# MUM: A Technique for Maximising the Utility of Macro-operators by Constrained Generation and Use

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

## Abstract

Research into techniques that reformulate problems to make general solvers more efficiently derive solutions has attracted much attention, in particular when the reformulation process is to some degree solver and domain independent. There are major challenges to overcome when applying such techniques to automated planning, however: reformulation methods such as adding macro-operators (macros, for short) can be detrimental because they tend to increase branching factors during solution search, while other methods such as learning entanglements can limit a planner's space of potentially solvable problems (its coverage) through over-pruning. These techniques may therefore work well with some domain-problem-planner combinations, but work poorly with others.

In this paper we introduce a new learning technique (MUM) for synthesising macros from training examples in order to improve the speed and coverage of domain independent automated planning engines. MUM embodies domain independent constraints for selecting macro candidates, for generating macros, and for limiting the size of the grounding set of learned macros, therefore maximising the utility of used macros. Our empirical results with IPC benchmark domains and a range of state of the art planners demonstrate the advance that MUM makes to the increased coverage and efficiency of the planners. Comparisons with a previous leading macro learning mechanism further demonstrate MUM's capability.

## Introduction

A fundamental problem solving technique is to reformulate a problem to make it easier to solve. In automated planning, where solution generation is known to be hard in general, techniques that reformulate planning domains have the potential to increase the speed of solution plan generation, and increase coverage, that is the number of planning problems that can be solved within some resource constraint. Where the reformulation involves encoding knowledge directly into the same language in which the problem definition is encoded, then planning engines do not need to be modified in order to exploit them.



Macro-operator generation is a well known technique for encapsulating sequences of original operators, so that they can be stored and used in future planning problems. Macro-operators (macros, for short) can be encoded in the same format as original operators and therefore can be used to reformulate planning problem definitions. The idea of using macros in planning dates back to 1970s where, for example, it was applied in STRIPS (Fikes and Nilsson 1971) and REFLECT (Dawson and Siklóssy 1977). More recently, systems such as MacroFF CA-ED version (Botea et al. 2005) or WIZARD (Newton et al. 2007) are able to extract macros and reformulate the original domain model, such that standard planning engines can exploit them.

On the other hand, macros can also be exploited by specifically enhanced algorithms. This is the case for MacroFF SOL-EP version (Botea et al. 2005) which is able to exploit offline extracted and ranked macros, and Marvin (Coles, Fox, and Smith 2007) that generates macros online by combining sequences of actions previously used for escaping plateaus. Such systems can efficiently deal with drawbacks of specific planning engines, in this case the FF planner (Hoffmann and Nebel 2001), however, their adaptability for different planning engines might be low.

Another type of additional knowledge that can be encoded into domain/problem definitions and has been exploited in classical planning are entanglements, relations between planning operators and predicates (Chrpa and McCluskey 2012). Outer entanglements (Chrpa and Barták 2009), one of the types of entanglements, capture causal relations between planning operators and initial or goal predicates which are used to prune some unpromising instances of planning operators. Deciding outer entanglements is, however, PSPACE-complete in general (Chrpa, McCluskey, and Osborne 2012), therefore they are extracted by an approximation algorithm which generally does not preserve completeness (solvability might be lost if incorrect entanglements are applied).

In this paper we introduce MUM, a new learning technique for synthesising macros from training examples in order to improve the speed and coverage of domain independent planning engines which input encodings in classical PDDL. MUM utilises constraints which are created through the generation of outer entanglements, then uses these entanglements for selecting macro candidates, for generating macros, and for limiting the size of the ground-

ing set of learned macros. There have been approaches which utilise macros and outer entanglements independently (Chrpa 2010a), however, in contrast to this, MUM is designed to directly exploit knowledge related to outer entanglements throughout the process of macro learning and use, thus maximising their utility. Also, MUM preserves completeness since outer entanglements are applied only on generated macros, so the original operators remain intact. We present an empirical evaluation on IPC benchmarks (from the IPC-7 learning track) using a range of 6 planners and 9 domains. Our empirical results demonstrate the advance that MUM makes to the increased coverage and efficiency of the planners, and this improvement is apparent accross planners and domains. Comparisons with a previous leading macro learning mechanism called WIZARD (Newton et al. 2007), further demonstrate MUMs capability.

## Background and Related Work

Classical planning (in state space) deals with finding a sequence of deterministic actions to transform the fully observable environment from some initial state to a desired goal state (Ghallab, Nau, and Traverso 2004).

In the set-theoretic representation *atoms*, which describe the environment, are propositions. *States* are defined as sets of propositions. *Actions* are specified via sets of atoms defining their preconditions, negative and positive effects (i.e., $a = (pre(a), eff^-(a), eff^+(a))$). An action $a$ is *applicable* in a state $s$ if and only if $pre(a) \subseteq s$. Application of $a$ in $s$ (if possible) results in a state $(s \setminus eff^-(a)) \cup eff^+(a)$.

In the classical representation atoms are predicates. A *planning operator* $o = (name(o), pre(o), eff^-(o), eff^+(o))$ is a generalised action (i.e. an action is a grounded instance of the operator), where $name(o) = op\_name(x_1, \ldots, x_k)$ ($op\_name$ is an unique operator name and $x_1, \ldots x_k$ are variable symbols (arguments) appearing in the operator) and $pre(o), eff^-(o)$ and $eff^+(o)$ are sets of (ungrounded) predicates with variables taken only from $x_1, \ldots x_k$.

A *planning domain model* is specified by a set of predicates and a set of planning operators. A *planning problem definition* is specified via a domain model, initial state and set of goal atoms. A *plan* is a sequence of actions. A plan is a *solution* of a planning problem if and only if a consecutive application of the actions in the plan (starting in the initial state) results in a state, where all the goal atoms are satisfied. An important class of predicates in this paper are static predicates. For a given domain model, a predicate is called *static* if it is not present in the effects of any operator.

