Journal of Artificial Intelligence Research 1 (1993) 1-15 Submitted 6/91; published 9/91

Planning with Critical Section Macros: Theory and Practice

Lukáš Chrpa CHRPALUK@FEL.CVUT.CZ
Faculty Of Electrical Engineering,
Czech Technical University in Prague

Mauro Vallati M.VALLATI@HUD.AC.UK
School of Computing and Engineering,
University of Huddersfield

Abstract

Macro-operators (macros) 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 by considering most frequent action sequences in training plans. Unfortunately, frequent action sequences might not capture meaningful activities as a whole, leading to a limited beneficial impact for the planning process.

In this paper, inspired by resource locking in critical sections in parallel computing, we propose a technique that generates macros able to capture whole activities in which limited resources (e.g., a robotic hand, or a truck) are used. Specifically, such a *Critical Section* macro starts by *locking* the resource (e.g., grabbing an object), continues by *using* the resource (e.g., manipulating the object) and finishes by *releasing* the resource (e.g., dropping the object). Hence, such a macro bridges states in which the resource is locked and cannot be used. We also introduce versions of Critical Section macros dealing with multiple resources and phased locks. Usefulness of macros is evaluated using a range of state-of-the-art planners, and a large number of benchmarks from the deterministic and learning tracks of recent editions of the International Planning Competition.

1. Introduction

Automated planning is an important research area of Artificial Intelligence that deals with the problem of finding a sequence of actions whose application in an initial state of the environment leads to a desired goal state (Ghallab, Nau, & Traverso, 2004). Automated planning has been successfully applied in challenging real-world domains, including drilling (Fox, Long, Tamboise, & Isangulov, 2018), transport (Vallati, Magazzeni, Schutter, Chrpa, & McCluskey, 2016; Cardellini, Maratea, Vallati, Boleto, & Oneto, 2021), smart grid (Thiébaux, Coffrin, Hijazi, & Slaney, 2013), UAV control (Ramírez, Papasimeon, Lipovetzky, Benke, Miller, Pearce, Scala, & Zamani, 2018), e-learning (Garrido, Morales, & Serina, 2012) and mining (Lipovetzky, Burt, Pearce, & Stuckey, 2014).

Since 1998, the International Planning Competition (IPC)[1] has been organised and is increasingly attracting the attention of the automated planning community. IPCs are a driving factor for developing domain-independent planning engines, predominantly based on heuristic search (Bonet & Geffner, 2001), that accept planning task specification in the PDDL language (Mcdermott, Ghallab, Howe, Knoblock, Ram, Veloso, Weld, & Wilkins, 1998) on the input and output a plan (if it exists). This decoupling between the planning

1. `http://ipc.icaps-conference.org`

engine and the planning task specification supports the use of reformulation techniques, that can automatically re-formulate planning models provided as input to the engines, while keeping to the same input language, in order to increase the efficiency of planning engines and increase the range of problems solved.

A very well-known reformulation approach is the generation of macro-operators, macros for short, that encapsulate sequences of (primitive) planning operators. Macros are encoded as ordinary planning operators and, hence, they can be easily added into planning 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 a smaller number of steps. This comes at the cost of increased branching factor, since macros often have much more instances than primitive 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 only if they are frequently used in solution plans (Hofmann, Niemueller, & Lakemeyer, 2017), their number of instances is small (Chrpa, Vallati, & McCluskey, 2014), or they address weaknesses of a specific planning engine (Coles, Fox, & Smith, 2007).

In this paper we present a detailed study of *Critical Section Macros (CSMs)*, that are inspired by parallel computing critical sections, and show their beneficial impact on the performance of domain-independent planning engines. In parallel computing, a *critical section* is a part of the code that deals with one or more shared resources such that the resources are initially locked (to prevent other processes or threads to concurrently access them), then processed and at the end they are released. In automated planning we can observe parts of plans that share similar characteristics of critical sections. For example, a hand of a robotic barman can be understood as a shared (or limited) resource, since the hand can hold at most one object at a time. Then a sequence of actions starting with the hand grasping a shaker, following with shaking the cocktail, pouring the cocktail into a glass and cleaning the shaker, and finishing by releasing the shaker can be referred to as a critical section. Assembling such an action sequence into a (Critical Section) macro bridges states in which the resource is locked and cannot be used (e.g. the hand cannot grasp a different object at the same time). Consequently, it might, to some extent, mitigate the incorrect assumption of some heuristics (e.g. delete-related ones) that a resource can be used with multiple objects simultaneously. Due to their nature, CSMs can sometimes replace primitive operators they are assembled from and/or can be effectively combined with other macro generating techniques.

Contributions of this paper, in which we significantly extend the work presented in our conference paper (Chrpa & Vallati, 2019), are as follows:

- We describe the concept of standard and phased *resource locks*.
- We define and describe the concept of *Critical Section Macros* (CSMs).
- We consider complex variants of CSMs, namely multi-locks CSMs and multi-phased lock CSMs.
- We introduce the concept of compound CSMs that, in a nutshell, account for incremental construction of more complex CSMs from simple ones.

- We consider the aggressive approach of CSM use in which we eliminate replaced original operators.
- We provide an extensive and detailed empirical evaluation on a broad range of domains.

The remainder of this work is organised as follows. Section 2 contextualises the proposed approach with regards to existing related work. The necessary background, in terms of classical planning, macros and outer entanglements, is given in Section 3. Critical Section Macros and their compound variant are formally introduced in Sections 4 and 5, and the approach for extracting such macros is provided in Section 6. The experimental analysis is presented in Section 7. Finally, in Section 8, conclusions are given.

2. Related Work

Using macros dates back to 1970s and 1980s. REFLECT (Dawson & Siklóssy, 1977) builds macro-operators from pairs of primitive operators that can be successively applied and share at least one argument. MORRIS (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-serialisable sub-goals (e.g. in Rubik's cube).

Recent planner-independent techniques aim at improving performance of any standard planner. MacroFF (Botea, Enzenberger, Müller, & Schaeffer, 2005) generates macros according to several pre-defined rules (e.g., the "locality rule") that apply on adjacent actions in training plans. Wizard (Newton, Levine, Fox, & Long, 2007) learns macros from training plans by exploiting genetic programming. Alhossaini and Beck (2013) select problem-specific macros from a given pool of macros (hand-coded or generated by another technique). Dulac, Pellier, Fiorino, and Janiszek (2013) exploit n-gram algorithm to analyse training plans to learn macros. DBMP/S (Hofmann et al., 2017) applies Map Reduce for learning macros from a larger set of training plans. The recent extension of DBMP/S supports ADL features (Hofmann, Niemueller, & Lakemeyer, 2020). CAP (Asai & Fukunaga, 2015) exploits component abstraction (introduced by MacroFF) for generating sub-goal specific macros.

MUM (Chrpa et al., 2014) leverages "outer entanglements", which are relations between operators and initial or goal atoms (Chrpa, Vallati, & McCluskey, 2018), as a heuristic for generating macros step by step (i.e., assembling two "macro candidates", which can be both primitive operators or macros, in a given step). Generated macros have limited number of instances, specifically, the number of macro instances has to be in the same order of magnitude as the number of primitive operator instances. Junghanns and Schaeffer (2001) introduce the concept of "tunnel macros" aiming at bypassing "uninteresting" intermediate states for the Sokoban game. The idea of "tunnel macros" was later elaborated in the context of breaking variable coupling in order to decompose abstracted planning tasks (Haslum, 2007). Conceptually, CSMs, which are the contribution of this paper, can be seen as "tunnel macros" in the sense of bypassing states in which the (limited) resource is being used. BloMa (Chrpa & Siddiqui, 2015) leverages block deordering, which rearranges plans into "blocks" that can no longer be deordered (Siddiqui & Haslum, 2012), for generating longer

macros. In particular, BloMa generates a large pool of macros from "macroblocks", which are derived from "blocks" by applying a set of rules, that is later reduced by applying (strict) frequency requirements. CSMs, in contrast to other macro generating techniques, have a specific structure capturing whole activities in which limited resources are used. Although the aim of generating meaningful long macros is similar to BloMa, CSMs are targeted to bypassing states in which limited resources are being used that in consequence might help, for example, delete-relaxation-based approaches to make better heuristic estimates. CSMs can be combined with "chaining" macro generation approaches such as MUM, as shown in our previous work (Chrpa & Vallati, 2019).

Macro generation is not limited to classical planning, a few recent works concern the use of macros in plan repair in numeric planning (Scala, 2014; Scala & Torasso, 2015).

Several works go in the opposite direction of macros. 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. In other words, the method identifies and removes macros from the planning task description. From another perspective, operator schema splitting (Areces, Bustos, Dominguez, & Hoffmann, 2014) deals with decomposing complex planning operators into simpler ones; basically, the contrary of what macro generation approaches are aiming at. It should be mentioned that these works do not contradict the usefulness of macro generation techniques, they just highlight the importance of careful selection of useful macros, or careful positioning of macros in domain models (Vallati, Chrpa, McCluskey, & Hutter, 2021) – and to some extent, the importance of the wider knowledge engineering process of encoding domain models (McCluskey, Vaquero, & Vallati, 2017).

3. Classical Planning

Classical planning is concerned with finding a (partially or totally ordered) sequence of actions transforming the static, deterministic and fully observable environment from the given initial state to a desired goal state (Ghallab et al., 2004; Mcdermott et al., 1998).

In the classical representation, a *planning task* consists of a *planning domain model* and a *planning problem*, where the planning domain model describes the environment and defines planning operators while the planning problem defines concrete objects, an initial state and a set of goals. The environment is described by *predicates* that are specified via a unique identifier and terms (variable symbols or constants). For example, a predicate (at ?t ?p), where at is a unique identifier, and ?t and ?p are variable symbols, denotes that a truck ?t is at a location ?p. Predicates thus capture general relations between objects.

Definition 1. *A **planning task** is a pair* $\Pi = (Dom_\Pi, Prob_\Pi)$ *where a **planning domain model*** $Dom_\Pi = (P_\Pi, Ops_\Pi)$ *is a pair consisting of a finite set of predicates* P_Π *and planning operators* Ops_Π*, and a **planning problem*** $Prob_\Pi = (Objs_\Pi, I_\Pi, G_\Pi)$ *is a triple consisting of a finite set of objects* $Objs_\Pi$*, initial state* I_Π *and goal* G_Π*.*

Let ats_Π *be the set of all **atoms** that are formed from the predicates* P_Π *by substituting the objects* $Objs_\Pi$ *for the predicates' arguments. In other words, an atom is an **instance** of a predicate (in the rest of the paper when we use the term instance, we mean an instance that is fully ground). A **state** is a subset of* ats_Π*, and the **initial state*** I_Π *is a distinguished*

state. The ***goal*** G_Π *is a non-empty subset of* ats_Π*, and a* ***goal state*** *is any state that contains the goal* G_Π*.*

Notice that the semantics of state reflects the full observability of the environment. That is, that for a state s, atoms present in s are assumed to be true in s, while atoms not present in s are assumed to be false in s.

Planning operators are the mean that allows to modify the environment. They consist of *preconditions*, i.e., what must hold prior an operators' application, and *effects*, i.e., what is changed after operators' application. Specifically, we distinguish between *delete effects*, i.e., what becomes false, and *add effects*, i.e., what becomes true after applying the operator. *Actions* are instances of planning operators, i.e., operators' arguments as well as corresponding variable symbols in operators' preconditions and effects are substituted by objects (constants). Planning operators capture general types of activities that can be performed. Planning operators can be instantiated to actions in order to capture given activities between concrete objects.

Definition 2. *A* ***planning operator*** $o = (name(o), pre(o), del(o), add(o))$ *is specified such that* $name(o) = op\_name(x_1, \ldots, x_k)$*, where op_name is a unique identifier and* $x_1, \ldots x_k$ *are all the variable symbols (arguments) appearing in the operator,* $pre(o)$ *is a set of predicates representing the precondition of* o*,* $del(o)$ *and* $add(o)$ *are sets of predicates representing the negative (or delete) and positive (or add) effects of* o*.*

Actions *are instances of planning operators that are formed by substituting objects, defined in a planning problem, for operators' arguments as well as for corresponding variable symbols in operators' preconditions and effects. An action* $a = (pre(a), del(a), add(a))$ *is* ***applicable*** *in a state* s *if and only if* $pre(a) \subseteq s$*. Application of* a *in* s*, if possible, results in a state* $(s \setminus del(a)) \cup add(a)$*.*

We define a ***transition function*** $\gamma : S \times A^n \to S$, where S is a set of states, A is a set of actions and n is a non-negative integer, as follows:

$$\gamma(s, \langle\rangle) = s$$
$$\gamma(s, \langle a \rangle) = (s \setminus del(a)) \cup add(a) \text{ if } a \text{ is applicable in } s, \text{ or undefined otherwise}$$
$$\gamma(s, \langle a_1, a_2, \ldots, a_n \rangle) = \gamma(\gamma(s, \langle a_1 \rangle), \langle a_2, \ldots, a_n \rangle), \text{ or undefined if } \gamma(s, \langle a_1 \rangle) \text{ is undefined}$$

A solution of a planning task is a sequence of actions transforming the environment from the given initial state to a goal state.

Definition 3. *A* ***plan*** *is a sequence of actions. A plan is a* ***solution*** *of a planning task* Π*, a* ***solution plan*** *of* Π *in other words, if and only if a consecutive application of the actions from the plan starting in the initial state of* Π *results in the goal state of* Π*.*

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 atoms p and q are ***mutex*** if and only if no reachable state contains both of them. We also say that a set of atoms is a ***mutex group*** if and only if in each reachable state at most one atom from the set is true. We say that an action a_i is an ***achiever*** for an action a_j if an only if $add(a_i) \cap pre(a_j) \neq \emptyset$. We also say that

actions a_i and a_j are ***independent*** if and only if $del(a_i) \cap (pre(a_j) \cup add(a_j)) = \emptyset$ and $del(a_j) \cap (pre(a_i) \cup add(a_i)) = \emptyset$. We also say that actions a_i and a_j are ***equivalent*** if and only if for each state s it is the case that $\gamma(s, a_i) = \gamma(s, a_j)$ or both $\gamma(s, a_i)$ and $\gamma(s, a_j)$ are undefined.

3.1 Running Examples

Throughout the paper, we will use examples from several benchmark domains that were introduced in the International Planning Competitions[2].

