation when a robotic hand picks-up a block ?x from the table, putdown(?x) refers to a situation when a robotic hand puts-down the block ?x it is holding to the table, unstack(?x,?y) refers to a situation when a robotic hand unstacks a block ?x from ?y, and stack(?x,?y) refers to a situation when a robotic hand stacks a block ?x to ?y.

3.1. *Macro-operators*

A macro-operator encapsulates a sequence of (primitive) planning operators and can be represented as an ordinary planning operator. In BlocksWorld, it may be observed that instances of the operator unstack(?x ?y) are followed by instances of the operator putdown(?x). Hence, it is reasonable to assemble these operators into a macro-operator unstack-putdown(?x ?y). Formally speaking, a macro-operator $o_{i,j}$ is constructed by assembling planning operators o_i and o_j (in that order) in the following way:

- $pre(o_{i,j}) = pre(o_i) \cup (pre(o_j) \setminus eff^+(o_i))$
- $eff^-(o_{i,j}) = (eff^-(o_i) \setminus eff^+(o_j)) \cup eff^-(o_j)$
- $eff^+(o_{i,j}) = (eff^+(o_i) \setminus eff^-(o_j)) \cup eff^+(o_j)$

Clearly, o_i must not delete any predicate required by o_j, otherwise corresponding instances of o_i and o_j cannot be applied consecutively. Longer macro-operators, i.e., those encapsulating longer sequences of original planning operators can be constructed by this approach iteratively.

Macro-operators can be thus understood as 'short-cuts' in the state space, which in some cases can speed up plan generation considerably [4, 27]. This property can be useful since by exploiting them it is possible to reach the goals in fewer steps. However, the number of instances of macro-operators is often higher than the number of instances of the original operators, because they usually have more arguments deriving from the arguments of the operators that are encapsulated. This increases the branching factor in the search space, which can slow down the planning process and, moreover, increase the memory consumption. Therefore, it is important that benefits of macro-operators outweigh their drawbacks. This problem is known as the *utility problem* [26].

Macro-operators, since they are encoded in the same way as 'normal' planning operators, can be added into planning domains and reformulated domains can be passed to any planning engine. Therefore, macro-operators are thoroughly studied [4,8,27]. For ASAP we used Chrpa's macro-operator learning approach [8]. This approach learns macro-operators from training plans by considering both adjacent actions, and non-adjacent actions which can be made adjacent by permutating the training plans (clearly the permutations considered must preserve the soundness of the plan). Macro-operators are generated according to several criteria such as whether instances of one operator frequently follows (or precedes) instances of the other operator, and whether the number of the macro-operator's arguments is small. To raise the efficiency of the planning process the approach also removes some primitive operators replaced by newly generated macro-operators. In our example, when the new macro-operator unstack-putdown(?x ?y) is created, then it may be observed that the primitive operator putdown(?x) is useless (unless an initial state consists of a situation where the robotic hand holds some block). Although in general this approach is incomplete (we might lose solvability of some problems), it has been demonstrated empirically that this approach works well [8].

3.2. *Entanglements*

Entanglements [6] are relations between planning operators and atoms (predicates). Entanglements aim to capture the causal relationships characteristic for a given class of planning problems which in many cases enable a reduction of the branching factor in the state space. There are two kinds of entanglements, outer and inner entanglements.

Outer entanglements [7] are relations between planning operators and initial or goal atoms (predicates) which refers to situations where to solve a given planning problem we need only such instances of operators where instances of a certain predicate in an operator's precondition or positive (add) effects respectively are present in the initial state or goal situation respectively. In BlocksWorld, it can be observed that unstacking blocks only occurs from their initial positions. In this case an 'entanglement by init' will capture that if an atom on(a b) is to be achieved for a corresponding instance of operator unstack(?x ?y) (unstack(a b)), then the atom is an initial atom. Similarly, it may be observed that stacking blocks only occurs to their goal positions. Then, an 'entanglement by goal' will capture that atom on(b a) achieved by a corresponding instance of operator stack(?x ?y) (stack(b a)) is a goal atom.

Encoding outer entanglements is done by introducing supplementary static predicates having the same arguments as predicates involved in the outer entanglement relations. Instances of these static predicates that correspond to instances of original predicates in the initial or goal state, are added to the initial state. These supplementary static predicates are added into preconditions of operators involved in the outer entanglement relations, so they allow only such operators' instances that follow conditions of outer entanglements. For detail, see [6].

Inner entanglements [6] are relations between pairs of planning operators and predicates which refer to situations where one operator is an exclusive 'achiever' or 'consumer' of a predicate to or from another operator. In the BlocksWorld it may be observed that operator pickup(?x) achieves predicate holding(?x) exclusively for operator stack(?x,?y) (and not for operator putdown(?x)), i.e., pickup(?x) is 'entangled by succeeding' stack(?x,?y) with holding(?x). Similarly, it may be observed that predicate holding(?x) for operator putdown(?x) is exclusively achieved by operator unstack(?x ?y) (and not by operator pickup(?x)), i.e., putdown(?x) is 'entangled by preceding' unstack(?x ?y) with holding(?x).

Encoding inner entanglements into planning domains and problems must ensure 'achiever' and 'requirer' exclusivity given by these inner entanglements. It is done by using specific predicates, 'locks', which prevent executing certain instances of operators in some stage of the planning process. An instance of an operator having a 'lock' in its precondition cannot be executed after executing an instance of another operator ('locker') having a 'lock' in its negative effects until an instance of some other operator ('releaser') having a 'lock' in its positive effects has been executed. For example, a situation where `pickup(?x)` is 'entangled by succeeding' `stack(?x,?y)` with `holding(?x)` is modelled such that `pickup(?x)` is a 'locker' for `putdown(?x)` and `stack(?x,?y)` is a 'releaser' for `putdown(?x)`. For details, see [6].

Entanglements are extracted from training plans, solutions of simpler planning problems, by checking whether for each operator/pair of operators and related predicates the entanglement conditions are satisfied in all the training plans. Since training plans may consist of 'flaws' (e.g. redundant actions) some error rate (often up to 10%) is allowed. Introducing error rate might result in detecting incorrect entanglements even for the training problems. Hence, the extracted entanglements are cross-validated on the (reformulated) training problems. This approach, however, does not guarantee completeness since even if extracted entanglements are valid for all the training problems, they might be invalid for some other problems in the same domain. In spite of this, it has been empirically shown that it does not happen (or happens very rarely) if the structure of the training problems is similar to the structure of testing ones [6].

Entanglements, as mentioned before, aim to reduce the branching factor in the search space by eliminating unpromising instances of planning operators or search alternatives. This has clearly a positive influence on the planning process. However, introducing supplementary predicates, especially those for encoding inner entanglements, may introduce overheads. Eliminating of some actions or search alternative might act adversarially against some planning techniques, for instance, delete-relaxed heuristics [19].

4. Integrated Planners

In this section, we give a brief description of each of the high performance domain-independent planners that are currently incorporated in ASAP. Much more detailed information are available from the referred papers.

Such systems were selected due to their good performances in International Planning Competitions (IPC) and the different techniques that they exploit.