### Macro-operators

Macro learning and use has been studied for several decades (Dawson and Siklóssy 1977; Korf 1985; Botea et al. 2005; Fikes and Nilsson 1971). Here we are concerned with macros that are encoded in the same way as ordinary planning operators but encapsulate sequences of planning operators. This gives the technique the potential of being *planner independent* as well as being domain independent. In the well known BlocksWorld domain (Slaney and Thiébaux 2001), we can observe, for instance, that the operator unstack(?x,?y) is often followed by the operator putdown(?x). Hence, it might be reasonable to create a macro unstack-putdown(?x,?y) which moves a block ?x from top of a block ?y directly to the table, bypassing the situation where the block ?x is held by the robotic hand. Formally, a macro $o_{i,j}$ is constructed by assembling planning operators $o_i$ and $o_j$ (in that order) in the following way:

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

For a macro to be sound, assuming we are working in domain models where positive and negative effects of all operators are disjoint, $o_i$ must not delete any predicate required by $o_j$. If soundness is violated then corresponding instances of $o_i$ and $o_j$ cannot be applied consecutively.

Longer macros, i.e., those encapsulating longer sequences of original planning operators can be constructed by this approach iteratively.

Macros can be understood as 'shortcuts' in the state space. This property can be useful since by exploiting them it is possible to reach the goals in less steps. However, the number of instances of macros is often higher than the number of instances of the original operators, because they usually have a large set of parameters, that derives from the parameters of the operators that are encapsulated. This increases the branching factor in the search space, which can slow down the planning process and, moreover, increase the memory consumption. Therefore, it is important that benefits of macros overweight their drawbacks. This problem is known as the *utility problem* (Minton 1988).

### Learning Macros

Macro learning techniques often follow the following rules, when a macro $o_{i,j}$ is being created from operator $o_i$ and $o_j$ (in that order):

1) $o_i$ adds a predicate needed in the preconditions of $o_j$
2) $o_{i,j}$ is not complex
3) the number of learned macros is small

Rule 1) refers to a sort of coherency between $o_i$ and $o_j$. Theoretically, it is possible to generate a macro from independent operators, however, in practice it is not a very useful approach because then the number of possible instances of the macro can be very high and, moreover, there may not be a clear motivation for executing such independent operators in a specific sequence. Rules 2) and 3) ameliorate the utility problem: complex macros can have many groundings which are likely to introduce overheads which outweigh the benefits of macro use, and similarly, generation of many macros may bring the same shortcomings. Recent related work confirms that the better option is to generate a few and shorter macros rather than many or longer macros. For macros which are generated to have the same format as original operators, and thus are potentially planner independent, relevant state-of-the-art systems include MacroFF (Botea et al. 2005), Wizard (Newton et al. 2007) and the system developed by Chrpa (2010b). The CA-ED version of MacroFF

uses a component abstraction technique for learning macros. In brief, abstract components are sets of static predicates referring to 'localities'. Static predicates provide a kind of matching between objects of different or the same types. An abstract component consists of such static predicates that, informally, can group objects into a single component. For instance, in the Depot's domain, assume that a hoist is placed at some location. Then at(?hoist, ?place) can form an abstract component since a hoist can be only at one place. On the other hand, supports(?camera, ?mode) cannot form an abstract component since a camera might support different modes as well as a mode can be supported by more different cameras. Abstract components are used to check the *locality rule* of a generated macro, that is, whether static predicates in a macro's precondition belong to the same abstract component. By pruning macros containing cycles and limiting numbers of arguments or predicates in macro preconditions, MacroFF is able to eliminate complex macros. Limiting the number of learned macros is done by selecting the $n$ most used macros in training plans (which are solutions of simple problems).

In Chrpa's approach, macros are learned from training plans by considering both adjacent actions, and non-adjacent actions which can be made adjacent by permutating the training plans (clearly the permutations considered must preserve the soundness of the plan). Macros are generated according to several criteria such as whether instances of one operator frequently follows (or precedes) instances of the other operator, and whether the number of macro's arguments is small. No limit on how many macros can be generated is given a priori, but the priority is given to macros that could replace some original operators. Removing original operators is, however, incomplete in general, although it has been empirically shown on IPC benchmarks that solvability is lost very rarely (Chrpa 2010b). On the other hand, whereas IPC benchmarks differ by number of objects, initial and goal situations often are found to be similar in structure.

Wizard is based on an evolutionary method carried out in a training phase, which computes macros by combining operators using a genetic algorithm. This approach allows Wizard to generate macros that refer to sequences of actions that do not appear in the considered training plans. Generated macros are then cross-validated on training problems and the fitness of the macros, i.e. their usefulness, is determined according to the performance of planners. Although macros generated by Wizard have shown to have good quality, learning time is often very high (typically tens of hours).

## Outer Entanglements

*Outer Entanglements* are relations between planning operators and initial or goal predicates, and have been introduced as a tool for eliminating potentially unnecessary instances of these operators (Chrpa and McCluskey 2012). In the BlocksWorld (Slaney and Thiébaux 2001) we may observe, for example, that *unstacking* blocks only occurs from their initial positions. In this case an *entanglement by init* will capture that if a predicate onblock(a,b) is to be achieved for a corresponding instance of operator unstack(?x,?y) (unstack(a,b)), then the predicate is present in the initial

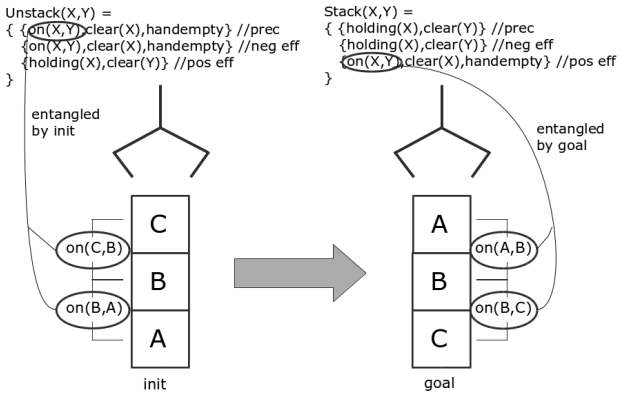


Figure 1: An illustrative example of outer entanglements.

state. Similarly, it may be observed that *stacking* blocks only occurs to their goal position. Then, an *entanglement by goal* will capture that a predicate onblock(b,a) achieved by a corresponding instance of operator stack(?x,?y) (stack(b,a)) is the goal one. Such an observation is depicted in Figure 1. Outer entanglements are defined as follows.

