Exploiting Block Deordering for Improving Planners Efficiency

Lukáš Chrpa
PARK Research group
School of Computing & Engineering
University of Huddersfield

Fazlul Hasan Siddiqui
NICTA Optimisation Research Group
Research School of Computer Science
The Australian National University, Australia

Abstract

Capturing and exploiting structural knowledge of planning problems has shown to be a successful strategy for making the planning process more efficient. Plans can be decomposed into its constituent coherent subplans, called *blocks*, that encapsulate some effects and preconditions, reducing interference and thus allowing more deordering of plans. According to the nature of blocks, they can be straightforwardly transformed into useful macro-operators (shortly, "macros"). Macros are well known and widely studied kind of structural knowledge because they can be easily encoded in the domain model and thus exploited by standard planning engines.

In this paper, we introduce a method, called BLOMA, that learns domain-specific macros from plans, decomposed into "macro-blocks" which are extensions of blocks, utilising structural knowledge they capture. In contrast to existing macro learning techniques, macro-blocks are often able to capture high-level activities that form a basis for useful longer macros (i.e. those consisting of more original operators). Our method is evaluated by using the IPC benchmarks with state-of-the-art planning engines, and shows considerable improvement in many cases.

1 Introduction

Capturing and exploiting structural knowledge of planning problems has shown to be a successful strategy for improving efficiency of the planning process. A well-known technique for encapsulating sequences of operators, macro-operators ("macros", for short), is a good example of such structural knowledge due to possibility to encode macros in the same format as original operators, so they can be used in a planner-independent way. Macros date back to 1970s where they were used, for example, in STRIPS [Fikes and Nilsson, 1971] and REFLECT [Dawson and Siklóssy, 1977]. Korf (1985) has shown that using macros can reduce the problem complexity in some cases which gives a good motivation for their use. Hence, many successful macro learning techniques have been developed in recent years (see the following section).

Analysis of training plans, usually solutions of simpler problems, is a key step in the macro learning process. Totally ordered plans, however, often "hide" some promising candidates for macros, since corresponding actions are not adjacent. Chrpa (2010) proposed a technique that as macro candidates considered actions non-adjacent in a given plan but adjacent in some of its permutations. Recently, a technique that decomposes plans into its constituent coherent subplans, *blocks* [Siddiqui and Haslum, 2012], was developed, and successfully applied in post-processing based plan quality optimisation [Siddiqui and Haslum, 2013]. Blocks can reduce interference between actions and thus allowing more deordering of plans, hence blocks can be exploited in the form of macros.

In this paper, we introduce *macro-blocks* that extend the blocks by considering relations between them in order to provide more promising macro candidates. In some domains, macro-blocks can capture longer subplans representing important activities. Then, we introduce a method, called BLOMA, that from macro-blocks achieved by decomposition of training plans extracts domain-specific macros. Given the property of macro-blocks, BLOMA can generate useful longer macros in problems whose structure relies on repetitive application of a larger sets of actions. Traditional macro learning techniques that are based on "operator chaining" approaches (i.e. assembling operators one by one) are often not able to find such long macros. BLOMA is evaluated by using the IPC benchmarks with state-of-the-art planning engines, and shows considerable improvement in many cases.

2 Related Work

Recent macro learning systems can be categorised as planner-independent or planner-specific. The CA-ED version of MacroFF [Botea *et al.*, 2005] generates macros according to several pre-defined rules (e.g., the "locality rule") and by exploring adjacent actions in plans. SOL-EP version of MacroFF learns macros from training plans and uses them for improving the performance of the FF planner [Hoffmann and Nebel, 2001]. Marvin [Coles *et al.*, 2007], which is built on top of FF, combines offline and online macro generating techniques in order to escape plateaus in heuristics landscape. Wizard [Newton *et al.*, 2007] learns macros by exploiting genetic programming. Alhossaini and Beck (2013) proposed a technique for efficient selection of problem-specific macros from a set of macros learnt by some of the existing techniques. A more recent macro learning method, called MUM [Chrpa *et al.*, 2014], is based on Chrpa's (2010) method considering non-adjacent actions in training plans, and exploits outer entanglements [Chrpa and McCluskey, 2012] as a heuristics for limiting the number of potential instances of macros.

Several works go in the opposite direction. Haslum and Jonsson (2000) proposed a method that identifies and removes "redundant actions" (i.e. actions whose effects can be achieved by sequences of other actions). Areces *et al.* (2014) proposed a technique for decomposing more complex operators into simpler ones.

3 Background

AI planning deals with finding a sequence of actions transforming the environment from an initial state to a desired goal state [Ghallab *et al.*, 2004].

