Meta-Search Through the Space of Representations and Heuristics on a Problem by Problem Basis

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

Two key aspects of problem solving are representation and search heuristics. Both theoretical and experimental studies have shown that there is no one best problem representation nor one best search heuristic. Therefore, some recent methods, e.g., portfolios, learn a good combination of problem solvers to be used in a given domain or set of domains. There are even dynamic portfolios that select a particular combination of problem solvers specific to a problem. These approaches: (1) need to perform a learning step; (2) do not usually focus on changing the representation of the input domain/problem; and (3) frequently do not adapt the portfolio to the specific problem. This paper describes a meta-reasoning system that searches through the space of combinations of representations and heuristics to find one suitable for optimally solving the specific problem. We show that this approach can be better than selecting a combination to use for all problems within a domain and is competitive with state of the art optimal planners.

Introduction

Automated planning now offers a wide range of domain-independent heuristics and search methods. Theoretical analysis can shed light on the impact of a single heuristic in different domains (Hoffmann 2005). However, it has been shown that a combination of heuristics improves the results over a single one (Helmert 2006). And analysing a combination of heuristics becomes a much harder task. Also, there is no known best combination for all problems within a domain and across domains. This has motivated research into techniques for selecting *good* combinations. There have been proposals to either manually define the best combinations (Richter and Westphal 2010), generate portfolios that combine planners (Cenamor, de la Rosa, and Fernández 2016; Helmert, Röger, and Karpas 2011; Gerevini, Saetti, and Vallati 2009; Núñez, Borrajo, and Linares-López 2015), or auto-tune the parameters of planners (Fawcett et al. 2011). Many of these techniques require off-line training and select combinations that depend on the planning domain but not on the particular problem to solve.

An additional and relevant aspect that has been much less studied in automated planning is the input representation. 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 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 been examined (Kautz and Selman 1992; Rintanen 2012).

In this paper, we present Meta-Search for Planning (MSP), an approach that, given a problem and domain, searches through the space of representations and heuristics to find a good combination to use in solving the problem. We 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. We also vary the procedure to obtain the internal representation.

We make the following claims: (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) this meta-level search can be made efficient enough to be competitive within the IPC constraints.

Meta-search

We will 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 ($\mathcal{MS}$) and meta-search operators ($\mathcal{MO}$), and the problem solving method: the search algorithm and the heuristics.

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 and heuristics, it also receives as inputs the sets of representation change operators that can be applied, R_e and R_i, the set of heuristics that can be used, $\mathcal{H}$, and the time bound, T. R_e is the set of representation changes that generate a new PDDL representation, referred to as *external* representation changes. R_i are the *internal* representation changes, that transform a PDDL representation into an internal one such as SAS$^+$. The last input is a time bound T, representing the maximum time to solve the problem.

MSP is given the maximum time it can use to solve a problem (e.g., 1800s as in the IPC). Within that time limit, it must use some of that time to select a good combination of representation changes 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 two time limits: one for the meta-search and the other as an estimate of the time to be given later to the planner. We assume $T/2$ for both.

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}; and some information related to the selected configuration for the planner, r_i^{end} and H^{end}, the procedures to generate the internal representation and the set of heuristics that were selected for the best state found. 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.

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

```
Require: domain $D$, problem $P$, PDDL rep. changes $R_e$, internal
    rep. changes $R_i$, set of heuristics $\mathcal{H}$, time bound $T$
Ensure: plan
1: $(D^{end}, P^{end}, r_i^{end}, H^{end}, \pi) \leftarrow$
2:     META-SEARCH($D, P, R_e, R_i, \mathcal{H}, T/2, T/2$)
3: if $\pi = \emptyset$ then
4:     $T \leftarrow T -$ consumedTime
5:     plan ← PLANNER($D^{end}, P^{end}, r_i^{end}, H^{end}, T$)
6: else
7:     plan ← $\pi$
8: return plan
```