**Definition 1.** *Let $P$ be a planning problem, where $I$ is the initial situation and $G$ is the goal situation. Let $o$ be a planning operator and $p$ be a predicate ($o$ and $p$ are defined in the domain model of $P$). We say that operator $o$ is **entangled by init (resp. goal)** with predicate $p$ in $P$ if and only if $p \in pre(o)$ (resp. $p \in eff^+(o)$) and there exists a plan $\pi$ that is a solution of $P$ and for every action $a \in \pi$ which is an instance of $o$ and for every grounded instance $p_{gnd}$ of the predicate $p$ it holds: $p_{gnd} \in pre(a) \Rightarrow p_{gnd} \in I$ (resp. $p_{gnd} \in eff^+(a) \Rightarrow p_{gnd} \in G$).*
*Henceforth, entanglements by init and goal are denoted as **outer entanglements**.* ■

Outer entanglements can be used to prune potentially unnecessary instances of planning operators. Given the example of the BlocksWorld domain (see Figure 1), we can see that since the unstack operator is entangled by init with the on predicate only the instances unstack(b,a) and unstack(c,b) follow the entanglement conditions and then we can prune the rest of unstack's instances because they are not necessary to find a solution plan. Similarly, we can see that since the stack operator is entangled by goal with the on predicate only the instances stack(a,b) and unstack(b,c) follow the entanglement conditions and then we can prune the rest of stack's instances. Usefulness of such pruning can be demonstrated in the following way. Given $n$ blocks we can have at most $n \cdot (n-1)$ instances of stack or unstack (we do not consider instances when the block is unstacked from or stacked on itself – e.g stack(a,a)). Considering outer entanglements we can have at most $n-1$ instances of stack or unstack (we consider situations where at most one block can be stacked on the top of another block and no block can be stacked on more than one block). In summary, while in the original setting the number of instances grows quadratically with the number of blocks, considering outer entanglements reduces the instances growth to linear.

## Reformulating Planning Problems by Outer Entanglements

Outer entanglements are directly encoded into a problem definition, so, similarly to the kind of macros we consider, are used as a problem reformulation technique. The way outer entanglements are encoded is inspired by one of their properties: given a static predicate $p$, an operator $o$ is entangled by init with $p$ if and only if $p \in pre(o)$ (Chrpa and Barták 2009). Introducing supplementary static predicates and putting them into preconditions of operators in the outer entanglement relation (both init and goal) will *filter* instances of these operators which do not follow the entanglement conditions. Formally, let $P$ be a planning problem, $I$ be its initial state and $G$ its goal situation. Let an operator $o$ be entangled by init (resp. goal) with a predicate $p$. Then the problem $P$ is reformulated as follows (Chrpa and McCluskey 2012):

1. Create a static predicate $p'$ (not already defined in the domain model of $P$) having the same arguments as $p$ and add $p'$ to the domain model of $P$.
2. Modify the operator $o$ by adding $p'$ into its precondition. $p'$ has the same arguments as $p$ which is in precondition (resp. positive effects) of $o$.
3. Add instances of $p'$ which correspond to instances of $p$ in $I$ (resp. in $G$) into $I$.

## Detecting Outer Entanglements

Deciding outer entanglements is PSPACE-complete in general, i.e., as hard as planning itself (Chrpa, McCluskey, and Osborne 2012). Detecting outer entanglements is thus done by using an approximation method which finds the entanglements in several training plans, solution of simpler planning problem, and assumes these entanglements hold for the whole class of planning problems defined in the same domain (e.g. IPC benchmarks) (Chrpa and McCluskey 2012). Although this approximate method may result in an incorrect assumption, it has been shown empirically using IPC benchmarks that it occurs very rarely, though whether the technique would show the same success rate in 'real-world' problems is an open question.

The method proceeds by iterating through the training plans counting how many times for a (operator, predicate) pair the outer entanglement conditions are violated. Because training plans might consist of 'flaws' (e.g. redundant actions) a *flaw ratio* $\eta$ is used to allow some percentage of errors (i.e. when the entanglement conditions are violated). An entanglement between an operator and a predicate is considered true if the number of operator's instances is non-zero and the ratio between errors and the number of operator's instances is smaller or equal to the flaw ratio. Of course, having the flaw ratio greater than zero might result in detecting incorrect entanglements even for training problems. Hence, these entanglements must be validated on the training problems and if some of the problems become unsolvable then the flaw ratio is decreased and the method is run again.

# The MUM Technique: Combining Outer Entanglements and Macros

The general idea of MUM is to utilise outer entanglements both in the macro generation and the macro use phase, in order to constrain which macros are generated, and the number of instances of macros in their use. Adding macros into a domain model does not affect completeness since macros can be understood as 'shortcuts' in the state space. Because deciding outer entanglements is intractable in general, an approximation method is used to extract them. However, there is no guarantee that all the extracted entanglements are valid. In other words, applying incorrect entanglements leads to losing solvability of some problems. If some instances of a macro are removed due to 'incorrect entanglements', however, completeness is not affected since the corresponding sequence of instances of original operators can be used instead. Hence, applying entanglements only on macros (and not on the original operator set) ensures completeness is preserved even in the case where some of the entanglements are incorrect.

