On the Online Generation of Effective Macro-operators

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

Macro-operator ("macro", for short) generation is a well-known technique that is used to speed-up the planning process. Most published work on using macros in automated planning relies on an offline learning phase where training plans, that is, solutions of simple problems, are used to generate the macros. However, there might not always be a place to accommodate training.

In this paper we propose OMA, an efficient method for generating useful macros without an offline learning phase, by utilising lessons learnt from existing macro learning techniques. Empirical evaluation with IPC benchmarks demonstrates performance improvement in a range of state-of-the-art planning engines, and provides insights into what macros can be generated without training.

1 Introduction

Macros are a well-known technique for encapsulating sequences of operators in the same format as original operators, and therefore can be used to reformulate planning domain models. 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]. Most published work on using macros as a planner-independent technique relies on a learning phase where training plans (solutions of simple problems) are used to generate the macros. It is the case of MacroFF [Botea *et al.*, 2005], Wizard [Newton *et al.*, 2007], and MUM [Chrpa *et al.*, 2014]. These approaches depend to some extent on the bias of the selection of the training problems as well as on the planners used to generate training plans. In real-world applications, however, it can be hard to identify, select, and properly manage a complete –and usually expensive– learning phase. Moreover, the real world is dynamic and changes frequently. In many applications, the structure and topology of problems that a planner is expected to solve can change over time, thus the training instances used for the offline learning may not be a representative example of testing problems. Nowadays, specially in planning applications, planners can be understood as "black boxes" taking domain and problem descriptions as inputs and providing solution plans as output. Hence, there is a good motivation for a macro learning technique that can be wrapped around an existing planning engine and will enhance its performance. Existing online techniques are mostly planner-dependent, such as Marvin [Coles *et al.*, 2007], which is built on top of the FF planner [Hoffmann and Nebel, 2001]. Although planner-dependent techniques can focus on drawbacks of planners, e.g., escaping plateaus efficiently, their adaptability for different planning engines is usually low. Finally, there is no single planner that outperforms all others in every known benchmark domain. Therefore, different planners can be useful in solving problems from different domains. Hence, there is a good motivation for developing a planner-independent method that can generate macros without an offline learning phase, but by analysing domain and problem description only.

In this paper we introduce OMA, a method for Online MAcro generation. Evidently, online generation requires to be efficient, i.e., able to generates macros in a very short time. Furthermore, the absence of training plans brings a challenge since we do not get useful information about structure of plans, i.e., which instances of which operators and how often they are applied consecutively. On the other hand, domain and problem models can still provide some information, e.g., dependencies between planning operators and, moreover, existing macro learning techniques can give us valuable insights into how useful macros can be generated. In particular, macros have to be sound and should be applicable at some point of the planning process. Also, macros are often useful if they capture a single and meaningful activity, and have a relatively small number of instances. OMA combines these ideas. We empirically evaluate it using International Planning Competition (IPC) benchmarks and a range of state-of-the-art planning engines. The empirical evaluation demonstrates performance improvement in considered planners, and provides insights into what macros can be generated without training.

2 Background

AI planning deals with finding a partially or totally ordered sequence of actions to transform the environment from an initial state to a desired goal state. Classical planning is a restricted form of AI planning, where environment is static and fully observable, and actions are deterministic and instantaneous [Ghallab *et al.*, 2004].

In the PDDL representation, the environment

is described by predicates. *States* are defined as sets of grounded predicates. We say that $o = (\mathit{name}(o), \mathit{pre}(o), \mathit{eff}^-(o), \mathit{eff}^+(o), \mathit{cost}(o))$ is a *planning operator*, where $\mathit{name}(o) = \mathit{op\_name}(x_1, \ldots, x_k)$ (*op_name* is an unique operator name and $x_1, \ldots x_k$ are variable symbols (arguments) appearing in the operator) and $\mathit{pre}(o), \mathit{eff}^-(o)$ and $\mathit{eff}^+(o)$ are sets of (ungrounded) predicates with variables taken only from $x_1, \ldots x_k$ representing o's precondition[1], negative, and positive effects, and $\mathit{cost}(o)$ is a numerical value representing o's cost[2]. *Actions* are grounded instances of planning operators. An action a is *applicable* in a state s if and only if $\mathit{pre}(a) \subseteq s$. Application of a in s (if possible) results in a state $(s \setminus \mathit{eff}^-(a)) \cup \mathit{eff}^+(a)$.