In the classical (STRIPS) representation the environment is described by *predicates*. *States* are defined as sets of grounded predicates. We say that $o = (\mathit{name}(o), \mathit{pre}(o), \mathit{del}(o), \mathit{add}(o))$ is a *planning operator*, where $\mathit{name}(o) = \mathit{op\_name}(x_1, \ldots, x_k)$ ($\mathit{op\_name}$ is an unique operator name and $x_1, \ldots x_k$ are variable symbols (arguments) appearing in the operator) and $\mathit{pre}(o), \mathit{del}(o)$ and $\mathit{add}(o)$ are sets of (ungrounded) predicates with variables taken only from $x_1, \ldots x_k$ representing o's precondition, delete, and add effects respectively. *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 $(s \setminus \mathit{del}(a)) \cup \mathit{add}(a)$.

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, initial state and set of goal predicates. Given a planning problem, a *plan* is a sequence of actions such that their consecutive application starting in the initial state results in a state containing all the goal predicates.

3.1 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*. Formally, a macro $o_{i,j}$ is constructed by assembling planning operators o_i and o_j (in that order) in the following way (o_i and o_j may share some arguments)[1]:

- $\mathit{pre}(o_{i,j}) = \mathit{pre}(o_i) \cup (\mathit{pre}(o_j) \setminus \mathit{add}(o_i))$
- $\mathit{del}(o_{i,j}) = (\mathit{del}(o_i) \setminus \mathit{add}(o_j)) \cup \mathit{del}(o_j)$
- $\mathit{add}(o_{i,j}) = (\mathit{add}(o_i) \setminus \mathit{del}(o_j)) \cup \mathit{add}(o_j)$

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, or to escape local heuristic minima. Macros, however, have often a high number of instances and thus they might considerably increase the size of grounded problem representation and be memory demanding. Therefore, it is important that benefits of macros exploitation outweigh their drawbacks. This problem is known as the *utility problem* [Minton, 1988].

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

3.2 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 Barták, 2009; Chrpa and McCluskey, 2012]. Such a technique is especially useful for limiting the number of instances of macros [Chrpa *et al.*, 2014].

We say that an operator is *entangled by init* (resp. *entangled by goal*) with a predicate, if there exists a plan where all the operator's instances require (resp., produce) instances of the predicate that correspond to initial (resp., goal) ones.

In the BlocksWorld domain [Slaney and Thiébaux, 2001], the operator unstack is entangled by init with the predicate on, since *unstacking* blocks is necessary only from their initial positions. Similarly, the operator stack is entangled by goal with the predicate on, since *stacking* blocks is necessary only to their goal position.

Outer entanglements have been used as a reformulation technique, as they can be directly encoded into a domain and problem model. The way outer entanglements are encoded is inspired by one of their properties: given a static predicate p_s[2], an operator o is entangled by init with p_s if and only if $p_s \in \mathit{pre}(o)$ [Chrpa and Barták, 2009]. Operators involved in the outer entanglement relation with a non-static predicate are modified by putting a "static twin" of the predicate into the precondition. Instances of the "static twin" corresponding with initial or goal instances of the predicate are added into the initial state. For detailed description of the reformulation approach, see [Chrpa and McCluskey, 2012]

3.3 Plan Deordering and Block Decomposition

A *partially ordered plan* (POP) is a tuple $(\mathcal{A}, \prec)$, where $\mathcal{A}$ is the set of plan actions and $\prec$ is a strict partial order on $\mathcal{A}$. $\prec^+$ denotes the transitive closure of $\prec$. A *linearization* of $(\mathcal{A}, \prec)$ is a total ordering of the actions in $\mathcal{A}$ that respects $\prec$. A POP provides a compact representation for multiple linearizations. We assume that every ordering constraint, $a_i \prec a_j$, in $(\mathcal{A}, \prec)$ must have at least one necessary reason, denoted by $\mathrm{Re}(a_i \prec a_j)$, for not violating $a_i \prec a_j$. Violating $a_i \prec a_j$ causes an action precondition to be unsatisfied before its execution in some linearizations of $(\mathcal{A}, \prec)$. Necessary reasons (with respect to an atom m) are of four types: $\mathrm{PC}(m)$ (producer–consumer of m), $\mathrm{CD}(m)$ (consumer–deleter of m), $\mathrm{DP}(m)$ (deleter–producer of m), and $\mathrm{DK}(m)$ (deleter–knight of m). Note that an ordering constraint can have several associated reasons of the same type but referring to different atoms. $\mathrm{PC}(m) \in \mathrm{Re}(a_i \prec a_j)$ states that a_i produces an atom m that a_j consumes. This relation is usually called a *causal link* from a_i to a_j for m [McAllester and Rosenblitt, 1991], and denoted by a triple $\langle a_i, m, a_j \rangle$. A causal link $\langle a_i, m, a_j \rangle$ is threatened if there is

