The Meta-Search Planner (MSP) at IPC 2018

Raquel Fuentetaja[1], Michael Barley[2], Daniel Borrajo[1], Jordan Douglas[2], Santiago Franco[3] and **Patricia Riddle[2]**

[1]Departamento de Informática. Universidad Carlos III de Madrid, Spain
[2]Department of Computer Science. University of Auckland, New Zealand
[3]School of Computing and Engineering, University of Huddersfield, United Kingdom

Abstract

This abstract describes the general behavior of MSP (Meta-Search Planner) for the IPC (International Planning Competition) 2018, optimal track. MSP is a meta-reasoning system that searches through the space of planners, representations and heuristics on a problem-by-problem basis. Given a planning problem, MSP perfoms two phases: meta-search phase and problem solving phase. The meta-search phase is a search process for selecting one combination suitable for optimally solving the specific problem. The solving problem phase is basically a call to the selected planner with the other elements of the selected combination.

Introduction

The Meta-Search Planner (MSP) presented at the 2018 IPC is based on an approach for optimal planning described previously in (Fuentetaja et al. 2018). Given a problem and domain, MSP searches through the space of representation changes and heuristics to find a good combination to use in solving the problem. The underlying ideas are the following: (1) it is seldom the case that a single representation or heuristic is best over all problems within a domain; (2) the choice of representations and heuristics can be modeled as a meta-level search through a space of combinations of representations and heuristics; and (3) the decision on which representation change and set of heuristics to use can be made on-line and on a problem by problem basis. The main differences between MSP and other methods such as portfolios (Cenamor, de la Rosa, and Fernández 2016; Helmert, Röger, and Karpas 2011; Gerevini, Saetti, and Vallati 2009; Núñez, Borrajo, and Linares-López 2015) are: (1) MSP does not perform a learning step; (2) porfolios do not usually focus on changing the representation of the input domain/problem; and (3) they frequently do not adapt the portfolio to the specific problem.

An important aspect of MSP is that it allows representation changes applied to the input representation. Representation changes have been much less studied in automated planning than other aspects such as heuristics and search algorithms. However, the same problem can be defined in different ways in PDDL (Planning Domain Description Language) (Ghallab et al. 1998), the standard language for compactly representing planning tasks. Each particular definition of a planning problem may have an impact on the planner performance. In fact, given the same combination of heuristics and search methods some representations will make it harder to solve the problem while others will facilitate its solution (Howe et al. 1999; Howe and Dahlman 2002; Riddle et al. 2016; Fuentetaja and de la Rosa 2016). Also, the *best* representation may not be the same for different problems within a domain.

Related also to the representation is the fact that most current planners transform the PDDL representation into more efficient internal representations such as propositional logic (Hoffmann and Nebel 2001) or SAS$^+$ (Bäckström and Nebel 1995). Thus, changes of representation can be performed either at the PDDL level, or in the procedure(s) involved in the transformation. For instance, in Fast Downward (FD) (Helmert 2006), one of the most influential planning platforms, these procedures are the translation and preprocessing steps to generate a SAS$^+$ representation. In SAT planning, the impact of different encodings has also been examined (Kautz and Selman 1992; Rintanen 2012).

In general, MSP can change the representation using various techniques as black-boxes. They range from simple ones like changing the order of actions in the PDDL domain file, à la (Vallati et al. 2015), to more complex ones like Baggy (Riddle et al. 2016), that reformulates problems into a bagged representation. It can also vary the procedure to obtain the internal representation.

MSP can use one or more underlying planners. The meta-search described in the original publication (Fuentetaja et al. 2018) uses the RIDA* planner (Barley, Franco, and Riddle 2014) for several purposes: evaluating meta-search states; selecting the set of appropiate heuristics for a meta-search state; and as the planner for solving the problem. The version presented at the IPC 2018 does not use RIDA*. Instead, it includes two planners: Optimal Fast Downward (FD) (Helmert 2006) and SYMBA* (Torralba, Linares-López, and Borrajo 2016). Thus, the evaluation of meta-search nodes and the selection of heuristics at the IPC 2018 is different from that described in the original article. Specifically, the evaluation of nodes is perfomed by a sampling process using the mentioned planners. The heuristics are included in the meta-

search states, so that they are evaluated together with the other state features by an evaluation function that is also different. The next sections describe the MSP version for IPC 2018 in more detail.