Representation Changes

The representation changes 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 heuristics for solving the problem. We define meta-search states as follows:

Definition 1 (meta-search state) *A meta-search state s is a tuple $s = (\langle r_e^1, \ldots, r_e^n \rangle, D_n, P_n, r_i, H)$, 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, $(D_n, P_n) = \sigma_{\langle r_e^1, \ldots, r_e^n \rangle}(D, P)$; $r_i \in R_i$ 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

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

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 allows us 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, an internal representation change, and all subsets of heuristics. Thus, the size of this space is exponential with respect to the number of representation changes and heuristics. An estimation of this size can be computed assuming that the same PDDL representation change $r_e \in R_e$ cannot be applied more than once (which is not necessarily true for all PDDL representation changes) and that all pairs $r_i = (t, p) \in R_i$ are possible (which is also not necessarily true). In this case, the size of the state space would be:

$$\sum_{k=0}^{|R_e|} \left[\binom{|R_e|}{k} \times k! \right] \times |R_i| \times 2^{|\mathcal{H}|}$$

where the first term (in square brackets) is the number of subsets of external representation changes with k elements, times the number of sequences (permutations) that can be generated for each of those subsets. The second and third terms are the number of internal representation changes and the number of possible subsets of heuristics respectively.

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 search efficiently. The initial meta-search state is $(\emptyset, D, P, r_i, \mathcal{H})$. It contains the empty sequence, the original domain and problem, a default value for r_i, and all the input heuristics.

Meta-search Operators

Given a state $s = (\langle r_e^1, \ldots, r_e^n \rangle, D_n, P_n, r_i, H)$ of the meta-search, there are three 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) 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 $mo = (mo_{r_e}, mo_{r_i}, mo_H)$, where the first component defines an external representation change $r_e \in R_e$ to be added to the sequence, the second component defines an internal representation change $r_i \in R_i$ to be used, and the last one defines a selection of heuristics $H \subseteq \mathcal{H}$.*

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

To improve search efficiency, we use a meta-search evaluation function that is based on the RIDA* planner (Barley, Franco, and Riddle 2014).[2] In this paper, RIDA* is used to perform sampling over the search space in order to generate three outputs. These outputs are the numerical information that we use to evaluate a given representation; a selection of heuristics appropriate for the planning task with that representation, and possibly a solution plan if one is found during the sampling. Since RIDA* works on top of FD, a meta-search state can only be evaluated by RIDA* if it has a SAS$^+$ representation. Hence, the meta-search operators always consist of one or more external representation changes and one pair of internal representation changes. The details on how meta-search states are evaluated are described later.

The call to RIDA* to evaluate a meta-search state returns the associated combination of heuristics, H_{RIDA^*}. Hence, the third component of meta-search operators is bound to that combination: $mo = (mo_{r_e}, mo_{r_i}, H_{\text{RIDA}^*})$.

Regarding the external changes (PDDL representation), R_e, we consider the following ones.[3] **baggy-all** applies bagging to all types in the domain according to the Baggy technique (Riddle et al. 2016), that consists of replacing by counters all objects of the same type whose name is not relevant. **alphabetical-inverse-order** renames each action in the domain file so planners that order actions based on alphabetical order (e.g. FD (Helmert 2006)) consider the actions in the reverse order. **alphabetical-random-order** is similar to the previous operator, but changing the action names randomly. **inverse-order** reverses the order of actions. **random-order** changes the order of actions randomly. And **neutral** makes no change to the PDDL representation.

So as stated earlier, the order, in which these representation change operators are applied, matters. As an example, FD can obtain different results by applying alphabetical-inverse-order followed by Baggy, than by applying Baggy followed by alphabetical-inverse-order, since Baggy generates actions that have different suffixes.