Metric-FF [18] extends the main ideas used in FF [19]. The FF's search strategy is a variation of hill-climbing over the space of the world states, and in FF the goal distance is estimated by solving a relaxed task for each successor world state. Compared to the first version of FF, Metric-FF is enhanced with goal orderings pruning technique and with the ordering knowledge provided by the goal agenda.

SatPlan [23] computes plans by encoding the planning problems into SAT problems that are then solved by a generic SAT solver. It first estimates an initial bound k on the length (or horizon) of the plan, then it encodes the planning problem with horizon equal to k into a CNF formula, and finally uses an incorporated SAT solver to solve the CNF formula. If the formula is satisfiable, the solution to the SAT problem is translated into a plan, otherwise, SatPlan generates another CNF using the current planning horizon increased by one (i.e. $k + 1$), and so on. In this work, SatPlan exploited the Precosat [1] SAT solver and the SAT-MAX-PLAN encoding [34], which is used for translating a planning problem into a SAT formula.

Mp [31] is a SAT-based planner that, differently from SatPlan, exploits an extremely compact SAT representation of the planning problem and an integrated SAT solver. Mp generates different SAT formulae, corresponding to different horizons. Such formulae are then solved in parallel, through a finely tuned scheduling technique, by the integrated solver. As soon as a formula is satisfied, all the process is stopped and the current solution plan is provided.

Lama-11 [29, 30] translates the PDDL problem specification into a multi-valued state variable representation ("SAS+") and searches for a plan in the space of the world states using a heuristic derived from the causal graph, a particular graph representing the causal dependencies of SAS+ variables. Its core feature is the use of a pseudo-heuristic derived from landmarks, propositions that must be true in every solution of a planning task. Moreover, a weighted A^* search is used with iteratively decreasing weights, so that the planner continues to search for plans of better quality.

LPG [14] uses stochastic local search in a space of partial plans represented through *linear action graphs*, which are variants of the very well known planning graph [2]. The search steps are certain graph modifications transforming an action graph into another one. LPG is an incremental planner that exploits restarts and noise for improving the quality of solutions found.

SGPlan [5] exploits a partition-and-resolve strategy to partition the mutual-exclusion constraints of a planning problem by its subgoals into subproblems. The subproblems are then individually solved using Metric-FF. All the subproblems solutions are then merged in order to solve the original planning problem; the violated global constraints are iteratively resolved while merging solutions.

Probe [25] implements a dual search architecture for planning that is based on the idea of *probes*: single action sequences computed without search from a given state that can quickly go deep into the state space, terminating either in the goal or in failure. It is a complete standard greedy best first search planner using the additive heuristic [3] with a single change: when a state is selected for expansion, it first launches a "probe" from the state to the goal. If the probe reaches the goal, the problem is solved and the solution is returned. Otherwise, the states expanded by probe are added to the open list, and control returns to the main loop.

5. The Proposed Approach

It is well known that different planning engines might have very different performance, and none of the existing planning engines outperforms the rest in all the (existing) domains. The same can be said about different domain encodings, since they have their advantages and peculiarities (as discussed before). The rationale behind ASAP is to select the best couple - planner, encoding - for each domain in order to (nearly) maximally exploit advantages of both.

The learning phase of ASAP is composed of four steps: (i) extraction of macro-operators and removal of useless primitive operators, (ii) detection of entanglements and encoding them into new planning domains/problems, (iii) generation of all the algorithms as couples $\langle e, p\rangle$, (iv) measurement of the performances of the available algorithms, and (v) selection of the most promising algorithm for solving the testing instances.

Macro-operators and entanglements are extracted using the approach described, respectively, in [8] and [6] on plans generated by Metric-FF, exploiting the original domain encodings, on training problems. Through these techniques ASAP is able to generate at most four new encodings per domain: Macros, which includes macro-operators and excludes some original operators; Inner, which includes inner entanglements; Outer, which includes outer entanglements; Both, which considers both inner and outer entanglements. The maximum number of algorithms per domain is 35, which arises from 7 included planners that can be used with 5 different encodings.

The current version of ASAP runs the available algorithms on training problems. The performances are measured in terms of CPU time required for solving each training instance, number of actions of the solutions found, and the number of solved problems. The performances of each algorithm $\langle e, p\rangle$ are then compared in order to select the most promising one to execute on testing problems. ASAP has two different versions: ASAPs which selects the algorithms for optimising runtimes, and ASAPq which optimises the quality of the plans (in classical STRIPS planning the quality is measured by plan length, i.e., shorter better).

For selecting the most promising algorithm in terms of runtime, ASAPs uses the time IPC score. It is a value, firstly introduced in IPC-6 [12], which considers runtimes and number of solved problems together. It is very useful because it synthesises different aspects of planners' performance in a single value, that can then be compared through different planners. ASAPs selects the couple which achieved the best IPC score on the learning problems; if more algorithms achieved the same

Table 1. For every domain, the couple selected by ASAPs and ASAPq, and the total number of available algorithms.

Domain	ASAPs	ASAPq	Total
Barman	$\langle$Outer, SGPlan$\rangle$	$\langle$Original, SGPlan$\rangle$	35
BlocksWorld	$\langle$Both, FF$\rangle$	$\langle$Both, FF$\rangle$	35
Depots	$\langle$Outer, FF$\rangle$	$\langle$Both, LPG$\rangle$	35
Gripper	$\langle$Outer, SGPlan$\rangle$	$\langle$Outer, Mp$\rangle$	14
Gold Miner	$\langle$Macro, FF$\rangle$	$\langle$Both, SatPlan$\rangle$	35
Matching-BW	$\langle$Macro, LPG$\rangle$	$\langle$Outer, LPG$\rangle$	35
Parking	$\langle$Original, FF$\rangle$	$\langle$Inner, Lama$\rangle$	21
Rovers	$\langle$Macro, LPG$\rangle$	$\langle$Macro, Lama$\rangle$	21
Satellite	$\langle$Outer, LPG$\rangle$	$\langle$Outer, LPG$\rangle$	21
Spanner	$\langle$Outer, Mp$\rangle$	$\langle$Original, LPG$\rangle$	14
TPP	$\langle$Both, LPG$\rangle$	$\langle$Outer, Lama$\rangle$	35

score some secondary criteria are used. These criteria include the number of solved problems, the number of problems in which the couple has been the fastest and the mean CPU time on solved problems.

The method used by ASAPq is similar, but it is considering the quality of plans (in terms of number of actions) instead of the CPU times. For the incremental planners[a], i.e. LPG and Lama, the best solution found within the CPU time limit is considered.

The time and quality IPC score are determined as defined for IPC-7 [10]. The time score of an algorithm $\mathcal{A}$ for a planning problem P is defined as $Score(\mathcal{A}, P)$, which is 0 if P is unsolved, and $1/(1 + \log_{10}(T_P(\mathcal{A})/T_P^*))$ otherwise, where T_P^* is the lowest measured CPU time to solve P and $T_P(\mathcal{A})$ denotes the CPU time required by $\mathcal{A}$ to solve problem P. Higher values of the speed score indicate better performance. The quality score is defined as $Score(\mathcal{A}, P)$, which is 0 if P is unsolved, and $Q_P^*/Q(\mathcal{A}_P)$ otherwise ($Q_P^* \leq Q(\mathcal{A}_P)$ for any $\mathcal{A}$). Quality is measured in terms of number of actions. The IPC score on a set of problems is given by the sum of the scores achieved on each considered problem.

