Improving Domain-Independent Planning via Critical Section Macro-Operators

Lukáš Chrpa
Faculty of Electrical Engineering
Czech Technical University in Prague &
Faculty of Mathematics and Physics
Charles University in Prague

Mauro Vallati
School of Computing and Engineering
University of Huddersfield

Abstract

Macro-operators, macros for short, are a well-known technique for enhancing performance of planning engines by providing "short-cuts" in the state space. Existing macro learning systems usually generate macros from most frequent sequences of actions in training plans. Such approach priorities frequently used sequences of actions over meaningful activities to be performed for solving planning tasks.

This paper presents a technique that, inspired by resource locking in critical sections in parallel computing, learns macros capturing activities in which a limited resource (e.g., a robotic hand) is used. In particular, such macros capture the whole activity in which the resource is "locked" (e.g., the robotic hand is holding an object) and thus "bridge" states in which the resource is locked and cannot be used. We also introduce an "aggressive" variant of our technique that removes original operators superseded by macros from the domain model. Usefulness of macros is evaluated on several state-of-the-art planners, and a wide range of benchmarks from the learning tracks of the 2008 and 2011 editions of the International Planning Competition.

Introduction

Automated Planning, in a nutshell, is about finding a sequence of actions whose application in an initial state of the environment leads to a desired goal state (Ghallab, Nau, and Traverso 2004). Whereas a lot of effort has been traditionally given to developing efficient planning engines, usually based on heuristic search (Bonet and Geffner 2001), another line of research focuses on increasing efficiency of the planning process by reformulating the domain knowledge, to obtain models that are more amenable for automated reasoners.

A very well-known reformulation approach is the generation of macro-operators, macros for short, that encapsulate sequences of (original) planning operators. Macros are encoded as ordinary planning operators and, hence, they can be added into domain models such that standard planning engines can straightforwardly take advantage of them. Macros, informally speaking, provide "short-cuts" in the state space and, consequently, planning engines can generate plans in less number of steps. This comes with the cost of increased branching factor since macros often have much more instances than ordinary operators and thus their use might introduce additional overheads as well as larger memory requirements. Although in theory the use of macros might reduce complexity of planning (Korf 1985), in practice macros are considered if their use is frequent (Hofmann, Niemueller, and Lakemeyer 2017), their number of instances is small (Chrpa, Vallati, and McCluskey 2014), or they address weaknesses of a specific planner (Coles, Fox, and Smith 2007).

In this paper, we introduce *Critical Section Macros*. Being inspired by resource locking in critical section in parallel computing, these macros capture whole activities in which a resource is used. For example, a robotic hand manipulates with objects (e.g. blocks in BlocksWorld, or shaker in Barman). When the robotic hand grasps an object it becomes "locked", i.e., no other object can be grasped by that hand, until the hand releases the object it holds. Hence, Critical Section Macros aim to capture sequences of operators such that the first operator locks a resource (e.g. a robotic hand grasps an object), the last operator releases the resource (e.g. the robotic hand drops the object), and the intermediate operators, if any, uses the resource (e.g. shaking a cocktail). From the technical perspective, Critical Section Macros "bridge" states in which the resource is locked and cannot be used. This is thought to be particularly beneficial for techniques that exploit delete-relaxation (Hoffmann and Nebel 2001) as they tend to incorrectly assume that a resource can be used by multiple activities (e.g. a robotic hand holding multiple objects) and thus provide largely incorrect heuristic estimations. As additional contributions, we illustrate how the proposed reformulation approach can be exploited in an "aggressive" variant, that removes the elementary operators that are included in the macros, and can be combined with more traditional techniques for generating macros. The usefulness of Critical Section Macros, in all the variants depicted above, is evaluated on several state-of-the-art planners, and a wide range of benchmarks from learning tracks of the International Planning Competition.

Related Work

Using macros dates back to 1970s and 1980s. REFLECT (Dawson and Siklóssy 1977) builds macro-operators from pairs of primitive operators that can be successively applied and share at least one argument. MOR-

RIS (Minton 1988) learns macro-operators from parts of plans appearing frequently (S-macros) or being potentially useful despite having low priority (T-macros). Macro Problem Solver (Korf 1985) learns macros for particular non-serializable sub-goals (e.g. in Rubik's cube).

Recent planner-independent techniques aim at improving performance of any standard planner. MacroFF (Botea et al. 2005) generates macros according to several pre-defined rules (e.g., the "locality rule") that apply on adjacent actions in training plans. Wizard (Newton et al. 2007) learns macros from training plans by exploiting genetic programming. Alhossaini and Beck (2013) selects problem-specific macros from a given pool of macros (hand-coded or generated by another technique). Dulac et al. (2013) exploits n-gram algorithm to analyze training plans to learn macros. DBMP/S (Hofmann, Niemueller, and Lakemeyer 2017) applies Map Reduce for learning macros from a larger set of training plans. CAP (Asai and Fukunaga 2015) exploits component abstraction (introduced by MacroFF) for generating sub-goal specific macros.

MUM (Chrpa, Vallati, and McCluskey 2014) exploits "outer entanglements" (Chrpa and McCluskey 2012) as a heuristics for generating macros with limited number of instances. BloMa (Chrpa and Siddiqui 2015) exploits block deordering (Siddiqui and Haslum 2012) for generating possibly longer macros. Our "critical section" macros share some characteristics with "block" macros generated by BloMa. BloMa, however, initially generates a large pool of macros that is later reduced by applying (strict) frequency requirements.