Regarding the translator and pre-processor pairs $(t, p) \in R_i$ we consider the following options. **Translator,** t: given that we are using the FD framework, each new PDDL representation must be translated into SAS$^+$. There are two options for the translation. The first one is the standard FD translator (Helmert 2006). A second option, that we will refer to as FD$^+$, is only available when Baggy is used. In this case, Baggy generates a list of invariants that are sent to the FD translator code. The planner uses these invariants in addi-

[2]RIDA* was denoted as RA* in its original paper (Barley, Franco, and Riddle 2014).

[3]We leave to future work using other representation changes available for optimal planning.

tion to its own. This can make many fewer SAS$^+$ variables, which can have a positive or negative effect on the heuristics. **Pre-processor,** p: again there are two choices. The first one 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 some heuristics, using the latter pre-processor has an advantage.

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

Meta-search Search Technique

We have opted for a 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 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 $(D^{end}, P^{end}, r_i^{end}, H^{end}, \pi)$. 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 domain and problem, as well as the FD translator and pre-processor. Thus, the initial state is $(\emptyset, D, P, (t = \text{FD}, p = \text{FD}), \mathcal{H})$. After that, the initial state is evaluated. The EVALUATE function returns a set of heuristics, 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).

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

```
Require: domain $D$, problem $P$, PDDL rep. changes $R_e$, internal rep. changes $R_i$, set of heuristics $\mathcal{H}$, meta-search time bound $T_m$, planning time bound $T_P$
Ensure: configuration, $(D^{end}, P^{end}, r_i^{end}, H^{end}, \pi)$
 1: $MO \leftarrow$ GENERATE-OPERATORS$(R_e, R_i)$
 2: init $\leftarrow (\emptyset, D, P, (t = \text{FD}, p = \text{FD}), \mathcal{H})$
 3: $T_E \leftarrow T_m/3$
 4: $H, f, \pi \leftarrow$ EVALUATE(init,$T_P, T_E, \mathcal{H}$)
 5: best $\leftarrow (\emptyset, D, P, (t = \text{FD}, p = \text{FD}), H)$
 6: best_$f \leftarrow f$
 7: succ $\leftarrow$ SUCCESSORS(init,$MO$)
 8: while $T_m$ not reached and $\pi = \emptyset$ and succ$\neq \emptyset$ do
 9:     $s = (\langle r_e^1, \ldots, r_e^n \rangle, D_n, P_n, r_i, 0) \leftarrow$ pop(succ)
10:     $H, f, \pi \leftarrow$ EVALUATE$(s, T_P, T_E, \mathcal{H})$
11:     $s \leftarrow (\langle r_e^1, \ldots, r_e^n \rangle, D_n, P_n, r_i, H)$     sets the value of H
12:     if $f >$best_$f$ then
13:         best $\leftarrow s$
14:         best_$f \leftarrow f$
15:         succ $\leftarrow$ SUCCESSORS$(s, MO)$
16: return $(D^{end}, P^{end}, r_i^{end}, H^{end}, \pi) \leftarrow$ CONFIG(best,$\pi$)
```

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. Currently, we are setting T_E to one-third of T_m, so that in the worst case we sample at least three different configurations. We show in the Experimental Results section that MSP usually explores more than three configurations.

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; or the last r_e is random, and an ancestor of s has already applied a random operator.

Meta-Search State Evaluation

To evaluate a meta-search state, s, meta-search calls EVALUATE$(s, T_P, T_E, \mathcal{H})$. Algorithm 3 shows its pseudocode. s is the state being evaluated, T_P is the estimated planner's time limit, T_E is the evaluation time limit, and $\mathcal{H}$ is the collection of candidate heuristics. Given s and T_E, we would like EVALUATE to return a reasonable estimate of the "goodness" of s within the time bound T_E. Ideally, the goodness measure would be how long FD will take, using A*, to solve that problem. However, that is hard to predict. Instead, we use a goodness measure that allows us to determine whether one meta-search state configuration makes A* more likely to solve the problem within the T_P time limit than another meta-search state configuration.