A *planning domain model* is specified by a set of predicates and a set of planning operators. A *planning problem* is specified via a domain model, an initial state, and a set of goal predicates. A *solution plan* of a planning problem is a sequence of actions such that their consecutive application starting from the initial state results in a state containing all the goal predicates. Static predicates are an important class in this paper. A predicate is *static* if it is not present in the effects of any operator in a given domain model.

A *Substitution* is a mapping from variable symbols to terms that is used to determine which arguments (variable symbols) operators share. Hereinafter, we will assume that different operators have distinct parameters as arguments unless stated otherwise. In the set operations over sets of ungrounded predicates, we assume that only predicates having the same name and the same arguments (i.e., the same parameter symbols ordered in the same way) are equivalent.

Naturally, operators create predicates required by other operators. This indicates how instances of these operators will be ordered in plans. Such a relation is denoted as being an *achiever* and is formally defined as follows.

Definition 1. *We say that an operator o_1 is an **achiever** for an operator o_2 with respect to a substitution Θ if and only if $\mathit{eff}^+(o_1) \cap \mathit{pre}(\Theta(o_2)) \neq \emptyset$.*

Also, we may observe that some predicates cannot be present together in any valid state (reachable by applying a sequence of actions from an initial state), which are known as *mutex predicates*.

Definition 2. *We say that grounded predicates p_g and q_g are **mutex** with respect to a problem P if and only if for every state s such that s is reachable from the initial state of P it is the case that $\{p_g, q_g\} \not\subseteq s$.*

Heuristically, we can determine whether grounded predicates p_g, q_g are mutex by checking the following conditions: i) p_g, q_g are not both present in an initial state, and ii) for every action a, if $p_g \in \mathit{eff}^+(a)$, then $q_g \in \mathit{eff}^-(a) \setminus \mathit{eff}^+(a)$, and analogously if $q_g \in \mathit{eff}^+(a)$, then $p_g \in \mathit{eff}^-(a) \setminus \mathit{eff}^+(a)$. This can be easily extended for determining whether ungrounded predicates are mutex.

[1] including negative equality

[2] For domain models, where action cost is not explicitly enabled, every operator o has implicitly $\mathit{cost}(o) = 1$.

2.1 Macro-operators

Macros can be 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*. A macro $o_{i,j}$ is constructed by assembling planning operators (or macros) o_i and o_j (in that order) with respect to a substitution Θ (o_i and o_j may share some arguments), i.e.,

$$\begin{aligned} o_{i,j} = & \mathit{name}(o_i) \cdot \mathit{name}(o_j), \\ & \mathit{pre}(o_i) \cup (\mathit{pre}(\Theta(o_j)) \setminus \mathit{eff}^+(o_i)), \\ & (\mathit{eff}^-(o_i) \setminus \mathit{eff}^+(\Theta(o_j))) \cup \mathit{eff}^-(\Theta(o_j)), \\ & (\mathit{eff}^+(o_i) \setminus \mathit{eff}^-(\Theta(o_j))) \cup \mathit{eff}^+(\Theta(o_j)), \\ & \mathit{cost}(o_i) + \mathit{cost}(o_j). \end{aligned}$$

To ensure correctness of planning with macros, each macro has to be sound. A macro $o_{i,j}$ is *sound* if and only if for all its grounded instances $(\mathit{eff}^-(a_i) \setminus \mathit{eff}^+(a_i)) \cap \mathit{pre}(a_j) = \emptyset$, where a_i and a_j are instances of o_i and $\Theta(o_j)$ respectively. If soundness of $o_{i,j}$ is violated o_i and o_j cannot be applied consecutively because o_i deletes a predicate required by o_j.

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 fewer steps. However, the number of instances of macros is often high, because they usually have a large set of parameters, that derives from the parameters of the original 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 exploitation outweigh their drawbacks. This problem is known as the *utility problem* [Minton, 1988].

2.2 Outer Entanglements

Outer Entanglements are relations between planning operators and initial or goal predicates, and have been introduced as a technique for eliminating potentially unnecessary instances of these operators [Chrpa and McCluskey, 2012].

Generally speaking, entanglements by init (hereinafter entInit) capture situations, where only instances of a given operator that requires instances of a certain predicate present in the initial state are needed to solve the problem. Similarly, entanglements by goal (entGoal) capture situations, where only instances of a given operator that achieves instances of a certain predicates present in the set of goal predicates are needed to solve the problem.