BlocksWorld (Bw, in short) is perhaps the best known domain in the planning community. In Bw, the task is to rearrange towers of blocks by a robotic hand. A "clear" block (i.e., nothing is stacked on that block) can be *picked up* from the table if the robotic hand is free. A "clear" block can be *unstacked* from another block if the robotic hand is free. A block can be *put down* on the table if held by the robotic hand. Similarly, a block can be *stacked* on another "clear" block if held by the robotic hand.

The *Transport* domain is a variant of a logistics domain in which packages have to be delivered by limited-capacity trucks from their locations of origin to their destination locations. A package can be *picked up* by a truck if they both are at the same location and the truck has sufficient capacity to carry the package. A package can be *dropped* by the truck if the package is inside the truck. A truck can *drive* from one location to another if the locations are connected by road.

In the *Storage* domain, crates have to be delivered by hoists from the areas of their origin to the areas of their destination. A "free" hoist can *pick up* a crate if the hoist is located in the area next to the area in which the crate is placed. A hoist can *drop* the crate to a free store area next to the area the hoist is located. A hoist can *move* to a free adjacent store area. A hoist can *go out* from the store area to an adjacent transit area and, finally, a hoist can *go in* to a free adjacent store area from the transit area the hoist is located.

The *Barman* domain models a robotic barman which has to prepare a collection of drinks and cocktails. The robotic barman has two hands that can be used to carry and manipulate containers (shots or shakers). Cocktails are prepared in a shaker to which the barman has to put required ingredients (two of them). Ingredients can be poured from dispensers to shots and then put to the shaker that has to be clean before use. The shot has to be clean as well before use unless it is used for the same ingredient as before.

3.2 Macro-actions and Macro-operators

Sequences of actions can be assembled into single actions, called macro-actions. Macro-actions can be understood as short-cuts in state-transition systems representing planning tasks. Plan generation might hence be done in a smaller number of steps. Macro-actions are formally defined as follows.

Definition 4. *Let $\langle a_1, \dots a_n \rangle$ be a sequence of actions. We say that $a_{1,\dots,n}$ is a* **macro-action** *over $\langle a_1, \dots a_n \rangle$ if and only if for each state s it is the case that $\gamma(s, a_{1,\dots,n}) = \gamma(s, \langle a_1, \dots a_n \rangle)$ or both $\gamma(s, a_{1,\dots,n})$ and $\gamma(s, \langle a_1, \dots a_n \rangle)$ are undefined.*

2. `http://ipc.icaps-conference.org`

Algorithm 1 Assembling Macro-actions from a sequence of actions

Require: $\langle a_1, \dots a_n \rangle$
Ensure: $a_{1,\dots,n}$
1: **function** AssembleTwo(a_i, a_j)
2: $pre(a_{i,j}) \leftarrow pre(a_i) \cup (pre(a_j) \setminus add(a_i))$
3: $del(a_{i,j}) \leftarrow (del(a_i) \setminus add(a_i)) \cup del(a_j)$
4: $add(a_{i,j}) \leftarrow (add(a_i) \setminus del(a_j)) \cup add(a_j)$
5: **return** $a_{i,j}$
6: **end function**

7: **for** $i = 1$ to $n - 1$ **do**
8: $a_{1,\dots,i+1} \leftarrow$AssembleTwo($a_{1\dots,i}, a_{i+1}$)
9: **end for**

We also say that $a_{1,\dots,n}$ is **sound** *if and only if for each $1 < i \leq n$ it is the case that $pre(a_i) \cap \bigcup_{j=1}^{i-1}(del(a_j) \setminus \bigcup_{l=j}^{i-1} add(a_l)) = \emptyset$.*

A macro-action $a_{1,\dots,n}$ can be assembled from a sequence of actions $\langle a_1, \dots a_n \rangle$ as shown in Algorithm 1. The algorithm iteratively assembles actions one by one until the required macro-action is created.

At a more general level, macro-operators encapsulate sequences of ordinary planning operators and as actions are grounded instances of planning operators, macro-actions are instances of macro-operators. Also, macro-operators are domain-specific rather than problem-specific and hence they can be added into a domain model and exploited in a *planner independent* way (e.g. encoded in PDDL). We generalise Definition 4 for macro-operators as follows.

Definition 5. *Let $\langle o_1, \dots o_n \rangle$ be a sequence of planning operators. We say that $o_{1,\dots,n}$ is a* **macro-operator** *over $\langle o_1, \dots o_n \rangle$ if for each instance $a_{1,\dots,n}$ of $o_{1,\dots,n}$ it is the case that $a_{1,\dots,n}$ is a macro-action over $\langle a_1, \dots a_n \rangle$, where for each $1 \leq i \leq n$ it holds that a_i is an instance of o_i.*

We also say that $o_{1,\dots,n}$ is **sound** *if and only if each of its instances is sound as well.*

Macro-operators, in the most general case, contain all the variable symbols (parameters) of operators they are assembled from. However, it is useful to bind some variables to be equal. For example, in BlocksWorld, assembling two operators pickup(?x) and stack(?z ?y) into a macro-operator makes sense if ?x and ?z are equal since we would like to stack the same block we picked up. Instead of explicitly putting equality constraints into macro-operators' preconditions, we unify equal variables by renaming substitutions. That is, in our example, renaming ?z to ?x, so the macro-operator pickup-stack(?x ?y) has two parameters.

Note that hereinafter we will use the term *macro* to refer to both macro-actions and macro-operators. Where not clear from the context we will use the notions macro-action/macro-operator to disambiguate. Another type of variable binding is *inequality constraints*, which might be necessary to handle a possible source of unsoundness of macro-operators (Chrpa, 2010b). For example, in the BlocksWorld 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 generalising this observation we can see that if some (different) variable symbols are substituted by the same constants, a macro-operator might become unsound. To avoid such cases, a constraint requiring different instantiation of affected variable symbols, i.e., the inequality constraint, is added into macro-operator's precondition (e.g. (not (= ?x ?y)) is added into pickup-stack(?x ?y)'s precondition).

3.3 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 et al., 2018).

Generally speaking, entanglements by init 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 capture situations where only instances of a given operator that achieves instances of a certain predicate present in the goal are needed to solve the problem.

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

Definition 6. *Let $\Pi = (D, P)$ be a planning task, where $P = (Obj, I, G)$ is a planning problem and $D = (P, O)$ is a domain model. We say that operator $o \in O$ is **entangled by init (resp., goal)** with a predicate $p \in P$ if and only if $p \in pre(o)$ (resp., $p \in add(o)$) and there exists a solution plan π of Π such that for every o's instance $a \in \pi$ and for every p's instance p_{gnd} it holds: $p_{gnd} \in pre(a) \Rightarrow p_{gnd} \in I$ (resp., $p_{gnd} \in add(a) \Rightarrow p_{gnd} \in G$).*

Outer entanglements have been used as a reformulation technique, as they can be directly encoded into a problem instance. The way outer entanglements are encoded is inspired by one of their properties: given a static predicate p_s (p_s is not a part of effects of any operator), an operator o is entangled by init with p_s if and only if $p_s \in pre(o)$ (Chrpa & 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.

Formally, let Π be a planning task, I be its initial state and G its goal. Let an operator o be entangled by init (resp., goal) with a predicate p. Then Π is reformulated as follows (Chrpa et al., 2018):

1. Add a new static predicate p' having the same arguments as p to the domain model of Π.

2. Add p' into o's precondition such that p' has the same arguments as p which is in precondition (resp., add effects) of o.

3. Add instances of p' which correspond to instances of p in I (resp., in G) into I.

To give an example, operator unstack(?x ?y) that is entangled by init with (on ?x ?y) is modified such that a new static predicate (ei_on ?x ?y) is added to its precondition. Then, assuming that, for example, (on A B) and (on B C) are present in the initial state of a planning task, the initial state of the reformulated planning task will contain (ei_on A B) and (ei_on B C).

4. Critical Section Macros (CSMs)

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 multiple different processes are concurrently modifying a shared 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 sections, i.e., locking a resource, using it, and releasing it. In BlocksWorld, 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, pouring the cocktail into a shot, cleaning the shaker before putting it back on the table (releasing robot's hand).

The rationale behind CSMs is to *bridge* resource use with a single macro. Such macros can hence capture a whole activity that deals with a limited resource (or more limited resources). This might be beneficial, for instance, for planning techniques incorporating delete-relaxation, i.e., ignoring delete effects of actions (Hoffmann & Nebel, 2001). For states in which a resource is available, delete-relaxation heuristics might incorrectly assume that the resource can be simultaneously used by multiple objects (e.g. a robotic hand holding multiple blocks at the same time), potentially leading to inaccurate heuristic estimates.

Technically speaking, a free resource, as well as a locked resource, is represented by corresponding atoms. For example, (handfree) represents a free resource (the robotic hand) while (holding a) represents a locked resource (the robotic hand carrying block a).

We recognise critical activities by identifying mutex atoms p and q that represent a free and a locked resource, respectively. We consider two different conditions for p and q such that one has to be met in order to recognise p and q as a lock. In the first condition, the arguments of q must be a superset (not necessarily proper) of the arguments of p. 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 fewer arguments (in order to recover a corresponding atom p after releasing the resource). In the second condition, p and q must have the same name and the same number of arguments but they differ at exactly one argument. Such a condition refers to a situation, for example,

where a predicate represents a capacity of a resource (e.g. a truck). One argument describes the capacity of the resource (in that argument in which p and q differ) while the other arguments describe the resource itself (e.g. a truck).

Locking and releasing resources is done by specific actions. An action that deletes p and adds q, a *locker*, locks a given resource. Analogously, an action that deletes q and adds p, a *releaser*, releases (unlocks) the given resource. An action having q in its precondition but not deleting it, a *user*, uses the given resource. This idea is formalised as follows (note that $args(p)$ represents the set of arguments of a predicate p).

Definition 7. *Let Π be a planning task and ats_Π be atoms and A_Π be actions (defined in Π). Let $p, q \in ats_\Pi$ be atoms that are mutex and where (i) $args(p) \subseteq args(q)$, or (ii) $name(p) = name(q)$ and $|args(p) \cap args(q)| + 1 = |args(p)| = |args(q)|$. We say that an action $a \in A_\Pi$ is a p,q-**locker** if $p \in del(a)$ and $q \in add(a)$. We also say that $a' \in A_\Pi$ is a p,q-**releaser** if $q \in del(a')$ and $p \in add(a')$. We also say that $a'' \in A_\Pi$ is a q-**user** if $q \in pre(a'') \setminus (del(a'') \setminus add(a''))$. We denote the pair p, q as a* **lock**.

The above definition considers single-phased locks, i.e., resources are locked/released by one action only. However, it might be the case that locking and/or releasing a resource has to be done in multiple steps. For illustration, multi-phase locks might account for capacity of a truck that might carry at most k packages. Then, a sequence of "pick" actions might account for a multi-phase locker and a sequence of "drop" actions might account for a multi-phase releaser. To consider multi-phased locks, we extend Definition 7 as follows.

Definition 8. *Let Π be a planning task and ats_Π be atoms and A_Π be actions (defined in Π). Let $p^1, \ldots, p^k \in ats_\Pi$ be atoms forming a mutex group and where (i) $args(p^1) \subseteq \cdots \subseteq args(p^k)$, or (ii) $name(p^1) = \cdots = name(p^k)$ and $|\bigcap_{i=1}^{k} args(p^i)| + 1 = |args(p^1)| = \cdots = |args(p^k)|$. We say that a sequence of actions $a^{l_1}, \ldots, a^{l_m}$ forms a $p^1, \ldots, p^k$-**locker** if for each $i \in \{1, \ldots, m\}$ it is the case that a^{l_i} is a $p^{c_i}, p^{c_{i+1}}$-locker, where $1 = c_1 < c_2 < \cdots < c_m < c_{m+1} = k$. Similarly, we say that a sequence of actions $a^{r_1}, \ldots, a^{r_n}$ forms a $p^1, \ldots, p^k$-**releaser** if for each $i \in \{1, \ldots, n\}$ it is the case that a^{r_i} is a $p^{d_{i+1}}, p^{d_i}$-releaser, where $k = d_1 > d_2 > \cdots > d_n > d_{n+1} = 1$. We denote the sequence $p^1, \ldots, p^k$ as a* **phased lock**.

We say that $p^1, \ldots, p^k$-locker and $p^1, \ldots, p^k$-releaser are **balanced** *if and only if $m = n$ and for the indices it is the case that $c_1 = d_{n+1}, c_2 = d_n, \ldots, c_{n+1} = d_1$.*

It should be noted that one action can be a locker/releaser or a part of action sequence forming a locker/releaser for more locks (single- or multi-phased).

In the following subsections, we describe in detail three kinds of CSMs, the simple ones that use a single-phased lock such that they start with a locker and end up with a releaser over that (single) lock, and the complex ones that involve multi-phased locks or multiple locks.

4.1 Single-phased Single-lock CSMs

A Single-phased Single-lock (SpSl) CSM, in general, encapsulates a sequence of actions starting with a *locker* and ending up with a *releaser*. Such an action sequence might also contain *users* and "gluing" actions. A gluing action, roughly speaking, is an action that

cannot be moved before the locker or after the releaser without compromising the sequence – by invalidating a precondition of some action or by achieving a different outcome than the original action sequence. To give an example, an action sequence – pick(truck pkg loc1), drive(truck loc1 loc2), drop(truck pkg loc2) – can form a CSM. The drive action is a gluing action as it neither can be moved before pick, since these actions are not independent because drive deletes pick's precondition, nor can be moved after drop, since drive achieves a fact required by drop. In another action sequence – pick(truck pkg loc1), drive(truck loc1 loc2), drive(truck2 loc2 loc3), drop(truck pkg loc2) –, we can see that we can move the latter drive action (drive(truck2 loc2 loc3)) after drop as it is not an achiever for drop and both actions are independent. Note that the latter drive action can also be moved before pick as it is independent with both pick and the former drive action (drive(truck loc1 loc2)) and none of these actions is an achiever for it (the latter drive action).

We define the notion of a "potentially gluing" action that unifies the conditions under which the action can be considered as "gluing" within a CSM.

Definition 9. *Let $a_{1,\dots,k}$ be a macro over $\langle a_1, \dots, a_k \rangle$. We say that an action a_i, where $1 < i < k$, is a* **potentially gluing action** *with respect to $a_{1,\dots,k}$ if the following conditions hold:*

- *there exists an action a_j, where $1 \leq j < i$, such that a_i and a_j are not independent or a_j is an achiever for a_i*
- *there exists an action a_m, where $i < m \leq k$, such that a_i and a_m are not independent or a_i is an achiever for a_m*

Formally, a CSM with a single-phased single lock is defined as follows.