noend 3 EVALUATE$(s, T_P, T_E, \mathcal{H})$

```
Require: meta-search state $s$, planning time bound $T_P$, evaluation time bound $T_E$, set of heuristics $\mathcal{H}$
Ensure: selected heuristics $H$, evaluation $f$, plan $\pi$
 1: $(\langle r_e^1, \ldots, r_e^n \rangle, D_n, P_n, r_i, \mathcal{H}) \leftarrow s$
 2: $S \leftarrow r_i(D_n, P_n)$
 3: $H, b, T_n, \pi \leftarrow$ RIDA$^*(S, T_E, \mathcal{H})$
 4: $f \leftarrow \log_b \frac{T_P}{T_n}$
 5: return $H, f, \pi$
```

If we knew how far A* would go within T_P for each of two given configurations, then the configuration where A* achieves a higher f-limit would be better than one that reaches a lower f-limit. We use a variation of RIDA* (Barley, Franco, and Riddle 2014) to evaluate configurations. RIDA* receives the SAS$^+$ representation of the planning task, the evaluation time limit and the set of heuristics. The SAS$^+$ representation is generated on line 2 and referred to as S.

RIDA* returns the following information about this state's representation: $\langle H, b, T_n, \pi \rangle$. $H \subseteq \mathcal{H}$ is the recommended subset of heuristics; b, the average heuristic branching factor when using H; T_n, the average time-per-node when using H; and π, the plan that RIDA* may have found while evaluating this configuration or $\emptyset$ otherwise. Using the formula found on line 4, EVALUATE uses the information returned by RIDA* to calculate the maximum f-limit f that FD is estimated to reach while solving the problem with this representation and H. This f-limit is our measure of the goodness of this state. As far as we are aware, this is the first use of maximum f-limit as a measure of goodness. Previous work using RIDA* used other less accurate measures of goodness. EVALUATE returns $\langle H, f, \pi \rangle$.

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 uses the new domain and problem definitions, as well as the selected translator and pre-processor and the heuristics to be used. We have selected RIDA*, which is built upon FD, as the planner to be used to perform optimal planning using A* with the maximum of the chosen (admissible) heuristics H. 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.

Experimental Results

Our goal with the experiments is to analyze whether the proposed meta-search algorithm could automatically detect configurations that work well for a specific problem. Our experimental setting included solving all problems from the optimal track of IPC'11 and '14, except for the domains using conditional effects from IPC'14 (491 problems in total). We used the same setting as in the IPC: time limit of 1800s and memory limit of 4Gb. The meta-search algorithm was programmed in Common Lisp, while RIDA* runs within the FD framework. We ran the experiments on a cluster with Intel XEON 2.93 Ghz nodes using Linux Ubuntu 12.04 LTS.

We ran the meta-search with a maximum time of 900s to find a good representation and set of heuristics, and we used the remaining time for solving the problem. We gave a maximum time of 300s to evaluate each meta-search state. The main goal of this paper is to study whether MSP can effectively find a good combination. We believe RIDA* already finds the best subset of heuristics for a given representation. Therefore, we used the heuristics in our version of RIDA*: 42 ga-PDBs (Edelkamp 2006) (generated by a genetic algorithm), LMCut (Helmert and Domshlak 2009), iPDB (Haslum et al. 2007) and hmax (Bonet and Geffner 2001). Including other heuristics or representation changes is left for future work.

We first compare the performance of MSP against three baselines that use the same RIDA* planner: (1) OFD: original representation and FD pre-processor; (2) Oh^2: original representation and h^2 pre-processor; and (3) Bh^2: bagged representation and h^2 pre-processor on the problems that can be bagged. The results can be observed in the first five columns of Table 1. MSP solves more problems than the base configurations. It solves 323 problems while the two executions of RIDA* on the original representation solved 276 and 315, respectively. In comparison with OFD, MSP gains 55 more problems but loses 8 by choosing an incorrect representation or by running out of time. The comparison is similar with Oh^2 where we gain 21 problems but lose 13. Only 14% (69 of 491) of all problems failed due to memory after meta-search finished. Currently, we do not include memory usage predictions into the decision making process when choosing a representation.