It is of course possible to learn macros and outer entanglements separately, and use them both to reformulate planning problems as distinct techniques. While using such an approach can bring promising results (Chrpa 2010a), the problems of completeness not being preserved, or macros having too many instances, remain. In contrary to this, MUM exploits outer entanglements directly during the macro learning process, so we believe that it will lead to generating better quality macros. Following these insights, a high level design of MUM is as follows:

1) Learn outer entanglements
2) Macro Generation: learn macros by exploiting the knowledge of outer entanglements
3) Macro Use: reformulate a problem definition with learned macros and their supporting outer entanglements

One useful result we can use in step 2), is that macros can inherit outer entanglements from original operators as follows (Chrpa 2010a):

- $o_{i,j}$ is entangled by init with $p$ iff $p \in pre(o_{i,j})$ and $o_i$ or $o_j$ is entangled by init with $p$.
- $o_{i,j}$ is entangled by goal with $p$ iff $p \in eff^{+}(o_{i,j})$ and $o_i$ or $o_j$ is entangled by goal with $p$.

## Estimating the potential number of instances of an operator

Inspired by abstract components used by MacroFF (Botea et al. 2005) to determine 'locality' of objects of different types, we propose an idea of operator *argument matching* which can be used to estimate the number of operator instances. Static predicates which have at least two arguments denote relations between objects. For example, in the well known Depots domain, the static predicate at(?hoist,?place) provides a relation between hoists and places. In particular, a hoist can be exactly at one place. This information can be useful for estimating how many reachable instances a planning operator (or macro) can have. For ex-

ample, Lift(?hoist,?crate,?surface,?place) can hypothetically have $\#hoists \cdot \#crates \cdot \#surfaces \cdot \#places$ instances. Knowing that the static predicate at(?hoist,?place) is in the precondition of Lift we can deduce that the number of Lift's instances is bounded by $\#hoists \cdot \#crates \cdot \#surfaces$ because the number of at's instances in the initial state is bounded by $\#hoists$.

Recall how outer entanglements are encoded, that is by introducing supplementary static predicates. If these introduced static predicates have at least two arguments, then they can be exploited in the same way as other static predicates in order to estimate operator instances. It holds that if $p \in pre(o)$ is static, then $o$ is entangled by init with $p$ (Chrpa and Barták 2009). In other words, static predicates are special cases of entanglement by init relations. If Lift(?hoist,?crate,?surface,?place) is entangled by init with a predicate on(?crate,?surface), then a static predicate on'(?crate,?surface) is created and put into Lift's precondition. For each problem, its initial state $I$ is modified to $I'$ by adding instances of on' such that $\forall x, y : on'(x, y) \in I' \Leftrightarrow on(x, y) \in I$. A similar modification takes place in the case where Lift(?hoist,?crate,?surface,?place) is entangled by init with a predicate at(?crate,?place). We can observe that the number of instances of on'(?crate,?surface) as well as the number of instances of at'(?crate,?place) (a supplementary static predicate derived from at(?crate,?place)) is bounded by the number of crates ($\#crates$). Straightforwardly, at most one crate can be stacked on a given surface and a crate can be in at most one place. With the knowledge of these entanglement relations involving the Lift operator, the estimation of the number of Lift's instances can be modified to $\mathcal{O}(\#crates)$.

Static predicates and outer entanglements provide matching between arguments of planning operators. For every operator we can construct an *argument matching graph* (or a *simple argument matching graph* if only static predicates are involved) as in the following formal definition:

**Definition 2.** *Let $o(x_1, \ldots, x_n)$ be a planning operator where $x_1, \ldots, x_n$ are its arguments.*
*Let $G = (N, E)$ be an undirected graph such that $N = \{x_1, \ldots, x_n\}$ and $(x_i, x_j) \in E$ if and only if $x_i \neq x_j$ and there is a static predicate $p$ such that $p \in pre(o)$ and $x_i, x_j \in args(p)$. Then, we denote $G$ as the **simple argument matching graph** of $o$.*
*Let $G = (N, E)$ be an undirected graph such that $N = \{x_1, \ldots, x_n\}$ and $(x_i, x_j) \in E$ if and only if $x_i \neq x_j$ and there is a predicate $p$ such that $x_i, x_j \in args(p)$ and $o$ is entangled by init or goal with $p$. Then, we denote $G$ as the **argument matching graph** of $o$.*

Henceforth, we will assume that the number of instances of predicates having at least one argument in the initial/goal state of a typical planning problem is bounded by $\mathcal{O}(n)$ ($n$ stands for the number of objects). This assumption is made according to the observation of standard planning benchmarks and will be used as a heuristic estimation of numbers of operator or predicate instances. Of course, this assumption might not be always true, hence a predicate which does not follow the assumption does not have to be considered when constructing the (simple) argument matching graph.

Using the previous assumption, the (simple) argument matching graph of an operator $o$ can be therefore used to estimate the number of $o$'s instances. Let $n$ be the number of objects defined in some planning problem, $o$ be a planning operator and $c$ be the number of components of the (simple) argument matching graph of $o$. Then, the number of $o$'s instances is $\mathcal{O}(n^c)$.