For example, in the BlocksWorld domain, the operator unstack is entInit with the predicate on, since we need to unstack blocks only from their initial positions. Similarly, the operator stack is entGoal with the predicate on, since we need to stack blocks only to their goal positions.

Definition 3. *Let P be a planning problem, where I is the initial state and G is a set of goal predicates. Let o be a planning operator and p be a predicate defined in P. We say that operator o is **entangled by init (resp., goal)** with predicate p if and only if $p \in \mathit{pre}(o)$ (resp., $p \in \mathit{eff}^+(o)$) and there exists a solution plan π of P such that for every o's instance $a \in \pi$ and for every p's grounded instance p_{gnd} it holds: $p_{gnd} \in \mathit{pre}(a) \Rightarrow p_{gnd} \in I$ (resp., $p_{gnd} \in \mathit{eff}^+(a) \Rightarrow p_{gnd} \in G$).*

Outer entanglements have been used as a reformulation technique, as they can be directly encoded into a problem and domain model. The way outer entanglements are encoded is inspired by one of their properties: given a static predicate p_s, an operator o is entangled by init with p_s if and only if $p_s \in pre(o)$ [Chrpa and Barták, 2009]. For an operator o being entangled by init (resp., goal) with a predicate p, a supplementary static predicate p' that "clones" p is introduced and put into $pre(o)$. Initial (resp., goal) instances of p are "cloned" and put into the initial state. For a formal description of the reformulation process, the interested reader is referred to [Chrpa and McCluskey, 2012].

2.3 Estimating the Number of Operator Instances

Estimating the number of instances of a macro is important for a more accurate determination of its utility [Chrpa *et al.*, 2014]. For example, in the Depots domain the operator Lift(?h - hoist ?c -crate ?s - surface ?p - place) can potentially have $\#hoists \cdot \#crates \cdot \#surfaces \cdot \#places$ instances. In Lift's precondition, there is a static predicate (at ?h ?p) constraining a hoist to be at exactly one place. Hence, the number of Lift's instances can drop to $\#hoists \cdot \#crates \cdot \#surfaces$. It can be observed that Lift is entangled by init with predicates (on ?c ?s) and (at ?c ?p). A crate can be initially stacked only on one surface (another crate or a pallet) and be at one place. Hence, if the entanglements are involved the potential number of Lift's instances can drop to $\#crates$.

Static predicates and outer entanglements thus provide matching between arguments (parameters) 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) such that arguments of the operator are nodes and edges are between nodes whose corresponding arguments appear together in a static predicate present in operator's precondition or in a predicate involved in the outer entanglement relation with the operator (for the formal definition, see [Chrpa *et al.*, 2014]).

Given the problem description, it is possible to determine whether the number of instances of static predicates or predicates involved in an entanglement relation in the initial or goal situation is bounded by $\mathcal{O}(n)$ (n stands for the number of objects). Predicates that do not follow this are not considered when constructing (simple) argument matching graphs. Let n be the number of objects defined in a 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 estimated as $\mathcal{O}(n^c)$. It will be useful for estimating impact of generated macros on branching factor.

3 Online Macros Learning

A wrapper, which enables a planning engine to exploit macros without altering the planner, can be designed as follows: i) macros are generated online by analysing the original domain and problem model; ii) reformulated models, that include extracted knowledge, are prepared; iii) the planning engine is run using reformulated models; iv) the solution plan is passed to a "macro unfolding" tool that replaces macros by corresponding sequences of actions. Hereinafter, we will focus on the OMA module for online macros generation.

OMA has only the problem and domain model that is available from the input to work with. In summary, macros are often useful when: i) they capture some "coherent and meaningful" activity (e.g., moving an object from its initial to its goal position), ii) they do not have a considerably larger number of instances than original operators, iii) the number of learnt macros is small. While iii) is easy to accommodate, there are some fundamental issues regarding i) and ii). To tackle these issues, we use insights from existing techniques, namely, MUM [Chrpa *et al.*, 2014], MacroFF [Botea *et al.*, 2005], and DHG [Armano *et al.*, 2004]. OMA incorporates the idea of using outer entanglements to reduce the number of instances of macros exploited by MUM. As in MUM, we apply entanglements only on macros (so completeness is not compromised). However, we cannot extract entanglements from training plans as MUM could. Instead, OMA makes an informed selection of them from the problem and domain model descriptions.