For further comparison we combined the results of our three baseline solvers to get a Virtual Best Solver (VBS) which is represented by "correctly choosing" between the three baseline solvers on a problem by problem basis. Column VBS of Table 1 shows the problems solved by any of the three baselines. Since this is done on each problem it can have a higher number than any of its baselines in the domain row, which happens in Woodworking11, Hiking14 and Tetris14. The VBS gives us an upper bound on how well MSP might perform, solving 343 problems. It is a lower upper bound, because we did not use all the other representation changes as baselines (such as *alphabetical inverse*). This shows we have room for improvement, and might be able to solve another 20 problems with further fine-tuning.

The average number of meta-search states evaluated by the meta-search across all problems is 5.8 ($\sigma^2 = 3.8$). In MSP almost all the time is spent on the evaluation of meta-search states by RIDA*. The average number of generated meta-search states is 11.2 ($\sigma^2 = 10.26$). The average time spent on meta-search is 642s ($\sigma^2 = 382.5$).[4] In 55% of the problems, meta-search did not use all of the 900s time limit.

Table 1 also shows comparisons to other state-of-the-art planners, such as FD with LMCut (LMh^2) and SYMBA* (Sy), the winner of IPC 2014 optimal track. Given that SYMBA* was the winner of IPC'14, we could have used SYMBA* within MSP. However, in its current form SYMBA* can only use one form of heuristic based on abstractions. Also, we have not determined how to use SYMBA* to compute estimations when using different representation changes. If we use RIDA* to make the estimations and SYMBA* as planner, MSP might frequently make poor estimations since the estimates are tightly linked to the planner. Thus, the use of SYMBA* is left for future work.

MSP solved 323 problems while LMCut only solved 264, a substantial difference, especially in the case of optimal planning. The biggest gains are in Barman, Childsnack, Hiking, and Visitall. Barman and Childsnack are solved because MSP selected Baggy which makes a big difference in these domains. Besides, MSP solves some Hiking and Visitall problems in the original representation that LMCut cannot. So RIDA*'s broader choice of heuristics is also a fac-

[4]The meta-search time limit is not strictly observed, but the total time limit for MSP is strictly controlled.

Table 1: The first seven columns present: coverage results comparing MSP, original representation with the FD pre-processor, OFD, and the h^2 pre-processor, Oh^2, Baggy with the h^2 pre-processor, Bh^2, the Virtual Best Solver, VBS, FD with LMCut and h^2 pre-processor, LMh^2, and SYMBA*, Sy. The remainder of the columns: selections made by MSP on the most relevant heuristics, internal and external representation changes (Original, O, Bagging, B, and Alphabetical-Inverse, AI). Each cell represents ⟨number of times selected and the problem was solved/number of times selected⟩.