6. Experimental Analysis

In this section, we present the results of a large experimental study examining the *effectiveness* of the knowledge generated and exploited by ASAP under the form of selected algorithms $\langle e, p\rangle$.

6.1. *Experimental Setup*

We considered problem instances from 11 well-known benchmark domains used in the learning tracks of International Planning Competitions: Barman (IPC7), BlocksWorld (IPC-7), Depots (IPC-7), Gripper (IPC-7), Gold-miner (IPC-6),

[a]An incremental planner produces a sequence of solutions with increasing plan quality which are generated with increasing CPU times.

Matching-BW (IPC-6), Parking (IPC-6/7), Rovers (IPC-7), Satellite (IPC-7), Spanner (IPC-7) and TPP (IPC-7). These domains were selected because they are suitable for reformulations. Some domains used in learning tracks are not suitable for extracting additional knowledge in the form of macros or entanglements since they have a very small number of operators (1 or 2). In other domains, since Metric-FF was not able to solve any training problem, ASAP was not able to derive any type of knowledge. In such domains, the comparison would be only between basic solvers, which is not the focus of this paper.

For each domain, we used the existing 30 benchmark problems as testing instances. As training problems we used 30 training problems from those provided by organisers, whenever available; otherwise we generated circa 30 instances (easier than the ones used for testing the approach) through available random generators.

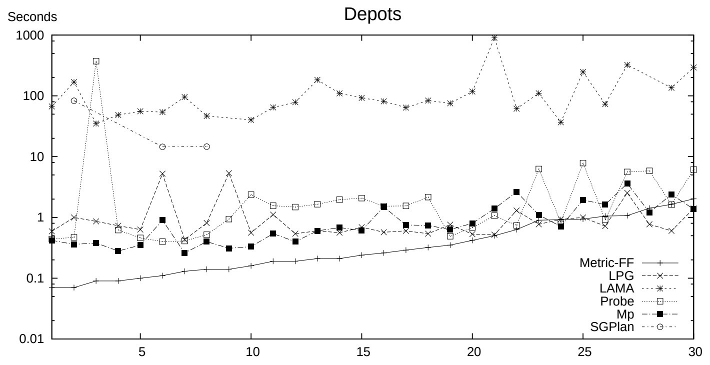
A runtime cutoff of 900 CPU seconds (15 minutes, as in the learning tracks of IPC) was used for both learning and testing runs. All the experiments were run on 3.0 Ghz machine CPU with 2GB of RAM.

6.2. *Results on Selected Domains*

Table 1 shows, for every domain, the algorithm selected by ASAPs and ASAPq, and the total number of available algorithms. It is interesting to note that the selected domains encodings changes frequently through benchmark domains. On the other hand, the planners most frequently included in the couples selected by ASAPs are Metric-FF and LPG. On the contrary, for optimising the quality, almost all the planners have been selected. The total number of available algorithms can be smaller than 35 in the case that macro-operators or some type of entanglements were not found.

Figure 1 shows results for a two-dimensional comparison, in terms of runtimes, done on the testing problems of Depots. In the top chart we are comparing the CPU times of all the algorithms sharing the same encoding (e) of the selected one, but exploiting different planners. SatPlan is not shown since it does not solve any testing problem. In the bottom chart we are comparing the algorithms sharing the same planner (p) but with different encodings. The impact of both the dimensions considered by the proposed approach is significant in terms of CPU time, and we experimentally observed that it is significant also in terms of quality of the solutions found.

For understanding the usefulness of the knowledge extracted under the form of domain encodings, for verifying the hypothesis that no single planner outperforms all the others in every considered benchmark domain, and for verifying that ASAP effectively selects the most promising algorithm for each selected domain, we have compared ASAP with the best performing basic planner for each considered domain. The best basic planner is exploiting the original domain formulation, and has been selected according to the IPC score achieved on the testing problems. The results of this experiment are shown in Table 2. The first interesting result is that both



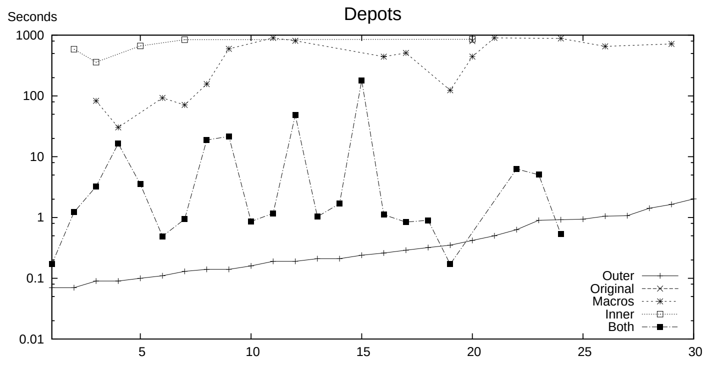


Fig. 1. CPU time (log. scale) of the selected couple with respect to the couples exploiting the same encoding (upper plot) and couples exploiting the same planner (lower plot) on benchmark problems of Depots domain. Note that the X-axes denote the problem number.

ASAPs and ASAPq have significantly better performance in terms of IPC score and number of solved problems, while considering all the selected domains together. While analysing the results for every single domain, ASAPs is almost always better than the best basic planner. In Parking the performance are exactly the same, ASAPs selected exactly FF and the original domain as the algorithm to exploit. Only in Barman and Spanner, the performance of ASAPs are worse than those of the best basic planner. In both the domains, this is due to the different behaviour of selected algorithms on training and testing instances. In Spanner, Mp is extremely quick on training instances, but on large testing problems it runs out of memory. In Barman, outer entanglements showed to be useful on smaller training problems,

Table 2. Time/quality IPC score (max score 30 per domain) and percentages of problems solved by ASAP and the best basic planner for the selected domains. BW, Gold-m and M-BW stand respectively for BlocksWorld, Gold-miner and Matching-BW.

Domain	BestS	Time IPC		% Solved	
		ASAPs	BestS	ASAPs	BestS
Barman	SGPlan	23.1	**28.1**	100.0	100.0
BW	Probe	**30.0**	5.8	**100.0**	66.7
Depots	Probe	**30.0**	8.7	100.0	100.0
Gripper	LPG	**30.0**	5.0	**100.0**	33.3
Gold-m	Mp	**30.0**	16.5	100.0	100.0
M-BW	Lama	**30.0**	8.7	**100.0**	80.0
Parking	FF	5.0	5.0	16.7	16.7
Rovers	LPG	**28.0**	22.0	93.3	93.3
Satellite	LPG	**29.0**	27.8	96.7	**100.0**
Spanner	LPG	10.6	**25.6**	36.7	**100.0**
TPP	Lama	**20.0**	3.4	**66.7**	33.3
All above		**265.7**	156.6	**82.7**	74.8

Domain	BestQ	Quality IPC		% Solved	
		ASAPq	BestQ	ASAPq	BestQ
Barman	SGPlan	30.0	30.0	100.0	100.0
BW	Probe	**29.5**	18.1	**100.0**	66.7
Depots	Probe	**30.0**	26.2	100.0	100.0
Gripper	LPG	**30.0**	8.3	**100.0**	33.3
Gold-m	LPG	**30.0**	29.9	100.0	100.0
M-BW	SatPlan	**26.4**	20.4	**90.0**	73.3
Parking	FF	3.8	**4.6**	13.3	**16.7**
Rovers	LPG	**29.3**	26.1	**100.0**	90.0
Satellite	LPG	28.8	**29.7**	96.7	**100.0**
Spanner	LPG	30.0	30.0	100.0	100.0
TPP	Lama	**30.0**	9.1	**100.0**	33.3
All above		**297.8**	232.4	**90.9**	73.9

but their usefulness has not been confirmed on the testing set.