Meta-search

We use the standard definition of a planning task as $\Pi = \{F, A, I, G\}$. F is a set of propositions, A is a set of actions, $I \subseteq F$ defines the initial state and $G \subseteq F$ defines the goals. A planner takes as input a planning task and returns a plan $\pi = \langle a_1, \ldots, a_n \rangle$ such that if applied to the initial state I, it will achieve the goals. That is, it will generate a state s_n after applying actions in π to I such that $G \subseteq s_n$. Actions have a cost, $c(a), \forall a \in A$. The cost of a plan is $c(\pi) = \sum_{a_i \in \pi} c(a_i)$. An optimal plan is one with minimum cost.

MSP can be described in terms of a generic search in a meta-search space followed by a call to a planner. It can be formally defined in terms of the meta-search state space ($\mathcal{MS}$), meta-search operators ($\mathcal{MO}$), and the problem solving method: the search algorithm and the heuristics.

We will denote as R_e the set of representation changes that generate a new PDDL representation, referred to as *external* representation changes. The changes that transform a PDDL representation into an internal one (such as SAS$^+$) are referred to as R_i, the *internal* representation changes. R_i is planner specific.

The input to MSP is a planning task described in terms of a domain D, and a problem P, both described in PDDL. Since MSP performs search in the space of representations, planners and heuristics, it also receives as inputs the set of external representation change operators that can be applied, R_e; the planners, Pl; the set of internal representation changes, R_i, and heuristics, $\mathcal{H}$, that every planner can use; and the time bound, T, represents the maximum time to solve the problem.

MSP is given the maximum time it can use to solve a problem. Within that time limit, it must use some of that time to select a good combination of representation changes, planners and heuristics, and use the remainder to apply that selection to solve the problem. This means that MSP must balance the benefits of spending more time in the meta-search against those of spending more time actually solving the problem. Picking the right balance is a difficult decision. Currently, MSP just splits the maximum time, T, in half. While the planner is guaranteed at least half of the total time, it will end up with more time whenever the selection process takes less than half. In other words, the planner will have a time limit of T - *consumedTime*, where *consumedTime* is the actual time consumed by the meta search.

Algorithm 1 shows a high level description of MSP that includes the call to meta-search and a call to the planner with the output of the meta-search (and the remaining time). The meta-search requires three time limits: one for the evaluation of states, another one for the meta-search and the other as an estimate of the time to be given later to the planner. The state evaluation time has been fixed to 75 seconds. For the other two, we assume $T/2$.

In the following, *end* refers to the best meta-search state found. The output of the meta-search contains: the final selected PDDL representation of the domain and problem, D^{end} and P^{end}; the selected planner pl^{end}, r_i^{end}, the internal representation, and H^{end}, the set of heuristics. The output also contains a plan π that will be empty if the problem is not solved during meta-search. In that case, the planner is called with the final domain and problem, the selected configuration and the remaining time. These components will be explained in detail in the following subsections.

Algorithm 1 MSP$(D, P, R_e, Pl, R_i, \mathcal{H}, T_E, T)$

Inputs: domain D, problem P, PDDL rep. changes R_e, planners Pl, internal rep. changes R_i and set of heuristics $\mathcal{H}$ for every planner, estimation time bound T_E, time bound T
Outputs: plan

```
(D^end, P^end, pl^end, r_i^end, H^end, π) ←
    META-SEARCH(D, P, R_e, Pl, R_i, H, T_E, T/2, T/2)
if π = ∅ then
    T ← T − consumedTime
    plan ← pl^end(D^end, P^end, r_i^end, H^end, T)
else
    plan ← π
return plan
```

Representation Changes

The general representation changes that can be included in MSP are of two types: external (R_e) and internal (R_i). Each external representation change operates at the PDDL level and generates a new PDDL domain, D, and problem, P. It can be defined as a function that operates in the space of valid PDDL descriptions, $\mathcal{P}$ (each element of $\mathcal{P}$ is a pair (D, P)): $\forall r_e \in R_e$, $r_e : \mathcal{P} \rightarrow \mathcal{P}$. Given that they operate over $\mathcal{P}$, the PDDL representation changes can be applied in sequence. We define the composition of two changes $\sigma_{\langle r_e^1, r_e^2 \rangle}$ over a pair (D, P) in the usual way: $\sigma_{\langle r_e^1, r_e^2 \rangle}(D, P) = (r_e^1 \circ r_e^2)(D, P) = r_e^2(r_e^1(D, P))$. The composition of these functions is not necessarily symmetric, so the order in which changes are performed is relevant.

For the internal representation, we are restricted to transformations into SAS$^+$. If $\mathcal{S}$ is the space of all SAS$^+$ descriptions, $\forall r_i \in R_i$, $r_i : \mathcal{P} \rightarrow \mathcal{S}$. In order to define each r_i, there are usually two processes applied consecutively to the original PDDL representation to generate the internal one in SAS$^+$. First, there is a *translation* process that generates an initial SAS$^+$ representation; and second, there is a *preprocessing* step that generates an optimized SAS$^+$. Our set of internal representation changes R_i contains pairs (t, p), a translator method t and a pre-processor method p.