		RIDA*						Heuristics			R_i				R_e		
											Translator		Preprocessor				
Domain	MSP	OFD	Oh^2	Bh^2	VBS	LMh^2	Sy	LM	ipdb	gapdb	FD	FD$^+$	FD	h^2	O	B	AI
Barman11	7	4	4	12	12	4	12	1/12	5/8	1/1	5/5	2/2	3/3	4/4	0/10	7/7	1/1
Elevators11	19	19	19	12	19	18	19	4/4	5/6	7/8	17/18	2/2	13/13	6/7	9/9	7/8	4/4
Floortile11	14	5	14	13	14	14	14	10/16	0/0	1/1	12/17	2/3	3/3	11/17	3/3	11/17	3/5
NoMystery11	17	20	17	17	20	14	16	0/0	2/5	2/5	17/19	0/1	15/16	2/4	15/15	2/5	0/0
Openstacks11	15	16	16		16	16	20	11/16	0/0	0/0	15/20	0/0	15/20	0/0	15/21	0/0	0/0
ParcPrinter11	17	13	17		17	17	17	0/3	0/0	0/0	17/20	0/0	11/14	6/6	17/20	0/0	0/0
Parking11	7	7	7		7	3	2	0/1	7/19	0/0	7/20	0/0	7/19	0/1	1/7	0/0	6/14
Pegsol11	17	18	18		18	18	20	0/3	0/0	14/14	17/20	0/0	7/10	10/10	8/11	0/0	9/9
Scanalyzer11	13	13	13		13	12	9	1/8	1/1	3/3	13/20	0/0	11/18	2/2	12/19	0/0	1/1
Sobokan11	20	20	20		20	20	20	0/0	9/9	4/4	20/20	0/0	10/10	10/10	12/12	0/0	6/6
Tidybot11	16	14	17		17	17	17	7/11	5/5	0/0	16/20	0/0	7/9	9/11	8/10	0/0	8/10
Transport11	10	10	10	6	10	6	11	0/4	0/0	5/9	10/16	0/4	10/12	0/8	9/11	1/9	0/0
Woodwork.11	14	11	15		16	15	20	2/7	0/0	3/5	14/20	0/0	7/13	7/7	12/16	0/0	2/4
Visitall11	18	18	16		18	10	12	0/0	0/0	2/4	18/20	0/0	17/18	1/2	16/17	0/0	2/3
Barman14	6	0	0	6	6	0	6	0/0	6/14	1/1	3/4	3/10	1/1	5/13	0/0	6/14	0/0
Childsnack14	8	0	0	8	8	0	4	0/8	0/0	8/12	5/14	3/6	0/8	8/12	0/8	8/12	0/0
Floortile14	18	4	17	20	20	17	20	16/18	0/0	0/0	16/18	2/2	0/0	18/20	0/0	18/20	5/7
GED14	19	19	19		19	15	20	0/0	10/11	7/8	19/20	0/0	13/14	6/6	11/12	0/0	6/6
Hiking14	16	16	16	16	17	9	19	0/0	8/12	7/7	14/17	2/3	6/8	10/12	9/9	7/11	1/1
Openstacks14	3	3	3		3	3	20	3/16	0/0	0/4	3/20	0/0	3/18	0/2	3/18	0/0	0/2
Parking14	6	6	6		6	3	4	0/1	6/19	0/0	6/20	0/0	6/19	0/1	3/9	0/0	3/11
Tetris14	10	8	9	8	10	9	10	0/4	7/10	0/1	7/13	3/4	4/5	6/12	2/3	7/13	2/3
Tidybot14	11	9	13		13	13	10	9/19	1/1	0/0	11/20	0/0	4/7	7/13	5/9	0/0	6/10
Transport14	7	9	9	5	9	6	9	1/8	0/0	3/8	6/19	1/1	6/15	1/5	6/13	1/4	0/3
Visitall14	15	14	15		15	5	8	0/5	0/0	2/2	15/20	0/0	15/20	0/0	14/19	0/0	1/1
Total (491)	323	276	315	123	343	264	339	164	120	97	452	39	302	189	281	123	101
% selected								23%	25%	20%	92%	8%	61%	39%	57%	25%	20%
Ratio solved when selected								0.56	0.6	0.75	0.66	0.5	0.65	0.66	0.68	0.64	0.7

tor. SYMBA* solves more problems than MSP, but most of this difference is caused by Openstacks14 where no good heuristic is known so that SYMBA*'s bidirectional search has a clear advantage. Without it MSP and SYMBA* would have solved 320 and 319 problems respectively. This is fairly amazing since SYMBA* has access to bidirectional search.