Classical Planning

The classical (STRIPS) representation considers static and fully observable environment, and deterministic and instantaneous action effects. The environment is described by first-order logic *predicates* defined as $p = \mathit{pred\_name}(x_1, \dots, x_n)$, where *pred_name* is a unique predicate name and $x_1, \dots x_n$ are variable symbols. *States* are defined as sets of *atoms* (grounded predicates whose variable symbols are substituted with constants - problem-specific objects). We say that $o = (\mathit{name}(o), \mathit{pre}(o), \mathit{del}(o), \mathit{add}(o))$ is a *planning operator*, where $\mathit{name}(o) = \mathit{op\_name}(x_1, \dots, x_k)$ (*op_name* is an unique operator name and $x_1, \dots x_k$ 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 $x_1, \dots x_k$ 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* $D = (P, O)$ is specified by a set of predicates (P) and a set of planning operators(O). A *planning task* $\Pi = (D, I, G)$ is specified via a domain model (D), initial state (I) and set of goal atoms (G). 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 atoms.

Given a planning task Π, we say that a state s' is *reachable* from a state s if and only if there exists a sequence of actions such that their consecutive application starting in s results in s'. We say that an action a_i is an *achiever* for an action a_j if an only if $\mathit{add}(a_i) \cap \mathit{pre}(a_j) \neq \emptyset$. We also say that actions a_i and a_j are *independent* if and only if $\mathit{del}(a_i) \cap (\mathit{pre}(a_j) \cup \mathit{add}(a_j)) = \emptyset$ and $\mathit{del}(a_j) \cap (\mathit{pre}(a_i) \cup \mathit{add}(a_i)) = \emptyset$.

Macro-operators

Macros represent sequences of (ordinary) planning operators. Advantageously, macros can be encoded in the same form as planning operators (i.e., having a precondition, add and delete effects). Hence, macros can be added into a domain model and thus can be exploited in a *planner independent* way (e.g. encoded in PDDL).

Formally, a macro $o_{i,j}$ is constructed by assembling planning operators o_i and o_j (in that order) as follows. Let Φ and Ψ be mappings between variable symbols (we need to appropriately rename variable symbols of o_i and o_j to construct $o_{i,j}$).

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

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

For a macro to be *sound*, no instance of $\Phi(o_i)$ can delete an atom required by a corresponding instance of $\Psi(o_j)$, otherwise they cannot be applied consecutively. Whereas it is obvious that if a predicate deleted by $\Phi(o_i)$ (and not added back) is the same (both name and variable symbols) as a predicate in the precondition of $\Psi(o_j)$ then the macro $o_{i,j}$ is unsound, another source of macro unsoundness is often not being even considered in literature. For example, in the Blocks-World domain, a macro pickup-stack(?x ?y) that has (clear ?x)(ontable ?x)(clear ?y)(handempty) in its precondition can be instantiated into pickup-stack(A A) that is applicable if (clear A)(ontable A)(handempty) is true in some state. However, actions (pickup A) and stack(A A) cannot be applied consecutively because (pickup A) deletes (clear A) which is required by stack(A A). By generalizing this observation we can see that if some (different) variable symbols are substituted by the same constants, a macro might become unsound. To avoid such cases, a constraint requiring different instantiation of affected variable symbols is added into macro's precondition (e.g. (not (= ?x ?y)) is added into pickup-stack(?x ?y)'s precondition).

Critical Section Macros

In parallel computing, critical sections are used to regulate the access to resources, to guarantee the integrity of the overall system by avoiding situations where many different processes are concurrently modifying a resource. To prevent other processes (or threads) to access the resource while it is in use, the resource is locked at the beginning of the critical section, then the required operations are made with the resource by the process that is locking it, and then –at the end of critical section– the resource is released and is therefore available for other processes.

In planning, we can observe that some subsequences of actions in plans replicate the underlying structure of critical section, i.e., locking a resource, using it, and releasing it. In Blocks-World, the robotic hand can be seen as a resource. When the robotic hand picks-up or unstacks a block it becomes "locked", that is, no other block can be carried by the robotic hand at that time. When the robotic hand stacks or puts-down the block it is holding, then the hand is "released", that is, it can be used to manipulate other blocks. A more complicated example can be found in the Barman domain, in which a robotic barman prepares cocktails. Here, a critical section activity, for instance, involves grabbing a shaker (locking robot's hand), shaking a cocktail, putting the cocktail to a shot, cleaning the shaker before putting it back on the table (releasing robot's hand).

Technically speaking, a free resource, as well as a locked resource, is represented by corresponding predicates. For example, (handfree) represents a free resource (robotic hand) while (holding ?x) represents a locked resource (robotic hand carrying a block). Also, corresponding instances of these predicates must be mutually exclusive (mutex for short), i.e., no resource can be both free and locked at the same time.

Definition 1. *Let* Π *be a planning task and* p *and* q *be predicates defined in the domain model of* Π*. Let* Φ, Ψ *be substitutions mapping variable symbols to variable symbols. We say that* p *and* q *with respect to* Φ *and* Ψ *are* ***mutex*** *in* Π *if and only if for all ground instances of* $\Phi(p)$ *and* $\Psi(q)$ *it is the case that they are not both true in all reachable states from the initial state of* Π*.*

Having mutex predicates p and q that represent a free and locked resource respectively, is a necessary condition for recognising "critical section" activities. Also, arguments of q must be a superset of arguments of p (they can have the same set of arguments). Arguably, since q represents a locked resource, it might contain additional arguments referring to why the resource is locked (e.g. a hand holding an object), however, it cannot contain less arguments (in order to recover a corresponding instance of p after releasing the resource).

Locking and releasing resources is done by specific planning operators. A planning operator that deletes p and adds a corresponding variant of q, a *locker*, locks a given resource. Analogously, a planning operator that deletes q and adds a corresponding variant of p, a *releaser*, releases (unlocks) the given resource. This idea is formalised as follows.