3.1 Identifying Outer Entanglements

Deciding outer entanglements is intractable (PSPACE-complete) in general: entanglements are heuristically extracted from training plans [Chrpa and McCluskey, 2012]. However, without training plans, we have to derive entanglements from the domain and problem description only.

It has been observed from training-based approaches (e.g., MUM) that a higher number of initial or goal instances of a predicate with respect to the number of objects relevant to the predicate makes the chance that the predicate is in an outer entanglement relation rise. Where we have all the possible initial or goal instances of a predicate, then the predicate is certainly involved in the entanglement relation, while having no initial or goal instance of the predicate, the predicate might be involved in the entanglement relation only with an operator that is not used in plan generation. We have to compromise in order to achieve a satisfactory level of accuracy of entanglement identification (i.e., the number of initial or goal instances of the predicate should not be too low) and exploiting the pruning power of outer entanglements (i.e., the number of initial or goal instances of the predicate should not be too high).

From the observation above, we derive the following formula that determines whether a predicate p is a suitable candidate for the outer entanglement relation. Notice that $\#p$ is the number of p's initial or goal instances. Assume each predicate argument is typed and it is possible to calculate the number of possible objects that can fill each argument. Then $\#x$ is defined as the maximum of the number of possible objects in each argument. (e.g., having a predicate (at ?truck ?place), and 10 trucks and 15 places, then $\#x$ = 15).

$$c_1 \cdot \#x \leq \#p \leq \ c_2 \cdot \#x, \ 0 < c_1 \leq c_2$$

Hence, the most suitable candidate for the outer entanglement is determined by having the number of initial or goal instances within "reasonable" bounds. c_2 should not be greater than 1.0 in order to sustain the accuracy of instances estimation by the Argument Matching Graphs (see Section 2.3). With a decreasing value of c_1 the risk of "false positives" (i.e., identification of incorrect entanglements) rises.

Algorithm 1 The OMA algorithm.

```
1: IdentifyEntanglements()
2: for all o, o' ∈ ops, o' ≠ o, not Conn(o), not Conn(o'), not
   HasEG(o), HasEI(o) or HasEG(o') do
3:    Φ = {Θ | o is an achiever for o' w.r.t Θ}
4:    while Φ ≠ ∅ do
5:       Θ = PopMostSpecific(Φ)
6:       ma = CreateMacro(o, o', Θ)
7:       if UsefulMacro(ma) and (AMG(ma)≤AMG(o) or
         AMG(ma)≤AMG(o')) then
8:          ops = ops ∪ {ma}
9:          break
10:      end if
11:   end while
12: end for
13: FilterMacros()
```

If a predicate is considered for the entInit (or entGoal) relation with two (or more) operators and one operator deletes an effect achieved by (one of) the other operator(s), then the predicate is unlikely to be in the entanglement relation with both. This is because such operators often take place in different phases of plans (e.g., in Depots, Lift acts in the initial stage and Drop acts in the final stage while both have the entanglement candidate (at ?crate ?surface) in their preconditions). In the EntInit case, one operator whose instances are more "likely" to be applicable in the initial state than instances of the other operator is a good candidate for the EntInit relation with the predicate.

Definition 4. *Let o_1 and o_2 be planning operators. o_1 is more* **likely** *to be applicable in the initial state than o_2, denoted as $o_1 \prec o_2$, is determined as follows. For every $p \in pre(o_1)$ such that for no substitution Θ, $\Theta(p) \in pre(o_2)$, it holds that $c_1 \cdot \#x \leq \#p \leq c_2 \cdot \#x$ (c_1, c_2, $\#p$ and $\#x$ have the same meaning as before).*

Assume we have a predicate p such that p is, according to the previous formula, a suitable candidate for the entanglement relation. Further, assume we have operators o_1 and o_2 such that p is present in both o_1 and o_2's precondition (resp., positive effects). Entanglements are determined according to following rules (second and third rules apply only to entInit):

- If there is no substitution Θ such that $\mathit{eff}^-(o_1) \cap \mathit{eff}^+(\Theta(o_2)) \neq \emptyset$ or $\mathit{eff}^+(o_1) \cap \mathit{eff}^-(\Theta(o_2)) \neq \emptyset$, then both o_1 or o_2 are entInit (resp., entGoal) with p.
- If $o_1 \prec o_2$ and $o_2 \not\prec o_1$, then o_1 is entInit with p.
- If $o_2 \prec o_1$ and $o_1 \not\prec o_2$, then o_2 is entInit with p.