States of the Meta-search

The meta-search states contain all the necessary information regarding problem representation and the set of heuristics used for solving the problem. We define meta-search states as follows:

Definition 1 (meta-search state) *A meta-search state s is a tuple $s = (\langle r_e^1, \ldots, r_e^n \rangle, D_n, P_n, pl, r_i, H)$, where $\langle r_e^1, \ldots, r_e^n \rangle$, $r_e^i \in R_e$ is a sequence of external representation changes; D_n and P_n are the resulting domain and*

problem generated by applying this sequence to the original domain and problem, $(D_n, P_n) = \sigma_{\langle r_e^1,\ldots,r_e^n\rangle}(D, P)$*;* $pl \in Pl$ *is a planner;* $r_i \in R_i$ *is an internal representation change; and* $H \subseteq \mathcal{H}$ *is a subset of heuristics.* [1]

Regarding the PDDL representation, the states contain both the sequence of representation changes and the obtained domain and problem. As we will explain later, the selected changes can affect the applicability of operators (e.g., some representation changes can only be applied once).

With relation to the internal representation, every state of the meta-search contains one translator and pre-processor pair $r_i = (t, p)$. Therefore, meta-search states define implicitly a SAS$^+$ representation, that can be obtained by applying the internal representation procedure to the resulting domain and problem: $r_i(D_n, P_n)$. This involves executing first the translator t and then the pre-processor p on the planning task defined by D_n and P_n. While we could implement all changes done at the PDDL level (R_e) as changes at the SAS$^+$ representation, we keep both kinds of changes independent. On one hand, changes are more easily performed at the PDDL level than at the SAS$^+$ level. On the other hand, and more importantly, keeping these changes at the PDDL level could allow to use other PDDL-based planners that do not work with SAS$^+$ representations.

The subset of heuristics $H \subseteq \mathcal{H}$ represents the selected heuristic for solving the task, defined as the maximum value over all the heuristics in H. Since we are doing optimal planning all heuristics in $\mathcal{H}$ should be admissible.

The meta-search state space is composed of all the possible combinations of sequences of external representation changes, a planner, an internal representation change and a subset of heuristics. We only consider the internal representation change and heuristics which are accepted by the chosen planner.

Given that this meta-search state space can be huge and that it will be traversed at problem solving time (on-line), the challenge consists of how to perform the search efficiently.

The initial meta-search state is $(\emptyset, D, P, pl, r_i, H)$. It contains the empty sequence, the original domain and problem, and default values for pl, r_i, and H.

Meta-search Operators

Given a state $s = (\langle r_e^1, \ldots, r_e^n\rangle, D_n, P_n, pl, r_i, H)$ of the meta-search, there are four types of modifications that can be applied to generate a new state: adding an additional change to the sequence of external changes; selecting a (possibly different) planner, selecting a (possibly different) internal representation change; and selecting a (possibly different) subset of heuristics. Thus, we define meta-search operators as follows.

Definition 2 (meta-search operator) *A meta-search operator is a tuple* $mo = (mo_{r_e}, mo_{pl}, mo_{r_i}, mo_H)$*, where the first component defines an external representation change* $r_e \in R_e$ *to be added to the sequence, the second component defines a planner* $pl \in Pl$*, the third component defines an internal representation change* $r_i \in R_i(pl)$ *to be used, and the last one defines a selection of heuristics* $H \subseteq \mathcal{H}(pl)$*.*

[1] n refers just to the number of elements in the sequence.

The operator $mo = (mo_{r_e} = r_e^{n+1}, mo_{pl} = pl', mo_{r_i} = r_i', mo_H = H')$ applied to the state s generates a new state $s' = (\langle r_e^1, \ldots, r_e^n, r_e^{n+1}\rangle, D_{n+1}, P_{n+1}, pl', r_i', H')$, where D_{n+1} and P_{n+1} are the domain and problem generated by the new sequence: $(D_{n+1}, P_{n+1}) = \sigma_{\langle r_e^1,\ldots,r_e^n,r_e^{n+1}\rangle}(D, P)$. pl' is a planner. The internal (SAS$^+$) representation to be used by the planner pl' will be the result of: $r_i'(\sigma_{\langle r_e^1,\ldots,r_e^n,r_e\rangle}(D, P))$. All three, pl', r_i' and H', can take the same value in consecutive states.