Definition 10. *Let Π be a planning task and ats_Π be atoms and A_Π be actions (defined in Π). Let $a_1, a_2, \dots, a_k \in A_\Pi$ be actions. We say that $a_{1,\dots,k}$ is a* ***Single-phased Single-lock Critical Section Macro*** *over a lock p, q ($p, q \in ats_\Pi$) if it is a macro over a sequence of actions $\langle a_1, \dots, a_k \rangle$ such that*

i) a_1 is a p, q-locker

ii) a_i, where $1 < i < k$, is either a q-user, or a potentially gluing action not being a q-user (a gluing action, for short)

iii) a_k is a p, q-releaser

Furthermore, we can classify SpSl CSMs into the following categories with regards to their shape.

Trivial – CSM contains only a locker and a releaser

Using – CSM contains also users, besides a locker and a releaser

Gluing – CSM contains also gluing operators, besides a locker and a releaser

Full – CSM contains all types of operators

For example, a macro pickup-stack(?x ?y), representing that a block ?x is moved from the table to a block ?y, is a CSM consisting of only a locker (pickup(?x)) and a releaser (stack(?x ?y)) and hence is classified as *trivial*. In Storage instead, a macro go_out-lift-go_in(?hoist ?storearea1 ?transitarea ?crate ?storearea2 ?place), representing that ?hoist initially moves from ?storearea1 to ?transitarea in order to lift ?crate from ?place in ?storearea2 and then moves back to ?storearea1, is a CSM consisting of a locker (go_out(?hoist ?storearea1 ?transitarea)), a releaser (go_in(?hoist ?transitarea ?storearea1)) and a user (lift(?hoist ?crate ?storearea2 ?transitarea ?place)) and hence is classified as *using*. In a variant of Transport in which each truck can carry at most one package, a macro pick-drive-drop(?truck ?package ?location1 ?location2), representing that ?truck delivers ?package from ?location1 to ?location2, is a CSM consisting of a locker (pick(?truck ?package ?location1)), a releaser (drop(?truck ?package ?location2)) and a gluing action (drive(?truck ?location1 ?location2)) and hence is classified as *gluing*.

4.2 Single-Phased Multi-lock Critical Section Macros

Operators involved in one CSM might be lockers or releasers for other CSMs. That said, a SpSl CSM might contain a part of another CSM (could be more than one). Consequently, some CSMs might overlap, i.e., sharing a common subsequence of actions, and such a situation might be impractical during planning, as the planner might use only one of these macros. To give an example, in the Storage domain we can find two SpSl CSMs (parameters are omitted for the sake of clarity): go_out-lift-go_in (see Section 4.1) and lift-go_in-drop, where the hoist lifts a crate, moves from the transit area to a store area and then drops the crate. These CSMs overlap in the lift and go_in actions.

To deal with this issue we introduce *Single-phased Multi-lock (SpMl) CSMs* that contain multiple lockers and releasers.

Definition 11. *Let Π be a planning task and ats_Π be atoms and A_Π be actions (defined in Π). Let $a_1, a_2, \ldots, a_k \in A_\Pi$ be actions. We say that $a_{1,\ldots,k}$ is a* ***Single-phased Multi-lock Critical Section Macro*** *over a set of locks $p^1, q^1, p^2, q^2, \ldots, p^l, q^l$ ($p^1, q^1, p^2, q^2, \ldots, p^l, q^l \in ats_\Pi$) if it is a macro over a sequence of actions $\langle a_1, \ldots, a_k \rangle$ such that*

(i) a_1 *is a p^i, q^i-locker*

(ii) a_i*, where $1 < i < k$, is either*

- *(a) a q^x-user, where there exists $j < i$ such that a_j is a p^x, q^x-locker and there exists $m > i$ such that a_m is a p^x, q^x-releaser*
- *(b) a p^v, q^v-releaser, where $1 \leq v \leq l$, there exists $1 \leq m < i$ such that a_m is a p^v, q^v-locker*
- *(c) a potentially gluing action which is neither a q^x-user nor a p^v, q^v-releaser as in (a) or (b), respectively (a gluing action, for short)*
- *(d) a p^v, q^v-locker, where $1 \leq v \leq l$, there exists m, where $i < m \leq k$, such that a_m is a p^v, q^v-releaser, and a_i is a q^x-user ($x \neq v$), a gluing action, or a p^x, q^x-releaser ($x \neq v$).*

(iii) a_k *is a p^j, q^j-releaser*

Let $locks(i) = \{a_j \mid j \leq i, a_j$ *is a* p^v, q^v*-locker*$, 1 \leq v \leq l\}$ *be a set of lockers up to the index* i *and* $rels(i) = \{a_j \mid j \leq i, a_j$ *is a* p^v, q^v*-releaser*$, 1 \leq v \leq l\}$ *be a set of releasers up to the index* i*. It is the case for each* $1 \leq i \leq k-1$ *that* $|locks(i)| > |rels(i)|$*.*

Returning to the above Storage domain example, we can construct a SpMl CSM – go_out-lift-go_in-drop (parameters are omitted for the sake of clarity) representing that a hoist moves to a transit area, lifts a crate from an adjacent store area, returns back to the store area the hoist was before and drops the crate to a (different) adjacent store area. By constructing such a CSM, we effectively deal with the above mentioned overlap.

Note that if a SpMl CSM starts with a p^i, q^i-locker and ends with a p^i, q^i-releaser, it also classifies as a SpSl CSM, where the other lockers and releasers count as users or gluing actions.

4.3 Multi-phased Single-lock Critical Section Macros

More complex situations involve multi-phased locks, that is, a sequence of actions need to be applied to get the lock and similarly a sequence of other actions need to be applied to release the lock. *Multi-phased Single-lock (MpSl) CSMs* are defined as follows.

Definition 12. *Let* Π *be a planning task and* ats_Π *be atoms and* A_Π *be actions (defined in* Π*). Let* $a_1, a_2, \ldots, a_k \in A_\Pi$ *be actions. We say that* $a_{1,\ldots,k}$ *is a* ***Multi-phased Single-lock Critical Section Macro*** *over a multi-phased lock* $p^1, p^2, \ldots, p^l$ *(*$p^1, p^2, \ldots, p^l \in ats_\Pi$*) if it is a macro over a sequence of actions* $\langle a_1, \ldots, a_k \rangle$ *such that*

(i) there exist $1 = l_1 < \cdots < l_m < r_1 < \ldots r_n = k$ *such that* $a^{l_1}, \ldots, a^{l_m}$ *forms a* $p^1, p^2, \ldots, p^l$*-locker and* $a^{r_1}, \ldots, a^{r_n}$ *forms a* $p^1, p^2, \ldots, p^l$*-releaser*

(ii) a_i*, where* $1 < i < k$ *and* $i \notin \{l_1, \ldots, l_m, r_1, \ldots, r_n\}$*, is a* p^x*-user* $(1 \leq x \leq l)$*, or a potentially gluing action not being a* p^x*-user (a gluing action, for short)*

For example, in the Transport domain, a MpSl CSM, is pick-pick-drive-drop-drive-drop. The macro represents that a truck initially picks up two packages at the same location and delivers the first package to a location adjacent to the location of origin, and the second package to a location adjacent to the first delivery location.

Note that if a MpSl CSM starts with a p^1, p^2-locker and finishes with a p^1, p^2-releaser (it is always the case for balanced locks), it is also a SpSl CSM (the other actions being part of the phased lock will count as gluing actions).

It is possible to define *Multi-phased Multi-lock (MpMl) CSMs* that, roughly speaking, would combine Definitions 11 and 12. To give an example, in the Barman domain, we can have two SpSl CSMs, i.e., 1) grasping a shot, filling a shot by an ingredient, pouring the shot to the shaker, cleaning the shot, filling the shot by another ingredient, pouring the shot to the shaker, cleaning the shot, leaving the shot, 2) grasping a shaker, shaking a cocktail, serving the cocktail, emptying, cleaning and leaving the shaker, and one MpSl CSM, i.e., 3) pouring the shot to the shaker, cleaning the shot, filling the shot with another ingredient, pouring the shot to the shaker, cleaning the shot, leaving the shot, grasping the shaker, shaking the cocktail, serving it and emptying the shaker. For the first and second CSM, the limited resource is a barman's hand which holds a shot in the first case and the shaker in the second case. For the third macro, the limited resource is the capacity of the shaker. The

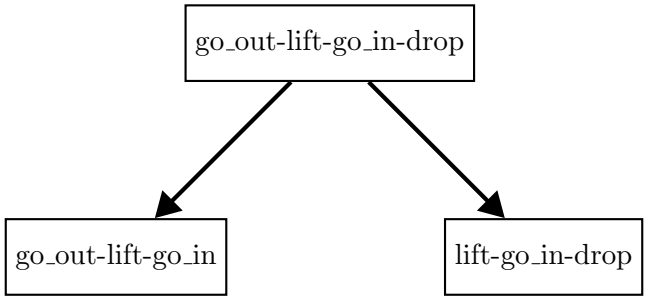

Figure 1: Example of a SpMl CSM in the Storage Domain.

shaker is filled by two pouring actions and emptied by the "pouring to shot" action and the "empty" action. It can be observed that the third macro overlaps with the other two. To deal with the overlap, we can generate a MpMl CSM, i.e., grasping a shot, filling a shaker with ingredients and leaving the shot, grasping the shaker, shaking a cocktail, serving the cocktail, cleaning the shaker and leaving the shaker.

5. Compound Critical Section Macros

As macros are encoded as ordinary actions, this fact can be leveraged for assembling more complex CSMs from simpler ones. In particular, we might only need a method that learns SpSl CSMs (we refer to this type of macros as simple CSMs) and by iterative use of the method, we can construct more complex CSMs (i.e., SpMl, MpSl or even MpMl CSMs). Also, as we learn CSMs from training plans, which might be noisy (e.g., contain some peculiarities such as very suboptimal action sequences, or redundant actions), it seems to be a more feasible option to learn CSMs incrementally as shorter macros might be easier to identify in noisy plans.

Initially, we define the notion of *compound CSMs* that captures the property where one CSM (strictly) incorporates a simpler CSM.

Definition 13. *Let $a_{1,\ldots,k}$ be a Critical Section Macro over the sequence of actions $\langle a_1, \ldots, a_k \rangle$. Let $a_{i,\ldots,i+m}$ be another Critical Section Macro over the sequence of actions $\langle a_i, a_{i+1} \ldots, a_{i+m} \rangle$ such that $1 \leq i < i+m \leq k$. Then, we say that $a_{1,\ldots,k}$ **incorporates** $a_{i,\ldots,i+m}$. If $(m-1) < k$, then we also say that $a_{1,\ldots,k}$ **strictly incorporates** $a_{i,\ldots,i+m}$*

*We say that a Critical Section Macro cm is **compound** if there exists a Critical Section Macro m such that cm strictly incorporates m.*

As can be observed from Definition 13, Compound CSMs form hierarchies which are defined by the relation of strict incorporation.

Definition 14. *Let m be a (compound) Critical Section Macro. We say that $\mathcal{H}(m)$ is a* **Critical Section Macro hierarchy** *over m, where $\mathcal{H}(m) = (N, E)$ is a directed graph such that $N = \{m_i \mid m \text{ incorporates } m_i\}$ is a set of nodes and $E = \{(m_i, m_j) \mid m_i, m_j \in N, m_i \text{ strictly incorporates } m_j, \nexists m_k : m_i \text{ strictly incorporates } m_k \wedge m_k \text{ strictly incorporates } m_j\}$ is a set of edges. We say that a Critical Section Macro m_p is* **primitive** *with respect to $\mathcal{H}(m)$ if and only if there is no outgoing edge from the m_p node in $\mathcal{H}(m)$.*

For the recent Storage example, the SpMl CSM – go_out-lift-go_in-drop – has two primitive successors in its CSM hierarchy, i.e., go_out-lift-go_in and lift-go_in-drop, as depicted in Figure 1.

Since macros are encoded as ordinary actions, we can leverage such a property to condense more complex CSMs by replacing a strictly incorporated action sequences forming CSMs by actual macros.

Definition 15. *Let $a_{1,\dots,i,\dots,i+m,\dots,k}$ be a Critical Section Macro over the sequence of actions $\langle a_1, \dots, a_i, \dots, a_{i+m}, \dots, a_k\rangle$. Let $a_{i,\dots,i+m}$ be another Critical Section Macro over the sequence of actions $\langle a_i, a_{i+1} \dots, a_{i+m}\rangle$ such that $a_{1,\dots,i,\dots,i+m,\dots,k}$ strictly incorporates $a_{i,\dots,i+m}$. We say that a Critical Section Macro $a_{1,\dots,\{i,\dots,i+m\},\dots,k}$, over $\langle a_1, \dots, a_{i,\dots,i+m}, \dots, a_k\rangle$, is a* **condensation** *of $a_{1,\dots,i,\dots,i+m,\dots,k}$ with respect to $a_{i,\dots,i+m}$.*

We say that a Critical Section Macro is **maximally condensed** *if its Critical Section Macro hierarchy contains exactly one node.*

Lemma 16. *For any triplet of Critical Section Macros, m_c, m, m', such that m_c is a condensation of m with respect to m', it is the case that m_c and m are equivalent.*

Proof. The claim immediately follows from Definition 4. □

The above definition gives an intuition of how more complex CSMs can be incrementally constructed from simpler CSMs. In particular, we show how more complex CSMs can be generated while having only a method for generating simple (SpSl) CSMs. The case of SpMl CSMs is shown in the following proposition.

Proposition 17. *For each Single-phased Multi-lock Critical Section Macro m, there exists an equivalent Single-Phased Multi-lock Critical Section Macro m' whose Critical Section Macro hierarchy $\mathcal{H}(m')$ has the following property. For each node m_n in $\mathcal{H}(m')$ (it holds that m' incorporates m_n) it is the case that a maximally condensed Critical Section Macro equivalent to m_n is a Single-phased Single-lock Critical Section Macro.*

Proof. Initially, we show that for each SpMl CSM m we can find an equivalent SpMl CSM m' such that for each lock considered in m, m' incorporates a SpSl CSM over that lock.

