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

Raquel Fuentetaja\textsuperscript{1}, Michael Barley\textsuperscript{2}, Daniel Borrajo\textsuperscript{1}, Jordan Douglas\textsuperscript{2}, Santiago Franco\textsuperscript{1} and Patricia Riddle\textsuperscript{2}

\textsuperscript{1}Departamento de Informática. Universidad Carlos III de Madrid, Spain
\textsuperscript{2}Department of Computer Science. University of Auckland, New Zealand
\textsuperscript{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 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 Liñares-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\textsuperscript{+} (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\textsuperscript{+} 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 \( \pi \) 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 (\( MS \)) and meta-search operators (\( MO \)), 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 may 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 manage 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 \( T - \text{consumedTime} \), where \( \text{consumedTime} \) 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, \( \text{end} \) 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, \( D^\text{end} \) and \( P^\text{end} \); and some information related to the selected configuration for the planner, \( r^\text{end}_e \) and \( H^\text{end} \). 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 \( \pi \) 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.

\begin{algorithm}
\caption{MSP\( (D, P, R_e, R_i, \mathcal{H}, T) \)}
\begin{algorithmic}[1]
\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
\State \( (D^\text{end}, P^\text{end}, r^\text{end}_e, H^\text{end}, \pi) \leftarrow \) \Meta-search\( (D, P, R_e, R_i, \mathcal{H}, T/2, T/2) \)
\If {\( \pi = \emptyset \)}
\State \( T \leftarrow T - \text{consumedTime} \)
\State \( \text{plan} \leftarrow \text{PLANNER}(D^\text{end}, P^\text{end}, r^\text{end}_e, H^\text{end}, T) \)
\Else
\State \( \text{plan} \leftarrow \pi \)
\State \text{return} plan
\EndIf
\end{algorithmic}
\end{algorithm}

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 \( \sigma(r_1, r_2) \) over a pair \( (D, P) \) in the usual way:
\[
\sigma(r_1, r_2)(D, P) = (r_1 \circ r_2)(D, P) = r_2^e(r_1^e(D, P)).
\]

For the internal representation, we are restricted to transformations into SAS+. If \( S \) is the space of all SAS+ descriptions, \( \forall r_i \in R_i, r_i : S \rightarrow S \). 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 pre-processing 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:

\textbf{Definition 1 (meta-search state)} A meta-search state \( s \) is a tuple \( s = \langle (r^1_e, \ldots, r^n_e), D_n, P_n, r_i, H \rangle \), where \( \langle r^1_e, \ldots, r^n_e \rangle, r^1_e \in R_e \) 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, \( (D_n, P_n) = \sigma(r^1_e, \ldots, r^n_e)(D, P) \); \( r_i \in R_i \) is an internal representation change; and \( H \subseteq \mathcal{H} \) is a subset of heuristics. \(^1\)

\(^1\)\( n \) 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 \( H \subseteq H \) 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 \( H \) 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:

\[
\sum_{k=0}^{\vert R_e \vert} \left( \frac{\vert R_e \vert}{k} \right) \times \vert R_i \vert \times 2^\vert H \vert
\]

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 \((\emptyset, D, P, r_i, H)\). 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 \( s = (r^1, \ldots, r^n, D_n, P_n, r_i, H) \) 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 \( mo = (mo_e, mo_i, mo_H) \), 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 \( H \subseteq H \).

The operator \( mo = (mo_e, r_i, mo_H = H') \) applied to the state \( s \) generates a new state \( s' = (r^1, \ldots, r^n, D_{n+1}, P_{n+1}, r'_i, H') \), where \( D_{n+1} \) and \( P_{n+1} \) are the domain and problem generated by the new sequence: \( (D_{n+1}, P_{n+1}) = \sigma(r^1, \ldots, r^n, r_i)(D, P) \). The internal (SAS\(^+\)) representation to be used by the planner will be the result of: \( r'_i(\sigma(r^1, \ldots, r^n, r_i)(D, P)) \). 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 RIDA\(^+\) call 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: \( mo = (mo_e, mo_i, H_{RIDA^+}) \).

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. *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-

---

2RIDA\(^+\) was denoted as RA\(^+\) in its original paper (Barley, Franco, and Riddle 2014).