This approach can be easily extended for situations where more than two operators are candidates for the entanglement relation with the same predicate.

3.2 Generating Macros

For OMA's purpose, we define a special class of macros, called "connected macros". A *connected macro* contains both the entInit and entGoal relations in which the same object is involved. Connected macros thus transform the object state from a given initial state to a required goal one (a similar idea was previously used as a heuristic for identifying useful macros in the work of [McCluskey and Porteous, 1997]). For example, in the Gripper domain, Pick-Move-Drop is a connected macro, since it moves a ball from its initial to its goal position.

OMA is depicted in Algorithm 1. As a preliminary step, we identify possible outer entanglements as described in the previous subsection. Lines 2–12 contain the main loop, where pairs of (different) operators (including macros generated in the iterative process) are checked whether they are suitable to become macros. The loop terminates when either all the pairs have been checked or a number (l) of extracted macros is reached. Since the macro generation process is performed online, it is of critical importance to avoid the generation of large number of macros. Check of limit l is not shown in Algorithm 1 for the sake of readability. For a potential macro candidate composed by o and o' (in this order), it must hold that neither o nor o' is a connected macro, o has not any entGoal, o has an entInit with a non-static predicate or o' has an entGoal. Connected macros already capture coherent activities, so they do not have to be expanded. In other cases, original operators are chained starting by one having an entInit (with a non-static predicate) or ending by one having an entGoal. Moreover, an original operator having an entGoal can be only the last in the chain. So, outer entanglements serve as a heuristics for devising macros capturing a "meaningful" activity, i.e., those that come out of initial state or achieves goal predicates.

We require that o is an achiever for o' (i.e., the "chaining rule" in MacroFF), so we compute all substitutions for which it holds (Line 3) (arguments appearing in the definition of o' can be substituted only for arguments appearing in the definition of o). If o is not an achiever for o', then no substitution is found (notice that if o achieves a nullary predicate for o', an empty substitution is found). The most specific substitution, i.e., the substitution which substitutes the largest number of arguments, is taken (Line 5). Then, a new macro ma is created (Line 6) with entanglements considered in its encoding. If ma passes the "usefulness check" (Line 7), ma is added into the set of operators and we go back to Line 2. If ma fails the "usefulness check" (Line 7), we go back to Line 4 and try the next most specific substitution, or (if no substitution left) try other macro candidates.

The usefulness of a potential macro ma is assessed by the following checks. **First**, o must not delete a predicate required by o', otherwise the macro ma is not sound. **Second**, ma cannot contain mutex predicates in its precondition, since otherwise no instance of ma is applicable in any point. A similar constraint has been used by DHG. **Third**, ma cannot contain consecutive inverse (original) operators, that is, operators which reverse each other effects. **Fourth**, no operator can be present more than once in ma. This is a stronger form of "anti-recursion rule" (i.e., preventing recursions within macros such as Lift-Load-Lift-Load) that was used by MUM. Our experience indicates that useful macros do not usually contain any of the original operators more than once. **Fifth**, ma cannot contain more variants of some static predicate in the precondition than o or o' contains. It is not allowed, for instance, to have predicates (at ?h ?p1) and (at ?h ?p2) in the precondition of a Lift-Load macro, since the

original operators Lift and Load contain only one variant of the at predicate. This is a stronger form of the "locality rule" used by Macro-FF, and helps macros to be focused on a single activity. **Sixth**, the number of components of Argument Matching Graph of ma (AMG(ma)) cannot be larger than that for o or that for o' (for original operators, we consider Simple Argument Matching Graphs). This constraint (derived by MUM) prevents generating macros with a large number of instances.

Out of the generated macros, OMA prefers (in this order) macros that have fewer possible instances, connected macros, or macros that contain less operators. Therefore, filtering is composed of the following steps. **First**, a macro ma such that AMG(ma)$\geq \frac{\sum_{o \in O} \text{AMG}(o)}{|O|}$ is filtered out (O is a set of original operators). **Second**, macros are ordered according to their "potential utility". A macro ma is considered as "better" than a macro ma' if: i) AMG(ma)<AMG(ma'), or ii) AMG(ma)=AMG(ma'), and ma is connected, while ma' is not, or iii) AMG(ma)=AMG(ma'), and Length(ma)<Length(ma'). Notice that Length(ma) denotes the number of original operators incorporated in ma. Finally, top k macros are selected (k is a OMA parameter)[3].