Without loss of generality let us assume that p, q is a lock and $\langle a_i, \dots, a_{i+r-1}, a_{i+r}, a_{i+r+1}, \dots, a_{i+m}\rangle$ is a subsequence of the sequence of actions m is assembled from, where a_i is a p, q-locker, a_{i+m} is a p, q-releaser and a_{i+r} is neither a p, q-user nor a gluing action (as in ii) of Definition 10). Hence, the action subsequence does not form a CSM because of a_{i+r}. As a_{i+r} is not a potentially gluing action (see Def. 9) there are two possibilities. If it is the case that a_{i+r} is independent with all actions $a_i, \dots, a_{i+r-1}$ and no action a_{i+j} $(0 \leq j < r)$ is an achiever for a_{i+r}, then we can move a_{i+r} before a_i. The independence condition ensures that a_{i+r} will not delete preconditions for $a_i, \dots, a_{i+r-1}$ as well as these actions will not delete add effects of a_{i+r}. Also, these actions are not important to satisfy the precondition of a_{i+r} as none is the achiever for it . Analogously, if it is the case that a_{i+r} is independent with all actions $a_{i+r+1}, \dots, a_{i+m}$ and an achiever for none of them, then we can move a_{i+r} after a_{i+m}.

Hence, m' can be constructed from m by moving the a_{i+r} actions as above. Note that since the a_{i+r} actions have to meet condition (ii) in Definition 11 they cannot be moved outside the macro (i.e., either before the first locker or after the last releaser).

Next, we show that for each node m_c of $\mathcal{H}(m')$ it is the case that its maximally condensed equivalent is a SpSl CSM. If m_c starts with a locker and ends with a releaser for the same lock(s), then m_c is also a SpSl CSM (see Definition 10). It is straightforward to see that a maximally condensed CSM, equivalent to m_c, is also a SpSl CSM. If m_c starts with a locker and ends with a releaser for at least one different lock, then there exist two CSMs that are successors of m_c in $\mathcal{H}(m')$ as follows (note that more than the two following successors might exist). Let us assume that m_c is a CSM over $\langle a_i, \ldots, a_j \rangle$. Let k be the smallest k such that $k > i$ and $\langle a_k, \ldots, a_j \rangle$ forms a CSM. Consequently, this CSM is a successor of m_c in $\mathcal{H}(m')$. Such k exists because if a_j is a releaser for a lock that a_i is not a locker for, then a different action in m_c has to be that locker. Note that a_k does not necessarily have to be that locker, it might be a locker for a different lock of m_c. We can also observe that there is a releaser a_u with $k \leq u < j$ for a lock for which a_i is a locker but a_j is not a releaser. If a_u was placed before a_k, then all locks would have been released before a_k and hence m_c would not have been a valid CSM (see Definition 11). A condensation of m_c with respect to $\langle a_k, \ldots, a_j \rangle$ is a SpSl CSM since the CSM over $\langle a_k, \ldots, a_j \rangle$ is a releaser of lock(s) that a_i is the locker for (as the releaser a_u is part of the incorporated CSM).

We can also find the largest l such that $l < j$ and $\langle a_i, \ldots, a_l \rangle$ forms a CSM, which is a successor of m_c in $\mathcal{H}(m')$. In analogy to the previous case, we can observe that there is a locker a_v with $i < v \leq l$ for a lock for which a_j is a releaser but a_i is not a locker. A condensation of m_c with respect to $\langle a_i, \ldots, a_l \rangle$ is a SpSl CSM since the CSM over $\langle a_i, \ldots, a_l \rangle$ is a locker of lock(s) that a_j is the releaser for.

It can be straightforwardly observed that a successor of SpMl CSM in a CSM hierarchy has at least one less lock and all primitive CSMs are SpSl CSMs (otherwise they would have had at least two successors as described above).

□

To understand the practical consequences of the above proposition (including the proof), we recall the Storage example. We also assume that we can only identify SpSl CSMs. Hence sequences in form – go_out-lift-go_in-drop – would not be immediately identified as SpMl CSMs. Yet we can identify subsequences – go_out-lift-go_in – and – lift-go_in-drop – as SpSl CSMs (see Figure 1). We can condense both subsequeces into macros. As the subsequences overlap, only one of them would be a part of the SpMl macro. For the first subsequence, we get CSM1-drop. CSM1 inherits the locker property from the lift action and hence CSM1-drop forms a compound SpSl CSM. For the second subsequence, we then get – go_out-CSM2. CSM2 inherits the releaser property from the go_in action and hence go_out-CSM2 also forms a compound SpSl CSM.

In a similar fashion, MpSl CSMs, where multi-phased locks are balanced, can be incrementally constructed from SpSl CSMs.

Proposition 18. *For each Multi-phased Single-lock Critical Section Macro m, where the lock is balanced, there exists an equivalent Multi-phased Single-lock Critical Section Macro m' whose Critical Section Macro hierarchy $\mathcal{H}(m')$ has the following property. For each node m_n in $\mathcal{H}(m')$ (it holds that m' incorporates m_n) it is the case that a maximally condensed Critical Section Macro equivalent to m_n is a Single-phased Single-lock Critical Section Macro.*

Proof. In analogy to proof of Proposition 17, we can obtain m' from m by moving actions that are neither users nor gluing actions before the lockers or after the releasers.

Let m' be a MpSl over $p^1, \ldots, p^k$-lock. Let m^i be a CSM over $p^i, \ldots, p^k$-lock with $1 \leq i < k$ such that m' incorporates m^i. We can observe directly from Definition 12 that m^i strictly incorporates m^j for every $1 \leq i < j < k$. We can also observe that a condensation of m^i with respect to m^{i+1} $(1 \leq i < k-1)$ is a maximally condensed SpSl CSM over p^i, p^{i+1}-lock (m^{i+1} becomes a p^{i+1}-user). Finally, we can observe that m^{k-1} is a SpSl CSM over (single-phased) p^{k-1}, p^k-lock, which is primitive in $\mathcal{H}(m')$. □

For MpSl CSMs, the CSM hierarchy is linear, so the bottom-up construction of MpSl CSMs is straightforward. In Transport we might, for example, initially generate a SpSl CSM – pick-drive-drop, which is condensed to CSM1. Then we might, for example, generate a compound SpSl CSM – pick-CSM1-drive-drop – which is equivalent to MpSl CSM – pick-pick-drive-drop-drive-drop. Technically, as we deal with balanced locks the macro in each level of the hierarchy is a SpSl CSM. Hence, it might be possible to identify MpSl CSMs with balanced locks at once even if we assume that we can only identify SpSl CSMs. In practice, however, the bottom-up incremental approach might be useful to deal with noise in training plans or to limit the number of macros' arguments (see Section 6).

By combining proposition 17 and 18 we can also construct MpMl CSMs, where multi-phased locks are balanced.

6. Constructing Critical Section Macros from Training Plans

To learn CSMs from training plans, we have to determine locks, lockers and releasers. Since macro-operators are a domain-specific (or class-of-problems-specific) kind of knowledge, locks, lockers and releasers have to be in lifted form. We capture the information in quadruples (p, q, O_l, O_r), where p, q are predicates and O_l, O_r are sets of operators defined in the given domain model. Then, for each p_g and q_g being instances of p and q, respectively, and forming a lock, it is the case that for each operator $o_l \in O_l$ there exists its instance being a p_g, q_g-locker and for each operator $o_r \in O_r$ there exists its instance being a p_g, q_g-releaser. The quadruples (p, q, O_l, O_r) are determined by considering whether each operator, defined in the domain model, that deletes p (resp. q) adds the corresponding variant of q (resp. p). Such operators then form the O_l (resp. O_r) set. On top of that, corresponding instances of p and q must not be simultaneously present in initial states of the considered planning tasks (otherwise the instances of p and q are not mutex). Note that our notion of mutex corresponds with the notion of Fact Alternating Mutex (Fiser & Komenda, 2018).

6.1 Constructing Single-phased Single-lock Critical Section Macros

Algorithm 2 describes the method for learning SpSl CSMs from a set of training plans. Initially, quadruples (p, q, O_l, O_r) are determined (Line 1). For each training plan, we determine all possible locker/releaser pairs (a_l, a_r) with corresponding instances of the locks (p_g, q_g) (Line 3). Then, we iterate through the locker/releaser pairs (Lines 4–14). Besides a_l (p_g, q_g-locker) and a_r (p_g, q_g-releaser) we consider q_g-users into a possible macro-action (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

Algorithm 2 Learning (SpSl) CSMs 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** *(optional) some gluing action added an extra argument* **then**
9: **continue**
10: **end if**
11: $ma \leftarrow$CreateMacro(in_ma)
12: ConsiderGoalAchieving(ma)
13: ConsiderConnected(ma)
14: AddMacro(mcr_db,ma)
15: **end for**
16: **end for**
17: FilterUnderrepresentedMacros(mcr_db,ν_1)
18: RefineMacroOperators(mcr_db)

19: **function** ConsiderDependent(in_ma,out_ma)
20: **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**
21: $out\_ma \leftarrow out\_ma \setminus \{a_k\}$
22: **end while**
23: **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**
24: $out\_ma \leftarrow out\_ma \setminus \{a_k\}$
25: **end while**
26: $in\_ma \leftarrow in\_ma \cup out\_ma$
27: **end function**

idea of how intermediate actions can be moved away, formalised in Definition 9, 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 (a similar approach has been used by MUM (Chrpa et al., 2014)). Those actions that cannot be moved away are *gluing* actions (see ii) in Definition 10). See Section 4.1 for details. Optionally, we might not consider macro-actions where some of the gluing actions introduced one or more extra arguments (Line 8) to prevent a possibly large number of instances of macro-operators, refined from such macro-actions (Line 18). After the macro-action is created by assembling the sequence of actions in in_ma (Line 11), it is checked whether it is *goal achieving*, i.e., whether some of its add effects are goal atoms. Goal achieving macro-operators (refined from goal-achieving

Algorithm 3 A high-level routine for learning SpSl CSMs

1: Generate training plans by solving training problem instances
2: Learn (SpSl) CSMs by applying Algorithm 2 and add the macro-operators into the domain model
3: Generate plans by solving training problem instances with the enhanced domain model
4: Eliminate macro-operators that are underrepresented in the plans, i.e., their number is below ν_2
5: Learn and apply Outer Entanglements on the remaining macro-operators

macro-actions), in fact, are entangled by goal with some predicate whose instances are present in the goal. On top of that, if a macro-action is goal achieving we check whether the macro-action is *connected*, i.e., its precondition atoms that contain some of the arguments of the goal achieving atoms are also present in the initial state. That is, we try to identify whether there are predicates a macro-operator (refined from connected macro-actions) is entangled by init with such that each of the predicates has at least one shared argument with a predicate the macro-operator is entangled by goal with. Note that the concept of connected macros was discussed by Chrpa, Vallati, and McCluskey (2015).

A macro database (*mcr_db*) groups together macro-actions that are assembled from the same action sequence, have the same argument binding, are goal achieving for the same predicates (or not goal achieving at all) and are connected by the same predicates (or not connected at all). Before refining macro-operators from the (remaining) groups of macro-actions in the macro database (Line 18), which is done analogously to other macro learning techniques (Botea et al., 2005; Chrpa, 2010b), macro-actions that are underrepresented, i.e., their number in the macro database is below a specified threshold ν_1, are filtered out (Line 17). Note that underrepresented macro-operators, in the Machine Learning terminology, can be understood as noise in training data. They are, for example, problem-specific macros that do not generalise for a class of planning problems, or capture 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 et al., 2014) also eliminate underrepresented macros for the same reason. If the macro database contains more groups referring to a macro assembled from the same action sequence (but there are differences in the constraints) we prefer the most constrained version of the macro (i.e., we prefer connected over goal achieving and goal achieving over unconstrained macros).

The whole procedure of learning (SpSl) CSMs is summarised in Algorithm 3 (the procedure is inspired by BloMa (Chrpa & Siddiqui, 2015)): It might often be the case that the number of learned macro-operators (after step 2) is rather large and there also might be overlaps in operator sequences macros are assembled from. Generating the plans again, but now with the enhanced domain model, will identify possibly useful and useless macros, according the frequency of their use in the plans (ν_2 is the threshold). Note that such a strategy has also been exploited by MacroFF (Botea et al., 2005) and BloMa (Chrpa & Siddiqui, 2015). Applying Outer Entanglements on the remaining macros has been shown to be a useful strategy to reduce the number of their possible instances (Chrpa, 2010a) and such a strategy was also used by BloMa (Chrpa & Siddiqui, 2015).

Algorithm 4 A high-level routine for learning compound CSMs

1: Generate training plans by solving training problem instances (with the enhanced domain model since the second iteration)
2: Learn CSMs by applying Algorithm 2 (macros in the current enhanced domain model are considered as ordinary operators)
3: If new macro-operators are generated, enhance the domain model and go to step 1
4: Generate plans by solving training problem instances with the enhanced domain model
5: Eliminate macro-operators that are underrepresented in the plans, i.e., their number is below ν_2
6: Learn and apply Outer Entanglements on the remaining macro-operators

6.2 Constructing Compound Critical Section Macros

Compound CSMs are composed from primitive actions/operators as well as other CSMs (see Definition 13). That said, compound CSMs can be constructed from simpler CSMs. The consequence of Proposition 17 is that we can incrementally build any "interesting" SpMl CSM[3] by repeatedly using the algorithm for identifying SpSl CSMs (Algorithm 2). Analogously, the consequence of Proposition 18 is that we can incrementally build any "interesting" MpSl CSM with a balanced lock. Consequently, we can incrementally build any "interesting" MpMl CSM with balanced multi-phased locks.

To summarise the procedure for incrementally generating compound CSMs as in Algorithm 4, SpSl CSMs are initially learned by Algorithm 2, then these CSMs are encoded into the domain model as ordinary operators, then Algorithm 2 is applied again to learn compound CSMs, which if generated are also encoded into the domain model, and the process continues until no new (compound) CSMs are generated. At the end, underrepresented CSMs are eliminated and Outer Entanglements applied on the remaining CSMs in the same way as the SpSl CSM generation procedure does in Algorithm 3.

Technically, in each iteration of the compound CSM learning method, we move up in CSM hierarchy. Initially, we can identify only SpSl CSMs. Note that SpSl CSMs might not be necessarily primitive, i.e., even a SpSl CSM might strictly incorporate another CSM. In our Storage example (see Figure 1), we can initially learn primitive SpSl CSMs – go_to_transit-lift-go_back and lift-go_back-drop. In the second iteration, we can learn SpMl CSM – go_to_transit-lift-go_back-drop.