There was no heuristic nor representation change that was selected for every problem in every domain. Roughly 27% of the problems (87 problems) were solved during meta-search state evaluation. However, if a problem is solved during meta-search state evaluation, we say that the meta-search state's representation changes were selected. As far as which heuristics were included in the selected combination on a problem basis: ga-PDBs was chosen 20%; iPDB 20%; LM-Cut, 10% and assigned another 13%;[5] and hmax, 1%. Regarding internal representation changes, FD$^+$ was chosen as translator on 8% of the problems, and h^2 as pre-processor on 39%. FD standard was chosen in the rest of cases as translator and pre-processor respectively. With relation to external representation changes (PDDL): Baggy was chosen in 25% of the problems; Alphabetical-Inverse, 20%; Inverse, 5%; and Random and Alphabetical Random, around 1% each. For each problem, more than one heuristic and external representation change could have been chosen, so these numbers do not need to add to 100%. The specific number of selections made per domain for the most relevant heuristics, internal and external representation changes are shown in Table 1. Each cell represents ⟨*number of times selected and the problem was solved/number of times selected*⟩. The row *Total* contains the total number of times the alternative was selected.[6] The last two rows represent the total percentage of times the alternative was selected and the ratio of solved problems when selected, respectively. In all domains, except Openstacks11, there is diversity in the selections. This justifies working on a problem by problem basis. The ratio of solved problems varies between 0.5 and 0.75.

Non-alphabetical reordering of the operators in the domain file makes no difference to the FD planner, as FD re-sorts the actions into alphabetical order. MSP discovers that the non-alphabetical reordering representation change does

[5]Because RIDA* could not return an evaluation in time, and in that case LMCut is selected as the default heuristic.

[6]Note that two or more heuristics and/or representation changes can be selected for a single problem.

not affect the results. However, non-alphabetical orderings might be relevant to other planners. Reordering of the operators using "alphabetical" does change the order in which alternatives are chosen by FD. These changes are chosen more frequently by MSP. Analysis of the results shows that changing the order of the operators alphabetically does affect the behavior of a number of the heuristics, including LMCut. So *alphabetical-inverse-order* can make a difference for optimal planning, which was discovered by MSP and was not what we expected.

For all the representation changes there are problems for which that change is a bad decision. MSP has the advantage of deciding when and when NOT to use them. In addition, these results were not tuned in any way for the paper. If we were competing in the IPC, with the FD planner, we would not use inverse, or random since they have little effect, and we would fine tune the time split between the meta-search and the actual problem solving. So it is very surprising that we can already get results that almost equal SYMBA*.

In order to analyze how good the selected MSP combination is compared to VBS, we assumed that the meta-search computation time is negligible. Thus, we compared MSP against the base configurations where their time limit is 900s (i.e., $T/2$). In this case MSP solves 321 problems and VBS solves 331, only 10 problems more than MSP. OFD-T/2 solves 267, Oh^2 solves 306 and Bh^2 solves 118. While MSP only solves two problems less than with 1800s, the other configurations solve up to 12 problems less.

In the MSP results we used a specific order of meta-search operators: Baggy options are preferred over the others, FD$^+$ is preferred over FD for the translator (though FD$^+$ can only be applied after Baggy), and h^2 is preferred over FD for the pre-processor. However, we ran an additional experiment using random operator orderings resulting in very similar numbers (MSP solved 324 problems with a random ordering).

Finally, in order to evaluate the impact of performing the meta-search and the goodness of the evaluation function, we carried out another experiment. MSP chose a combination by using a random walk over the same meta-search space. The number of solved problems is 256, considerably less than 323. This result confirms that our meta-search scheme and evaluation function are an improvement over using random decisions.

Related Work

The work reported in the paper relates to several others in at least two different aspects: change of representation or reformulation; and generating good combinations of solvers/heuristics. Examples of reformulation techniques at the PDDL level are the generation of macro-operators (Fikes, Hart, and Nilsson 1972), bagging similar objects (Riddle et al. 2016; Fuentetaja and de la Rosa 2016), action splitting (Areces et al. 2014), or just changing the order of domain elements in the PDDL file (Vallati et al. 2015). Most work on reformulation in planning has been at the domain level, looking for the best representation for all problems within a domain. However, finding the best reformulation for all problems in the same domain does not always work, especially when the problems' distribution varies on two or more dimensions (Riddle, Holte, and Barley 2011). Also, there has been little work on changing representation for each problem, except for systems that learn macro-operators on-line in the satisficing planning (Coles and Smith 2007).