[2]A predicate is static if not present in effects of any operator

any possibility of m being deleted and there is no producer to reproduce it before the execution of a_j. $\mathrm{CD}(m)$ states that a_j deletes an atom m that a_i consumes. Therefore, unless a_j is ordered after a_i, m could be unsatisfied before the execution of a_i in some linearizations of $(\mathcal{A}, \prec)$. $\mathrm{DP}(m)$ states that a_i deletes an atom m that a_j produces, and that is consumed by some action a_k with $a_j \prec a_k$. Therefore, unless a_i is ordered before a_j, m could be unsatisfied before the execution of a_k in some linearizations of $(\mathcal{A}, \prec)$. Note that it is not necessary to order a producer and deleter if no step that may occur after the producer in the plan depends on the produced atom. Finally, $\mathrm{DK}(m)$ states that a_i deletes an atom m that a_j produces, and a_i threats a causal link $\langle a_x, m, a_y \rangle$ in some linearizations of $(\mathcal{A}, \prec)$ unless a_i is ordered before a_j. Hence, a_j acts as a white knight to $\langle a_x, m, a_y \rangle$.

The validity of a POP can be defined in two equivalent ways: (1) a POP is valid iff every linearisation of its actions is a valid sequential plan, under the usual STRIPS execution semantics; and (2) a POP is valid if every action precondition is supported by an unthreatened causal link.

A *block* [Siddiqui and Haslum, 2012] is a constituent coherent subplan, i.e., a subset of actions, that must not be interleaved with actions outside the block.

Definition 1. *Let $(\mathcal{A}, \prec)$ be a partially ordered plan. A* ***block*** *w.r.t. $\prec$ is a subset $b \subset \mathcal{A}$ of actions such that for any two actions $a, a' \in b$, there exists no action $a'' \in (\mathcal{A} - b)$ such that $a \prec^+ a'' \prec^+ a'$.*

Blocks, like ordinary actions, have preconditions, add, and delete effects that are a subset of the union of those of its constituent actions. This enables blocks encapsulating some effects and preconditions, reducing interference, and thus allowing more deordering of plans.

A *decomposition* of a plan into blocks is recursive, i.e., a block can be wholly contained in another. However, blocks cannot be partially overlapping. The semantics of a partially ordered block decomposed plan is defined by restricting its linearisations (for which it must be valid) to those that respect the block decomposition, i.e., that do not interleave actions from disjoint blocks.

Deordering converts a sequential plan into a POP, but the conventional deordering approach restricts the deordering to only the cases where individual actions are independent and thus non-interfering. *Block deordering* [Siddiqui and Haslum, 2012] eliminates that restriction by forming blocks which allows to remove some ordering constraints. It enables deordering in many cases where it is impossible in the standard interpretation of plans. Maximising such deordering helps to exhibit the plan structure more clearly.

The block deordering procedure [Siddiqui and Haslum, 2012] automatically finds a block decomposition of a plan that maximises deordering of a partially ordered plan. This procedure works in a check-and-remove fashion: Firstly, it checks the reasons (PC, CD, DP, and DK) behind every necessary ordering $a_i \prec a_j$ within the current plan structure, and forms two blocks b_i and b_j with the initial element a_i and a_j respectively. Then the blocks gradually expand in opposite directions picking steps one after another from the plan structure until those reasons (and newly added reasons) behind the ordering no longer exist (due to the encapsulation within the blocks) or the expansion has reached the boundary. If no reason is left at the end, the ordering is removed as well, and this process is repeated until all the necessary orderings have been checked or the allotted time is up. As a simple example, Figure 1(i) shows a sequential plan for a small BlocksWorld problem. This plan cannot be deordered into a conventional POP, because each action in plan has a reason to be ordered next to another. Block deordering, however, is able to break the ordering ($a_2 \prec a_3$) by removing the only reason PC(handempty) based on the formation of two blocks b_1 and b_2 as shown in (ii). Neither of the two blocks delete or add the atom "handempty" (though it is a precondition of both). This removes the interference between them, and allows the two blocks to be executed in any order but not in an iterleaving way. Therefore, the possible linearisations of the block decomposed partially ordered plan are only (a_1, a_2, a_3, a_4) and (a_3, a_4, a_1, a_2).