3We 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 \( m_{or} = \text{baggy-all} \) and \( m_{or} = (t = \text{FD}^+, p = \text{FD}) \), which applies Buggy 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 \( R_c \), the internal representation changes \( R_i \), the set of heuristics \( H \), 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 \((D^{end}, P^{end}, r^{end}_i, r^{end}_e, H^{end}, \pi)\). META-SEARCH

```plaintext
noend2 META-SEARCH(D, P, R_c, R_i, H, T_m, T_P)

Require: domain D, problem P, PDDL rep. changes R_c, internal
rep. changes R_i, set of heuristics H, meta-search time bound
T_m, planning time bound T_P

Ensure: configuration, \((D^{end}, P^{end}, r^{end}_i, r^{end}_e, H^{end}, \pi)\)

1: \( MO \leftarrow \text{GENERATE-OPERATORS}(R_c, R_i) \)
2: \( \text{init} \leftarrow (\emptyset, D, P, (t = \text{FD}, p = \text{FD}), H) \)
3: \( T_E \leftarrow T_m/3 \)
4: \( H, f, \pi \leftarrow \text{EVALUATE}(\text{init}, T_P, T_E, H) \)
5: \( \text{best} \leftarrow (\emptyset, D, P, (t = \text{FD}, p = \text{FD}), H) \)
6: \( \text{best}_f \leftarrow f \)
7: \( \text{succ} \leftarrow \text{SUCCESSORS}(\text{init}, MO) \)
8: \( \text{while } T_m \text{ not reached and } \pi = \emptyset \text{ and } \text{succ} \neq \emptyset \) do
9: \( s = ([r^{1}_1, \ldots, r^{m}_1], D_n, P_n, r_i, 0) \leftarrow \text{POP(succ)} \)
10: \( H, f, \pi \leftarrow \text{EVALUATE}(s, T_P, T_E, H) \)
11: \( s' \leftarrow ([r^{1}_1, \ldots, r^{m}_1], D_n, P_n, \pi, H) \) sets the value of H
12: \( \text{if } f > \text{best}_f \text{ then} \)
13: \( \text{best} \leftarrow s \)
14: \( \text{best}_f \leftarrow f \)
15: \( \text{succ} \leftarrow \text{SUCCESSORS}(s, MO) \)
16: \( \text{return } (D^{end}, P^{end}, r^{end}_i, r^{end}_e, H^{end}, \pi) \leftarrow \text{CONFIG(best, } \pi) \)
```

first generates the operators, \( MO \), given the possible representation changes \( R_c \) 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 \((\emptyset, D, P, (t = \text{FD}, p = \text{FD}), H)\). After that, the initial state is evaluated. The EVALUATE function returns a set of heuristics, an evaluation of the state and a plan \( \pi \) (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. 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(\(s, T_P, T_E, H\)). Algorithm 3 shows its pseudo-code. \( s \) is the state being evaluated, \( T_P \) is the estimated planner’s time limit, \( T_E \) is the evaluation time limit, and \( H \) is the collection of candidate heuristics. Given \( s \) and \( T_E \), we would like \( \text{EVALUATE} \) 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.

```plaintext
noend3 EVALUATE(s, T_P, T_E, H)

Require: meta-search state s, planning time bound T_P, evaluation
time bound T_E, set of heuristics H

Ensure: selected heuristics H, evaluation f, plan \( \pi \)

1: \( s = ([r^{1}_1, \ldots, r^{n}_1], D_n, P_n, r_i, H) \leftarrow s \)
2: \( S \leftarrow r_i(D_n, P_n) \)
3: \( H, b, T_n, \pi \leftarrow \text{RIDA}^*(S, T_E, H) \)
4: \( f \leftarrow \log_b T_P^* \)
5: \( \text{return } H, f, \pi \)
```

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\textsuperscript{*} 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 \( \pi \), the plan that RIDA\textsuperscript{*} may have found while evaluating this configuration or \( \emptyset \) otherwise. Using the formula found on line 4, EVALUATE uses the information returned by RIDA\textsuperscript{*} 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\textsuperscript{*} 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\textsuperscript{*}, which is built upon FD, as the planner to be used to perform optimal planning using A\textsuperscript{*} 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\textsuperscript{*} 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\textsuperscript{*} already finds the best subset of heuristics for a given representation. Therefore, we used the heuristics in our version of RIDA\textsuperscript{*}: 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\textsuperscript{*} planner: (1) OFD: original representation and FD pre-processor; (2) Oh\textsuperscript{2}: original representation and \( h^2 \) pre-processor; and (3) Bh\textsuperscript{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\textsuperscript{*} 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\textsuperscript{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 (\( \sigma = 3.8 \)). In MSP almost all the time is spent on the evaluation of meta-search states by RIDA\textsuperscript{*}. The average number of generated meta-search states is 11.2 (\( \sigma = 10.26 \)). The average time spent on meta-search is 642s (\( \sigma = 382.5 \)). 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\textsuperscript{2}) and SYMBA\textsuperscript{*} (Sy), the winner of IPC 2014 optimal track. Given that SYMBA\textsuperscript{*} was the winner of IPC’14, we could have used SYMBA\textsuperscript{*} within MSP. However, in its current form SYMBA\textsuperscript{*} can only use one form of heuristic based on abstractions. Also, we have not determined how to use SYMBA\textsuperscript{*} to compute estimations when using different representation changes. If we use RIDA\textsuperscript{*} to make the estimations and SYMBA\textsuperscript{*} as planner, MSP might frequently make poor estimations since the estimates are tightly linked to the planner. Thus, the use of SYMBA\textsuperscript{*} 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, Childsnacll, Hiking, and Visi-tall. Barman and Childsnacll are solved because MSP selected Baggy which makes a big difference in these domains. Besides, MSP solves some Hiking and Visi-tall problems in the original representation that LMCut cannot. So RIDA\textsuperscript{*}’s broader choice of heuristics is also a fac-