ASAPq is worse in two of the selected domains; in Parking and in Satellite, mainly due to the smaller number of problems solved by the selected algorithms. Another interesting result that we can derive from Table 2 is that every considered planner achieves the best Time/Quality IPC score on at least one considered domain, and that the single best planner of a domain is usually different than the one included in the couple selected by ASAP. This means that entanglements and macro-operators have a very significant impact on the planners' performance, and that the impact varies notably from planner to planner.

In order to understand the accuracy of the algorithm selection, we compared the performance of the three best algorithms. In Tables 3 and 4 we present the results of this comparison, in terms of time and quality IPC score, that has been done on the benchmark problems of the selected domains. The performances are shown in terms

Table 3. Time IPC score (max score 30 per domain), average CPU time/plan quality and percentages of problems solved by the first 3 couples with respect to the IPC score, for the selected domains. The * indicates the algorithm selected by ASAPs/ASAPq.

Optimising runtimes				
Domain	**Algorithm**	**IPC**	**Mean CPU**	**% Solved**
Barman	⟨Original, SGPlan⟩	28.1	–	100.0
	⟨Outer, SGPlan⟩*	23.1	–	100.0
	⟨Outer, Lama⟩	0.0	–	0.0
BlocksWorld	⟨Both, FF⟩*	30.0	1.3	100.0
	⟨Outer, Probe⟩	20.6	3.8	100.0
	⟨Outer, LPG⟩	19.8	8.1	100.0
Depots	⟨Outer, FF⟩*	28.2	0.5	100.0
	⟨Outer, LPG⟩	21.4	0.9	100.0
	⟨Both, LPG⟩	21.2	1.1	100.0
Gripper	⟨Outer, Mp⟩	30.0	8.0	100.0
	⟨Outer, SGPlan⟩*	26.7	11.0	100.0
	⟨Outer, LPG⟩	23.9	15.3	100.0
Gold-miner	⟨Macro, FF⟩*	29.9	0.02	100.0
	⟨Outer, FF⟩	21.8	0.08	100.0
	⟨Inner, FF⟩	18.8	0.1	100.0
Matching-BW	⟨Macro, LPG⟩*	24.6	1.9	100.0
	⟨Both, LPG⟩	21.6	5.4	96.7
	⟨Macro, FF⟩	17.0	42.6	76.7
Parking	⟨Original, FF⟩*	4.6	376.9	16.7
	⟨Inner, Lama⟩	2.6	659.8	13.3
	⟨Original, Probe⟩	2.5	308.7	10.0
Rovers	⟨Macro, LPG⟩*	28.0	45.6	93.3
	⟨Original, LPG⟩	22.0	92.9	93.3
	⟨Outer, LPG⟩	15.1	261.6	86.7
Satellite	⟨Outer, LPG⟩*	23.8	75.9	96.7
	⟨Original, LPG⟩	22.1	92.4	100.0
	⟨Macro, LPG⟩	17.8	85.1	66.7
Spanner	⟨Original, LPG⟩	25.6	100.4	100.0
	⟨Outer, LPG⟩	24.6	70.4	100.0
	⟨Outer, Mp⟩*	10.6	17.3	36.7
TPP	⟨Outer, LPG⟩	23.2	34.2	100.0
	⟨Both, LPG⟩*	20.0	14.7	66.7
	⟨Outer, Probe⟩	19.9	61.3	100.0

of IPC score, average CPU time (quality) and solved problems. The * indicates the algorithm selected by ASAP. The mean CPU time/quality are calculated on instances solved by all the three best algorithms of the given domain. We would remark that ASAP selects the most promising algorithm on the basis of the results achieved on the learning problems, while in Tables 3 and 4 the comparison is made by ordering the algorithms on the results that they achieved on the testing instances. Concerning the planners included in ASAP, all of them appear at least once. We can then derive that all the planners are able to efficiently exploit, at least on one domain, the knowledge extracted under the form of different encodings.

Considering the runtime optimisation, the planner that appears most frequently

Table 4. Quality IPC score (max score 30 per domain), average CPU time/plan quality and percentages of problems solved by the first 3 couples with respect to the IPC score, for the selected domains. The * indicates the algorithm selected by ASAPs/ASAPq.

Optimising Quality				
Domain	**Algorithm**	**IPC**	**Mean Qual**	**% Solved**
Barman	⟨Original, SGPlan⟩*	30.0	–	100.0
	⟨Outer, SGPlan⟩	27.4	–	100.0
	⟨Outer, Lama⟩	0.0	–	0.0
BlocksWorld	⟨Both, FF⟩*	29.5	231.5	100.0
	⟨Outer, LPG⟩	28.3	243.0	100.0
	⟨Outer, Probe⟩	27.5	248.7	100.0
Depots	⟨Outer, LPG⟩	29.4	120.3	100.0
	⟨Both, LPG⟩*	29.2	120.9	100.0
	⟨Both, Probe⟩	27.7	127.1	100.0
Gripper	⟨Outer, Mp⟩*	30.0	566.8	100.0
	⟨Outer, LPG⟩	28.6	593.8	100.0
	⟨Outer, SGPlan⟩	25.8	659.9	100.0
Gold-miner	⟨Both, SatPlan⟩*	30.0	23.4	100.0
	⟨Macro, Lama⟩	30.0	23.4	100.0
	⟨Macro, LPG⟩	30.0	23.4	100.0
Matching-BW	⟨Outer, SatPlan⟩	26.4	57.2	93.3
	⟨Both, SatPlan⟩	26.4	57.2	93.3
	⟨Outer, LPG⟩*	25.8	61.1	90.0
Parking	⟨Original, FF⟩	4.6	78.0	16.7
	⟨Inner, Lama⟩*	3.8	82.0	13.3
	⟨Inner, FF⟩	2.9	74.5	10.0
Rovers	⟨Macro, Lama⟩*	27.5	674.3	100.0
	⟨Macro, LPG⟩	23.7	669.0	93.3
	⟨Outer, Lama⟩	21.5	657.1	83.3
Satellite	⟨Original, LPG⟩	29.7	675.5	100.0
	⟨Outer, LPG⟩*	28.8	673.1	96.7
	⟨Original, SGPlan⟩	13.7	742.5	50.0
Spanner	⟨Outer, LPG⟩	30.0	266.0	100.0
	⟨Original, LPG⟩*	30.0	266.0	100.0
	⟨Original, Mp⟩	11.0	266.0	36.7
TPP	⟨Outer, Lama⟩*	27.1	422.1	100.0
	⟨Both, Lama⟩	21.0	387.1	73.3
	⟨Outer, Probe⟩	19.7	570.4	100.0

in Table 3 is LPG, followed by Metric-FF, Probe and SGPlan. Thus, we can derive that LPG and Metric-FF are the planners that better exploit macro-operators and entanglements for improving runtime. This is quite surprising if we consider that LPG and Metric-FF are the oldest planners included in ASAP, and that they appeared (in particular FF) more rarely in Table 2. One could argue that, since the plans found by Metric-FF were used for reformulating the domains, the fact that Metric-FF performs well while exploiting entanglements or macro-operators, is not surprising. From this perspective, it is worth noting that LPG is the planner which is able to better exploit this additional knowledge, and that the plans found by Metric-FF were used due to their good quality and to the relatively low CPU time required for finding them. If we focus on the best algorithms for optimising quality of the solutions, the planner which appears most frequently in Table 4 is again LPG, but in this case it is followed by Lama-11. While LPG is often the best basic solver with regard to the plan quality, as shown in Table 2, Lama is the best one only in TPP. It seems then reasonable to deduce that the impact of macro-operators and entanglements is strong on the performance of Lama.