Besides generating "interesting" complex CSMs, the incremental method for generating compound macros has two practical benefits. One benefit is about being more robust to noise in training plans. Shorter macros can be identified more easily. For longer macros the issue is that, besides higher "chance" for noise to prevent identifying such macros, longer macros are (much) less frequent despite being possibly (very) useful. Hence, it is more likely that trying to identify more complex CSM at once might fail due to noise in training plans. The other, perhaps not so obvious, benefit is in keeping the number of candidate macros reasonably low. If we generate all possible CSMs (including the complex ones) at once, the number of macros might be (very) high. Since between different CSMs there will be a lot

3. For each SpMl CSM which cannot be incrementally built there exists an equivalent SpMl CSM we can incrementally build (see Proposition 17 for details).

of overlaps (a simple example of overlaps can be seen in the Storage domain), only a small number of candidate CSMs will be useful. However, with a large number of complex CSMs even training tasks might be difficult to solve for planners and hence step 4 of Algorithm 4 would have likely failed in some occasions.

6.3 Aggressive Approach

Adding sound macro-operators into a domain model does not compromise completeness. On the other hand, the size of (grounded) representation can considerably grow as macros often have more instances than ordinary operators, due to a 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. CSMs 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 a variant of the Transport domain, where all locations are connected with each other, a CSM pick-drive-drop replaces original operators pick and drop (unless some truck initially carries some package or it is required that some truck carries a package in the goal).

The aggressive version of our CSM approach consists of the following steps:

1. Generate CSMs (simple or compound) and add them into the domain model.
2. Remove the first and the last operators of each CSM (i.e., lockers and releasers) from the domain model.
3. Generate plans for the training problem instances 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, by removing original operators the aggressive approach can (sometimes considerably) reduce the size of the representation, and prune the search space. In the above Transport example, removing the pick and drop operators prunes out states in which a package is inside a truck 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).

7. Experimental Evaluation

This experimental analysis aims at: (i) assessing the capabilities of the introduced approaches to extend domain models with CSMs that can improve the performance of domain-independent planning engines; (ii) comparing the performance gap between the complete

and incomplete (aggressive) CSMs, and (iii) comparing the proposed CSMs with state-of-the-art techniques, MUM (Chrpa et al., 2014) and BloMa (Chrpa & Siddiqui, 2015).

7.1 Benchmarks and Planning Engines

We considered a range of well-known benchmark domains from both deterministic and learning tracks of IPCs. In particular: Elevators, Floortile, GED, Hiking, Termes, and Transport from the deterministic track of IPCs 2011, 2014 and 2018, and Barman, Blocksworld (Bw), Depots, Gold Miner (Gold), Gripper, Matching-Bw, Rovers, Sokoban, and Thoughtful from the learning track of IPCs 2008 and 2011. We have also considered the Storage domain from IPC 2006, that was used for evaluating the BloMa technique (Chrpa & Siddiqui, 2015). We did not include in the experimental analysis domains from the aforementioned competitions, where no CSMs were generated, and domains that include unsupported ADL features (we partially support only negative preconditions). Note that we modified the Transport domain such that the Drive action has constant cost of 20 since our implementation does not fully support cost increase by numeric fluents.

As testing instances, for each domain we used those exploited in IPCs. There are 20 instances for the domains included in the deterministic tracks (except Storage), and 30 instances for the learning track benchmarks and Storage.

We selected 8 state-of-the-art planning engines, according to their results in the IPCs while considering a diverse range of planning techniques they leverage, namely: *FF* (Hoffmann & Nebel, 2001), *LAMA* (Richter & Westphal, 2010), *Probe* (Lipovetzky, Ramirez, Muise, & Geffner, 2014), *MpC* (Rintanen, 2014), *Mercury* (Katz & Hoffmann, 2014), *Yahsp3* (Vidal, 2014), *FDSS 2018* (Seipp & Röger, 2018) and *Dual BFWS* (Lipovetzky, Ramirez, Frances, & Geffner, 2018). We aimed at including planning engines exploiting very different planning techniques, to better capture the impact of the generated macros on a wide range of approaches.

7.2 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 a time limit of 900 seconds and a memory limit of 4 GB is applied (as in the learning tracks of IPCs). All the experiments were conducted on Intel Xeon E5-2620 v4 2.10 GHz with 32GB RAM.

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 $PAR10(p, \Pi) = t$, otherwise $PAR10(p, \Pi) = 10T$ (i.e., 9000s in our case).

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

7.3 Learning

Similarly to other macro learning techniques (Chrpa et al., 2014; Chrpa & Siddiqui, 2015), we considered 6 training tasks per each domain such that their plan length was mostly within 30-80 actions[4]. One training plan was considered per training task.

For each individual domain, out of all considered planners, a planner which generates best quality training plans, i.e., the minimum sum of the lengths of the plans or the minimum sum of the costs of the plans (depending whether action costs are/not considered), is selected to generate training plans for that domain. This methodology, used also by BloMa (Chrpa & Siddiqui, 2015), follows an intuition that good quality training plans yield to most promising knowledge for all planners rather than when each planner generates training plans for itself, which has been empirically verified in the recent work (Chrpa & Vallati, 2019).

The thresholds for underrepresented macros, ν_1 and ν_2 (see Algorithms 2,3 and 4) were set according to results of preliminary experiments. In particular, ν_1 was set to maximum of 1/2 of the number of the training tasks and 1/3 occurrences of the most frequent macro, while ν_2 was set to the number of training tasks. The thresholds are set more conservatively than in the BloMa approach (Chrpa & Siddiqui, 2015) as CSMs are more "focused". The number of occurrences of the most frequent macro identified in training plans indicates how many times a useful macro should occur. Setting the ν_1 threshold to 1/3 of the occurrences of the most frequent macro turned up to be a reasonable compromise (note that BloMa considered the threshold of 1/2 of the occurrences of the most frequent macro). In Floortile, for example, the most frequent macro has 108 occurrences while the second most frequent macro has 23 occurrences (there were 29 detected macros in total). Although out of these 29 macros only 16 passed the lower threshold, i.e., 1/2 of the training tasks, they still possessed considerable burden for the planning process. The ν_1 threshold, however, eliminated all the identified macros, except the most frequent one, which had improved the planning process (especially, in the aggressive version). The lower threshold of ν_1, that is, 1/2 of the number of the training tasks is very conservative, on the other hand, it can still effectively filter out useless macros that occur once or twice and rather than general knowledge they reflect peculiarities of individual training plans. On the other hand, such a low threshold allows to consider possibly useful macros. A good example can be found in the Hiking domain, where one useful macro has initially only 4 occurrences (the most frequent macro has 7). Note that that macro after generating training plans with macros (step 3 in Algorithm 3) occurred 12 times in the (macro-enhanced) training plans and hence the macro passed the ν_2 threshold. That threshold (ν_2) ensures that there is at least one macro per training problem in average which has shown to be reasonable as it eliminates possibly useless macros but conservative enough to keep more complex (often compound) macros that are useful but not that frequent.

The learning process usually took at most several seconds with some notable exceptions such as compound CSM generation in Sokoban and Transport, where the learning process took more than 10 minutes. Also, in Hiking during compound CSM generation the planner (FF) crashed (buffer overflow) in Step 4 of Algorithm 4. Note that the plan generation is

4. 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. For the deterministic track domains, we used problem instances from the optimal track, if available, or the satisficing track otherwise.

the most computationally expensive part of the learning process, and can be done offline – so it does not have any impact on the performance of the planning engines that are going to exploit generated macros[5].

7.4 Complete (Conservative) Approaches

Table 1 gives an overview of the average performance achieved by the considered planning engines on the selected benchmarks, when using the original domain model (O) or the models extended with the learnt CSMs. In particular, extended with simple CSMs with (C) and without argument limit (nC), i.e., whether the extra argument check on Line 8 of Algorithm 2 is enabled or not, and compound CSMs with (cC) and without argument limit (ncC). Detailed results, showing planning engine by domain performance, are provided in Appendix A.1.

Considering the coverage results shown in Table 1, it is easy to notice that the use of models extended with CSMs can usually considerably improve the performance of domain-independent planning engines. Only in four domains, namely Termes, Transport, Goldminer and Thoughtful, the use of macros leads to a slight underperformance when compared to the original domain model. The only case in which the use of macros is significantly reducing the solving capabilities of planning engines is in Transport, when compound CSMs without argument limits are used: learnt macros have a very large number of instances, that makes basically impossible to solve any instance due to its grounded size. Overall, CSMs have a very beneficial impact on planners' performance on the considered benchmarks.

Imposing argument limits for macros is not new (Botea et al., 2005; Chrpa, 2010b) and the rationale behind it is to avoid generating macros with a possibly huge number of instances. Hence it is not a big surprise that CSMs do often perform better with argument limit than without it (it is more apparent for compound CSMs, especially in the Transport domain). However, there are some exceptions to the rule. One example is the GED domain. By lifting the argument limit constraint, we are able to learn a useful CSM which captures genome cutting and splicing (in this order). On top of that, arguments of that macro are strongly intertwined by learnt entanglements and hence such a macro despite having extra arguments has only a few instances. Another example, where lifting the argument limit constraint was beneficial, concerns the Hiking domain, where a useful compound CSM was generated. The macro, assembled from 12 primitive operators, captures the activity of transporting and setting up a tent in a hikers' walking destination, performing the walk and returning the tent back to its former place. Contrary to the GED macro, arguments of the Hiking macro are only loosely intertwined. Hence, the number of instances of the Hiking macro is relatively large (about 45% of the number of instances of the original operators). Yet, such a drawback of the macro is outweighed by its length and by its (very) frequent use in plans.

Table 1 also shows PAR10 and IPC quality results. The use of models enhanced with macros is usually reflected in better PAR10 score; this comes as no surprise, since PAR10 mixes runtime and coverage – that are exactly the aspects that macros aim at improving. With regards to IPC quality, that measures a trade-off between quality of generated plans and coverage, we can see that the use of macros does not have a detrimental impact on

5. Our code and benchmarks can be found at: `https://github.com/lchrpa/CSMs`.

	Coverage					PAR10					IPC quality				
Domain	O	C	nC	cC	ncC	O	C	nC	cC	ncC	O	C	nC	cC	ncC
Elevators	16.8	19.4	19.4	18.5	18.5	1494.8	299.2	300.6	713.6	703.3	15.3	16.6	16.6	14.6	14.6
Floortile	8.4	8.5	8.5	8.5	8.5	5251.2	5192.2	5190.8	5191.2	5189.7	8.4	6.4	6.4	6.4	6.4
GED	13.8	-	15.6	-	15.6	2830.1	-	1984.1	-	1984.4	12.4	-	14.5	-	14.5
Hiking	13.5	17.0	13.3	-	17.8	3000.8	1382.3	3107.9	-	1034.1	12.4	12.2	9.4	-	11.4
Termes	6.0	5.3	5.5	5.0	3.6	6320.3	6669.4	6542.5	6787.6	7380.4	5.3	4.4	4.7	3.5	2.6
Transport	14.1	10.3	11.6	10.1	0.0	2697.6	4464.6	3858.1	4521.1	9000.0	12.7	8.9	10.1	8.9	0.0
Barman	7.1	21.9	22.0	30.0	30.0	6918.8	2521.0	2509.7	47.7	42.3	5.6	21.1	21.3	30.0	30.0
Bw	16.4	22.6	22.6	22.9	22.6	4137.1	2261.1	2268.3	2188.9	2268.8	11.7	21.3	21.3	21.4	21.3
Depots	10.9	11.8	11.8	11.8	11.9	5833.0	5535.0	5535.1	5538.3	5501.1	9.7	11.3	11.3	11.3	11.4
Gold	29.8	28.6	29.5	28.6	29.5	77.8	433.0	158.0	433.0	158.1	26.2	24.0	22.9	24.0	22.9
Gripper	0.9	11.1	11.0	11.0	11.0	8753.6	5706.3	5746.3	5746.1	5746.5	0.8	11.1	11.0	11.0	11.0
Match-Bw	15.6	27.0	27.0	27.0	27.0	4330.9	914.7	914.3	914.1	914.9	13.0	26.3	26.3	26.3	26.3
Rovers	23.3	24.3	24.1	24.3	24.0	2193.9	1902.6	1946.0	1910.8	1974.1	22.6	23.4	23.3	23.7	23.4
Sokoban	25.5	27.0	23.9	26.8	23.9	1373.7	935.8	1963.7	1004.5	1963.3	23.0	23.8	20.3	23.5	20.3
Storage	22.8	22.1	22.1	23.1	23.1	3272.5	3559.1	3559.1	3125.9	3124.5	20.2	20.2	20.2	21.7	21.7
Thoughtful	19.0	18.6	18.6	18.6	18.6	3305.5	3420.8	3421.1	3422.2	3422.2	18.3	18.0	18.0	18.0	18.0
Overall	13.5	16.2	16.1	16.6	15.8	3497.5	2738.4	2744.6	2596.9	2961.8	12.1	14.8	14.6	15.3	14.4

Table 1: Average Results: (O)riginal models with CSM enhanced models, namely CSMs with (C) and without argument limit (nC). Compound CSMs with (cC) and without argument limit (ncC). Totals do not include GED and Hiking domains, for which some set of macros were not generated. Top part of the table shows results on deterministic benchmarks, excl. Storage (20 instances per domain), bottom on learning, incl. Storage (30). Coloured cells indicate best performance for the corresponding metric.

the quality of generated plans: this can be better observed in domains where coverage is similar, like Thoughtful and Storage.

With regards to the generated CSMs, Table 2 provides an overview of the extracted set of macros, in terms of the number of CSMs per type and equality of learnt macro sets among the different approaches (where applicable). As it is apparent, in many of the considered benchmarks, the macro sets are identical: this is partially due to the structure of the domain models that might allow only some specific kinds of CSMs. Regarding types of SpSl CSMs (or simple CSMs, in other words) we can observe that Trivial CSMs considerably improve planners' performance in Bw and Matching-Bw domains. In other domains, where Trivial CSMs were generated the performance difference (with respect to the original domains) is negligible. For the Using CSMs, we can observe a great performance improvement in Barman while, in contrary, decrease of performance in Termes. Similarly, Gluing CSMs work very well in Gripper but underperform in Transport. With regards to complex CSMs, we can observe that MpMl CSMs perform very well in Hiking and Barman while the use of MpSl CSMs in Termes and Transport leads to (considerable) performance decrease. These domains are, however, the only domains in which MpMl or MpSl CSMs were generated and thus the observations we made here are inconclusive in general. Regarding SpMl CSMs we observed an increase of performance in Rovers and Storage while in Termes the performance was worse. In summary, there does not seem to be any (significant) correlation between the type of CSMs and its effect on planners' performance.