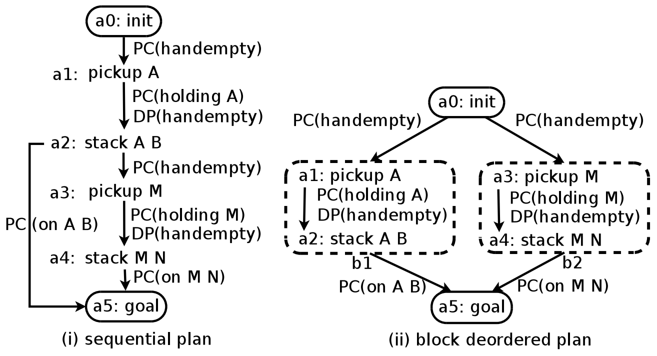

Figure 1: A sequential plan and its block deordering. Precedences are labelled with their reasons: producer–consumer (i.e., a causal link), denoted $\mathrm{PC}(p)$; deleter–producer, denoted $\mathrm{DP}(p)$; and consumer–deleter, denoted $\mathrm{CD}(p)$.

The nature of blocks is very similar to macros. Since block deordering tends to produce blocks that localise interactions between actions as much as possible, they often capture some coherent activities that can form a basis for useful macros. In addition, the block deordering algorithm returns also a justification for correctness of the block ordering, by labelling ordering constraints with their reasons (as mentioned above).

4 Macro-block Formulation

We extend the blocks to *macro-blocks* by considering different relations between them (as defined below) in order to provide larger subplans revealing important structural properties that often capture more complex activities that are frequently used in plans (e.g. mixing a cocktail and cleaning the shaker afterwards – as observed in the Barman domain).

Having a block deordered plan, we define relations of *immediate predecessor and immediate successor* of a block. Ordering between blocks is determined by any of the necessary reasons (PC, CD, DP, and DK) as stated in Section 3.3. Let $\mathrm{IP}(b)$ be a set of blocks being ordered immediately before b with respect to the transitively reduced ordering. Also, let $\mathrm{IS}(b)$ be a set of blocks that are ordered immediately af-

Algorithm 1 Computing extended blocks.

```
1: $B_{ext} \leftarrow B_{basic}$
2: while $\exists b_i, b_j \in B_{ext} : IP(b_j) = \{b_i\}, IS(b_i) = \{b_j\}$ do
3:     $B_{ext} \leftarrow B_{ext} \cup \{b_i \cdot b_j\} \setminus \{b_i, b_j\}$
4: end while
```

ter b with respect to the transitively reduced ordering. The *causal followers* of a producer a_p with respect to an atom m, $\mathrm{CF}_{\langle m,a_p\rangle}$, are a set of actions $\{a_p, a_j, ..., a_k\} \setminus \{a_\mathrm{I}, a_\mathrm{G}\}$ (a_I and a_G are special "init" and "goal" actions respectively, where a_I produces initial atoms, and a_G consumes the goal atoms) such that $\{\langle a_p, m, a_j\rangle, ..., \langle a_p, m, a_k\rangle\}$ are the causal links. For example, the atom $m =$ (holding A) in the block deordered plan of Figure 1 is associated with one causal link: $\langle a_1, m, a_2\rangle$, which form the causal followers $\mathrm{CF}_{\langle(\text{holding A}),a_1\rangle} = \{a1, a2\}$. The *causal follower blocks* involve a set of blocks related to the given causal link rather than a set of actions. In particular, the causal follower blocks for an atom m and its producer a_p are defined as $\mathrm{CFB}_{\langle m,a_p\rangle} = \{b \mid b \in B, \mathrm{CF}_{\langle m,a_p\rangle} \cap b \neq \emptyset\}$ (B is a set of blocks). For example, the causal follower blocks of the atom $m =$ (holding A) and its producer a_1 in the block deordered plan of Figure 1, are $\mathrm{CFB}_{\langle(\text{holding A}),a_1\rangle} = \langle\{b1\}\rangle$, since all the actions of $\mathrm{CF}_{\langle(\text{holding A}),a_1\rangle}$ are captured by the block $b1$.

Let B_{basic} be the set of blocks acquired by applying the block deordering approach. *Extended blocks* B_{ext} are generated iteratively from the basic blocks as depicted in Algorithm 1. In other words, extended blocks encapsulate non-branching subsequences of blocks, and therefore, can better capture some non-trivial activities. It should be noted that if no deordering is possible, B_{ext} will consist of a single (extended) block encapsulating the whole plan.