**Algorithm 1** Our method for generating macros

```
1:  macros := {}
2:  repeat
3:     candidates := ComputeMacroCandidates()
4:     candidates := SortMacroCandidates(candidates)
5:     candSelected := false
6:     while not candSelected and not Empty(candidates)
       do
7:        cand := PopFirst(candidates)
8:        mcr := GenerateMacro(cand)
9:        if not Uninformative(mcr) and not Repetitive-
          ness(mcr) and AMGComponentCheck(mcr) then
10:          candSelected := true
11:       end if
12:    end while
13:    if candSelected then
14:       UpdatePlans(mcr)
15:       macros := macros ∪{mcr}
16:    end if
17: until no more macros have been generated or the no. of
    generated macros has reached a prescribed limit
18: FilterGeneratedMacros(macros)
```

## Generating Macros

Algorithm 1 describes how MUM generates macros (it is the detail of step 2 in the high level design of MUM provided earlier in the text). Computing macro candidates (Line 3) is done according to Chrpa's approach (2010b). This approach considers two actions adjacent not only in the original plans but also their potential permutations. These actions must be related, i.e., one achieves a predicate (or predicates) required by the other's precondition. Plan permutations are computed according to the property of 'action independence' which allows the swapping of adjacent actions in plans following this property. We shall see later that the outer loop (Line 2) enables the possibility that generated macros are not of restricted length (that is, they can encapsulate more than two original operators).

Considering the sorting procedure in Line 4, let $o_i$ and $o_j$ be two operators making up a macro candidate, and $o_{i,j}$ be the macro generated by assembling $o_i$ and $o_j$ (in this order). We say that an operator $o$ has a *relational entanglement* by init (goal) if $o$ is entangled by init (goal) with a non-static predicate $p$ which has at least two arguments. The macro candidates are then sorted (Line 4) as follows. The macro candidates are ranked according to the following conditions.

(1) $o_i$ has a relational entanglement by init

(2) $o_j$ has a relational entanglement by goal

If both (1) and (2) are satisfied then the macro candidate is put to the top rank, if either (1) or (2) is satisfied then the candidate is put into the middle rank, otherwise the candidate is put into the bottom rank. If more than one candidate lies within the same rank, then those whose instances of corresponding operators can become macros (macro-actions) in the training plans more times are preferred.

The reason or this ranking is that if both (1) and (2) are satisfied, then the macro is supposed to modify the state of an object (or objects) directly from its (their) initial to its (their) goal state. Such a macro is, from our point of view, very useful. A good example is the macro Pick-Move-Drop in the Gripper domain. Similarly, if only one of (1) and (2) is satisfied, the macro is supposed to start from the initial object state or to reach the goal object state. In other words, in the Gripper domain, we prefer a potential macro Move-Drop before a potential macro Move-Pick.

Within the inner loop, the macro candidates are checked in the given order whether they follow three tests (Line 9), i.e., whether the macro is *uninformative*, *repetitive* and its number of possible instances remains within the same order as for the original operators (or macros) the macro is assembled from (the *AMGComponentCheck*). A potential macro $o_{i,j}$ is uninformative if and only if $\mathit{eff}^+(o_{i,j}) \subseteq \mathit{pre}(o_{i,j})$. Clearly, such a macro is of no utility since its application will not create any new atom (predicate). Repetitiveness of the potential macro is determined by repeating subsequences of original operators, for example, Move-Move or Lift-Load-Lift-Load. The last check, *AMGComponentCheck*, refers to whether the potential macro will lead to a significant increase of the branching factor during search. For this, we have to construct the argument matching graphs of $o_i, o_j$ and $o_{i,j}$. For original operators which will be intact in the reformulated domain models, the simple argument matching graphs will be considered. Let $comp(o)$ be the number of components of the (simple) argument matching graph of $o$ (depends whether $o$ is an original operator or a macro). The *AMGComponentCheck* succeeds if and only if $comp(o_{i,j}) \leq comp(o_i)$ or $comp(o_{i,j}) \leq comp(o_j)$. Failure of the *AMGComponentCheck* means that the number of instances of the potential macro $o_{i,j}$ can be significantly higher than the number of instances of $o_i$ and $o_j$, which is undesirable.

If all the tests (Line 9) are successful, then the macro is considered and the training plans are updated by replacing the instances of macro candidates by the new macro (for details, see (Chrpa 2010b)). If no macro candidate has passed through all the tests, or the number of generated macros has reached the limit, then the main loop (Lines 6–17) terminates.

Finally, the generated macros are filtered (Line 18) according to following conditions. If $comp(o_{i,j}) > comp(o_i)$ or $comp(o_{i,j}) > comp(o_j)$, then $o_{i,j}$ is removed. Note that the *AMGComponentCheck* used in the generation phase is a weaker condition than this, and allows situations where $\min(comp(o_i), comp(o_j) < comp(o_{i,j}) \leq \max(comp(o_i), comp(o_j))$. The reason is that $o_{i,j}$ may be used as a 'building block' for a longer macro which can be very useful (for illustration, see the example below) . In the case where one or both of the components of a new macro $m_l$ is also a macro (call it $m_s$), it is desirable to keep at most one of these. $m_l$ does not pass the filter test of Line 18 if $comp(m_l) > comp(m_s)$. $m_l$ is also filtered if $comp(m_l) = comp(m_s)$ and the number of $m_l$'s instances is not greater than the number of $m_s$'s instances in the updated training plans. Otherwise $m_l$ is kept and $m_s$ is filtered out.

| | FF | LAMA | LPG | Mp | Probe | Tot |
|---|---|---|---|---|---|---|
| Barman | – | – | – | – | – | – |
| Blocks | – | 2 | – | – | 3 | 3 |
| Depots | 2 | – | – | 2 | 2 | 2 |
| Gripper | 1 | 1 | 1 | 1 | 1 | 1 |
| Parking | – | – | – | – | – | – |
| Rovers | 2 | 2 | 1 | 2 | 1 | 4 |
| Satellite | 1 | 2 | 1 | 1 | 2 | 2 |
| Spanner | 2 | 1 | 2 | 1 | 1 | 3 |
| TPP | 1 | 1 | 1 | 1 | 1 | 2 |

Table 1: Number of macros extracted by MUM on a given domain for a single planner and across all the planners (Tot).

