AGAP: As Good As Possible

Lukáš Chrpa and **Mauro Vallati**
PARK Group
School of Computing and Engineering
University of Huddersfield
{l.chrpa, m.vallati}@hud.ac.uk

Abstract

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, extracted in the form of macro-operators or entanglements.

In this paper we propose AGAP, 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 the quality of the solution plans.

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 formalization, and that can be used for optimizing the planner behaviour.

These observations motivate the idea of extracting additional knowledge about the planning domains and automatically selecting the most promising planning algorithm, exploiting such knowledge, for a given domain.

In this paper we propose AGAP, 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), and (ii) explores the 2 dimensional space encodings (e)–planners (p), and then (iii) selects the best algorithm $\langle e, p\rangle$ for optimising the quality of the solution plans.

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.

AGAP is an evolution of the system proposed in (Vallati, Chrpa, and Kitchin 2013); AGAP is focused on quality of the solutions, and considers a different set of planners and a larger set of reformulation techniques. 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 (Howe et al. 1999; Roberts and Howe 2007), PbP2 (Gerevini, Saetti, and Vallati 2009; 2011) and Fast Downward Stone Soup (Helmert, Röger, and Karpas 2011; Seipp et al. 2012), with some significant differences.

The major difference between all the approaches above and AGAP 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 AGAP 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 learned 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 tech-

niques included in the well known domain-independent planner Fast Downward (Helmert 2006). Their approach is domain-independent, it does not extract any additional knowledge from the planning domains (in the form of 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 planning problem reformulations, then we describe the AGAP approach and finally we give conclusions.

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.

Macro-operators

A macro-operator, shortly macro, encapsulates a sequence of (primitive) planning operators and can be represented as an ordinary planning operator. In the well-known planning domain 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 `unstack-putdown(?x ?y)`. Creating macros (Dawson and Siklóssy 1977), which can be understood as 'shortcuts' in the state space, is therefore a well known and studied approach which in some cases can speed up plan generation considerably (Newton et al. 2007; Botea et al. 2005). Macros can be added into planning domains and reformulated domains can be passed to any planning engine. To raise the efficiency of the planning process, an approach (Chrpa 2010) besides generating new macros also removes some primitive operators which are very likely useless. In our example, if the new macro `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).

Entanglements

Entanglements (Chrpa and McCluskey 2012) 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 are relations between planning operators and initial or goal atoms (predicates) which refer 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 an initial state or goal situation respectively. In the 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 into planning domains and problems is done by introducing static predicates which allow only instances following conditions of outer entanglements (for details, see (Chrpa and McCluskey 2012)).

Inner entanglements 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 'consumer' exclusivity given by these inner entanglements. It is done by using specific predicates, 'locks', which prevents 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 modeled such that `pickup(?x)` is a 'locker' for `putdown(?x)` and `stack(?x,?y)` is a 'releaser' for `putdown(?x)`. For details, see (Chrpa and McCluskey 2012).

Entanglements are extracted from training plans, solutions of simpler problems, in such a way that we check for each operator/pair of operators and related predicates whether the entanglement conditions are satisfied in all the training plans (some error rate might be allowed since training plans may consist of 'flaws' such as redundant actions). Although applying extracted entanglements might cause loss of solvability of some non-training problems, 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 testing problems one. For deeper insights, see (Chrpa and McCluskey 2012).

Combining Macros and Entanglements

It is well known that adding macros into planning domain models may significantly increase the branching factors because macros, especially more complex ones, can have much more instances than primitive operators. Outer entanglements, on the other hand, eliminate some unpromising operators' instances. AGAP considers two different approaches for combining them.

MUM (Chrpa, Vallati, and McCluskey 2014) is a technique for generating macros which exploits outer entanglements. For generating a set of macro candidates, it investigates dependencies and independencies of actions in plans (Chrpa 2010). Then, MUM selects 'proper' macros according to number of components of argument matching graphs. In an argument matching graph of a macro, nodes are macro's arguments and edges between nodes indicate that corresponding arguments appear together in a static or 'entangled' predicate (clearly, such predicates are defined in macro's scheme). Heuristically, it is assumed that smaller number of argument matching graph components implies smaller number of macro's instances. Note that MUM applies (outer) entanglements only on macros (not the primitive operators), so completeness is not compromised even if some detected entanglements are incorrect. *Aggressive MUM* is a technique similar to MUM, but it applies outer entanglements also on primitive operators, which results in potential incompleteness.

'*Combo*' encapsulates a technique for macro generation from inner entanglements (Chrpa et al. 2013). The technique exploits the 'natural property' of inner entanglements, i.e., that one operator achieves a predicate (or predicates) for another operator. To generate a macro it must be ensured that no other operator has to be applied in between. In particular, a possible achiever for the second operator does not 'clobber' any predicate for the first one; if the first operator is an achiever for some other operator (not the second one), the second operator is not a 'clobberer' for that operator; the first operator does not achieve predicates for multiple instances of the second operator. Taking into account exclusivity of predicate achievement or requirement given by inner entanglements, we can remove one or both primitive operators after a macro is generated (Chrpa et al. 2013). After generating macros (and removing concerned primitive operators) outer entanglements are applied on macros as well as on the rest of primitive operators.