Definition 2. *Let* $\Pi = (D, I, G)$ *be a problem instance and* $D = (P, O)$ *be a domain model. Let* $p, q \in P$ *be predicates such that* p *and* q *with respect to substitutions* Φ, Ψ *are mutex in* Π *and* $args(\Phi(p)) \subseteq args(\Psi(q))$*. We say that an operator* $o \in O$ *is a* p, q***-locker*** *if* $\Phi(p) \in del(\Theta(o))$ *and* $\Psi(q) \in add(\Theta(o))$ *(*Θ *is a renaming substitution). We also say that* $o' \in O$ *is a* p, q***-releaser*** *if* $\Phi(q) \in del(\Theta'(o'))$ *and* $\Psi(p) \in add(\Theta'(o'))$ *(*Θ' *is a renaming substitution). We also say that* $o'' \in O$ *is a* p, q***-user*** *if* $\Phi(q) \in pre(\Theta''(o'')) \setminus (del(\Theta''(o'')) \cup add(\Theta''(o'')))$ *(*Θ'' *is a renaming substitution).*

Algorithm 1 Learning Critical Section Macros from training plans

```
1:  $cs \leftarrow \{(p, q, O_l, O_r) \mid o_l \in O_l$ is a $p, q$-locker; $o_r \in O_r$ is a $p, q$-releaser$\}$
2:  for each $\langle a_1, \ldots, a_n \rangle$ in Training_Plans do
3:      $lr\_pairs \leftarrow \{(a_l, a_r, p_g, q_g) \mid (p, q, O_l, O_r) \in cs; r > l; a_l, a_r, p_g, q_g$ are instances of $o_l \in O_l, o_r \in O_r, p, q$ respectively; $p_g \in del(a_l) \cap add(a_r); q_g \in add(a_l) \cap del(a_r)\}$
4:      for each $(a_l, a_r, p_g, q_g) \in lr\_pairs$ do
5:          $in\_ma \leftarrow \{a_k \mid l < k < r; q_g \in pre(a_k)\} \cup \{a_l, a_r\}$
6:          $out\_ma \leftarrow \{a_k \mid l < k < r; a_k \notin in\_ma\}$
7:          ConsiderDependent($in\_ma$,$out\_ma$)
8:          if no "gluing" action added extra argument then
9:              $o^m \leftarrow$CreateMacro($in\_ma$)
10:             ConsiderGoalAchieving($o^m$)
11:             AddMacro($mcr\_db$,$o^m$)
12:         end if
13:     end for
14: end for
15: FilterUnderrepresentedMacros($mcr\_db$)

16: function CONSIDERDEPENDENT($in\_ma$,$out\_ma$)
17:     while $\exists k : \{i \mid a_i \in out\_ma; i < k\} = \emptyset$ and $\{a_i \mid a_i \in in\_ma; i < k; a_i$ is an achiever for $a_k$ or $a_i$ is not independent with $a_k\} = \emptyset$ do
18:         $out\_ma \leftarrow out\_ma \setminus \{a_k\}$
19:     end while
20:     while $\exists k : \{j \mid a_j \in out\_ma; j > k\} = \emptyset$ and $\{a_j \mid a_j \in in\_ma; j > k; a_k$ is an achiever for $a_j$ or $a_j$ is not independent with $a_k\} = \emptyset$ do
21:         $out\_ma \leftarrow out\_ma \setminus \{a_k\}$
22:     end while
23:     $in\_ma \leftarrow in\_ma \cup out\_ma$
24: end function
```

The above definition considers one phase single locks. That is, that resources are locked/released by one operator (rather than their sequence) and only one lock (i.e., instance of q) is acquired. The definition can be extended to consider multiple-phase and multiple locks, however, for practical reasons (i.e., such situations are very uncommon if any in standard benchmarks used in the international planning competition) and for the sake of clarity we resort to the cases covered by Definition 2.

Constructing Critical Section Macros

The rationale behind the Critical Section Macros is to *bridge* resource use with a single macro. Delete-relaxation (Hoffmann and Nebel 2001) is a popular approach for many state-of-the-art planning engines. Delete-relaxation, roughly speaking, ignores delete effects of planning operators. Consequently, mutex relations between grounded predicates are also ignored. In situations of resource locking, the difference between delete-relaxed approximation and reality can be considerably large, hence undermining the usefulness of the heuristic evaluation. For example, in delete-relaxation, the robotic hand can hold all the blocks at the same time. As it is apparent, such discrepancies can easily cause local minima on landscape of heuristic functions based on delete-relaxation (Hoffmann 2011).

Critical Section Macros encapsulate sequences of operators such that they start with p, q-lockers and end with p, q-releasers. For longer sequences, p, q-users and "gluing" operators are present in between. For example, a macro pickup-stack is a Critical Section Macro consisting of only a locker (pickup) and a releaser (stack). In Barman, for example, a macro grasp-fillshot-leave is a Critical Section Macro consisting of a locker (grasp), a releaser (leave) and a user (fillshot). In Gripper, a macro pick-move-drop is a Critical Section Macro consisting of a locker (pick), a releaser (drop) and a gluing operator (move).