From the point of view of the encodings of the domains, there are no significant differences between runtime and quality, the Outer entanglements appear most frequently in Tables 3 and 4. The less useful encodings are the Inner entanglements, that rarely allowed algorithms considering them to achieve good results.

In terms of runtimes, ASAPs usually selects the best algorithm, which performs significantly better than the second and the third ones. In three domains, Barman, TPP and Gripper, ASAPs selected the second one. It is worth noting that in Gripper domain the performance of the best performing algorithm is similar to the ones of the second. On training problems, their performance were still very close, but the second one had slightly better results. In TPP, the selected algorithm is solving 20 testing problems out of 30. In all of them it is the fastest planner (quality score is exactly 20.0). Interestingly, on the remaining 10 problems, LPG crashed due to memory errors. We believe that this is a bug in the planner and that, without this bug, the algorithm selected by ASAPs would be the best one also in that domain. Finally, in Barman we already discussed the different behaviour of SGPlan on training and testing instances. ASAPs selected the third algorithm only in Spanner: the selected algorithm is fast, but it run out of memory on large instances.

In terms of quality, ASAPq selects the best algorithm in seven domains; in Depots, Parking and Satellite the selected couple is the second one, in Matching-BW the third one. On the other hand, the three best couples usually achieve similar quality score results, this behaviour is quite different between quality and runtimes. It should be noted that in Spanner, because of the structure of the domain, all the valid plan solutions have the same number of actions.

The results shown in Tables 3 and 4 indicate that the approach used for selecting the most promising algorithm, even if not very sophisticated, usually scales well with increasing problem instance size. The couples selected on training instances, easier than testing ones, are achieving very good results also on significantly harder testing instances. The main noticed issue is related to the memory usage of planner, that heavily depend on the planning approach exploited by the system and that is unpredictable on unseen problems.

6.3. *Impact of Reformulation*

ASAP exploits two different reformulation approaches, namely macros and entanglements. Thus, it would be interesting to understand how such reformulations affect the overall performance of ASAP. It should be noted that considering different reformulations has also an impact on the learning process, since the number of algorithms

Table 5. For every domain, the Time/Quality score achieved by allowing ASAP to select only couples that include the original domain model (Original), original and macros (Macros), and all the possible encodings (All).

	Time score			Quality score		
Domain	Original	Macro	All	Original	Macro	All
Barman	28.1	28.1	23.1	30.0	30.0	30.0
Bw	5.6	5.6	30.0	18.1	14.7	29.5
Depots	4.4	4.4	30.0	9.0	9.0	30.0
Gripper	17.7	17.7	29.5	8.3	8.3	30.0
GoldM	15.9	30.0	30.0	28.6	30.0	30.0
MatchBw	9.2	30.0	30.0	21.6	21.9	27.7
Parking	5.0	5.0	5.0	4.9	4.9	3.8
Rovers	22.0	28.0	28.0	23.0	29.9	29.9
Satellite	22.1	17.8	23.8	29.9	29.9	28.9
Spanner	27.1	27.1	26.9	30.0	30.0	30.0
TPP	0.0	15.6	20.0	9.1	0.0	30.0
Total	157.1	209.3	**276.2**	212.2	208.5	**299.7**

is increased and, moreover, their performance might become harder to predict.

In order to understand the impact of the reformulation techniques, we performed the following experiment. We considered three different sets of $\langle e, p\rangle$ algorithms: (i) e includes only original domain models; (ii) e includes also models enhanced with macros; (iii) e includes all the possible reformulations, i.e. original models, macros and entanglements. ASAP selected, for each set of algorithms, the most promising one, according to the performance on the training problems. Table 5 shows the performance of the corresponding selected algorithms on the testing instances. As expected, considering all the possible reformulation usually leads to better results, for both quality and CPU-time. In terms of speed, it seems that both macros and entanglements have a significant impact on the performance. In many domains, considering also macros give an interesting speed-up to ASAPs. On the other hand, it should be noted that in some domains, considering more formulations can lead to worse performance. This is due to the fact that training problems are easier than testing ones, therefore the exploitation of knowledge can have a different impact on planners performance. Moreover, exploiting macros can lead to an increase of memory usage, due to the fact that the branching factor in the search space increases. Thus, it happens that some planners quickly run out of memory on larger instances. This makes harder to predict the performance of algorithms on testing problems and, therefore, can lead to inefficient algorithm selection.

Exploiting macros can eventually decrease the quality of the solutions found. This is due to the fact that macros are sequences of operators, executed as a single one, that allows the planner to reach the goal in fewer search steps. Therefore, shorter plans that involve single actions might not be considered. Mainly for this reason, ASAPq is usually not exploiting macros, and its performance on the "Macro" set are similar to those achieved while using only original domain models. On the other hand, the impact of entanglements on quality score is remarkable.

Table 6. Quality score and the % of solved testing problems of ASAP trained on 10, 20 and 30 problems. Only domains in which the selected algorithm changes, across the different training processes, are considered. $*$ indicates that the score is the same, but the selected algorithm is different.

	Quality Score			Solved		
Domain	5	15	30	5	15	30
Bw	28.7	29.8	28.7	100.0	100.0	100.0
Depots	29.4	29.2	29.4	100.0	100.0	100.0
Gripper	28.6	30.0	30.0	100.0	100.0	100.0
GoldM	30.0$*$	30.0	30.0	100.0	100.0	100.0
MatchBw	21.6	26.1	26.1	83.3	90.0	90.0
Parking	0.0	4.6	3.8	0.0	16.7	13.3
Rovers	21.5	27.5	27.5	83.3	100.0	100.0
Satellite	29.7	29.7	28.8	100.0	100.0	96.7
TPP	3.9	0.0	29.9	13.3	0.0	100.0
Total	193.4	206.9	**234.2**	75.6	78.5	**88.9**

6.4. *Importance of the Number of Training Problems*

In order to understand if small sets of training problems can be sufficient to select the best performing algorithm for test problems that are larger than the training ones, we have compared the performance of ASAP trained using the default number of 30 training problems, and using half and one-sixth of these training problems. The range of the problem size is the same for each of the three sets of training problems. The results of this analysis are in Table 6. Of course, the lower the number of training problems is, the cheaper the training of ASAP is. On the other hand, the algorithm selected using few training problems can sometime be much less performant than the one selected by considering larger training sets. In particular, we observed that ASAPs is always selecting the same algorithm, regardless to the number of training problems considered. This clearly indicates that in the considered domains there is always a single algorithm which is always, or almost always, achieving the best performance on the training problems. Interestingly, this is not true for ASAPq: selecting the most promising algorithm for optimising plans quality appears to be more complex. In several domains, namely Matching-bw, Parking, Rovers and TPP, training only on 5 problems can lead to remarkable performance worsening. On the other domains, the performance are similar. Generally speaking, Table 6 confirms that using a larger number of training problems is helpful for selecting the most promising algorithms.