	Simple CSM				Simple CSM (NL)				Compound CSM				Compound CSM (NL)			
Domain	Triv	Use	Glu	Full	Triv	Use	Glu	Full	SpSl	SpMl	MpSl	MpMl	SpSl	SpMl	MpSl	MpMl
Elevators	0	1	1	0	Simple CSM				Simple CSM*				Simple CSM*			
Floortile	0	1	0	0	Simple CSM				Simple CSM				Simple CSM			
GED	N/A				0	0	1	0	N/A				Simple CSM			
Hiking	0	1	0	1	0	1	0	2	N/A				0	0	0	1
Storage	1	1	0	0	Simple CSM				1	1	0	0	Compound CSM			
Termes	0	3	0	0	0	4	0	0	6	1	0	0	6	0	1	0
Transport	0	0	2	0	0	0	2	0	Simple CSM				2	0	1	0
Barman	0	2	0	0	Simple CSM				2	0	0	1	Compound CSM			
Bw	3	0	0	0	Simple CSM				Simple CSM				Simple CSM			
Depots	2	0	0	0	Simple CSM				Simple CSM				Simple CSM			
Gold	0	1	0	0	0	0	0	1	Simple CSM				Simple CSM (NL)			
Gripper	0	0	1	0	Simple CSM				Simple CSM				Simple CSM			
Matching-Bw	6	0	0	0	Simple CSM				Simple CSM				Simple CSM			
Rovers	1	1	0	0	Simple CSM				1	1	0	0	Compound CSM			
Sokoban	1	0	0	0	1	0	1	0	Simple CSM				Simple CSM (NL)			
Thoughtful	1	0	0	0	Simple CSM				Simple CSM				Simple CSM			

Table 2: Numbers of generated macros per type. NL represents "no argument limit". N/A represents "no macro generated". The set name in the row (e.g. Simple CSM) represents that the given set is the same as the one stated. * means that the given set differs from the stated one by outer entanglements.

Table 3 shows how the coverage performance of the considered planning engines are affected by the use of domain models enhanced by CSMs and Compound CSMs. For both set of macros, we considered the version with argument limit as, according to Table 1, they allow to deliver the best overall performance. The coverage results confirm that the generated macros are usually beneficial for the planning engines in the majority of the benchmark domains. Interestingly, the impact of macros seems to be more related to the domain model structure, rather than on the specific characteristics of planning engines. In benchmark domains where macros are beneficial, the improvement is usually noticeable for most (all) of the planning engines. As previously discussed, in four domains, namely Termes, Transport, Gold-miner and Thoughtful, the use of macros leads to a slight underperformance when compared to the original domain model.

To give a better overview of the overall performance of the original models and of the models enhanced with macro actions, Figure 2 provides a graphical representation of coverage performance. Given a considered planning model, the figure shows, for each domain, the percentage of planning engines that solved few instances ($< 33\%$), around half of the instances (between 33% and 66%), and most of the instances ($> 66\%$). The figure presents the results achieved by the considered planners running with the original domain models (a), and with the two best CSM enhanced models –as for Table 1: CSMs with argument limit (b), and compound CSMs with argument limit (c). As it is apparent, the enhanced models are generally allowing planning engines to solve a larger number of instances within the allowed CPU-time limit.

As a rule of thumb a macro that is goal achieving and captures frequent and non-trivial activity usually considerably improves planners' performance. Besides the above GED and Hiking examples, our approach can generate a useful compound CSM in the Barman domain. The macro, assembled from 14 primitive operators, captures the whole process of cocktail preparation, i.e., putting required ingredients into the shaker, shaking

Elevators			
	O	C	cC
FF	0	20	20
LAMA	20	20	19
Probe	19	20	20
MpC	15	15	18
Mercury	20	20	20
Yahsp	20	20	13
BFWS	20	20	20
FDSS	20	20	18

Floortile			
	O	C	cC
FF	2	2	2
LAMA	6	7	7
Probe	4	4	4
MpC	20	20	20
Mercury	7	7	7
Yahsp	6	6	6
BFWS	2	2	2
FDSS	20	20	20

Hiking			
	O	C	cC
FF	9	18	-
LAMA	19	15	-
Probe	20	20	-
MpC	0	7	-
Mercury	12	20	-
Yahsp	17	18	-
BFWS	11	18	-
FDSS	20	20	-

Transport			
	O	C	cC
FF	0	1	1
LAMA	17	15	13
Probe	16	13	13
MpC	0	0	0
mMrcury	20	11	11
Yahsp	20	8	8
BFWS	20	20	20
FDSS	20	14	15

Termes			
	O	C	cC
FF	0	0	0
LAMA	14	14	13
Probe	7	6	5
Mp	0	0	0
Mercury	13	9	10
Yahsp	0	0	0
BFWS	2	3	3
FDSS	12	10	9

Barman			
	O	C	cC
FF	0	30	30
LAMA	2	30	30
Probe	5	5	30
MpC	0	0	30
Mercury	26	30	30
Yahsp	0	30	30
BFWS	0	20	30
FDSS	24	30	30

Bw			
	O	C	cC
FF	0	8	8
LAMA	28	29	29
Probe	27	30	30
MpC	0	29	29
Mercury	19	30	30
Yahsp	28	27	28
BFWS	4	3	3
FDSS	25	25	26

Depots			
	O	C	cC
FF	1	11	11
LAMA	1	0	0
Probe	30	30	30
MpC	2	2	2
Mercury	0	0	0
Yahsp	22	20	20
BFWS	11	16	16
FDSS	20	15	15

Gold			
	O	C	cC
FF	30	27	27
LAMA	30	30	30
Probe	30	30	30
MpC	30	30	30
Mercury	30	26	26
Yahsp	28	26	26
BFWS	30	30	30
FDSS	30	30	30

Gripper			
	O	C	cC
FF	0	28	28
LAMA	7	30	30
Probe	0	0	0
MpC	0	0	0
Mercury	0	25	25
Yahsp	0	0	0
BFWS	0	5	5
FDSS	0	1	0

Matching-Bw			
	O	C	cC
FF	13	27	27
LAMA	26	29	29
Probe	13	30	30
MpC	0	23	23
Mercury	10	22	22
Yahsp	21	26	26
BFWS	12	29	29
FDSS	30	30	30

Rovers			
	O	C	cC
FF	1	7	8
LAMA	30	30	29
Probe	30	29	30
MpC	16	17	16
Mercury	28	29	29
Yahsp	30	30	30
BFWS	21	22	22
FDSS	30	30	30

Sokoban			
	O	C	cC
FF	19	23	23
LAMA	21	20	20
Probe	25	30	29
MpC	30	30	30
Mercury	23	24	23
Yahsp	26	29	29
BFWS	30	30	30
FDSS	30	30	30

Storage			
	O	C	cC
FF	18	19	19
LAMA	19	20	21
Probe	21	29	29
MpC	29	18	18
Mercury	20	18	21
Yahsp	22	21	23
BFWS	30	30	29
FDSS	23	22	25

Thoughtful			
	O	C	cC
FF	17	20	20
LAMA	25	23	23
Probe	21	23	23
MpC	0	0	0
Mercury	21	17	17
Yahsp	8	6	6
BFWS	30	30	30
FDSS	30	30	30

Table 3: Coverage Results: (O)riginal models, CSMs with argument limit (C), Compound CSMs with argument limit (cC). GED domain is omitted since no macros for those sets were generated. First row of the table shows results on deterministic benchmarks; the other rows shows results on learning benchmarks. Coloured cells indicate best performance.

the cocktail and cleaning the shaker afterwards for further use. On the flip side, macros that are complex, having a lot of instances and are only occasionally used in plans have usually detrimental effects on planners' performance. It can be seen, for example, in Sokoban, where one compound CSM involves two move operators in between of two push operators. Such a macro has potentially a lot of instances while it is only occasionally used in plans. In Transport, for the no argument limit compound CSM set, the issue is much more apparent. One potentially problematic macro has four drive operators in between pickup and drop operators. Another potentially problematic macro has four drive operators in between two pickup and two drop operators. Both macros have rather huge number of instances while not being used in plans very often. This highlights one of the weaknesses of the proposed CSMs: they may not work well in domains where the critical sections are of variable length, in terms of actions that are between the locker and the releaser.

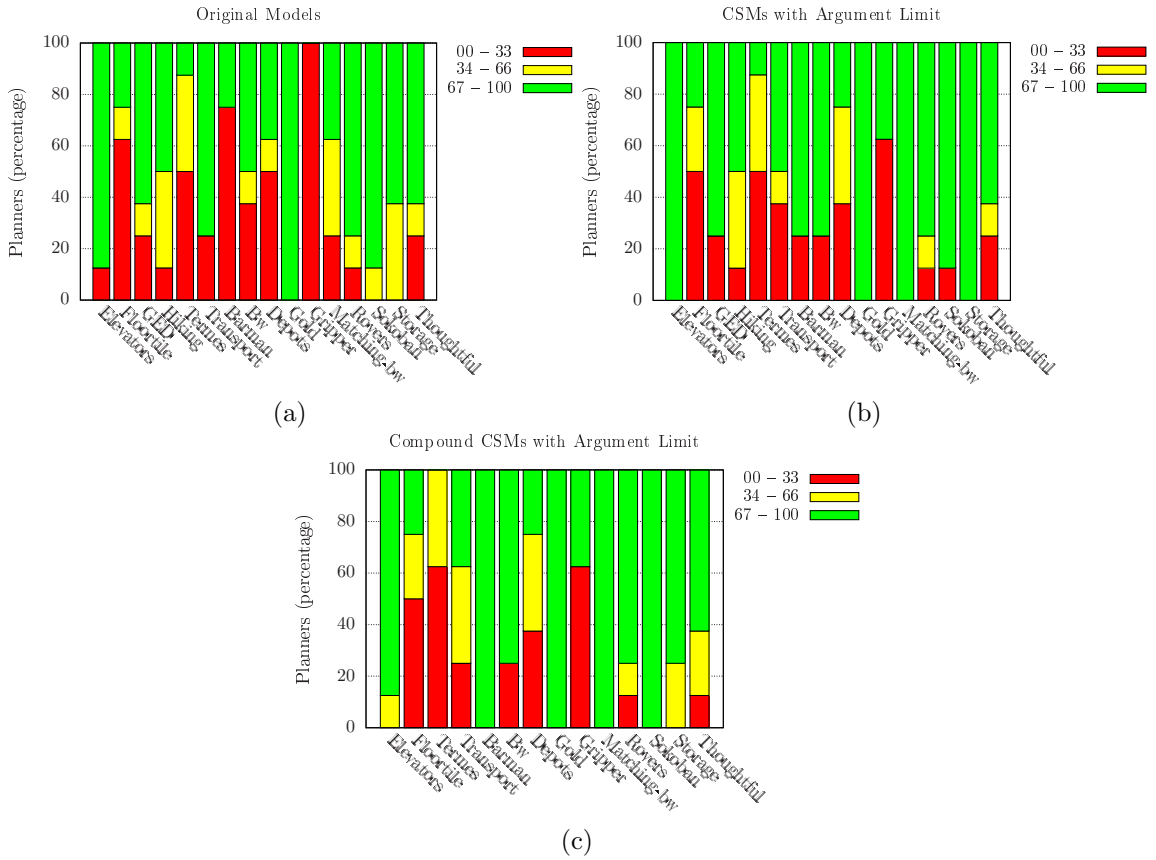

Figure 2: For each domain, the percentages of planners able to solve: more than 66% (green); between 34 and 66 % (yellow); and less than 33% of the instances (red). (a) using the original domain model, (b) using models reformulated with CSMs with argument limit, and (c) using models reformulated with Compound CSMs with argument limit.

7.4.1 Aggressive (Incomplete) Approaches

Despite the excellent improvements that can be achieved by the complete approaches, as shown in the previous section, one may argue that the aggressive approach presented in Section 6.3 may further improve the benefits of using CSMs – even though at the cost of compromising completeness. Table 4 gives an overview of the average performance achieved by the considered planning engines on the selected benchmarks, when using the original domain model (O) or the models extended with CSMs learnt by using the Aggressive version of the approaches. The table includes only domains in which the aggressive approach could remove original operators without losing solvability of any training task. Detailed results, showing planning engine by domain performance, are provided in Appendix A.2.

As expected, the use of the aggressive approach can lead to a considerable improvement of planning performance, when compared to the original models with a notable exception of the Transport domain. In most of the cases, the use of the aggressive approach allows the planning engine to solve all the benchmark instances and, noteworthy, each planner managed to solve testing instances in Floortile and Depots in less than 1 second in average

	Coverage					PAR10					IPC quality				
Domain	O	AC	AnC	AcC	AncC	O	AC	AnC	AcC	AncC	O	AC	AnC	AcC	AncC
Floortile	8.4	20.0	20.0	20.0	20.0	5251.2	0.1	0.1	0.1	0.1	8.3	18.2	18.2	18.2	18.2
Hiking	13.5	-	16.0	-	-	3000.8	-	1884.4	-	-	12.6	-	13.1	-	-
Transport	14.1	-	-	-	0.0	2697.6	-	-	-	9000.0	14.1	-	-	-	0.0
Barman	7.1	26.4	26.4	30.0	30.0	6918.8	1100.5	1100.1	3.5	3.3	5.6	26.4	26.4	30.0	30.0
Bw	16.4	25.6	25.6	25.6	25.6	4137.1	1324.8	1324.9	1324.6	1324.3	12.1	24.5	24.5	24.5	24.5
Depots	10.9	30.0	30.0	30.0	30.0	5833.0	0.1	0.1	0.1	0.1	7.2	29.8	29.8	29.8	29.8
Gripper	0.9	30.0	30.0	30.0	30.0	8753.6	44.1	44.5	44.2	43.3	0.9	29.9	29.9	29.9	29.9
Match-Bw	15.6	28.1	28.1	28.1	28.1	4330.9	562.5	562.5	562.5	562.5	12.3	27.5	27.5	27.5	27.5
Overall	9.9	26.7	26.7	27.3	27.3	5870.8	505.4	505.4	322.5	322.3	7.7	26.0	26.0	26.6	26.6

Table 4: Average Results: Results comparing the (O)riginal models with aggressive CSM enhanced models, namely CSMs with (AC) and without argument limit (AnC). Compound CSMs with (AcC) and without argument limit (AncC). Overall averages do not include Hiking and Transport domains, for which some set of macros were not generated. Top part of the table shows results on deterministic benchmarks (20 instances per domain), bottom on learning (30). Coloured cells indicate best performance for the corresponding metric.