Algorithm 1 describes the method for learning Critical Section Macros from training plans. Quadruples (p, q, O_l, O_r) (Line 1) are determined by considering whether each operator deleting p (or q) adding a corresponding variant of q (or p), in other words, each operator having p or q in its effects is either a p, q-locker or p, q releaser. Also, if corresponding instances of p and q are not simultaneously present in initial states of training tasks (and testing tasks), then p and q are mutex (with respect to corresponding substitutions). For each training plan, we determine all possible locker/releaser pairs (a_l, a_r) with corresponding instances of the involved predicates (p_g, q_g) (Line 3). Then, we iterate through the locker/releaser pairs (Lines 4–13). Besides a_l (p_g, q_g-locker) and a_r (p_g, q_g-releaser) we consider p_g, q_g-users into a possible macro (Line 5). Other actions placed in between a_l and a_r are checked whether they can be moved away (either before a_l or after a_r). This is done by the ConsiderDependent function. The idea of how intermediate actions can be moved away is based on the observation that: if for two adjacent actions a, a' in a plan (in this order), it is the case that a and a' are independent and a is not an achiever for a', then a, a' can be swapped without compromising the correctness of the plan (similar approach has been used by MUM (Chrpa, Vallati, and McCluskey 2014)). Those actions that cannot be moved away are *gluing* actions. If none of the gluing actions introduces an extra argument, in other words, does not have to reason with additional objects than the locker, releaser and the user actions (Line 8), then the action sequence in consideration can be considered as a macro (Line 9). After the macro is created, it is checked whether it is *goal achieving*, i.e., whether some of its add effects are goal atoms. Goal achieving macros, achieving instances of p that are present in the goal, extend the ordinary macros by introducing a supplementary static predicate p^G, added into macro's precondition, that has the same variable symbols as p in the macro's add effects but a different name. Also, a problem-instance is modified such that for each instance of $p \in G$, a corresponding instance of p^G is added to I. Noteworthy, such a concept is analogous to the use of outer entanglements (Chrpa and McCluskey 2012).

Macros that are "underrepresented", i.e., their number in the "macro database" is below a specified threshold are filtered out. Underrepresented macros, in a Machine Learning terminology, are noise in training data. That are, for example, problem-specific macros that do not generalize for a class of planning problems, or macros capturing peculiarities in training plans. Such macros are very unlikely to be beneficial. Other macro learning techniques such as MacroFF (Botea et al. 2005) or MUM (Chrpa, Vallati, and McCluskey 2014) also eliminate underrepresented macros from the same reason.

For macros that are assembled from the same operators but differ by being goal achieving, the most restrictive macro (achieving most goals) is only considered.

Aggressive Approach

Adding (sound) macros into a domain model does not compromise completeness. On the other hand, the size of the (grounded) representation can considerably grow as macros often have more instances than ordinary operators, due to the larger number of arguments. Consequently, planners might suffer with increased memory requirements and with extra burden in pre-processing.

To mitigate such an issue, original operators that are effectively replaced by macros can be removed from the domain model. Critical Section Macros have a good potential to replace original operators that operate with particular resources because the activities these macros represent have to be usually performed either as whole or not at all. For example, in Gripper, a Critical Section Macro pick-move-drop replaces original operators pick and drop (unless some robot initially holds some ball or it is required that some robot holds some ball in a goal state).

The aggressive version of our Critical Section Macro approach consists of the following steps:

1. Generate Critical Section Macros (Algorithm 1) and add them into the domain model.
2. Remove lockers and releasers of each macro from the domain model.
3. Generate plans for the training tasks with the modified domain model.
4. If some task cannot be solved, then fail (removed original operators are necessary).
5. Otherwise analyse the plans and eventually remove also those operators whose instances are never used in these plans.

The aggressive approach can compromise completeness as it can remove original operators that might be necessary to solve some (non-training) tasks. On the other hand, the aggressive approach by removing original operators can (sometimes considerably) reduce the size of the representation. In the Gripper example, removing the pick and drop operators prunes out states in which a ball is held by a robotic gripper and thus considerably reduces the size of (grounded) representation. The risk of making a task unsolvable can be alleviated by incorporating aggressive approaches into portfolios containing conservative components (e.g., the original model, a model with macros but containing all original operators).

Combination with other Approaches

Critical Section Macros focus on capturing activities in which a resource is locked. Other approaches consider different criteria for macro generation such as frequency of operators' "consecutivity" or the possible number of macros' instances. In particular, "chaining" approaches such as MacroFF (Botea et al. 2005) or MUM (Chrpa, Vallati, and McCluskey 2014), which construct macros iteratively, have a good potential to complement our approach as they can possibly chain the activities into longer and more useful macros.

Combining Critical Section Macros with other approaches (e.g., MUM) can be done straightforwardly. Critical Section Macros can be generated, as described in the previous sections, and then added to the domain model. Such enhanced domain model is then used for the training of a different macro generation approach. In fact, for the other macro learning approach, Critical Section Macros can be considered as original operators, and will be treated as original operators. On the other hand, as some macro learning approaches perform filtering of unpromising macros it might be useful to consider Critical Section Macros as macros in certain occasions. In particular, MUM filters out macros that are replaced by longer macros (e.g. a move-drop macro can be replaced by the pick-move-drop macro). Also, MUM uses "entanglements" to eliminate possibly unpromising instances of macros. For such occasions, we consider Critical Section Macros as macros. In all the remaining cases, Critical Section Macros are considered as ordinary operators, so they cannot be filtered out from different reasons.

Experimental Results

The purpose of this experimental analysis is to i) evaluate planners' performance on Critical Section macros as well as their combination with MUM (both conservative and aggressive versions), ii) compare them against related state-of-the-art techniques, MUM (Chrpa, Vallati, and McCluskey 2014) and BloMa (Chrpa and Siddiqui 2015) and iii) analyse impact of quality of training plans on generated macros utility . We considered domains from the learning track of IPC 2008 and 2011.

We have selected 6 state-of-the-art planning engines, according to their results in recent IPCs, and to the exploited planning techniques, namely: *LAMA* (Richter and Westphal 2010), *Probe* (Lipovetzky et al. 2014), *MpC* (Rintanen 2014), *Yahsp3* (Vidal 2014), *FDSS 2018* (Seipp and Röger 2018) and *Dual BFWS* (Lipovetzky et al. 2018).

Evaluation Metrics. Three metrics were used to evaluate planners' performance, namely *coverage* (number of solved problems), *PAR10 score* and *IPC quality score*. For each testing task time limit of 900 seconds and memory limit of 4 GB is applied (as in the learning tracks of IPCs). All the experiments were conducted on Intel Xeon E5 2.0 Ghz, Debian 9.