\textsuperscript{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, Lm$h^2$, and SYMBmA*. 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).

<table>
<thead>
<tr>
<th>Domain</th>
<th>MSP</th>
<th>OfD</th>
<th>Oh^2</th>
<th>Bh^2</th>
<th>VBS</th>
<th>Lm$h^2$</th>
<th>Sy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barman11</td>
<td>7</td>
<td>4</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>1/12</td>
</tr>
<tr>
<td>Elevators11</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>4/4</td>
</tr>
<tr>
<td>Floor tile11</td>
<td>14</td>
<td>5</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>10/16</td>
</tr>
<tr>
<td>No Mystery11</td>
<td>17</td>
<td>20</td>
<td>17</td>
<td>20</td>
<td>14</td>
<td>16</td>
<td>0/0</td>
</tr>
<tr>
<td>Openstacks11</td>
<td>15</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>20</td>
<td>11/16</td>
</tr>
<tr>
<td>Parking11</td>
<td>17</td>
<td>13</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>0/3</td>
</tr>
<tr>
<td>Parking11</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>2/1</td>
<td>0/1</td>
</tr>
<tr>
<td>Pegsol11</td>
<td>17</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>20</td>
<td>0/3</td>
<td>14/14</td>
</tr>
<tr>
<td>Scanalyzer11</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>12</td>
<td>9</td>
<td>1</td>
<td>1/3</td>
</tr>
<tr>
<td>Sohank11</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>0/0</td>
</tr>
<tr>
<td>Tidybot11</td>
<td>16</td>
<td>14</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>7/11</td>
<td>5/5</td>
</tr>
<tr>
<td>Transport11</td>
<td>10</td>
<td>10</td>
<td>6</td>
<td>10</td>
<td>6</td>
<td>11</td>
<td>0/4</td>
</tr>
<tr>
<td>Workwood11</td>
<td>14</td>
<td>11</td>
<td>15</td>
<td>16</td>
<td>20</td>
<td>27/0</td>
<td>0/35</td>
</tr>
<tr>
<td>Visitall11</td>
<td>18</td>
<td>18</td>
<td>16</td>
<td>18</td>
<td>10</td>
<td>12</td>
<td>0/0</td>
</tr>
<tr>
<td>Barman14</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>6/0</td>
<td>6/14</td>
</tr>
<tr>
<td>Childmack14</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>4</td>
<td>0/0</td>
<td>8/12</td>
</tr>
<tr>
<td>Floor tile14</td>
<td>18</td>
<td>4</td>
<td>17</td>
<td>20</td>
<td>17</td>
<td>20</td>
<td>16/18</td>
</tr>
<tr>
<td>GED14</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>15</td>
<td>20</td>
<td>0/0</td>
</tr>
<tr>
<td>Hiking14</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>17</td>
<td>19</td>
<td>9/0</td>
<td>20/11</td>
</tr>
<tr>
<td>Openstacks14</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>20</td>
<td>3/16</td>
<td>0/0</td>
</tr>
<tr>
<td>Parking14</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>0/1</td>
<td>6/19</td>
</tr>
<tr>
<td>Tetris14</td>
<td>10</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>9</td>
<td>12/0</td>
<td>0/1</td>
</tr>
<tr>
<td>Tidybot14</td>
<td>11</td>
<td>9</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>10</td>
<td>9/19</td>
</tr>
<tr>
<td>Transport14</td>
<td>7</td>
<td>9</td>
<td>5</td>
<td>9</td>
<td>6</td>
<td>1/3</td>
<td>0/0</td>
</tr>
<tr>
<td>Visitall14</td>
<td>15</td>
<td>14</td>
<td>15</td>
<td>15</td>
<td>5</td>
<td>8/0</td>
<td>0/5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Domain</th>
<th>Translator</th>
<th>Preprocessor</th>
<th>$R_a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barman11</td>
<td>7/7</td>
<td>3/1</td>
<td>1/11</td>
</tr>
<tr>
<td>Elevators11</td>
<td>7/7</td>
<td>3/1</td>
<td>1/11</td>
</tr>
<tr>
<td>Floor tile11</td>
<td>7/7</td>
<td>3/1</td>
<td>1/11</td>
</tr>
<tr>
<td>No Mystery11</td>
<td>7/7</td>
<td>3/1</td>
<td>1/11</td>
</tr>
<tr>
<td>Openstacks11</td>
<td>7/7</td>
<td>3/1</td>
<td>1/11</td>
</tr>
<tr>
<td>Parking11</td>
<td>7/7</td>
<td>3/1</td>
<td>1/11</td>
</tr>
<tr>
<td>Parking11</td>
<td>7/7</td>
<td>3/1</td>
<td>1/11</td>
</tr>
<tr>
<td>Pegsol11</td>
<td>7/7</td>
<td>3/1</td>
<td>1/11</td>
</tr>
<tr>
<td>Scanalyzer11</td>
<td>7/7</td>
<td>3/1</td>
<td>1/11</td>
</tr>
<tr>
<td>Sohank11</td>
<td>7/7</td>
<td>3/1</td>
<td>1/11</td>
</tr>
<tr>
<td>Tidybot11</td>
<td>7/7</td>
<td>3/1</td>
<td>1/11</td>
</tr>
<tr>
<td>Transport11</td>
<td>7/7</td>
<td>3/1</td>
<td>1/11</td>
</tr>
<tr>
<td>Workwood11</td>
<td>7/7</td>
<td>3/1</td>
<td>1/11</td>
</tr>
<tr>
<td>Visitall11</td>
<td>7/7</td>
<td>3/1</td>
<td>1/11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Domain</th>
<th>Ratio solved when selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barman11</td>
<td>0.56</td>
</tr>
<tr>
<td>Elevators11</td>
<td>0.7</td>
</tr>
<tr>
<td>Floor tile11</td>
<td>0.66</td>
</tr>
<tr>
<td>No Mystery11</td>
<td>0.65</td>
</tr>
<tr>
<td>Openstacks11</td>
<td>0.68</td>
</tr>
<tr>
<td>Parking11</td>
<td>0.64</td>
</tr>
<tr>
<td>Parking11</td>
<td>0.7</td>
</tr>
<tr>
<td>Pegsol11</td>
<td>0.66</td>
</tr>
<tr>
<td>Scanalyzer11</td>
<td>0.65</td>
</tr>
<tr>
<td>Sohank11</td>
<td>0.68</td>
</tr>
<tr>
<td>Tidybot11</td>
<td>0.64</td>
</tr>
<tr>
<td>Transport11</td>
<td>0.64</td>
</tr>
<tr>
<td>Workwood11</td>
<td>0.7</td>
</tr>
<tr>
<td>Visitall11</td>
<td>0.66</td>
</tr>
</tbody>
</table>