Regarding the external changes (PDDL representation), R_e, the MSP version presented at the 2018 IPC only considers *baggy-all* and *neutral*. *baggy-all* applies bagging to all types in the domain according to the Baggy technique (Riddle et al. 2016). This technique consists of replacing by counters all objects of the same type whose name is not relevant. *neutral* makes no change to the PDDL representation.

As planners we consider FD optimal (Helmert 2006) and SYMBA* (Torralba, Linares-López, and Borrajo 2016).

Regarding the translator and pre-processor pairs $(t, p) \in R_i$ we consider the following options. For the FD planner, we defined one translator and two preprocessors. The translator is the standard FD translator (Helmert 2006). For the preprocessors the first option is the standard FD pre-processor. The second one is the h^2-based one defined in (Alcázar and Torralba 2015), that we will refer to as h^2. For the SYMBA* planner we only consider its standard translator and preprocessor.

Finally, we consider the following heuristics: for FD, potentials (Seipp, Pommerening, and Helmert 2015), operator counting (Florian Pommerening and Bonet 2014), ipdb (Haslum et al. 2007), lmcut (Helmert and Domshlak 2009) and blind; for SYMBA* only its default heuristic. For the version presented at the competition the set of heuristics of meta-search states contains just one element. Thus, only one heuristic is selected at each meta-search node.

As an example, a meta-search operator can be $mo_{r_e} =$ *baggy-all*, $mo_{pl} =$ FD, $mo_{r_i} = (t =$ FD$^+, p =$ FD$)$, and $mo_H = \{$potentials$\}$ which applies Baggy to produce a new PDDL representation, selects the planner FD, transforms that into SAS$^+$ using the FD$^+$ translator and the standard FD preprocessor and selects the heuristic *potentials*.

Meta-search Search Technique

MSP performs greedy search with a technique similar to enforced-hill-climbing (Hoffmann and Nebel 2001). Algorithm 2 shows a high-level description of the meta-search. It takes as input a domain D, a problem P, the PDDL representation changes R_e, the planners Pl, the internal representation changes R_i, the set of heuristics $\mathcal{H}$, a meta-search time bound T_m, which will be the maximum time the whole meta-search process can take and a planning time bound T_P, which is the minimum time that the meta-search process assumes the planner will have to solve the problem. The META-SEARCH returns the configuration of the best meta-search state $(D^{end}, P^{end}, pl^{end}, r_i^{end}, H^{end}, \pi)$.

META-SEARCH first generates the operators, $\mathcal{MO}$, given the possible representation changes R_e and R_i, and builds the initial state. We start the search with the original do-

Algorithm 2 META-SEARCH($D, P, R_e, Pl, R_i, \mathcal{H}, T_E, T_m, T_P$)

Inputs: domain D, problem P, PDDL rep. changes R_e, Planners Pl, internal rep. changes R_i, set of heuristics $\mathcal{H}$, estimation time bound T_E, meta-search time bound T_m, planning time bound T_P

Outputs: configuration, $((D^{end}, P^{end}, pl^{end}, r_i^{end}, H^{end}), \pi)$

```
1: MO ←GENERATE-OPERATORS(R_e, Pl, R_i, H)
2: init ← (∅, D, P, SymBA*, (t = SymT, p = SymP), SymH)
3: f, π ← EVALUATE(init, T_E)
4: best ← init
5: best_f ← f
6: succ ← SUCCESSORS(init, MO)
7: while T_m not reached and π = ∅ and succ ≠ ∅ do
8:    s ← pop(succ)
9:    f, π ← EVALUATE(s, T_E)
10:   if f > best_f then
11:      best ← s
12:      best_f ← f
13:      succ ← SUCCESSORS(s, MO)
14: return (best = (D^end, P^end, pl^end, r_i^end, H^end), π)
```

main and problem, and the SYMBA* planner together with its translator, preprocessor and heuristics. Then, the initial state is evaluated. The EVALUATE function returns an evaluation of the state and a plan π (that will be empty if no plan is found during evaluation). Then, it generates its successors (the SUCCESSORS function returns a list of meta-search states), and starts evaluating each successor in order. As soon as it finds a state with a better evaluation than its parent's, it stops evaluating the current set of successors, and continues the search, generating the successors of that state. MSP finishes the meta-search if: the time bound is reached; a plan is found while evaluating a meta-search state; or if the list of successors is empty (it did not find a better meta-search state than its parent at any level).

An important decision is on the maximum time EVALUATE will be allowed to evaluate a meta-search state, T_E. If EVALUATE has too little time, the quality of its answers will be poor. If it has too much time, then the meta-search will not be able to search very many meta-search states in this space.