Macro-blocks are constructed according to the following eight rules we have specified. Each rule is applied over both sets of basic (B_{basic}) and extended blocks (B_{ext}). The macro-block construction rules are as follows:

$$\begin{array}{ll} \mathrm{R1} : \langle b\rangle & \mathrm{R5} : \langle \mathrm{R4}, \mathrm{R2}\rangle \\ \mathrm{R2} : \langle \mathrm{IP}(b), b\rangle & \mathrm{R6} : \langle \mathrm{R4}, \mathrm{R3}\rangle \\ \mathrm{R3} : \langle b, \mathrm{IS}(b)\rangle & \mathrm{R7} : \langle \mathrm{R4}, \mathrm{R4}\rangle \\ \mathrm{R4} : \langle \mathrm{IP}(b), b, \mathrm{IS}(b)\rangle & \mathrm{R8} : \mathrm{CFB}_{\langle m,a_p\rangle} \end{array}$$

The above rules can be divided into three groups, namely the primary rules (R1 to R4), the secondary rules (R5 to R7), and the causal rule (R8). Applied to all blocks in $B_{basic} \cup B_{ext}$, the primary, secondary, and causal rules can produce duplicates; of course, only unique macro-blocks are kept. The primary rules generate macro-blocks considering basic and extended blocks and their immediate predecessors and successors. Secondary rules chain exactly two macro-blocks generated by primary rules. The causal rule (R8) constructs macro-blocks based on causal links, specifically for each atom m and its producer a_p. These macro-blocks are not limited to any fixed length. The resultant macro-block, after applying any rule, may not capture some intermediate blocks, i.e., blocks that are ordered in between. Assume that $\mathrm{IS}(b_x) = \mathrm{IP}(b_y) = \{b_p, b_q\}$, and b_p and b_q are unordered. Applying R4 over b_p results in a set of blocks that do not contain the intermediately placed block b_q. We incorporate the intermediate blocks after applying each rule, i.e., b_q will be considered in a macro-block constructed from b_p by applying R4. If intermediate blocks are not considered, then macro-blocks do not represent consistent subplans.

Algorithm 2 The high-level design of BLOMA

```
1: $\Pi \leftarrow$ GenerateTrainingPlans()
2: $\mathcal{B} \leftarrow$ GenerateMacroBlocks($\Pi$)
3: $M \leftarrow$ ExtractMacros($\mathcal{B}$)
4: EnhanceDomain($M$)
5: $\Pi \leftarrow$ GenerateTrainingPlans()
6: FilterMacros($\Pi$)
7: LearnEntanglements($\Pi$)
```

5 Generating Macros from Macro-Blocks

Blocks aim to remove some of the ordering constraints (see Section 3.3), and thus reduce some interference between actions. Extended blocks put together blocks that have "no other alternative", in other words, blocks that are strictly consecutive in block deordered plans. Extended blocks are thus supposed to capture larger activities (e.g. shaking a cocktail and cleaning the shaker). Macro-blocks encapsulate both basic and extended blocks and relations between them. Considering relations between these blocks is somewhat related to the "chaining" approaches utilised by most of the existing macro learning techniques. In short, macro-blocks thus capture structural knowledge in form of coherent subplans that frequently occur in plans. These subplans can be linearised and thus can be exploited in the form of macros.

Algorithm 2 depicts the high-level implementation of BLOMA. Firstly, given a set of training planning problems (simpler but not trivial), training plans are generated. Then, BLOMA processes the training plans and generates macro-blocks as described in Section 4. Macro-blocks that consist of more than one action are linearised and then assembled into macros as described in Section 3.1. Macros that appear frequently in macro-blocks are added into the domain model. Then, training plans are re-generated using the macro enhanced domain model. Macros that do not appear or appear infrequently in the re-generated training plans are filtered out. In fact, we let the planner "decide" which macros are useful for it. The same strategy was used by MacroFF [Botea *et al.*, 2005]. As a final step of BLOMA macro-specific entanglements (i.e. those where only macros are involved) are learnt. Entanglements are learnt by checking whether for each operator and related predicates the entanglement conditions are satisfied in all the training plans. Some error rate (typically 10%) is allowed. For details, see [Chrpa and McCluskey, 2012]. As MUM does [Chrpa *et al.*, 2014] BLOMA uses entanglements for efficient pruning of unnecessary instances of macros.

Whether a macro candidate or macro is "frequent" is determined relatively. Let $f^b(m)$ be a number of occurrences of a macro candidate m in the set of macro-blocks $\mathcal{B}$. Then, a macro candidate $m \in M$ (M is the set of macros) is considered as frequent if $f^b(m) \geq p_b \max_{x \in M} f^b(x)$, where $0 < p_b \leq 1$. Similarly, let $f^p(o)$ be a number of occurrences of an operator or macro o in macro enhanced training plans Π. Then, a macro m is considered as frequent if $f^p(m) \geq p_p \max_{x \in O} f^p(x)$, where $0 < p_p \leq 1$ and O is a set of operators (including macros) defined in the planning domain. Clearly, setting p_b, p_p too high might cause filtering some useful macros out, while setting them too low might cause keeping useless macros.