6.5. *ASAP versus PbP2*

To evaluate the effectiveness of our approach against the state-of-the-art of learning-based planners, we compared ASAP with PbP2. It is the winner of the learning track of the last IPC, the IPC-7, which was held in 2011. For this comparison we used exactly the same benchmark domains and problems that were used for the IPC-7. PbP2 exploited the same knowledge that it used for the competition, while ASAP

Table 7. Time/quality IPC score (max score 30 per domain), average CPU time/plan quality and percentages of problems solved by ASAP and PbP2 for the selected domains. BW stands for BlocksWorld.

Domain	Time IPC		Mean CPU		% Solved	
	ASAPs	PbP2s	ASAPs	PbP2s	ASAPs	PbP2s
Barman	11.0	**30.0**	72.9	**2.0**	100.0	100.0
BW	**30.0**	16.4	**1.3**	9.9	100.0	100.0
Depots	**30.0**	8.8	**0.5**	76.7	**100.0**	86.7
Gripper	**30.0**	24.7	**11.0**	18.3	100.0	100.0
Parking	3.6	**8.0**	455.3	**172.8**	16.7	**26.7**
Rovers	19.8	**26.2**	58.4	**18.5**	**93.3**	90.0
Satellite	22.9	**30.0**	71.1	**28.3**	96.7	**100.0**
Spanner	10.1	**30.0**	17.3	**7.0**	36.7	100.0
TPP	**20.0**	16.5	**14.7**	113.8	66.7	**83.3**
All	177.4	**190.6**	–	–	78.9	**87.0**

Domain	Quality IPC		Mean Quality		% Solved	
	ASAPq	PbP2q	ASAPq	PbP2q	ASAPq	PbP2q
Barman	29.8	**29.9**	452.8	**449.3**	100.0	100.0
BW	**29.9**	25.7	**231.5**	269.9	100.0	100.0
Depots	**30.0**	19.2	**118.1**	163.8	**100.0**	86.7
Gripper	**30.0**	28.9	**566.8**	588.7	100.0	100.0
Parking	3.7	**5.0**	82.0	**68.0**	13.3	**20.0**
Rovers	**27.4**	27.3	**693.9**	708.4	100.0	100.0
Satellite	26.8	**28.2**	790.8	**784.3**	96.7	**100.0**
Spanner	30.0	30.0	326.0	326.0	100.0	100.0
TPP	**29.5**	13.4	**343.3**	370.1	**100.0**	50.0
All	**237.1**	207.6	–	–	**90.0**	84.0

was trained on 30 problems, easier than the testing ones, that were generated by using the problem generators provided by the organisers.

Table 7 shows performance in terms of time/quality IPC score, mean CPU time (quality) and percentage of solved problems on benchmark problems of the selected domains. The mean CPU time/quality are calculated on instances solved by both the approaches. The mean on all the domains is not indicated because, given the great variability of both CPU time and plan quality across the domains, it is not informative. The results indicate that ASAP performs better than PbP2 in terms of quality of the solution plans, but it achieves slightly worse results in terms of runtimes. ASAPs is significantly faster than PbP2s in four of the selected domains. In particular, ASAPs is significantly faster (more than 2 orders of magnitude) in Depots, where the entanglements give a great speedup. We noticed that ASAPs is significantly slower than PbP2s in Barman and Spanner. In these domains, as discussed in previous sections, our approach selected algorithms with good performance on training problems, that are not confirmed on testing ones.

Given the fact that some planners (LPG, Metric-FF, Lama and SGPlan) that are included in the ASAP system are also included in PbP2, it is interesting to analyse the domains in which PbP2 selected as a member of the portfolio the planner selected in the algorithm of ASAP. In Barman, both ASAPs and PbP2s are exploiting SGPlan, but PbP2s was able to extract some macro-operators that improve the performance of the planner in that domain. PbP2s configured a portfolio composed of only LPG (without macros) in Rovers, Satellite and Spanner domains. In Rovers and Satellite, ASAPs also selected an algorithm which included LPG, while in Spanner Mp was preferred. In all of them PbP2s is faster than ASAPs. It should be noted that in those domains PbP2s exploits a domain-specific configuration of the parameters of LPG, obtained by [37], while ASAPs runs the default configuration of LPG. Since the use of reformulated problems is useful for LPG in such domains (this can be derived by comparing Tables 2, 3 and 4), and considering that PbP2s did not include macros in the corresponding portfolios, we believe that the results achieved in Rovers and Satellite are due to the speedup allowed by the tuned configuration of LPG. On the other hand, in BlocksWorld, Depots and TPP ASAPs selected algorithms which include Metric-FF and LPG, that are available in PbP2s but are not selected; in these domains the performance improvement given by the entanglements is very significant.

In terms of quality of the solution plans, ASAPq achieved better results in five of the selected domains. In two domains, namely TPP and Depots, ASAPq has significantly better results than PbP2q. We also noticed that, counterintuitively, in Rovers, the macros are helpful for improving the quality of the solution plans.

The portfolios configured by PbP2q on the IPC-7 domains are usually composed of either 2 or 3 different planners. It could happen that all the included planners are useful, or that just a subset of them is actually exploited for finding solution plans on testing problems; it is unclear what the real contribution of each planner is to the portfolio. For this reason a comparison between the planners selected by PbP2q and ASAPq is not possible. Only in the Spanner domain PbP2q exploits a portfolio that is composed of a single planner, LPG, without macros. ASAPq selects an algorithm which is composed of LPG and the original domain encoding, which leads the systems to achieve exactly the same results. In the Barman, Depots, Parking, Rovers, Satellite and TPP, the planner included in the algorithm selected by ASAPq is present in the portfolio configured by PbP2q. Finally, in BlocksWorld ASAPq is selecting Metric-FF, which is considered in PbP2q but not included in the portfolio, and in Gripper our approach is selecting a planner, Mp, which is not considered by PbP2q.