Domain	AC	AnC	AcC	AncC
Floortile	⋆	⋆	⋆	⋆
Hiking	-	⊙	-	-
Transport	-	-	-	⊙
Barman	⋆	⋆	◇	◇
Bw	⋆	⋆	⋆	⋆
Depots	⋆	⋆	⋆	⋆
Gripper	⋆	⋆	⋆	⋆
Matching-Bw	⋆	⋆	⋆	⋆

Table 5: Comparison of sets of learned macros by the aggressive version, namely CSMs with (AC) and without argument limit (AnC). Compound CSMs with (AcC) and without argument limit (AncC). For a given domain, sets with the same symbol are identical.

(see Appendix A.2 for details). However, we have to emphasise that there is no guarantee about the completeness of such models (besides the training tasks). The aggressive approach can be used together with complete approaches in portfolios. In such a setting, we can benefit from performance boost of the aggressive approach while alleviating the risk of tasks becoming unsolvable as the complete approaches can still solve the task. In its most straightforward implementation, such a portfolio can run for half of the available CPU-time a planning engine on the enhanced domain model, and the remaining half on the original domain model. Considering the results shown for AncC in Table 4, the described portfolio would allow to maintain the excellent performance of planning engines on the vast majority of the benchmark domains, and would also allow planning engines to solve instances from the Transport domain.

Table 5 shows how the sets generated by the different approaches relate to each other. In most of the cases, the aggressive approaches are generating the same sets of macros; the only difference can be found in the Barman domain.

7.4.2 Comparison Against the State of the Art

To contextualise the performance of the proposed complete and incomplete techniques to learn CSMs, this section provides a comparison against the state of the art of techniques for learning macros for classical planning. In this analysis we consider the MUM and the BloMa approaches, that have been shown to consistently generate high quality macros. MUM (Chrpa et al., 2014) is a state-of-the-art chaining macro generation approach (macros are being constructed iteratively), which aims at keeping the number of possible instances of macros at most at the same order of magnitude as the number of instances of primitive operators they are assembled from. BloMa (Chrpa & Siddiqui, 2015) generates macros from macro-blocks, which are part of plans that are causally bound (cannot be further de-ordered). BloMa thus aims at generating longer macros that capture meaningful activities, an analogous philosophy to CSMs. For the comparison, we selected the most promising set of both conservative and aggressive CSMs by considering only the training instances. In domains with more different sets of CSMs, we selected the one having minimum product of the ratios of the number of steps in plans (macros count as one step) and the number of instantiated actions with respect to the original models for the training problem instances. The intuition behind the selection is that minimising the number of steps needed to generate plans as well as the number of actions (instances of both of macros and primitive operators) is usually beneficial for planning engines.

Table 6 gives an overview of the average performance achieved by the considered planning engines on all the selected benchmarks, when running on the original domain model, the model enhanced by macros learnt using MUM, BloMa, (conservative) CSMs (CS), and aggressive CSMs (ACS). Detailed results, showing planning engine by domain performance, are provided in Appendix A.3. The presented results indicate that ACS is generally the best option in terms of coverage and runtime: it allows planning engines to consistently deliver their best performance. There are, however, two exception to the rule. In Hiking, CS performance are slightly better while, in Transport, ACS "inherited" macros that have way too many instances, which prohibited to solve any problem. With regards to MUM and BloMa, both CS and ACS are usually able to generate macros that better improve the performance of planning engines on average. In some domains, differences are very significant, such as in Hiking, Barman and Matching-Bw. In a few cases, the models enhanced using MUM or BloMa macros allow planning engine to outperform CS and ACS, albeit marginally.

Now, we give a more detailed overview on difference between MUM, BloMa and (A)CS. MUM failed to generate macros in 6 domains. In domains such as Barman or Hiking, MUM's "instance wise" nature prevented macro generation while in Bw, interestingly, failing to identify outer entanglements in the initial step of the macro learning process resulted in failure to generate any macro. BloMa, on the other hand, succeeded to generate macros in all domains, except Floortile. All the methods generated the same set of macros in Depots and Gripper (note that small discrepancies in results are caused by different ordering of elements in enhanced domain models). In Matching-Bw, for instance, MUM and BloMa generated only 2 macros each, both being subsets of 6 macros generated by CS. BloMa

	Coverage					PAR10					IPC quality				
Planner	O	M	B	CS	ACS	O	M	B	CS	ACS	O	M	B	CS	ACS
Elevator	16.8	-	16.4	19.4	-	1494.8	-	1697.8	299.2	-	14.7	-	14.4	15.3	-
Floortile	8.4	8.8	-	8.5	20.0	5251.2	5081.4	-	5192.2	0.1	8.1	8.1	-	6.2	18.0
GED	13.8	13.0	15.0	15.6	-	2830.1	3172.8	2284.8	1984.1	-	11.6	11.1	9.7	13.7	-
Hiking	13.5	-	0.3	17.8	16.0	3000.8	-	8887.5	1034.1	1884.4	12.4	-	0.2	11.6	12.5
Termes	6.0	3.6	5.4	5.3	-	6320.3	7395.0	6610.6	6669.4	-	5.5	2.3	4.2	4.6	-
Transport	14.1	-	13.0	11.6	0.0	2697.6	-	3223.1	3858.1	9000.0	13.0	-	11.4	10.3	0.0
Barman	7.1	-	9.9	30.0	30.0	6918.8	-	6169.8	49.1	3.5	5.6	-	8.6	30.0	30.0
Bw	16.4	-	14.9	22.9	25.6	4137.1	-	4588.8	2189.8	1324.8	10.8	-	12.0	20.2	23.3
Depots	10.9	11.8	12.1	11.8	30.0	5833.0	5542.3	5424.8	5535.0	0.1	7.2	8.2	8.3	8.2	29.7
Gold	29.8	27.0	30.0	29.5	-	77.8	910.9	0.2	158.0	-	26.0	22.6	20.9	22.5	-
Gripper	0.9	11.0	11.0	11.1	30.0	8753.6	5746.9	5746.5	5706.3	44.1	0.8	11.0	11.0	11.1	29.5
Match-Bw	15.6	17.5	4.6	27.0	28.1	4330.9	3764.1	7613.5	914.7	562.5	11.8	13.9	3.6	24.0	26.7
Rovers	23.3	20.9	20.4	24.3	-	2193.9	2878.6	3027.9	1910.8	-	22.5	18.1	19.7	23.6	-
Sokoban	25.5	-	25.0	27.0	-	1373.7	-	1529.2	935.8	-	23.0	-	20.4	23.8	-
Storage	22.8	23.5	23.6	23.1	-	3272.5	2943.5	2878.1	3125.9	-	19.3	21.1	21.6	21.1	-
Thoughtful	19.0	19.3	18.8	18.6	-	3305.5	3232.6	3384.3	3420.8	-	18.1	18.1	17.9	17.7	-

Table 6: Average Results: Results comparing the (O)riginal models with macro enhanced models, namely (M)UM, (B)loMa, CSMs (CS) and the Aggressive CSMs (ACS). "-" indicates that no macros has been learnt by the considered approach. Top part of the table shows results on deterministic benchmarks, excl. Storage (20 instances per domain), bottom on learning, incl. Storage (30). Coloured cells indicate best performance for the corresponding metric.

learned macros that were "submacros" of SpSl CSMs learned in Hiking and Barman[6]. In Hiking, such a "submacro" had a very detrimental effect on planners' performance. In Storage, BloMa generated a different SpMl CSM than CS did (and BloMa's macro slightly outperformed ours).

The above observations show that state-of-the-art techniques such as MUM or BloMa are able to learn some CSMs, although as our experiments shown, only short ones. The results shown that CSMs are usually (albeit not always) beneficial to a wide range of planning techniques and hence methods tailored for generating CSMs can be very useful for classes of domains in which limited resources are considered.

8. Conclusion

Critical Section Macros, introduced and described in this paper, are inspired by their counterpart in parallel computing. They capture the whole activities in which limited resources are being used (e.g., a robotic hand holding an object). We described three variants of CSMs, the simple one with Single-phased Single-lock, and two more complex variants, one concerning multiple locks (i.e., dealing with multiple limited resources) and the other concerning multi-phased locks (e.g., a truck gets full after loading several packages). Further, we introduced the concept of compound CSMs that allows constructing more complex CSMs incrementally starting from simple ones using the method for generating SpSl CSMs and

6. By "submacros" we mean macros assembled from subsequences of operators the CSMs we refer to were assembled

have shown theoretically that by such a method we can generate all "interesting" SpMl CSMs and MpSl CSMs with balanced locks (and consequently also MpMl CSMs with balanced locks). We presented algorithms for learning (both simple and compound) CSMs from training plans, and described an aggressive variant of CSMs, where at least first and last operators from which CSMs are assembled are removed from the domain model. This can compromise the completeness of the use of CSMs, but has the potential to lead to even more substantial performance improvements.

Our empirical evaluation demonstrated the usefulness of CSMs across a wide range of benchmark domains and planning engines. The use of CSMs allowed planning engines to outperform the original model, MUM macros (Chrpa et al., 2014) and BloMa macros (Chrpa & Siddiqui, 2015) in 11 out of 16 domains. Compound CSMs considerably increased planners' performance in Barman and Hiking domains. The aggressive version of CSMs achieved the most impressive results in Floortile, Barman and Depots. The results indicate that long, goal achieving and frequent CSMs tend to improve planners' performance the most (as in Barman and Hiking).

In the future, we plan to investigate possibilities of adapting CSMs into numerical and temporal domains, possibly leveraging the notion of numerical entanglements (Chrpa, Scala, & Vallati, 2015). This may prove beneficial as numerical fluents often refer to resource use and hence the concept of CSMs can be successfully applied in numeric planning. As CSMs capture whole activities in which one or more resources are used, it might be natural to use them in temporal planning too.

Acknowledgements

We would like to thank the anonymous reviewers for their constructive feedback that helped to considerably improve the paper.

This research was funded by the Czech Science Foundation (project no. 18-07252S) and by the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics". Mauro Vallati was supported by a UKRI Future Leaders Fellowship [grant number MR/T041196/1].

References

Alhossaini, M. A., & Beck, J. C. (2013). Instance-specific remodelling of planning domains by adding macros and removing operators. In *Proceedings of SARA*, pp. 16–24.

Areces, C., Bustos, F., Dominguez, M., & Hoffmann, J. (2014). Optimizing planning domains by automatic action schema splitting. In *Proc. of ICAPS*, pp. 11–19.

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

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

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

Cardellini, M., Maratea, M., Vallati, M., Boleto, G., & Oneto, L. (2021). In-station train dispatching: A PDDL+ planning approach. In *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS*, pp. 450–458.

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

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

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

Chrpa, L., & Vallati, M. (2019). Improving domain-independent planning via critical section macro-operators. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, pp. 7546–7553.

Chrpa, L., Scala, E., & Vallati, M. (2015). Towards a reformulation based approach for efficient numeric planning: Numeric outer entanglements. In *Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS*, pp. 166–170. AAAI Press.

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

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

Chrpa, L., Vallati, M., & McCluskey, T. L. (2015). On the online generation of effective macro-operators. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, pp. 1544–1550.

Chrpa, L., Vallati, M., & McCluskey, T. L. (2018). Outer entanglements: a general heuristic technique for improving the efficiency of planning algorithms. *J. Exp. Theor. Artif. Intell.*, *30*(6), 831–856.

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

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

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

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

Fiser, D., & Komenda, A. (2018). Fact-alternating mutex groups for classical planning. *J. Artif. Intell. Res.*, *61*, 475–521.

Fox, M., Long, D., Tamboise, G., & Isangulov, R. (2018). Creating and executing a well construction/operation plan.. US Patent App. 15/541,381.

Garrido, A., Morales, L., & Serina, I. (2012). Using AI planning to enhance e-learning processes. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS*. AAAI.

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

Haslum, P. (2007). Reducing accidental complexity in planning problems. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pp. 1898–1903.

Haslum, P., & Jonsson, P. (2000). Planning with reduced operator sets. In *Proceedings of AIPS*, pp. 150–158.

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

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

Hofmann, T., Niemueller, T., & Lakemeyer, G. (2020). Macro operator synthesis for adl domains. In *Proceedings oo the European Conference on Artificial Intelligence (ECAI)*.

Junghanns, A., & Schaeffer, J. (2001). Sokoban: Enhancing general single-agent search methods using domain knowledge. *Artif. Intell.*, *129*(1-2), 219–251.

Katz, M., & Hoffmann, J. (2014). Mercury planner: Pushing the limits of partial delete relaxation. In *The Eighth IPC. Description of Participant Planners of the Deterministic Track*, pp. 43–47.

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

Lipovetzky, N., Burt, C. N., Pearce, A. R., & Stuckey, P. J. (2014). Planning for mining operations with time and resource constraints. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Lipovetzky, N., Ramirez, M., Frances, G., & 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*.

Lipovetzky, N., Ramirez, M., Muise, C., & 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*, pp. 6–7.

McCluskey, T. L., Vaquero, T. S., & Vallati, M. (2017). Engineering knowledge for automated planning: Towards a notion of quality. In *Proceedings of the Knowledge Capture Conference, K-CAP*, pp. 14:1–14:8.

Mcdermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., & Wilkins, D. (1998). PDDL - The Planning Domain Definition Language. Tech. rep. TR-98-003, Yale Center for Computational Vision and Control,.

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

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

Ramírez, M., Papasimeon, M., Lipovetzky, N., Benke, L., Miller, T., Pearce, A. R., Scala, E., & Zamani, M. (2018). Integrated hybrid planning and programmed control for real time UAV maneuvering. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS*, pp. 1318–1326.

Richter, S., & 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*, pp. 66–70.

Scala, E. (2014). Plan repair for resource constrained tasks via numeric macro actions. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*.

Scala, E., & Torasso, P. (2015). Deordering and numeric macro actions for plan repair. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pp. 1673–1681.