Using basic blocks leads to generating a subset of macros that can be generated by using extended blocks, which often results in failing to generate any macro. Although exploiting extended blocks yields to a relatively small number of considered macro candidates, they often provide a basis for good quality macros. This is because extended blocks often capture complex single activities. Such macros are usually not generated by "chaining-based" approaches. Macro-blocks, on the other hand, provide a larger number of suitable macro candidates than extended blocks, since R2-R8 allow weaker forms of block chaining. R2-R8 in fact incorporate a variant of "chaining" approaches which might lead into generating similar macros as the existing techniques do.

Following the above observations, BLOMA works in a two-phased way. Initially, BLOMA tries to generate macros from extended blocks. If no macro is generated, then BLOMA uses macro-blocks to generate macros.

6 Experimental Analysis

We experimentally evaluated BLOMA in order to demonstrate how it improves against the original and MUM enhanced domain and problem models. We used all the domains from the learning track of IPC-7. The results are analysed in terms of providing insights into possible impact of generated macros to the planning process.

6.1 Learning

We generated 6 training problems for each domain. The training problems were rather simple but not trivial, so the plan length was mostly within 40-80 steps. For learning, we have used 4 state-of-the-art planners that accommodate various planning techniques: LAMA [Richter and Westphal, 2010], MpC [Rintanen, 2014], Probe [Lipovetzky *et al.*, 2014] and Mercury [Katz and Hoffmann, 2014]. For each domain, we considered that planner that generated the best quality (shortest) training plans. One plan was considered per each training problem. The parameters p_b and p_p were both set to 0.5. The learning process took from couple of seconds to couple of minutes, where generating training plans consume the majority of the learning time.

6.2 Comparison

We use IPC score as defined in the learning track of IPC-7 [Coles *et al.*, 2012]. For an encoding e of a problem p, $IPC(p, e)$ is 0 if p is unsolved in e, and $1/(1 + \log_{10}(T_{p,e}/T^*_p))$, where $T_{p,e}$ is the CPU-time needed to solve p in e and T^*_p is the smallest CPU-time needed to solve p in any considered encodings, otherwise. As in the learning track, the time limit was 15 minutes per problem. All the experiments were run on Intel Xeon 2.53 Ghz with 2GB of RAM, CentOS 6.5. Apart of the four planners considered for learning, we also used Yahsp3 [Vidal, 2014] and Bfs-f [Lipovetzky *et al.*, 2014].

	Coverage			Δ IPC	
	O	M	B	M	B
Barman					
Lama	0	-	30	-	+30.0
Mercury	23	-	30	-	+9.8
MpC	0	-	0	-	0.0
Probe	3	-	22	-	+19.7
Yahsp	0	-	11	-	+11.0
Bfs-f	30	-	30	-	-6.5
BlocksWorld					
Lama	23	-	25	-	+3.3
Mercury	19	-	8	-	-11.1
MpC	0	-	0	-	0.0
Probe	24	-	25	-	+3.5
Yahsp	28	-	22	-	-7.5
Bfs-f	0	-	10	-	+10.0
Depots					
Lama	0	2	2	+2.0	+2.0
Mercury	0	0	0	0.0	0.0
MpC	18	24	24	+8.6	+8.6
Probe	30	30	30	+4.2	+4.2
Yahsp	21	20	20	+1.0	+1.0
Bfs-f	4	21	21	+17.8	+17.8
Gripper					
Lama	0	30	30	+30.0	+30.0
Mercury	0	4	4	+4.0	+4.0
MpC	0	0	0	0.0	0.0
Probe	0	5	5	+5.0	+5.0
Yahsp	0	0	0	0.0	0.0
Bfs-f	0	0	0	0.0	0.0
Parking					
Lama	3	-	0	-	-3.0
Mercury	6	-	3	-	-3.1
MpC	5	-	0	-	-5.0
Probe	3	-	4	-	+1.2
Yahsp	0	-	4	-	+4.0
Bfs-f	5	-	5	-	-0.9
Rovers					
Lama	27	29	25	+3.6	-3.3
Mercury	24	26	29	+6.7	+9.5
MpC	5	5	5	-0.1	0.0
Probe	28	27	19	-1.0	-11.9
Yahsp	30	30	30	+0.6	-0.4
Bfs-f	0	0	0	0.0	0.0
Satellite					
Lama	3	26	18	+23.7	+15.7
Mercury	19	11	13	-10.7	-9.2
MpC	1	0	0	-1.0	-1.0
Probe	0	2	0	+2.0	0.0
Yahsp	16	27	16	+4.7	-6.8
Bfs-f	0	0	0	0.0	0.0
Spanner					
Lama	0	0	-	0.0	-
Mercury	0	0	-	0.0	-
MpC	30	30	-	+1.7	-
Probe	0	0	-	0.0	-
Yahsp	0	0	-	0.0	-
Bfs-f	0	0	-	0.0	-
TPP					
Lama	16	15	17	-2.0	+1.0
Mercury	19	16	20	-4.0	+2.7
MpC	9	11	20	+2.0	+14.0
Probe	12	14	17	+2.2	+7.1
Yahsp	30	30	30	-0.7	+0.4
Bfs-f	15	15	8	-0.7	-8.3