7. Discussion

As the results shown in the previous section indicated, an accurate selection of a single algorithm allows ASAP to achieve better results than the portfolio-based planning system PbP2 in terms of plan quality, and to be very close to PbP2 in terms of CPU time. We believe that a domain-specific portfolio can be completely exploited when the different planners can run in *pure* parallel, so when several cores are available. In case a single core is available, using different planners together can

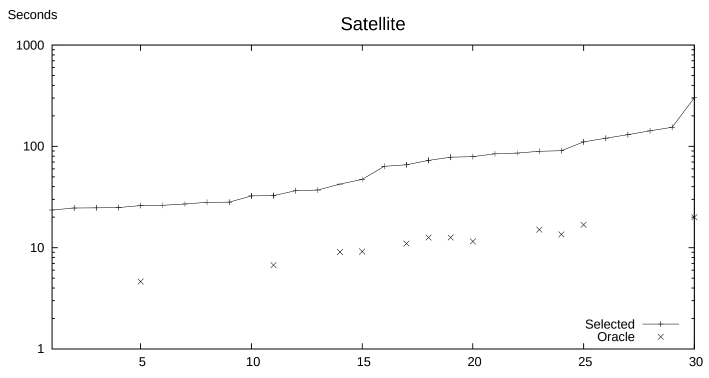


Fig. 2. CPU time (log. scale) of the selected couple w.r.t. the Oracle on benchmark problems of Satellite domain.

slow down the performance of the best one on the given domain.

The algorithms selected by ASAP have shown very good performance on testing problems. As indicated by the results shown in Tables 3 and 4, there is very little space for further improvements in terms of CPU time. However, in Matching-BW and Satellite domains the selected algorithms still achieved the best results in total in comparison to the others, but for some problems the results of the selected couple were significantly worse. For instance, as depicted in Figure 2, we observed that in Satellite domain about 40% of testing problems could be solved significantly faster by a different algorithm $\langle \mathsf{Macros}, \mathsf{LPG} \rangle$ then the selected one $\langle \mathsf{Outer}, \mathsf{LPG} \rangle$. It indicates that we need to somehow classify problems even within the same domain. An initial idea assumes that each class of the problem has a different algorithm assigned to it which provides the best results. In the learning stage, we can identify which couple works best on a particular problem and hence we might determine classes of problems (the number of classes is equal to the number of algorithms which were best on at least one problem). However, it might happen that some classes will be small (containing only a few problems) or very similar which may prevent identification of their characteristic properties. Therefore, a reasonable way for mitigating such an issue is in moving problems from 'small' classes to larger ones where the corresponding couple is closest to the best. However, an efficient classifier for planning problems has not been developed yet.

Interestingly, the need for classification seems to be restricted to runtime optimization. By analysing the performance of algorithms, we observed that in terms of plan quality, usually the couple that achieves the best result is finding the best solution for all, or almost all, the testing problems. In the Matching-BW domain the best algorithm is outperformed by a different one only on three problems out of the 30 considered.

Regarding the exploitation of the knowledge extracted in the form of macro-operators and entanglements, we noticed that Metric-FF and LPG are the ones that, in terms of runtimes, exploited it more efficiently. Considering the results shown in Table 2, in BlocksWorld, Depots, Gold-miner and Matching-BW, Metric-FF and LPG are not the best planners; but while exploiting the additional knowledge, they outperform the others, as shown in Tables 3 and 4. Also in terms of quality, the impact of the additional knowledge extracted in the form of new encodings is significant. This is surprising if we consider that macro-operators and entanglements are designed especially for improving the performance of planners in terms of time needed for finding a satisficing solution. A very interesting example of this behaviour is given by Lama in the domain TPP, in which the exploitation of entanglements, either Both or Outer, lets the planner improve its performance by more than the 60%; all the plans found by Lama and the original domain encoding have lower quality.

The results also showed that using different encodings than the original one usually improves the performance of the planners. This demonstrates the importance of reformulation techniques in planning, which was one of the goals of this work, although a particular reformulation technique might not lead to performance improvement in some domains and for some planners. Therefore, in our empirical study we did not consider some of domains (e.g. nPuzzle), where no knowledge could be extracted, so ASAP just selects the best planner for the original domain encoding.

We are also aware of other possibilities of extracting domain knowledge. Besides other techniques for extracting macros such as Macro-FF [4] and Wizard [27], an interesting possibility is combining macros and entanglements. An ad-hoc approach where macros and outer entanglements has been discussed in [9] and has shown some promising results.

Finally, we notice that since the current learning phase of ASAP requires running all the available algorithms, namely every planner for every encoding (35 in the current version), the time spent for the learning phase could be high. In the worst case, all the algorithms fail on all the training problems, it requires about 260 hours. The empirical evaluation presented in Section 6.4, shows that for selecting the most promising algorithm, in terms of runtime, usually a very small number of training problems is needed, at least on the considered benchmark domains. On the other hand, this is not true while optimising for plan quality. Results show in Table 6 confirms that the algorithm selection performed by ASAPq is more accurate when based on the results collected across a larger number of training problems. Therefore, it seems reasonable to use, while optimising plan quality, some sort of heuristic which can prune unpromising couples before or at an early stage of the learning phase. A simple approach may be based on an 'incremental pruning' strategy, i.e., solve a part of the training problems with all the couples, after that remove the worst ones and continue solving another part of the training problems with the remaining couples, then prune the worst and so on. A more sophisticated approach could be based on exploiting knowledge we have about planners and/or encodings.

8. Conclusion and Future Work

In this paper we have presented ASAP, an automatic algorithm selection approach for planning. ASAP is based on the idea of extracting additional knowledge from a domain, in the form of macro-operators and entanglements, combining such knowledge with existing planning systems for generating new algorithms, and selecting the most promising algorithm for solving problems from the given domain. ASAP has two different versions: ASAPs which selects the most promising algorithm for optimising runtimes, and ASAPq which optimises the quality of the solution plans.

An experimental analysis conducted on a total of 11 well-known benchmark domains and that involved a large number of planning problems, has shown that (i) the impact of the considered dimensions on the performances of the algorithms is significant, (ii) the technique used for selecting the most promising algorithm to exploit on testing problems is very accurate, (iii) ASAPs is competitive with the state-of-the-art of learning-based planning systems, PbP2s, in terms of runtime, (iv) ASAPq outperformed PbP2q in terms of quality of solution plans.

Future work includes further experimental analysis, in particular for understanding if learning-based approaches exploiting domain-specific portfolios would always be outperformed by accurate and efficient automatic algorithm selection based planners, while sharing the same planners and the same additional knowledge, on single core machines. A specific experimental analysis is also needed for having a better understanding of the impact of problems reformulation on the different planning systems; a system for predicting this impact would lead to a great reduction of the learning time needed for selecting the algorithm to use for a specific domain. Moreover, we are interested in combining the approach used for reformulating planning problems with existing techniques for generating macro-operators (e.g., Wizard [27], Macro-FF [4]). Another direction for further research is investigating the extraction and exploitation of additional knowledge in the form of parameters configuration of the domain independent highly parameterised included planners. Such optimisation, exploited also in PbP2 and ParLPG [16, 37], has shown to provide significant performance improvements.

We noticed that the major limitation of ASAP is that, in its current version, it is heavily dependent on Metric-FF. The solutions found by this planner on a small set of training problems are analysed for extracting additional knowledge. It could happens that Metric-FF is not able to solve any non-trivial training problem. To avoid this situation, we are planning to extend the techniques used for extracting macro-operators [8] and entanglements [6] in order to exploit plans produced by different planners. This could also lead to the derivation of more specific additional knowledge that, potentially, could further increase planner performance.