	IPC score		Solved		Runtime	
Domain	P	O	P	O	P	O
Floortile	29.9	**47.7**	32	**48**	28.0	**21.6**
GED	**80.0**	76.4	**86**	84	**69.4**	72.5
Hiking	**62.9**	60.0	**64**	61	70.8	**65.4**
Parking	11.2	**25.0**	12	**25**	206.7	**148.0**
Transport	**43.9**	41.5	**44**	43	**73.5**	78.9
Planner	P	O	P	O	P	O
ArvandHerd	32.6	**36.9**	34	**38**	153.2	**146.3**
Bfs-f	16.7	**20.6**	18	**21**	62.6	**44.3**
Cedalion	36.5	**53.5**	38	**54**	107.8	**98.1**
Freelunch	11.9	**12.7**	12	**13**	30.3	**28.4**
Mpc	**44.3**	34.6	**45**	35	42.3	**35.4**
Probe	34.1	**40.8**	36	**43**	73.2	**71.7**
Yahsp3	**51.7**	51.4	55	**57**	**14.5**	27.3

Table 1: Cumulative IPC score, coverage, and average runtime of planners exploiting original (P) and OMA enhanced (O) problems. Average runtime considers only problems solved by both P and O, only. Results are cumulative with regards to domains (top table) or planners (bottom table). The best results in boldface.

4 Experimental Analysis

The aims of this experimental analysis are: i) to evaluate the effectiveness of OMA, with respect to increasing the speed and coverage of plan generation over a range of domains and planning engine combinations, and ii) to compare OMA to the state-of-the-art of conventional macro generation methods which have the benefit of training plans. All the experiments were run on 3.0 Ghz machine CPU with 4GB of RAM. The constants c_1, c_2, l and k (see the previous section) were set to 0.4, 1.0, $\min(8, 2 \cdot |ops|)$ and $\min(4, |ops|)$ respectively for all the benchmarks. They were set by considering results on a small set of problems/domains. In this experimental analysis, IPC score as defined in IPC-7 is used [Coles *et al.*, 2012]. 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 CPU-time 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.

4.1 Results of Macros Use

Since macros primarily aim at speeding-up the planning process, we decided to use the same methodology for evaluating the performance as the one used in the Agile track of the last International Planning Competition (IPC-8). We have considered all the benchmark domains from this track. As benchmarking planners, we chose the top 7 performing planners of the Agile track, namely, Yahsp3, Mpc, Probe, Bfs-f, Cedalion, Freelunch, and Arvandherd. The interested reader can find description of planners in the IPC booklet [Vallati *et al.*, 2014]. Different versions of a planner were not included. A runtime cutoff of 300 CPU seconds (as in Agile track) was used.

[3]If k is too high, we might keep many useless macros.

CPU-time required by OMA for learning and unfolding macros is usually smaller than 0.1 seconds, and is included. In Floortile, GED, and Parking, macros were found for each problem, while in Transport and Hiking, respectively in 5 and 13 instances. Usually, between 1 and 3 macros were generated per problem, and they included 2 or 3 operators. On the other hand, in Barman, Child-Snack, Tetris, Thoughtful, and Visitall no macros were found in any of the benchmarks, often because no entanglements were generated. The remaining domains include features that are not currently supported (e.g., conditional effects). Notice that failing to find macros leaves the domain and problem encoding the same as the original one, and has negligible impact on CPU time.

Results of the evaluation are presented in Table 1, where we compared the performance achieved by planners exploiting OMA against their performance on original formulations. Values consider domains in which OMA generated at least one macro for at least one problem, only. Generally, we can see that macros improved performance of the planners except Mpc, where macros performed very poorly in the GED domain. From the domains perspective, the performance in Floortile and Parking have been improved considerably for most of the planners, while in the rest of the domains it has remained nearly the same. Regarding plan quality, 47% of the solved problems have same quality, while using macros improved quality in 22%. Concerning domains, only in GED we observed that some planners provided often worse quality solutions when using macros. Figure 1 provides a better overview of the impact of OMA online macros on planners' performance. It can be seen that online macros improves performance of the planners in most of the cases, both in terms of runtime and coverage (see the right edge of the graph). It is worth noting that many original problems are quickly solved, without leaving much space for improvements.