As an example to demonstrate how our method works we take the Gripper domain model (the version from the IPC-7 learning track). We identified three outer entanglements: Pick is entangled by init with at(?r,?room) and free(?r,?g) and Drop is entangled by goal with at(?r,?room). The numbers of components of the simple argument matching graphs are as follows: $comp(move) = 3$ and $comp(pick) = comp(drop) = 4$. After calculating the macro candidates we selected at first a potential macro Pick-Drop since it is the highest candidate after sorting. This macro is, however, uninformative since Pick and Drop are reverting each other effects in this case. Then, a potential macro Move-Drop is selected. It passed all the checks, $comp(move - drop) = 4$ (we consider the non-simple argument matching graph), so the macro is considered and the training plans are updated by replacing corresponding instances of Move and Drop with the new macro Move-Drop. Iterating around the outer repeat loop of Algorithm 1, macro-candidates are recalculated using the re-represented training plans. After the candidates have been sorted, the potential macro Pick-Move-Drop is selected. It passed all the checks, where $comp(pick - move - drop) = 2$, hence the macro is considered and the training plans are updated. Furthermore, two more macros Move-Pick-Move-Drop and Pick-Move-Drop-Move-Pick-Move-Drop such that $comp(pick - move - drop - move - pick - move - drop) = comp(move - pick - move - drop) = 3$ are considered. Since the prescribed limit of considered macros was 4, which is reached, we proceed to the filtering stage. The macros Move-Pick-Move-Drop and Pick-Move-Drop-Move-Pick-Move-Drop are pruned since their common 'submacro' Pick-Move-Drop has less components of argument matching graph. The macro Pick-Move-Drop has less components than the original operator Pick and the macro Move-Drop. Hence, the macro Pick-Move-Drop is

| | Solved | | # Fastest | | IPC score | |
|---|---|---|---|---|---|---|
| | O | M | O | M | O | M |
| FF | 1 | 36 | 0 | 36 | 0.5 | 36.0 |
| LAMA | 39 | 116 | 5 | 113 | 30.4 | 115.7 |
| LPG | 99 | 86 | 89 | 24 | 97.9 | 67.1 |
| Mp | 8 | 41 | 2 | 41 | 6.9 | 41.0 |
| Probe | 80 | 86 | 20 | 75 | 63.4 | 84.3 |
| Total | 227 | **365** | 116 | **289** | 199.1 | **344.1** |

Table 2: Number of solved problems, number of problems solved faster and IPC score achieved by considered planners on all the benchmark problems, while exploiting the original formulation (O) or the formulation extended with MUM macros (M), whenever available.

kept while its 'submacro' Move-Drop is pruned. After the filtering stage only Pick-Move-Drop remains which is is then added (with entanglements) into the original Gripper domain model.

## Experimental Analysis

The aims of this experimental analysis were to (a) evaluate the effectiveness of the MUM system with respect to increasing the speed and coverage of plan generation over a range of domains and planning engine combinations, and (b) to compare it to a similar state-of-the-art method. For this purpose, we use the well-known benchmark instances used in the learning track of the last International Planning Competition (IPC-7) (Coles et al. 2012) that was held in 2011. Such problems are from the following domains: Barman, BlocksWorld (BW), Depots, Gripper, Parking, Rovers, Satellite, Spanner and TPP. As benchmarking planners we chose Metric-FF (Hoffmann 2003), LPG-td (Gerevini, Saetti, and Serina 2003), LAMA-11 (Richter and Westphal 2010; Richter, Westphal, and Helmert 2011), Mp (Rintanen 2012) and Probe (Lipovetzky and Geffner 2011). All the planners successfully competed in the IPCs and exploit different techniques for finding satisficing plans. A runtime cutoff of 900 CPU seconds (15 minutes, as in learning tracks of IPC) was used for both learning and testing runs. All the experiments were run on 3.0 Ghz machine CPU with 4GB of RAM. In this experimental analysis, IPC score as defined in IPC-7 are used. For a planner $\mathcal{C}$ and a problem $p$, $Score(\mathcal{C}, p)$ is 0 if $p$ is unsolved, and $1/(1 + \log_{10}(T_p(\mathcal{C})/T_p^*))$, where $T_p(\mathcal{C})$ is the cpu time needed by planner $\mathcal{C}$ to solve problem $p$ and $T_p^*$ is the cputime needed by the best considered planner, otherwise. The IPC score on a set of problems is given by the sum of the scores achieved on each considered problem.

As the training set for learning macros for a given planner on a specific domain consists of about 5-8 simpler problems that were generated using the problem generators provided by the organisers. For a single planner and a single domain, the learning process required couple of seconds including generating the training plans and the execution of MUM.

**Results of Macro Generation:** In Table 1 the number of macros generated by MUM for a specific planner, on a given domain, is shown. Generally, the results show a good spread across planners and domains. Also, the total number of macros extracted per domain is usually lower than the prescribed limit of 4. Often the plans generated by different planners have similar structure, reflecting the domain's structure, resulting in the opportunity to generate only a low number of different macros. In Barman and Parking MUM did not generate any macro. In Barman we observe that there is not a sequence of operators which is frequently used in a plan, since the ways for generating different cocktails differ quite significantly, hence the reason why no macros were generated. In Parking, no outer entanglements were extracted, therefore potential macros had more possible instances (according to the number of components of their argument matching graphs), and so were filtered out. In BW and Depots, MUM did not generate macros for some planners because the training plans produced by these planners were of low quality (they were too long) which prevented MUM extracting useful outer entanglements at the start of the process. According to that, potential macros were not constrained enough (the number of components of argument matching graph was high), so they were filtered out.