Table 1: Comparison between original (O), MUM (M) and BLOMA (B) on the IPC-7 learning track domains. "-" stands for "no macros found". Δ IPC stands for a difference of IPC score of the original and the corresponding macro enhanced encodings.

Table 1 presents the results of BLOMA in comparison to the original problem encodings and macros generated by MUM. BLOMA and MUM generated the same sets of macros in Depots and Gripper. By exploiting extended blocks, macros have been generated in Barman, BW, Rovers and TPP. In Spanner, no macro has been generated. Positive results have been achieved in Barman, Depots, Gripper and TPP. In the rest of domains the results were rather mixed.

Mixed results point to the fact that different planning techniques have often a different "response" to macros. This observation is, of course, not very surprising. Macros are often not supportive if the original problems are solved quickly (in a few seconds) since macros are more demanding in the preprocessing stage. Letting a planner learn macros for itself is, however, occasionally helpful, although in the Satellite domain, MpC learnt a good macro for itself. On the other hand, when training plans are of poor quality, generated macros are very poor as well. For example, in TPP, Probe learnt a very poor macro.

In Barman, the success of BLOMA rests in finding an 8-step long macro that captures an important activity – shaking a cocktail, pouring it into a shot and cleaning the shaker afterwards. In Gripper, BLOMA (as well as MUM) found a useful 3-step long macro that directly delivers an object from its initial to its goal location. In other domains, BLOMA found only 2-step macros capturing only partial activities (e.g. loading a truck). In TPP, BLOMA generated a "recursive" macro drive-drive that a bit surprisingly contributed considerably to MpC and Probe's performance.

Apart from Barman, we identified other domains, namely Scanalyzer, Storage and the IPC-2 version of Gripper, where BLOMA found longer macros. Most problems from these domains (all problems in the IPC-2 Gripper domain) are easy to solve, that is, the planners needed at most a few seconds. With higher pre-processing requirements for macros, there is no space for improvement, although the performance was rarely considerably worse. In Storage, the learnt macro helped Probe to solve 7 more problems, Bfs-f and LAMA to solve 2 more problems, however, the macro caused MpC to solve 6 less problems. In Scanalyzer, the learnt macro was specific for "2-cycle problems" [Helmert and Lasinger, 2010]. While considering the macro, MpC solved 1 more problem and Mercury has better performance on harder problems. On the contrary, Bfs-f solved 1 less problem.

6.3 Discussion

As discussed before, BLOMA is particularly useful in domains where longer macros capturing important activities can be identified. Traditional "chaining-based" approaches (e.g. MUM) often fail in these occasions. In other domains, BLOMA performs with mixed results.

Number of instances of macros might cause issues with memory consumption as well as might make pre-processing more difficult. A typical example is the Parking domain that represents a combinatorial problem of re-arranging cars in a parking lot. Therefore, macros despite being frequently used in plans might have detrimental effect on performance. From this perspective, approaches such as MUM that consider only macros with a relatively small number of instances are efficient. However, as showed in this paper, they are not able to generate longer macros that despite a larger number of instances can be very beneficial.

On the other hand, we have observed that in some cases macros have arguments that are not necessary to be kept explicitly, so the number of instances might be reduced considerably. In particular, if a state of some object remains the same, i.e., no predicates containing this object are added or deleted, then it is not necessary to keep this object as an argument of the macro. For example, having a macro that represents an activity of delivering a package by a truck and returning the truck to the place of package's origin. The truck in fact does not change its state after applying the macro. Of course, some truck must be in the place of package's origin which has to be reflected in macro's precondition. It can be done by using existential quantifiers. Although they are supported in PDDL, many planning engines do not support such a feature.