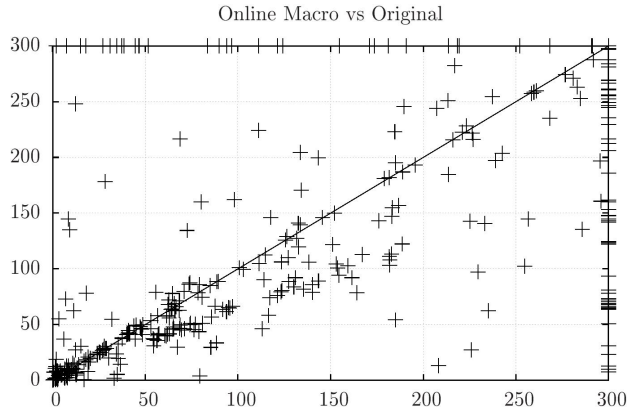

Figure 1: CPU time of planners exploiting problems extended with online macros (y-axis) w.r.t original problem formulation (x-axis) for all the considered benchmarks.

	Δ IPC score		Δ Solved	
Domain	M	O	M	O
Bw	30.0	**73.3**	19	**49**
Depots	**24.3**	0.2	**13**	-5
Gripper	**59.0**	-0.7	**63**	4
Parking	–	**22.7**	–	**21**
Rovers	**6.5**	–	**12**	–
Satellite	-2.9	–	**2**	–
Spanner	**12.8**	–	**16**	–
TPP	**14.6**	3.1	**9**	1
Planner	M	O	M	O
FF	**35.5**	3.0	**35**	2
LAMA	**85.3**	19.1	**77**	14
LPG	-30.8	**9.5**	-13	**4**
Mp	**34.1**	17.9	**33**	18
Probe	20.9	**40.3**	6	**23**

Table 2: Δ of performance between planners exploiting macros and original domain models. M stands for offline macros learned by MUM technique, O for online macros.

4.2 Comparison with other Online Technique

We compared OMA with Marvin [Coles *et al.*, 2007], a state-of-the-art planner-specific online macro learning technique. For the comparison, we used the same benchmarks and the same methodology as in the Agile track of the IPC-8 (as in the previous subsection). Although Marvin is able to carry macros forward (i.e., macros learnt in one problem can be used for solving another problem), we did not use this feature in order to provide fair comparison.

On considered benchmarks the overall difference of IPC scores with respect to original encodings is -20.7 for Marvin and -4.5 for OMA (we used the Marvin implementation of FF). OMA did not perform very well in Hiking and Floortile, while Marvin was considerably worse in the Barman domain, where OMA did not generate any macro. It is worth noting that FF did not solve a large number of benchmark instances and, therefore, it might not be sufficient to draw conclusions.

4.3 Comparison with Offline Technique

In order to compare OMA with a state-of-the-art approach that extract macros offline, we compare it against MUM [Chrpa *et al.*, 2014]. For avoiding issues related to a possible selection bias of training problems, and to consider a broader range of benchmarks, the comparison has been run using the same setup that Chrpa *et al.* used in their analysis. Therefore, we use all the benchmark instances used in the learning track of IPC-7. As benchmarking planners we use Metric-FF [Hoffmann and Nebel, 2001], LPG-td [Gerevini *et al.*, 2003], LAMA-11 [Richter and Westphal, 2010], Mp [Rintanen, 2012] and Probe [Lipovetzky and Geffner, 2011]. A runtime cutoff of 900 CPU seconds (as in learning tracks of IPC) was used. Table 2 shows the results of such comparison, in terms of improvement gap between performance achieved by using original domain formulation and reformulation that include macros (respectively, MUM or OMA). Remarkably, OMA macros are able to improve the performance of planners in terms of both coverage and runtime, and never decrease their overall performance. Surprisingly, out of 5 planers, two of them (LPG and Probe) performed better on online macros than offline ones generated by MUM. In terms of per-domain analysis, in Barman none of the methods extracted any macro. OMA performed considerably well in BW and Parking and considerably outperformed MUM. In BW, in some cases no macro was extracted by MUM because of the poor quality of training plans. In the other domains OMA's performance was nearly the same as for original problems.