Penalised Average Runtime (PAR10) score is a metric usually exploited in machine learning and algorithm configuration techniques. This metric trades off coverage and runtime for solved problems: if a planner p solves a problem instance Π in time $t \leq T$ ($T = 900$s in our case), then $\mathit{PAR10}(p, \Pi) = t$, otherwise $\mathit{PAR10}(p, \Pi) = 10T$ (i.e., 9000s in our case).

IPC quality score is defined as in the learning track of IPC-7 (Coles et al. 2012) as follows. For an encoding e of a problem instance Π, $\mathit{IPC}(\Pi, e)$ is 0 if Π is unsolved in e, and $(m^*_{\Pi,e}/m_\Pi))$, where $m_{\Pi,e}$ is the cost of the plan of Π in e and m^*_Π is the smallest cost of the plan of Π in any considered encodings, otherwise.

Learning. We have considered two methodologies, one that have been used by MUM (Chrpa, Vallati, and McCluskey 2014) and one that have been used by BloMa (Chrpa and Siddiqui 2015). For both methodologies, we considered 6 training tasks per each domain such that their plan length was mostly within 40-80 actions[1]. Both methodologies consider one training plan per a training task.

The MUM methodology uses the same planner for generating training plans as for solving testing tasks. In other words, planners learn macros for themselves (and not for other planners). This methodology follows an intuition that most promising knowledge for a given planner can be extracted by analysing its outputs (plans).

The BloMa methodology, in contrast, selects a planner which generates, for a given domain, best quality training plans (e.g. the shortest plans). This methodology follows an intuition that good quality training plans yield to most promising knowledge for all planners.

The threshold for "underrepresented" macros was set to 6 (as the number of training tasks). The learning process took at most several seconds.

Results. The results for domains in which Critical Section Macros were generated are summarised in Table 1 and 2 for the MUM learning methodology (i.e., a planner learns for itself) and the BloMa learning methodology (i.e., considering best quality training plans), respectively. We can observe that macros learnt by the BloMa methodology perform better across the considered domains and planners. For example, in Parking, Critical Sequence Macros generated from poor quality training plans capture meaningless activities (e.g. moveCarToCar-moveCarToCar) while no macros were generated from the best quality training plans. Intuitively, better quality training plans carry better information that can be exploited by a range of planning engines.

The conservative variant of Critical Section Macros outperforms BloMa (in PAR10) in about 72% of cases considering the MUM learning methodology while in about 63% of cases considering the BloMa learning methodology. With regards to MUM, the results are mixed, in about 50% of cases the conservative variant of Critical Section Macros outperforms MUM (in both methodologies). Combination of Critical Section Macros and MUM (the conservative version) outperforms both MUM and Critical Section Macros in about 60% of cases (overall). The aggressive versions outperform the corresponding conservative versions in about 90% of cases in which the aggressive versions generated

[1]For the IPC 2011 domains, we used provided problem generators while for the IPC 2008 domains, we selected the tasks from the provided sets of "bootstrap" tasks.