Impact of particular macros depends on a planning technique exploiting them. In Depots, macros bypass situations where a crate is held by a hoist. It is well known that a hoist can hold at most one crate at time, however, delete-relaxed heuristics ignores such a constraint which might lead into having many local minima in heuristic landscape [Hoffmann, 2011]. Macros that reduce the complexity of Planning Graph (e.g. smaller number of layers, less mutexes) seem to be beneficial for techniques such as those incorporated in MpC as observed in Depots and TPP. We believe that classifying macros according to their features will be helpful for selecting planner-specific macros that will be tailored for a given planning technique.

7 Conclusions

In this paper, we presented BLOMA that learns planner-independent macros from macro-blocks, which can be extracted from training plans, encapsulating constituent coherent subplans. Such an approach is beneficial especially for domains (e.g. Barman), where an important activity repeatedly applied in plans can be encapsulated by (longer) macros. We have shown empirically that BLOMA can considerably improve the performance in many cases.

There are two major limitations. Firstly, a higher number of macros' instances might be detrimental to the planning process. However, all the arguments of macros do not have to be always explicitly defined. To address this we have to use existential quantifiers in macros' preconditions, however, such a feature is not widely supported by planning engines. Alternatively, we might learn HTN methods rather than macros. Secondly, some macros are not supportive for certain planning techniques (e.g. they might "jump" into local heuristics minima). Hence, classifying macros according to their features might reveal what kind of macros has positive/negative impact on certain planning techniques.

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. NICTA is funded by the Australian

Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program

References

[Alhossaini and Beck, 2013] Maher A. Alhossaini and J. Christopher Beck. Instance-specific remodelling of planning domains by adding macros and removing operators. In *Proceedings of SARA*, pages 16–24, 2013.

[Areces *et al.*, 2014] Carlos Areces, Facundo Bustos, Martín Dominguez, and Jörg Hoffmann. Optimizing planning domains by automatic action schema splitting. In *Proceedings of ICAPS*, 2014.

[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.

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

[Chrpa *et al.*, 2014] Lukás Chrpa, Mauro Vallati, and Thomas Leo McCluskey. Mum: A technique for maximising the utility of macro-operators by constrained generation and use. In *Proceedings of ICAPS*, pages 65–73, 2014.

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

[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] Amanda Coles, Andrew Coles, Angel G. Olaya, Sergio Jiménez, Carlos L. Lòpez, Scott Sanner, and Sungwook 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.

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

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

[Haslum and Jonsson, 2000] Patrik Haslum and Peter Jonsson. Planning with reduced operator sets. In *Proceedings of AIPS*, pages 150–158, 2000.

[Helmert and Lasinger, 2010] Malte Helmert and Hauke Lasinger. The scanalyzer domain: Greenhouse logistics as a planning problem. In *ICAPS*, pages 234–237, 2010.

[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.

[Hoffmann, 2011] J. Hoffmann. Analyzing search topology without running any search: On the connection between causal graphs and h+. *Journal Artificial Intelligence Research (JAIR)*, 41:155–229, 2011.

[Katz and Hoffmann, 2014] Michael Katz and Joerg Hoffmann. Mercury planner: Pushing the limits of partial delete relaxation. In *The Eighth IPC. Description of Participant Planners of the Deterministic Track*, pages 43–47, 2014.

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

[Lipovetzky *et al.*, 2014] Nir Lipovetzky, Miquel Ramirez, Christian Muise, and Hector Geffner. Width and inference based planners: Siw, bfs(f), and probe. In *The Eighth IPC. Description of Participant Planners of the Deterministic Track*, pages 6–7, 2014.

[McAllester and Rosenblitt, 1991] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proc. 9th National Conference on Artificial Intelligence*, 1991.

[Minton, 1988] Steven 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, 2014] Jussi Rintanen. Madagascar: Scalable planning with sat. In *The Eighth IPC. Description of Participant Planners of the Deterministic Track*, pages 66–70, 2014.

[Siddiqui and Haslum, 2012] F.H. Siddiqui and P. Haslum. Block-structured plan deordering. In *AI 2012: Advances in Artificial Intelligence (Proc. 25th Australasian Joint Conference*, volume 7691 of *LNAI*, pages 803–814, 2012.

[Siddiqui and Haslum, 2013] Fazlul Hasan Siddiqui and Patrik Haslum. Plan quality optimisation via block decomposition. In *Proceedings of IJCAI 2013*, 2013.

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

[Vidal, 2014] Vincent Vidal. Yahsp3 and yahsp3-mt in the 8th international planning competition. In *The Eighth IPC. Description of Participant Planners of the Deterministic Track*, pages 64–65, 2014.