In order to avoid visiting some uninteresting or impossible combinations, we prune any state s when any of the following conditions apply: the same meta-search operator was applied in any ancestor of s; the last r_e is Baggy, and any ancestor of s has already applied Baggy in any form; the last r_e is Baggy, and any previously evaluated state in the search tree has already tried to apply Baggy and the problem could not be bagged.

Meta-Search State Evaluation

To evaluate a meta-search state, s, the meta-search calls EVALUATE(s, T_E). Algorithm 3 shows its pseudo-code. s is the state being evaluated and T_E is the evaluation time limit. Given s and T_E, EVALUATE returns a reasonable estimate of the "goodness" of s within the time bound T_E.

The evaluation of meta-search states is a sampling process that executes the planner defined in the corresponding state until reaching the estimation timeout. Our measure of

Algorithm 3 EVALUATE(s, T_E)

Inputs: meta-search state s, evaluation time bound T_E

Outputs: evaluation f, plan π

```
1: (⟨r_e^1, ..., r_e^n⟩, D_n, P_n, pl, r_i, H) ← s
2: S ← r_i(D_n, P_n)
3: max_flimit, π ← pl(S, T_E, H)
4: return f = max_flimit, π
```

the goodness for meta-search states is how far the search went when sampling; that is, the maximum f-level of the expanded nodes. The basic idea is that the configuration that achieves a higher f-level is better than one that reaches a lower f-level. If a plan is found during the evaluation, that plan is also returned.

Planner

The meta-search algorithm returns the expected best combination of representation, translation, and pre-processing techniques, plus selected heuristics and possibly a plan. If the meta-search finds a solution plan while evaluating any state, MSP just returns it. Otherwise, the planner is executed with the new domain and problem definitions, as well as the selected translator, pre-processor and heuristic.

If the representation chosen was a bagged representation, then the solution will need to be translated back into the original representation as well. Luckily this is a very fast process, linear in the size of the solution path.

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, 2–6.

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS+ planning. *Computational Intelligence* 11(4):625–655.

Barley, M.; Franco, S.; and Riddle, P. 2014. Overcoming the utility problem in heuristic generation: Why time matters. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 38–46.

Cenamor, I.; de la Rosa, T.; and Fernández, F. 2016. The IBaCoP planning system: Instance-based configured portfolios. *Journal of Artificial Intelligence Research* 56:657–691.

Florian Pommerening, Gabriele Roeger, M. H., and Bonet, B. 2014. Lp-based heuristics for cost-optimal planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 226–234. AAAI Press.

Fuentetaja, R., and de la Rosa, T. 2016. Compiling irrelevant objects to counters. special case of creation planning. *AI Communications* 29(3):435–467.

Fuentetaja, R.; Barley, M.; Borrajo, D.; Douglas, J.; Franco, S.; and Riddle, P. 2018. Meta-search through the space of representations and heuristics on a problem by problem basis. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*.

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)*, 350–353.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, 1007–1012.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, 162–169.

Helmert, M.; Röger, G.; and Karpas, E. 2011. Fast downward stone soup: A baseline for building planner portfolios. In *Working Notes of the ICAPS Workshop on Planning and Learning (PAL)*, 28–35.

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

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

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

Howe, A. E.; Dahlman, E.; Hansen, C.; Scheetz, M.; and von Mayrhauser, A. 1999. Recent advances in AI planning. In *Proceedings of the 5th European Conference on Planning (ECP)*, 62–72. Springer-Verlag.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *Proceedings of the 10th Europan Conference on Artificial Intelligence*, 359–363. New York, NY, USA: John Wiley & Sons, Inc.

Núñez, S.; Borrajo, D.; and Linares-López, C. 2015. Automatic construction of optimal static sequential portfolios for AI planning and beyond. *Artificial Intelligence* 226:75–101.

Riddle, P.; Douglas, J.; Barley, M.; and Franco, S. 2016. Improving performance by reformulating PDDL into a bagged representation. In *Working Notes of the ICAPS Workshop on Heuristics and Search for Domain-Independent Planning (HDIP)*.

Rintanen, J. 2012. Planning as satisfiability: Heuristics. *Artificial Intelligence* 193(Supplement C):45 – 86.

Seipp, J.; Pommerening, F.; and Helmert, M. 2015. New optimization functions for potential heuristics. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, 193–201. AAAI Press.

Torralba, Á.; Linares-López, C.; and Borrajo, D. 2016. Abstraction heuristics for symbolic bidirectional search. In *Proceedings of IJCAI'16*.

Vallati, M.; Hutter, F.; Chrpa, L.; and McCluskey, T. 2015. On the effective configuration of planning domain models. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*.