The second type of relevant related work generates good combinations of different problem-solving elements. Some work tunes parameters of solvers, as FD-AUTOTUNE (Fawcett et al. 2011). Others combine heuristics in different queues (Richter and Westphal 2010), compute the sum or maximum of several heuristics (Haslum, Bonet, and Geffner 2005; Domshlak, Karpas, and Markovitch 2010; Haslum et al. 2007), or search in the space of sets of heuristics (Barley, Franco, and Riddle 2014). Finally, other works combine different solvers by generating portfolios that are configured for all domains (Helmert, Röger, and Karpas 2011; Núñez, Borrajo, and Linares-López 2015), for a specific domain (Gerevini, Saetti, and Vallati 2009), or even for a specific problem (Cenamor, de la Rosa, and Fernández 2016). PBP (Gerevini, Saetti, and Vallati 2009) is one of the few works that have proposed a domain-dependent configuration that mixed representation changes (as macro-operators) and problem solvers. As with most portfolio approaches, it requires some training steps before problem solving. Dynamic portfolios (Cenamor, de la Rosa, and Fernández 2016), usually mix off-line meta-reasoning, by learning a predictive model, that they then apply to a problem on-line to decide which combination is appropriate for it. In summary, to our knowledge, no previous approach has attempted to generate on-line (on a problem basis) good combinations of both representation and heuristics.

Concluding Remarks

We have presented MSP, an automated technique that solves optimal planning tasks by performing an on-line meta-search for a good combination of representation and heuristics. It represents a new way to solve problem solving tasks by automatically generating a good combination of representation and heuristics. MSP avoids the expensive learning processes used by other approaches, such as portfolios. Results show that it is competitive with state-of-the-art planners in terms of coverage in many IPC domains.

As future work, we would like to extend this work to satisficing planning, include some estimations based on memory consumption, include other kinds of base search mechanisms (as symbolic planning), add more meta-search operators that implement other representation changes and/or heuristics, and make states' evaluation more efficient.

Acknowledgements

M. Barley was supported by the Air Force Office of Scientific Research, Asian Office of Aerospace Research and Development (AOARD) under award number FA2386-15-1-4069. This work has been partially supported by spanish MICINN project TIN2014-55637-C2-1-R.

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.

Areces, C.; Bustos, F.; Dominguez, M.; and Hoffmann, J. 2014. Optimizing planning domains by automatic action schema splitting. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 11–19.

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.

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

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.

Coles, A., and Smith, A. 2007. Marvin: A heuristic search planner with online macro-action learning. *Journal of Artificial Intelligence Research* 28:119–156.

Domshlak, C.; Karpas, E.; and Markovitch, S. 2010. To max or not to max: Online learning for speeding up optimal planning. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, 1071–1076.

Edelkamp, S. 2006. Automated pattern database design. In *Working Notes of the AAAI'06 Workshop on Heuristic Search, Memory-Based Heuristics and their Applications*, 7–14. AAAI Press.

Fawcett, C.; Helmert, M.; Hoos, H.; Karpas, E.; Röger, G.; and Seipp, J. 2011. FD-Autotune: Domain-specific configuration using fast downward. In *Working Notes of the ICAPS-2011 Workshop on Planning and Learning (PAL)*, 13–20.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3:251–288.

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

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.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. In *Proceedings of the 20th AAAI Conference on Artificial Intelligence*, 1343–1348.

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.

Hoffmann, J. 2005. Where ignoring delete lists works: local search topology in planning benchmarks. *Journal of Artificial Intelligence Research* 24:685–758.

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.

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

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)*.

Riddle, P.; Holte, R.; and Barley, M. 2011. Does representation matter in the planning competition? In *Proceedings of the 9th Symposium on Abstraction, Reformulation and Approximation (SARA)*, 90–98.

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

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)*.