4.4 Discussion

In summary, OMA demonstrated that in most cases we can improve the performance of planning engines by exploiting macros learned in a very quick and effective pre-processing step. The time needed for identifying macros and extending the domain and problem models is marginal (usually tens of milliseconds), which, in contrary to offline macro learning methods brings a lot of flexibility. The results indicate that OMA can be successful on a number of domains (e.g., Parking, BW), in which it was able to generate very useful macros, while in the others achieved performance is comparable – neither much better nor much worse – to those achievable by using original models. Of course, this observation might be related to a kind of conservative strategy OMA uses for generating and filtering macros – heavily focusing on the number of their possible instances.

The space of macros OMA can generate is huge (infinite, if macros' length is not limited). The only actual hard constraint that can be used for pruning the "macro" space is soundness (i.e., within the macro, an operator cannot delete a predicate required by another operator). Even macros having mutex predicates in their preconditions might be useful, for instance, while computing heuristics on delete-relaxed Planning Graph, as showed by Newton *et al.* (2007). Overall, the rules OMA used for generating and filtering macros provided a good guidance for identifying useful macros.

We observed that OMA does not perform well in 'Logistic'-like domains. It often did not generate macros such as load-move-unload, or generated some counter-intuitive and possibly useless macros (e.g. move-load-unload). In the Gripper domain, MUM generated only the macro pick-

move-drop, while OMA generated macros such as pick-drop. Therefore, the rules for pruning the space of macros do not filter all the potentially useless macros; extending domain models with useless macros lead to decrementing performance. However, we should not forget that the usefulness of types of macro might heavily depend on the specific formulation of the domain (e.g., vehicle capacity, road network) and, to some extent, on the planner that will exploit them.

5 Conclusions

In this paper we introduced OMA, a method for online generation of macros that can be wrapped around standard planning engines. An extensive empirical evaluation shows that OMA improves the performance of planning engines by reducing the time to generate plans and increasing the number of solved problems, on benchmarks from IPC-8 (agile track) and IPC-7 (learning track). Interestingly, OMA outperformed Marvin –a planner-specific macro generating method– and in some cases also MUM –an offline macro generating method– despite their clear advantages. OMA is thus a good alternative for situations, where it is not possible to use an offline technique to generate macros.

As part of future research, we want to investigate the exploitation of predictive models based on planning features – real numbers that characterise some aspects of a planning problem – [Cenamor *et al.*, 2013; Fawcett *et al.*, 2014] for improving the selection of macros.

Acknowledgements The research was funded by the UK EPSRC Autonomous and Intelligent Systems Programme (grant no. EP/J011991/1). The authors would like to acknowledge the use of the University of Huddersfield Queensgate Grid in carrying out this work.

References

[Armano *et al.*, 2004] G. Armano, G. Cherchi, and E. Vargiu. Automatic generation of macro-operators from static domain analysis. In *Proceedings of ECAI*, pages 955–956, 2004.

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

[Cenamor *et al.*, 2013] I. Cenamor, T. De La Rosa, and F. Fernández. Learning predictive models to configure planning portfolios. *Proceedings of ICAPS-PAL*, pages 14–22, 2013.

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

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

[Chrpa *et al.*, 2014] L. Chrpa, M. Vallati, and T. L. McCluskey. MUM: A technique for maximising the utility of macro-operators by constrained generation and use. In *Proceedings of ICAPS*, pages 65–73, 2014.

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

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

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

[Fawcett *et al.*, 2014] C. Fawcett, M. Vallati, F. Hutter, J. Hoffmann, H. H. Hoos, and K. Leyton-Brown. Improved features for runtime prediction of domain-independent planners. In *Proceedings of ICAPS*, pages 355–359, 2014.

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

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

[Ghallab *et al.*, 2004] M. Ghallab, D. Nau, and P. Traverso. *Automated planning, theory and practice*. Morgan Kaufmann, 2004.

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

[Lipovetzky and Geffner, 2011] Nir Lipovetzky and Hector Geffner. Searching for plans with carefully designed probes. In *Proceedings of ICAPS*, pages 154–161, 2011.

[McCluskey and Porteous, 1997] T. L. McCluskey and J. M. Porteous. Engineering and compiling planning domain models to promote validity and efficiency. *Artificial Intelligence*, 95(1):1–65, 1997.

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

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

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

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

[Vallati *et al.*, 2014] M. Vallati, L. Chrpa, and T. L. McCluskey. *The Eighth International Planning Competition. Description of Participant Planners of the Deterministic Track*. 2014.