**Results of Macro Use:** Cumulative results of the evaluation are presented in Table 2, with the performances exploited in the domain model enhanced with macros, compared to the original problem formulation. Values are computed only in domains in which MUM was able to find a set of macros for the given planner. The results show that, in general, exploiting the domain model extended with the extracted macros leads to a performance improvement in terms of number of solved problems, number of problems solved faster and IPC score (that is, improved coverage and speed). The exception to this are all the domains using the LPG planner. To investigate why, we ran LPG with entanglement-supported macros generated by other planners, and observed that the performance of LPG was usually better than the ones achieved on the original domain model. Hence, we postulate this behaviour is caused by (i) low quality solutions found for training problems by LPG (ii) the use of a local search algorithm in LPG being oversensitive to even a marginal increase of the branching factor caused by added macros.

**Results of Comparison with Wizard:** In order to evaluate the effectiveness of MUM with regards to the related state-of-the-art techniques for planner-independent generation of macros, we compared it to Wizard (Newton et al. 2007). For training Wizard about 90 problems per domain were used, divided in three sets accordingly to their complexity (assessed by the number of involved objects). Default parameters configuration was used. For a single planner, the process of generating macros on all the considered domains took between two and ten cpu-time days. Table 3 shows the detailed results achieved by each planner on the considered domains, while exploiting the original domain formulation, the domain model extended with macros found by Wizard and the formulation that includes the entanglement-supported macros extracted by MUM. Since Wizard is able to provide at most three different sets of macro operators, resulting from the execution of different sets of genetic operators, in Table 3 we reported only the

| | Solved | | | # Fastest | | | IPC score | | |
|---|---|---|---|---|---|---|---|---|---|
| **Barman** | O | W | M | O | W | M | O | W | M |
| FF | 0 | – | – | 0 | – | – | 0.0 | – | – |
| LAMA | 0 | – | – | 0 | – | – | 0.0 | – | – |
| LPG | 0 | – | – | 0 | – | – | 0.0 | – | – |
| Mp | 0 | 0 | – | 0 | 0 | – | 0.0 | 0.0 | – |
| Probe | 1 | 1 | – | 1 | 0 | – | **1.0** | 0.9 | – |
| **BW** | O | W | M | O | W | M | O | W | M |
| FF | 0 | – | – | 0 | – | – | 0.0 | – | – |
| LAMA | 18 | 0 | 27 | 2 | 0 | 25 | 13.2 | 0.0 | **26.8** |
| LPG | 20 | 30 | – | 0 | 30 | – | 9.2 | **30.0** | – |
| Mp | 0 | 0 | – | 0 | 0 | – | 0.0 | 0.0 | – |
| Probe | 20 | 18 | 30 | 0 | 0 | 30 | 13.6 | 11.1 | **30.0** |
| **Depots** | O | W | M | O | W | M | O | W | M |
| FF | 1 | 2 | 4 | 0 | 0 | 4 | 0.5 | 1.1 | **4.0** |
| LAMA | 3 | – | – | 3 | – | – | 3.0 | – | – |
| LPG | 13 | – | – | 13 | – | – | 13.0 | – | – |
| Mp | 6 | 5 | 19 | 1 | 2 | 17 | 4.7 | 3.9 | **18.6** |
| Probe | 30 | 30 | 30 | 1 | 0 | 29 | 23.0 | 23.6 | **29.9** |
| **Gripper** | O | W | M | O | W | M | O | W | M |
| FF | 0 | – | 25 | 0 | – | 25 | 0.0 | – | **25.0** |
| LAMA | 0 | – | 26 | 0 | – | 26 | 0.0 | – | **26.0** |
| LPG | 10 | – | 22 | 10 | – | 12 | 10.0 | – | **18.0** |
| Mp | 0 | 0 | 0 | 0 | 0 | 0 | 0.0 | 0.0 | 0.0 |
| Probe | 0 | – | 0 | 0 | – | 0 | 0.0 | – | 0.0 |
| **Parking** | O | W | M | O | W | M | O | W | M |
| FF | 7 | 0 | – | 7 | 0 | – | **7.0** | 0.0 | – |
| LAMA | 4 | – | – | 4 | – | – | 4.0 | – | – |
| LPG | 0 | – | – | 0 | – | – | 0.0 | – | – |
| Mp | 0 | – | – | 0 | – | – | 0.0 | – | – |
| Probe | 3 | – | – | 3 | – | – | 3.0 | – | – |
| **Rovers** | O | W | M | O | W | M | O | W | M |
| FF | 0 | – | 0 | 0 | – | 0 | 0.0 | – | 0.0 |
| LAMA | 6 | – | 29 | 0 | – | 29 | 4.1 | – | **29.0** |
| LPG | 28 | – | 27 | 26 | – | 2 | **27.9** | – | 21.1 |
| Mp | 1 | – | 0 | 1 | – | 0 | **1.0** | – | 0.0 |
| Probe | 20 | – | 11 | 19 | – | 1 | **20.0** | – | 9.4 |
| **Satellite** | O | W | M | O | W | M | O | W | M |
| FF | 0 | 0 | 7 | 0 | 0 | 7 | 0.0 | 0.0 | **7.0** |
| LAMA | 2 | – | 18 | 0 | – | 18 | 1.9 | – | **18.0** |
| LPG | 30 | – | 9 | 30 | – | 0 | **30.0** | – | 4.0 |
| Mp | 0 | 0 | 0 | 0 | 0 | 0 | 0.0 | 0.0 | 0.0 |
| Probe | 0 | – | 0 | 0 | – | 0 | 0.0 | – | 0.0 |
| **Spanner** | O | W | M | O | W | M | O | W | M |
| FF | 0 | – | 0 | 0 | – | 0 | 0.0 | – | 0.0 |
| LAMA | 0 | – | 0 | 0 | – | 0 | 0.0 | – | 0.0 |
| LPG | 30 | – | 26 | 22 | – | 8 | **29.2** | – | 22.0 |
| Mp | 0 | 20 | 20 | 0 | 0 | 20 | 0.0 | 18.7 | **20.0** |
| Probe | 0 | – | 0 | 0 | – | 0 | 0.0 | – | 0.0 |
| **TPP** | O | W | M | O | W | M | O | W | M |
| FF | 0 | – | 0 | 0 | – | 0 | 0.0 | – | 0.0 |
| LAMA | 13 | 13 | 16 | 3 | 0 | 15 | 11.2 | 10.8 | **15.9** |
| LPG | 1 | – | 2 | 1 | – | 2 | 0.8 | – | **2.0** |
| Mp | 1 | 8 | 2 | 0 | 7 | 1 | 0.9 | **8.0** | 1.9 |
| Probe | 10 | 6 | 15 | 0 | 0 | 15 | 6.8 | 3.4 | **15.0** |