The Proposed Approach

AGAP includes the following existing high performance domain-independent planners: Lama-11 (Richter and Westphal 2010; Richter, Westphal, and Helmert 2011), LPG (Gerevini, Saetti, and Serina 2003), Metric-FF (Hoffmann 2003), Mp (Rintanen 2012), Probe (Lipovetzky and Geffner 2011) and SGPlan (Chen, Wah, and Hsu 2006). We selected them due to their good performances in International Planning Competitions (IPC) and the different techniques they exploit.

The learning phase of AGAP is composed of four steps: (i) extraction of additional knowledge (macros and entanglements), (ii) generate new encodings of 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.

Through these techniques AGAP is able to generate at most four new encodings per domain: Outer entanglements (only), MUM, Aggressive MUM and Combo. The maximum number of algorithms per domain is 30, which arises from 6 included planners that can be used with 5 different encodings.

The current version of AGAP runs the available algorithms on training problems. The performances are measured in terms of CPU time required for solving each training instance, quality 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.

For selecting the most promising algorithm, AGAP uses the quality IPC score. It is a value, firstly introduced in IPC-6 (Fern, Khardon, and Tadepalli 2011), which considers qualities and number of solved problems together. It is very useful because it synthesizes different aspects of planners' performance in a single value, that can then be compared through different planners. AGAP selects the couple which achieved the best IPC score on the learning problems; if more algorithms achieved the same 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. For the incremental planners[1], i.e. LPG and Lama, the best solution found within the CPU time limit is considered.

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.

The learning phase of AGAP requires running all the available algorithms, namely every planner for every encoding (30 in the current version), the time spent for the learning phase could be high. Therefore, it seems reasonable to use 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.

References

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-FF: Improving AI planning with automatically

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

learned macro-operators. *Journal of Artificial Intelligence Research (JAIR)* 24:581–621.

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

Chrpa, L., and McCluskey, T. L. 2012. On exploiting structures of classical planning problems: Generalizing entanglements. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI-12)*, 240–245. IOS Press.

Chrpa, L.; Vallati, M.; McCluskey, T. L.; and Kitchin, D. E. 2013. Generating macro-operators by exploiting inner entanglements. In *Proceedings of SARA*.

Chrpa, L.; Vallati, M.; and McCluskey, T. L. 2014. Mum: A technique for maximising the utility of macro-operators by constrained generation and use. In *Proceedings of ICAPS*.

Chrpa, L. 2010. Generation of macro-operators via investigation of action dependencies in plans. *Knowledge Engineering Review* 25(3):281–297.

Dawson, C., and Siklóssy, L. 1977. The role of preprocessing in problem solving systems. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI-77)*, 465–471. William Kaufmann.

Fern, A.; Khardon, R.; and Tadepalli, P. 2011. The first learning track of the international planning competition. *Machine Learning* 84:81 – 107.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research (JAIR)* 20:239 – 290.

Gerevini, A.; Saetti, A.; and Vallati, M. 2009. An automatically configurable portfolio-based planner with macro-actions: PbP. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS-09)*, 19–23. AAAI Press.

Gerevini, A.; Saetti, A.; and Vallati, M. 2011. PbP2: Automatic configuration of a portfolio-based multiplanner. In *Booklet of the 7th International Planning Competition*.

Helmert, M.; Röger, G.; and Karpas, E. 2011. Fast Downward Stone Soup: A baseline for building planner portfolios. In *Proceedings of the ICAPS-11 Workshop of AI Planning and Learning (PAL)*.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research (JAIR)* 26:191–246.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research (JAIR)* 20:291–341.

Howe, A.; Dahlman, E.; Hansen, C.; vonMayrhauser, A.; and Scheetz, M. 1999. Exploiting competitive planner performance. In *Proceedings of the 5th European Conference on Planning (ECP-99)*, 62–72. Springer-Verlag.

Hutter, F.; Hoos, H. H.; Leyton-Brown, K.; and Stützle, T. 2009. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research (JAIR)* 36:267 – 306.

Lipovetzky, N., and Geffner, H. 2011. Searching for plans with carefully designed probes. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS-11)*. AAAI press.

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

Newton, M. A. H.; Levine, J.; Fox, M.; and Long, D. 2007. Learning macro-actions for arbitrary planners and domains. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS-07)*, 256–263. AAAI press.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research (JAIR)* 39:127–177.

Richter, S.; Westphal, M.; and Helmert, M. 2011. Lama 2008 and 2011. In *Booklet of the 7th International Planning Competition*.

Rintanen, J. 2012. Engineering efficient planners with SAT. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI-12)*, 684–689. IOS Press.

Roberts, M., and Howe, A. 2007. Learned models of performance for many planners. In *Proceedings of the ICAPS-07 Workshop of AI Planning and Learning (PAL)*.

Roberts, M., and Howe, A. 2009. Learning from planner performance. *Artificial Intelligence* 173(56):536 – 561.

Seipp, J.; Braun, M.; Garimort, J.; and Helmert, M. 2012. Learning portfolios of automatically tuned planners. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS-12)*.

Vallati, M.; Chrpa, L.; and Kitchin, D. E. 2013. An automatic algorithm selection approach for planning. In *IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI)*, 1–8.

Wilcoxon, F., and Wilcox, R. 1964. Some rapid approximate statistical procedures. Technical report, American Cyanamid Co.