Note that two or more heuristics and/or representation changes can be selected for a single problem.
not always work, especially when the problems’ distribution at the domain level, looking for the best representa-
et al. 2015). Most work on reformulation in planning has
lar objects (Riddle et al. 2016; Fuentetaja and de la Rosa
operators (Fikes, Hart, and Nilsson 1972), bagging simi-
niques at the PDDL level are the generation of macro-
tions of solvers/heuristics. Examples of reformulation tech-
ponents or reformulation; and generating good combina-
atics. It represents a new way to solve problem solving tasks
aries on two or more dimensions (Riddle, Holte, and Bar-
ley 2011). Also, there has been little work on changing rep-
resentation 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 ele-
ments. 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 port-
folios 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). PnP (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 repre-
sentation and heuristics. MSP avoids the expensive learning processes used by other approaches, such as portfolios. Re-

Finally, in order to evaluate the impact of performing 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 h² is preferred over FD for the pre-processor. However, we ran an additional experiment using random operator orderings resulting in very similar num-
bers (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 oth-
ers in at least two different aspects: change of repre-
separation or reformulation; and generating good combina-
tiques at the PDDL level are the generation of macro-
operators (Fikes, Hart, and Nilsson 1972), bagging simi-
lar objects (Riddle et al. 2016; Fuentetaja and de la Rosa
2016), action splitting (Areces et al. 2014), or just chang-
ing 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 representa-
tion 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

Acknowledgements

M. Barley was supported by the Air Force Office of Sci-
entific 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