Table 3: Number of solved problems, number of problems solved faster and IPC score achieved by considered planners on the benchmark domains, while exploiting the original formulation (O), the formulation extended with macros extracted by Wizard (W) or the formulation extended with macros extracted by MUM (M).

results achieved by the best one. In the results, Wizard is not able to generate macros in more cases (domain, planner) than our method. We also observed that wizard macros are usually 'long', in the sense that are composed by several operators. Moreover, some macros generated by Wizard seem somewhat counter-intuitive and therefore they do not seem to be very useful. According to the results shown in Table 3, we can derive that on the considered domains: (i) MUM is able to generate macros for most of the considered planners and domains; (ii) MUM is usually able to generate macros that improve the performance of the given planner; (iii) the macros generated by MUM have a greater impact on planners' performance than Wizard's.

One unwelcome side-effect of the use of macros is that their exploitation could have a negative impact on the quality of solution plans. In our experimental analysis we did not observe a significant difference between the quality of plans found by exploiting the original domain formulation and the domain model extended with macros; the average number of actions of plans is 416 in the former case and 422 in the latter. Therefore the macros generated by our approach are often able to improve the runtime of planners without decreasing the quality of solutions.

## Conclusions

In this paper we presented MUM, a technique for learning macros while maximising their utility by keeping them informative, though constrained. The number of possible instantiation of macros is limited by using outer entanglements in their generation and use. Moreover, MUM preserves completeness since only instances of macros are pruned. Macros generated by MUM generally improve the performance of planning engines by reducing the time to generate plans and increasing the number of solved problems, on the benchmark instances of the learning track of IPC-7. Also, macros generated by MUM have achieved impressive results in comparison to the macro learning system WIZARD (Newton et al. 2007) and we have evidence to show that this is not at the cost of poor quality solutions. Our experiments also give us useful insights into the current limitations of MUM: poor quality solutions used in training (as is the case with LPG) led to reduced performance, and in 2 of the 9 domains (Barman and Parking) no useful macro could be generated using our technique.

For future work, we are interested in investigating possibilities of learning problem- (or problem class-) specific macros. Inspired by work of Alhossaini and Beck (2013) which selects appropriate problem-specific macros from sets of macros learnt by existing approaches (e.g Wizard) we believe that implementing similar ideas on MUM will result in further improvements. We believe that extending our technique to support, for instance, ADL features (e.g. conditional effects) or durative actions (temporal planning), which is planned in future, will bring improvements to such kinds of planning techniques.

### Acknowledgements

The research was funded by the UK EPSRC Autonomous and Intelligent Systems Programme (grant no.

EP/J011991/1).

## References

Alhossaini, M. A., and Beck, J. C. 2013. Instance-specific remodelling of planning domains by adding macros and removing operators. In *Proceedings of SARA*.

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-FF: improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research (JAIR)* 24:581–621.

Chrpa, L., and Barták, R. 2009. Reformulating planning problems by eliminating unpromising actions. In *Proceedings of SARA*, 50–57.

Chrpa, L., and McCluskey, T. L. 2012. On exploiting structures of classical planning problems: Generalizing entanglements. In *Proceedings of ECAI*, 240–245.

Chrpa, L.; McCluskey, T. L.; and Osborne, H. 2012. Reformulating planning problems: A theoretical point of view. In *Proceedings of FLAIRS*, 14–19.

Chrpa, L. 2010a. Combining learning techniques for classical planning: Macro-operators and entanglements. In *Proceedings of ICTAI*, volume 2, 79–86.

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

Coles, A.; Coles, A.; Olaya, A. G.; Jiménez, S.; Lòpez, C. L.; Sanner, S.; and Yoon, S. 2012. A survey of the seventh international planning competition. *AI Magazine* 33:83–88.

Coles, A.; Fox, M.; and Smith, A. 2007. Online identification of useful macro-actions for planning. In *Proceedings of ICAPS*, 97–104.

Dawson, C., and Siklóssy, L. 1977. The role of preprocessing in problem solving systems. In *Proceedings of IJCAI*, 465–471.

Fikes, R., and Nilsson, N. J. 1971. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3/4):189–208.

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

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning, theory and practice*. Morgan Kaufmann Publishers.

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

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

Korf, R. 1985. Macro-operators: A weak method for learning. *Artificial Intelligence* 26(1):35–77.

Lipovetzky, N., and Geffner, H. 2011. Searching for plans with carefully designed probes. In *Proceedings of ICAPS*.

Minton, S. 1988. Quantitative results concerning the utility of explanation-based learning. In *Proceedings of AAAI*, 564–569.

Newton, M. A. H.; Levine, J.; Fox, M.; and Long, D. 2007. Learning macro-actions for arbitrary planners and domains. In *Proceedings of ICAPS*, 256–263.

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

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

Rintanen, J. 2012. Engineering efficient planners with SAT. In *Proceedings of ECAI*, 684–689.

Slaney, J., and Thiébaux, S. 2001. Blocks world revisited. *Artificial Intelligence* 125(1-2):119–153.