Planner	Coverage							PAR10							IPC Quality						
	O	M	B	C	CM	AC	ACM	O	M	B	C	CM	AC	ACM	O	M	B	C	CM	AC	ACM
barman																					
lama	2	-	19	30	30	30	30	8428	-	3653	385	8.6	11	1.8	1.7	-	18.9	26.4	29.9	29.9	29.9
probe	2	-	7	2	3	24	23	8427	-	7022	8455	8136	2104	2379	1.4	-	4.8	1.7	2.6	23.9	22.8
MpC	0	-	0	0	0	0	0	9000	-	9000	9000	9000	9000	9000	0.0	-	0.0	0.0	0.0	0.0	0.0
yahsp	0	-	-	0	0	0	0	9000	-	-	9000	9000	9000	9000	0.0	-	-	0.0	0.0	0.0	0.0
BFWS	0	-	0	3	0	12	30	9000	-	9000	8148	9000	5606	1.2	0.0	-	0.0	2.8	0.0	12.0	29.2
FDSS	20	-	24	30	30	30	30	3234	-	2164	330	166	16	2.1	17.5	-	23.9	29.3	29.3	29.3	29.3
bw																					
lama	28	29	0	28	29	28	28	681	411	9000	711	379	617	616	24.5	22.2	0.0	22.1	23.4	23.0	23.0
probe	25	-	30	30	30	30	30	1679	-	156	215	195	0.4	0.3	21.4	-	28.4	27.2	27.1	27.5	29.9
MpC	0	-	0	30	30	11	18	9000	-	9000	177	173	5700	3600	0.0	-	0.0	12.3	28.7	3.7	16.7
yahsp	27	24	22	26	26	30	30	948	1833	2437	1239	1227	0.1	0.1	5.9	19.6	16.4	22.4	20.3	27.1	27.2
BFWS	3	-	0	0	0	0	0	8106	-	9000	9000	9000	9000	9000	3.0	-	0.0	0.0	0.0	0.0	0.0
FDSS	22	20	21	19	22	30	30	2506	3104	2780	3387	2492	11	11	18.0	14.9	14.7	14.7	15.1	23.4	23.4
depots																					
lama	0	-	0	2	0	30	30	9000	-	9000	8417	9000	0.4	0.3	0.0	-	0.0	1.4	0.0	29.5	29.9
probe	30	30	30	30	30	30	30	39	40	21	88	42	0.3	0.1	26.1	26.4	27.0	25.8	27.4	28.0	29.4
MpC	17	25	0	15	24	30	30	3985	1635	9000	4589	1944	0.4	0.1	11.7	18.6	0.0	11.2	17.7	28.7	29.8
yahsp	21	-	12	5	5	30	30	2809	-	5558	7545	7555	1.4	0.1	2.3	-	1.8	0.7	0.7	27.4	30.0
BFWS	9	15	3	15	14	30	30	6383	4577	8108	4555	4892	0.1	0.2	5.9	7.6	1.3	9.1	6.4	28.1	28.1
FDSS	19	-	0	14	12	30	30	3838	-	9000	5024	5602	0.6	0.5	18.9	-	0.0	12.7	11.1	27.5	28.4
gripper																					
lama	6	30	17	30	30	30	30	7342	101	4242	160	105	4.8	4.2	5.6	29.9	16.9	29.9	29.9	25.7	25.7
probe	0	0	0	0	-	30	30	9000	9000	9000	9000	-	16	17	0.0	0.0	0.0	0.0	-	29.9	29.9
MpC	0	0	0	0	0	30	30	9000	9000	9000	9000	9000	48	1.3	0.0	0.0	0.0	0.0	0.0	29.7	29.8
yahsp	0	-	0	0	0	0	30	9000	-	9000	9000	9000	9000	0.2	0.0	-	0.0	0.0	0.0	0.0	30.0
BFWS	0	5	5	0	5	27	27	9000	7527	7539	9000	7527	1372	1382	0.0	4.9	4.9	0.0	4.9	26.9	26.9
FDSS	0	14	0	9	13	30	-	9000	5082	9000	6492	5361	7.3	-	0.0	13.9	0.0	8.9	12.9	29.9	-
matching-bw																					
lama	26	30	-	30	29	30	30	1202	2.8	-	2.5	301	0.1	0.1	18.3	21.9	-	20.4	20.9	23.4	29.5
probe	13	23	0	22	23	-	-	5108	2123	9000	2434	2115	-	-	9.7	19.2	0.0	20.3	20.5	-	-
MpC	0	0	-	26	24	-	-	9000	9000	-	1294	1885	-	-	0.1	0.1	-	22.7	20.8	-	-
yahsp	0	25	-	26	26	30	30	9000	1523	-	1201	1204	0.1	0.1	0.1	14.0	-	17.3	17.2	22.4	29.8
BFWS	11	15	0	27	29	30	30	5748	4604	9000	987	444	0.1	0.1	8.8	12.3	0.0	21.5	25.2	27.4	27.5
FDSS	30	30	30	30	30	30	30	2.2	2.0	1.8	2.3	1.7	0.2	0.2	22.4	21.4	21.8	20.4	21.0	24.6	29.4
parking																					
lama	21	-	0	-	-	-	-	2896	-	9000	-	-	-	-	20.0	-	0.0	-	-	-	-
probe	2	-	0	0	0	0	0	8436	-	9000	9000	9000	9000	9000	2.0	-	0.0	0.0	0.0	0.0	0.0
MpC	5	-	0	0	-	-	-	7539	-	9000	9000	-	-	-	5.0	-	0.0	0.0	-	-	-
yahsp	0	-	0	0	-	0	0	9000	-	9000	9000	-	9000	9000	0.0	-	0.0	0.0	-	0.0	0.0
BFWS	20	-	0	-	-	-	-	3087	-	9000	-	-	-	-	19.9	-	0.0	-	-	-	-
FDSS	10	-	0	-	-	-	-	6155	-	9000	-	-	-	-	9.4	-	0.0	-	-	-	-
rovers																					
lama	30	30	30	30	30	30	30	153	134	129	136	116	101	81	27.0	26.7	26.7	28.4	28.7	29.3	29.5
probe	29	26	29	30	28	29	29	699	1541	640	400	954	624	620	28.1	25.4	28.3	28.9	27.2	28.0	28.3
MpC	9	3	5	7	4	8	4	6512	8181	7602	7060	7898	6783	7913	8.7	2.1	2.6	6.9	2.9	6.6	2.6
yahsp	30	30	30	30	30	30	30	19	19	21	18	18	18	19	26.7	26.6	26.5	29.6	28.1	29.2	29.5
BFWS	17	12	1	19	16	-	-	4197	5596	8718	3590	4449	-	-	16.7	11.7	0.9	18.2	15.4	-	-
FDSS	30	30	24	30	30	30	30	372	306	2207	354	280	301	274	27.4	27.4	21.9	29.1	29.0	28.9	28.7
sokoban																					
lama	20	-	21	22	19	-	-	3017	-	2731	2431	3326	-	-	17.9	-	17.7	18.3	14.2	-	-
probe	25	-	27	29	28	-	-	1539	-	924	329	631	-	-	21.0	-	23.2	23.5	23.0	-	-
MpC	30	-	30	30	29	-	-	1.3	-	1.0	2.7	303	-	-	27.6	-	28.3	24.5	24.7	-	-
yahsp	25	-	25	28	27	-	-	1527	-	1577	724	966	-	-	17.2	-	13.3	25.6	24.5	-	-
BFWS	30	-	30	30	30	-	-	1.2	-	3.0	1.9	2.3	-	-	28.5	-	26.3	27.8	28.3	-	-
FDSS	30	-	30	30	30	-	-	22	-	13	37	30	-	-	27.3	-	20.2	24.0	24.5	-	-

Table 1: Coverage, average PAR10 score (in seconds), and IPC quality score of the (O)riginal, (M)UM, (B)LoMa, (C)ritical Section Marcos, Aggressive Critical Section Macros (AC) and their combination with MUM (CM, ACM respectively) encodings, using the MUM learning methodology. "-" denotes that no macros have been generated. Gray indicates cases where C or CM model allow the planner to outperform the Original, MUM, and BloMa models in terms of PAR10. Light gray is used for cases where the improvement is achieved only by AC or ACM.

macros and successfully removed the replaced original operators.