Finally, we are considering including different algorithm selection techniques in ASAP. The current one is mainly based on IPC score, which considers performance and number of solved problems together. The exploitation of more sophisticated score systems could improve the selection accuracy. Alternative selection techniques could be based, for instance, on the well-known PAR10 score, or on statistical analysis. Moreover, the problem of selecting the best couple can also be seen as an automatic configuration problem, by adding to existing planners a parameter for selecting the encoding of a domain. In this way we can reuse the well-known techniques of algorithm configuration (e.g., [21, 22]) for finding a good domain-specific configuration, which also considers the encoding.

Acknowledgments

The authors would like to acknowledge the use of the University of Huddersfield Queensgate Grid in carrying out this work.

References

1. A. Biere, P{re,ic}oSAT@SC'09 In *SAT Competition*, 2009.
2. A. L. Blum and M. L. Furst, "Fast Planning Through Planning Graph Analysis", *Artificial Intelligence*, 90:281–300, 1997.
3. B. Bonet and H. Geffner, "Planning as heuristic search", *Artificial Intelligence* 129:5–33, 2001.
4. A. Botea, M. Enzenberger, M. Müller and J. Schaeffer, "Macro-FF: Improving AI planning with automatically learned macro-operators", *Journal of Artificial Intelligence Research (JAIR)* 24:581–621, 2005.
5. Y. Chen, B. W. Wah and C. W. Hsu, "Temporal planning using subgoal partitioning and resolution in SGPlan", *Journal of Artificial Intelligence Research (JAIR)* 26:323–369, 2006.
6. L. Chrpa and T. L. McCluskey, "On exploiting structures of classical planning problems: Generalizing entanglements", In *Proc. of the 20th European Conference on Artificial Intelligence (ECAI)*, 240–245, 2012.
7. L. Chrpa and R. Barták, 'Reformulating planning problems by eliminating unpromising actions', in *Proceedings of SARA 2009*, pp. 50–57, (2009).
8. L. Chrpa, "Generation of macro-operators via investigation of action dependencies in plans", *Knowledge Engineering Review* 25(3):281–297, 2010.
9. L. Chrpa, "Combining learning techniques for classical planning: Macro-operators and entanglements", In *Proceedings of ICTAI*, volume 2, 79–86, 2010.
10. A. Coles, A. Coles, A. G. Olaya, S. Jiménez, C. L. Linares, S. Sanner and S. Yoon, "A survey of the seventh international planning competition", *AI Magazine* 33:83–88, 2012.
11. M. T. Cox, "Metacognition in computation: A selected research review", *Artificial Intelligence*. 169 (2), 104-141. 2005
12. A. Fern, R. Khardon and P. Tadepalli, "The first learning track of the international planning competition", *Machine Learning* 84:81–107, 2011.
13. E. Fink, "Changes of Problem Representation: Theory and Experiments", Springer, 2002.
14. A. Gerevini, A. Saetti and I. Serina, "Planning through stochastic local search and

temporal action graphs", *Journal of Artificial Intelligence Research (JAIR)* 20:239–290, 2003.

15. A. Gerevini, A. Saetti and M. Vallati, "An automatically configurable portfolio-based planner with macro-actions: PbP", In *Proc. of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, 19–23, 2009.
16. A. Gerevini, A. Saetti and M. Vallati, "PbP2: Automatic configuration of a portfolio-based multiplanner", In *Booklet of the 7th International Planning Competition.* 2011.
17. M. Helmert, "The Fast Downward planning system", *Journal of Artificial Intelligence Research (JAIR)* 26:191–246, 2006.
18. J. Hoffmann, "The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables", *Journal of Artificial Intelligence Research (JAIR)* 20:291–341, 2003.
19. J. Hoffmann and B. Nebel, "The FF planning system: Fast Plan Generation Through Heuristic Search", *Journal of Artificial Intelligence Research (JAIR)* 14:253–302, 2001.
20. A. Howe, E. Dahlman, C. Hansen, A. vonMayrhauser and M. Scheetz, "Exploiting competitive planner performance", In *Proc. of the 5th European Conference on Planning (ECP)*, 62–72, 1999.
21. F. Hutter, H.H. Hoos, K. Leyton-Brown, "Sequential Model-Based Optimization for General Algorithm Configuration", In *Proceedings of the conference on Learning and Intelligent OptimizatioN (LION 5)*, 507–523, 2011.
22. F. Hutter, H.H. Hoos, K. Leyton-Brown and T. Stützle, "ParamILS: An automatic algorithm configuration framework", *Journal of Artificial Intelligence Research (JAIR)* 36:267–306, 2009.
23. H. Kautz, B. Selman and J. Hoffmann, "SatPlan: Planning as satisfiability", In *Abstract Booklet of the 5th International Planning Competition*, 2006.
24. M. G. Lagoudakis, M. L. Littman and R. Parr, "Selecting the right algorithm", In *Proceedings of the 2001 AAAI Fall Symposium Series: Using Uncertainty within Computation*, AAAI Press, Menlo Park, CA, 2001.
25. N. Lipovetzky and H. Geffner, "Searching for plans with carefully designed probes", In *Proc. of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*, 2011.
26. S. Minton, "Quantitative Results Concerning the Utility of Explanation-Based Learning", In *Proc. of the 7th National Conference on Artificial Intelligence (AAAI)*, 564–569, 1988.
27. M. A. H. Newton, J. Levine, M. Fox and D. Long, "Learning macro-actions for arbitrary planners and domains", In *Proc. of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, 256–263, 2007.
28. J.R. Rice, "The algorithm selection problem", *Advances in Computers* 15:65118. 1976
29. S. Richter and M. Westphal, "The LAMA planner: Guiding cost-based anytime planning with landmarks", *Journal of Artificial Intelligence Research (JAIR)* 39:127–177, 2010.
30. S. Richter, M. Westphal and M. Helmert, "Lama 2008 and 2011", In *Booklet of the 7th International Planning Competition*, 2011.
31. J. Rintanen, "Engineering efficient planners with SAT", In *Proc. of the 20th European Conference on Artificial Intelligence (ECAI)*, 684–689, 2012.
32. M. Roberts and A. Howe, "Learned models of performance for many planners", In *Proc. of the ICAPS-07 Workshop of AI Planning and Learning (PAL)*, 2007.
33. M. Roberts and A. Howe, "Learning from planner performance", *Artificial Intelligence* 173(56):536–561, 2009.
34. A. Sideris and Y. Dimopoulos, 2010. Constraint Propagation in Propositional Plan-

ning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS-10)*.
35. J. Seipp, M. Braun, J. Garimort and M. Helmert, "Learning portfolios of automatically tuned planners", In *Proc. of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, 2012.
36. K. Talamadupula, J. Benton, P. W. Schermerhorn, S. Kambhampati and M. Scheutz, "Integrating a Closed World Planner with an Open World Robot: A Case Study." In *proceeding of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010)*, Atlanta, Georgia. 2010.
37. M. Vallati, C. Fawcett, A. Gerevini, H. Hoos and A. Saetti, "Automatic Generation of Efficient Domain-Specific Planners from Generic Parametrized Planners", In *Proc. of the Sixth Annual Symposium on Combinatorial Search (SoCS)*, 2013.
38. M.M. Veloso, and P. Stone, "FLECS: Planning with a Flexible Commitment Strategy." *Journal of Artificial Intelligence Research*, 3:25-52. 1995.