Interestingly, in 6 domains (Barman, BlocksWorld, Depots, Gripper, Matching-Bw, Rovers), the aggressive variant (the BloMA learning methodology) generates Critical Section Macros that can replace all primitive operators operating with a resource. Only in Sokoban, original operators cannot be removed as it renders training tasks unsolvable (a macro push-push cannot replace the push operator). Combination of aggressive Critical Section Macros with MUM, especially when using the BloMa learning methodology, leads to further performance boost. In Bar-

Planner	Coverage							PAR10							IPC Quality						
	O	M	B	C	CM	AC	ACM	O	M	B	C	CM	AC	ACM	O	M	B	C	CM	AC	ACM
barman																					
lama	2	-	14	30	30	30	30	8428	-	5022	396	8.5	11	1.9	1.7	-	13.9	26.4	30.0	30.0	30.0
probe	2	-	24	0	30	30	30	8427	-	2044	9000	43	231	0.5	1.4	-	22.5	0.0	29.9	29.9	29.9
MpC	0	-	0	0	30	1	30	9000	-	9000	9000	11	8700	0.6	0.0	-	0.0	0.0	30.0	1.0	30.0
yahsp	0	-	29	30	30	30	30	9000	-	442	253	0.3	10	0.1	0.0	-	21.5	29.1	30.0	30.0	30.0
BFWS	0	-	30	20	30	30	30	9000	-	4.9	3007	2.2	1.8	0.1	0.0	-	30.0	18.7	28.2	28.2	28.2
FDSS	20	-	25	30	30	30	30	3234	-	1820	310	152	15	1.9	17.6	-	24.9	29.3	29.4	29.4	29.4
bw																					
lama	28	-	29	28	29	28	30	681	-	380	711	404	616	0.4	15.3	-	12.1	12.7	13.0	12.8	30.0
probe	25	-	29	30	30	30	30	1679	-	558	214	172	0.4	0.3	21.4	-	27.3	27.2	27.0	27.3	29.9
MpC	0	-	0	30	30	11	20	9000	-	9000	172	168	5700	3012	0.0	-	0.0	9.5	22.0	2.2	20.0
yahsp	27	-	24	26	27	30	30	948	-	1848	1235	950	0.1	0.2	4.4	-	13.7	18.1	18.5	20.5	30.0
BFWS	3	-	1	1	3	30	30	8106	-	8700	8709	8114	58	3.2	0.6	-	0.1	0.4	1.3	28.2	29.9
FDSS	22	-	22	21	21	30	30	2506	-	2493	2806	2794	11	0.6	14.2	-	10.4	12.0	11.2	16.3	30.0
depots																					
lama	0	0	0	2	0	30	30	9000	9000	9000	8417	9000	0.5	0.3	0.0	0.0	0.0	1.4	0.0	29.5	29.9
probe	30	30	30	30	30	30	30	39	38	43	87	37	0.3	0.1	26.2	26.8	26.7	25.8	27.6	28.0	29.4
MpC	17	25	23	14	24	30	30	3985	1649	2221	4883	1947	0.4	0.1	11.7	18.6	17.1	10.6	17.9	28.7	29.8
yahsp	21	20	20	5	20	30	30	2809	3050	3061	7550	3060	1.4	0.1	2.3	3.2	3.2	0.7	3.2	27.4	30.0
BFWS	9	13	14	15	15	30	30	6383	5180	4873	4550	4620	0.1	0.1	5.3	9.3	9.0	7.8	10.8	20.4	29.7
FDSS	19	12	13	12	12	30	30	3838	5597	5325	5592	5601	0.6	0.4	18.9	11.2	12.2	11.0	11.2	27.5	28.4
gripper																					
lama	6	30	30	24	27	30	30	7342	101	103	1926	985	4.8	4.1	5.6	30.0	30.0	24.0	27.0	25.8	25.8
probe	0	0	0	0	0	30	30	9000	9000	9000	9000	9000	17	17	0.0	0.0	0.0	0.0	0.0	29.9	29.9
MpC	0	0	0	0	0	30	30	9000	9000	9000	9000	9000	48	1.3	0.0	0.0	0.0	0.0	0.0	29.7	29.8
yahsp	0	0	0	0	0	0	30	9000	9000	9000	9000	9000	9000	0.2	0.0	0.0	0.0	0.0	0.0	0.0	30.0
BFWS	0	5	5	0	5	27	27	9000	7529	7526	9000	7525	1371	1381	0.0	4.9	4.9	0.0	4.9	26.9	26.9
FDSS	0	15	14	9	10	30	30	9000	4802	5081	6493	6202	7.2	6.4	0.0	14.9	13.9	8.9	9.9	29.9	29.9
matching-bw																					
lama	26	26	0	30	30	30	30	1202	1202	9000	2.6	2.2	0.1	0.1	21.2	20.7	0.0	25.3	25.8	27.4	27.4
probe	13	18	0	30	30	30	30	5108	3610	9000	0.7	1.5	0.1	0.1	7.1	11.2	0.0	29.0	28.3	28.8	28.9
MpC	0	0	0	25	17	15	20	9000	9000	9000	1582	3982	4500	3000	0.0	0.0	0.0	17.3	13.7	9.0	19.9
yahsp	0	28	0	25	25	30	30	9000	639	9000	1501	1512	0.1	0.1	0.0	20.9	0.0	21.9	20.3	27.7	27.7
BFWS	11	15	0	27	29	30	30	5748	4604	9000	985	442	0.1	0.1	8.8	12.3	0.0	21.5	25.2	27.4	27.5
FDSS	30	30	15	30	30	30	30	2.2	1.8	4505	2.4	1.9	0.1	0.1	24.2	24.6	10.0	22.5	23.4	25.9	25.9
rovers																					
lama	30	30	30	30	30	30	30	153	123	163	118	91	110	81	26.6	26.2	26.3	29.5	29.6	28.8	29.0
probe	29	23	22	29	30	28	29	699	2392	2789	675	361	907	634	28.1	22.5	20.8	28.0	29.3	27.0	28.1
MpC	9	6	8	7	7	8	8	6512	7333	6786	7060	7058	6787	6788	8.3	5.6	7.5	6.6	6.9	6.3	6.7
yahsp	30	30	30	30	30	30	30	19	18	17	18	18	18	18	26.2	26.6	26.1	29.1	29.8	28.6	29.3
BFWS	17	11	2	18	15	21	19	4197	5881	8441	3854	4734	3016	3615	16.7	10.8	1.9	17.3	14.4	19.9	17.9
FDSS	30	30	23	30	30	30	30	372	302	2477	354	258	348	248	27.4	27.3	20.8	29.1	28.9	28.6	28.7
sokoban																					
lama	20	-	12	22	19	-	-	3017	-	5414	2431	3326	-	-	18.5	-	9.8	18.8	14.6	-	-
probe	25	-	27	29	28	-	-	1539	-	918	329	631	-	-	20.9	-	22.1	24.1	23.1	-	-
MpC	30	-	30	30	30	-	-	1.3	-	1.1	2.6	4.7	-	-	27.6	-	28.3	24.5	25.3	-	-
yahsp	25	-	26	28	27	-	-	1527	-	1297	722	965	-	-	17.2	-	13.8	25.6	24.5	-	-
BFWS	30	-	30	30	30	-	-	1.2	-	2.5	1.8	2.2	-	-	28.5	-	26.3	27.8	28.3	-	-
FDSS	30	-	30	26	29	-	-	22	-	12	1220	329	-	-	27.4	-	20.3	20.8	23.6	-	-

Table 2: Coverage, average PAR10 score (in seconds), and IPC quality score of the (O)riginal, (M)UM, (B)LoMa, (C)ritical Section Marcos, Aggressive Critical Section Macros (AC) and their combination with MUM (CM, ACM respectively) encodings, using the BloMa learning methodology. "-" denotes that no macros have been generated. Gray indicates cases where C or CM model allow the planner to outperform the Original, MUM, and BloMa models in terms of PAR10. Light gray is used for cases where the improvement is achieved only by AC or ACM.

man, BlocksWorld, Depots, Gripper, Matching-Bw, there exists at least one planner that solves each "enhanced" testing task within 1 second in average (that is considerably better than for the original tasks). Critical Section Macros are particularly useful in Barman, where we could learn two "long" macros, one which puts two ingredients into the shaker and cleans the shot used to fill the shaker, and the other which shakes the cocktail, pours it into the required shot and cleans the shaker afterwards. When combined with MUM, a "supermacro" assembling these two macros together was generated (hence, a cocktail can be made in one step). In Rovers, Critical Section Macros capture more marginal activities (sampling and dropping rock or soil) and thus the macros are improving performance marginally.

The IPC quality score indicates that the expectable plan quality degradation by macro use is marginal (the score is often close to the coverage). In Bw and Depots, plan quality even increases when macros are used.

We can observe that if Critical Section Macros capture more complex activities such as in Barman their impact on performance tend to be larger. In contrast, if Critical Section Macros capture non-frequent activities (i.e., macros are used a few times in plans) their impact is marginal such as in Rovers. Replacing all original operators dealing with a resource in the aggressive variant usually results in a considerable performance improvement. In the conservative variant, reasoning with such a resource is not completely eliminated as the original operators remain in the domain model and hence planning engines might not fully benefit from Critical Section Macros. Finally, using better quality training plans leads to more representative macros.

Conclusion

In this paper, we introduced Critical Section Macros that are inspired by resource locking in critical sections in parallel computing. Roughly speaking, these macros capture whole activities in which a resource is locked (e.g., shaking a cocktail and cleaning the shaker afterwards). The results have shown the usefulness of Critical Section Macros, especially in domains where non-trivial activities can be captured (e.g. Barman) or where original operators dealing with resources can be removed (e.g. BlocksWorld, Gripper).

In future, we would like to extend the methods for temporal planning, since we believe that the nature Critical Section Macros can capture useful activities also in partially ordered action sequences common in temporal plans.

Acknowledgements

This research was funded by the Czech Science Foundation (project no. 18-07252S). Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

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*, 16–24.

Asai, M., and Fukunaga, A. 2015. Solving large-scale planning problems by decomposition and macro generation. In *ICAPS*, 16–24.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1-2):5–33.

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

Chrpa, L., and Siddiqui, F. H. 2015. Exploiting block deordering for improving planners efficiency. In *IJCAI*, 1537–1543.

Chrpa, L.; Vallati, M.; and McCluskey, T. L. 2014. MUM: a technique for maximising the utility of macro-operators by constrained generation and use. In *ICAPS*, 65–73.

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.

Dulac, A.; Pellier, D.; Fiorino, H.; and Janiszek, D. 2013. Learning useful macro-actions for planning with n-grams. In *ICTAI*, 803–810.

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

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

Hofmann, T.; Niemueller, T.; and Lakemeyer, G. 2017. Initial results on generating macro actions from a plan database for planning on autonomous mobile robots. In *ICAPS*, 498–503.

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

Lipovetzky, N.; Ramirez, M.; Muise, C.; and Geffner, H. 2014. Width and inference based planners: Siw, bfs(f), and probe. In *The Eighth IPC. Description of Participant Planners of the Deterministic Track*, 6–7.

Lipovetzky, N.; Ramirez, M.; Frances, G.; and Geffner, H. 2018. Best-first width search in the ipc2018: Complete, simulated, and polynomial variants. In *The Ninth International Planning Competition. Description of Participant Planners of the Deterministic Track.*

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.

Rintanen, J. 2014. Madagascar: Scalable planning with sat. In *The Eighth IPC. Description of Participant Planners of the Deterministic Track*, 66–70.

Seipp, J., and Röger, G. 2018. Fast downward stone soup 2018. In *The Ninth International Planning Competition. Description of Participant Planners of the Deterministic Track.*

Siddiqui, F., and Haslum, P. 2012. Block-structured plan deordering. In *25th Australasian Joint Conference*, volume 7691 of *LNAI*, 803–814.

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