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

Siddiqui, F., & Haslum, P. (2012). Block-structured plan deordering. In *25th Australasian Joint Conference*, Vol. 7691 of *LNAI*, pp. 803–814.

Thiébaux, S., Coffrin, C., Hijazi, H., & Slaney, J. (2013). Planning with mip for supply restoration in power distribution systems. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Vallati, M., Chrpa, L., McCluskey, T. L., & Hutter, F. (2021). On the importance of domain model configuration for automated planning engines. *J. Autom. Reason.*, *65*(6), 727–773.

Vallati, M., Magazzeni, D., Schutter, B. D., Chrpa, L., & McCluskey, T. L. (2016). Efficient macroscopic urban traffic models for reducing congestion: A PDDL+ planning approach. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pp. 3188–3194. AAAI Press.

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*, pp. 64–65.

Appendix A. Detailed Results

In the following, we provide the detailed results of the performed experiments.

A.1 Conservative Version

	Coverage					PAR10					IPC quality				
Planner	O	C	nC	cC	ncC	O	C	nC	cC	ncC	O	C	nC	cC	ncC
elevators															
ff	0	20	20	20	20	9000	22	18	92	38	0.0	19.5	19.5	17.3	17.3
lama	20	20	20	19	19	36	10	11	482	482	19.9	14.3	14.3	11.1	11.1
probe	19	20	20	20	20	602	25	44	23	29	18.4	18.7	18.7	17.3	17.3
Mp	15	15	15	18	18	2256	2258	2258	911	911	13.7	12.5	12.5	17.1	17.1
mercury	20	20	20	20	20	4.1	13	14	40	28	19.7	15.7	15.7	13.9	13.9
yahsp	20	20	20	13	12	23	41	33	3227	3650	11.9	19.4	19.4	11.1	10.3
BFWS	20	20	20	20	20	7.0	5.3	5.6	5.1	5.4	19.6	16.6	16.6	16.7	16.7
FDSS	20	20	20	18	19	30	19	21	929	483	19.5	16.4	16.4	11.9	12.7
floortile															
ff	2	2	2	2	2	8100	8107	8102	8105	8105	2.0	1.7	1.7	1.7	1.7
lama	6	7	7	7	7	6367	5897	5895	5895	5897	6.0	4.8	4.8	4.8	4.8
probe	4	4	4	4	4	7204	7237	7234	7236	7221	4.0	2.8	2.8	2.8	2.8
Mp	20	20	20	20	20	0.1	0.1	0.1	0.1	0.1	19.9	14.4	14.4	14.4	14.4
mercury	7	7	7	7	7	5889	5874	5874	5874	5871	7.0	4.6	4.6	4.6	4.6
yahsp	6	6	6	6	6	6347	6322	6321	6319	6323	6.0	5.2	5.2	5.2	5.2
BFWS	2	2	2	2	2	8102	8100	8100	8100	8100	2.0	1.8	1.8	1.8	1.8
FDSS	20	20	20	20	20	0.6	0.3	0.3	0.3	0.3	20.0	15.9	15.9	15.9	15.9
GED															
ff	0	-	6	-	6	9000	-	6306	-	6306	0.0	-	6.0	-	6.0
lama	20	-	20	-	20	3.0	-	6.2	-	5.9	18.3	-	19.6	-	19.6
probe	20	-	20	-	20	55	-	73	-	74	15.2	-	16.6	-	16.6
Mp	2	-	2	-	2	8101	-	8100	-	8100	2.0	-	2.0	-	2.0
mercury	20	-	20	-	20	3.8	-	5.9	-	5.6	19.1	-	18.2	-	18.2
yahsp	11	-	20	-	20	4113	-	6.6	-	5.8	9.5	-	17.3	-	17.3
BFWS	17	-	17	-	17	1359	-	1363	-	1366	16.7	-	17.0	-	17.0
FDSS	20	-	20	-	20	6.1	-	12	-	12	18.3	-	19.6	-	19.6
Hiking															
ff	9	18	12	-	17	5077	963	3699	-	1378	8.3	15.7	8.1	-	9.5
lama	19	15	11	-	20	566	2312	4111	-	57	19.0	9.4	6.4	-	11.6
probe	20	20	20	-	20	13	7.1	102	-	1.9	18.2	15.9	18.5	-	10.2
Mp	0	7	1	-	10	9000	5850	8550	-	4501	0.0	7.0	0.4	-	8.1
mercury	12	20	13	-	18	3769	29	3217	-	936	11.0	13.9	10.9	-	13.8
yahsp	17	18	14	-	20	1457	949	2811	-	26	12.2	8.0	6.8	-	17.7
BFWS	11	18	16	-	17	4065	905	1804	-	1350	10.9	14.3	9.4	-	8.9
FDSS	20	20	19	-	20	59	43	569	-	23	19.2	13.5	14.8	-	11.2
storage															
ff	18	19	19	19	19	5403	4953	4954	4967	4966	17.6	17.3	17.3	17.8	17.8
lama	19	20	20	21	21	4957	4508	4508	4101	4101	16.6	17.8	17.8	19.9	19.9
probe	21	29	29	29	29	4050	507	505	476	469	20.8	26.8	26.8	26.4	26.4
Mp	29	18	18	18	18	454	5400	5400	5400	5400	25.9	15.6	15.6	17.5	17.5
mercury	20	18	18	21	21	4515	5400	5400	4066	4065	19.5	16.2	16.2	20.2	20.2
yahsp	22	21	21	23	23	3612	4050	4050	3172	3165	15.7	19.5	19.5	22.1	22.1
BFWS	30	30	30	29	29	1.6	33	34	476	484	27.4	28.8	28.8	27.4	27.4
FDSS	23	22	22	25	25	3187	3622	3622	2349	2346	17.8	19.2	19.2	22.5	22.3
termes															
ff	0	0	1	0	0	9000	9000	8554	9000	9000	0.0	0.0	1.0	0.0	0.0
lama	14	14	14	13	12	2747	2784	2754	3249	3647	12.0	12.2	12.1	8.3	8.5
probe	7	6	7	5	6	5900	6363	5884	6819	6331	6.5	5.2	5.8	3.2	4.4
Mp	0	0	0	0	0	9000	9000	9000	9000	9000	0.0	0.0	0.0	0.0	0.0
mercury	13	9	9	10	1	3196	5044	4978	4610	8550	12.3	6.7	7.5	7.2	0.8
yahsp	0	0	0	0	0	9000	9000	9000	9000	9000	0.0	0.0	0.0	0.0	0.0
BFWS	2	3	2	3	2	8100	7655	8107	7662	8110	1.7	2.8	1.8	2.5	1.9
FDSS	12	10	11	9	8	3619	4509	4063	4961	5405	10.2	8.6	9.7	6.6	5.0
transport															
ff	0	1	4	1	0	9000	8570	7270	8575	9000	0.0	1.0	3.8	1.0	0.0
lama	17	15	19	13	0	1442	2385	558	3228	9000	15.8	13.0	17.2	11.3	0.0
probe	16	13	18	13	0	1944	3234	1165	3229	9000	13.8	12.2	16.8	12.2	0.0
Mp	0	0	0	0	0	9000	9000	9000	9000	9000	0.0	0.0	0.0	0.0	0.0
mercury	20	11	9	11	0	13	4129	4984	4131	9000	20.0	5.8	4.6	5.8	0.0
yahsp	20	8	3	8	0	1.5	5555	7700	5598	9000	16.2	8.0	2.5	8.0	0.0
BFWS	20	20	20	20	0	15	30	23	26	9000	18.4	19.1	17.6	19.1	0.0
FDSS	20	14	20	15	0	165	2814	165	2382	9000	17.4	12.2	18.1	13.4	0.0

Table 7: Results comparing the (O)riginal models with CSM enhanced models, namely CSMs with (C) and without argument limit (nC). Compound CSMs with (cC) and without argument limit (ncC).

Tables 7 and 8 show detailed results of planners' performance achieved by the considered planners on the considered benchmark domains, when using the original domain model (O) or the models extended with CSMs with (C) and without argument limit (nC), i.e., whether the "extra argument" check on Line 8 of Algorithm 2 is enabled or not, and Compound CSMs with (cC) and without argument limit (ncC).

A.2 Aggressive Version

Tables 9 and 10 show detailed results of planners' performance achieved by the considered planners on the considered benchmark domains, when using the original domain model (O) or the models extended with the aggressive variant of CSMs with (AC) and without argument limit (AnC), i.e., whether the "extra argument" check on Line 8 of Algorithm 2 is enabled or not, and Compound CSMs with (AcC) and without argument limit (AncC).

A.3 Comparing against the state of the art

Tables 11 and 12 show detailed results of planners' performance achieved by the considered planners on the considered benchmark domains, when using the original domain model (O) or the models extended with the (M)UM macros, (B)LoMa macros, the conservative variant of CSMs (the most promising sets of CSMs were considered), the aggressive variant of CSMs (the most promising sets of the aggressive CSMs were considered).

	Coverage					PAR10					IPC quality				
Planner	O	C	nC	cC	ncC	O	C	nC	cC	ncC	O	C	nC	cC	ncC
barman															
ff	0	30	30	30	30	9000	1.1	0.9	0.1	0.1	0.0	30.0	30.0	30.0	30.0
lama	2	30	30	30	30	8418	183	372	12	13	1.7	26.4	26.4	30.0	30.0
probe	5	5	5	30	30	7581	7596	7607	47	44	3.4	5.0	5.0	30.0	30.0
Mp	0	0	0	30	30	9000	9000	9000	145	146	0.0	0.0	0.0	30.0	30.0
mercury	26	30	30	30	30	1307	51	74	14	14	18.0	28.8	28.8	29.9	29.9
yahsp	0	30	30	30	30	9000	185	150	0.3	0.3	0.0	29.1	29.1	30.0	30.0
BFWS	0	20	21	30	30	9000	3006	2727	3.0	2.7	0.0	19.9	20.9	30.0	30.0
FDSS	24	30	30	30	30	2044	146	147	160	118	22.0	29.9	29.9	29.9	29.9
bw															
ff	0	8	8	8	8	9000	6602	6602	6601	6602	0.0	8.0	8.0	8.0	8.0
lama	28	29	29	29	29	656	345	366	349	346	25.4	24.8	24.8	24.8	24.8
probe	27	30	30	30	30	1101	83	111	68	129	25.0	29.2	29.2	29.2	29.2
Mp	0	29	29	29	29	9000	435	439	430	437	0.0	29.0	29.0	29.0	29.0
mercury	19	30	30	30	30	3327	19	24	21	19	8.2	30.0	30.0	30.0	30.0
yahsp	28	27	27	28	27	631	933	933	660	931	7.6	27.0	27.0	28.0	27.0
BFWS	4	3	3	3	3	7814	8109	8109	8107	8109	3.4	2.5	2.5	2.5	2.5
FDSS	25	25	25	26	25	1568	1563	1562	1275	1577	23.6	20.2	20.2	19.8	20.2
depots															
ff	1	11	11	11	12	8703	5789	5792	5797	5511	1.0	10.8	10.8	10.8	11.8
lama	1	0	0	0	0	8724	9000	9000	9000	9000	1.0	0.0	0.0	0.0	0.0
probe	30	30	30	30	30	36	14	26	26	26	27.5	28.8	28.8	28.8	28.8
Mp	2	2	2	2	2	8400	8400	8400	8400	8400	1.6	2.0	2.0	2.0	2.0
mercury	0	0	0	0	0	9000	9000	9000	9000	9000	0.0	0.0	0.0	0.0	0.0
yahsp	22	20	20	20	20	2479	3036	3020	3028	3032	16.8	19.2	19.2	19.2	19.2
BFWS	11	16	16	16	16	5759	4309	4311	4306	4308	9.6	15.3	15.3	15.3	15.3
FDSS	20	15	15	15	15	3563	4732	4732	4749	4732	20.0	14.0	14.0	14.0	14.0
gold															
ff	30	27	29	27	29	11	921	331	928	333	30.0	27.0	22.8	27.0	22.8
lama	30	30	28	30	28	0.1	77	600	77	600	21.5	25.5	22.8	25.5	22.8
probe	30	30	30	30	30	0.0	0.0	0.0	0.0	0.0	30.0	23.5	22.0	23.5	22.0
Mp	30	30	30	30	30	0.0	0.0	0.0	0.0	0.0	29.3	21.2	22.2	21.2	22.2
mercury	30	26	30	26	30	1.3	1242	0.4	1241	0.3	23.0	24.6	19.7	24.6	19.7
yahsp	28	26	29	26	29	610	1224	332	1222	331	20.0	18.5	27.0	18.5	27.0
BFWS	30	30	30	30	30	0.0	0.0	0.0	0.0	0.0	30.0	27.2	22.9	27.2	22.9
FDSS	30	30	30	30	30	0.1	0.2	0.2	0.2	0.2	28.8	27.2	24.9	27.2	24.9
gripper															
ff	0	28	28	28	28	9000	686	712	713	735	0.0	28.0	28.0	28.0	28.0
lama	7	30	30	30	30	7029	69	75	71	63	6.5	30.0	30.0	30.0	30.0
probe	0	0	0	0	0	9000	9000	9000	9000	9000	0.0	0.0	0.0	0.0	0.0
Mp	0	0	0	0	0	9000	9000	9000	9000	9000	0.0	0.0	0.0	0.0	0.0
mercury	0	25	25	25	25	9000	1653	1664	1665	1654	0.0	25.0	25.0	25.0	25.0
yahsp	0	0	0	0	0	9000	9000	9000	9000	9000	0.0	0.0	0.0	0.0	0.0
BFWS	0	5	5	5	5	9000	7516	7519	7520	7520	0.0	5.0	5.0	5.0	5.0
FDSS	0	1	0	0	0	9000	8726	9000	9000	9000	0.0	1.0	0.0	0.0	0.0
matching-bw															
ff	13	27	27	27	27	5160	900	900	900	900	10.6	27.0	27.0	27.0	27.0
lama	26	29	29	29	29	1201	301	301	301	301	23.1	27.1	27.1	27.1	27.1
probe	13	30	30	30	30	5104	0.3	0.4	0.3	0.3	7.3	30.0	30.0	30.0	30.0
Mp	0	23	23	23	23	9000	2105	2104	2103	2105	0.0	23.0	23.0	23.0	23.0
mercury	10	22	22	22	22	6025	2425	2425	2426	2425	8.2	21.9	21.9	21.9	21.9
yahsp	21	26	26	26	26	2725	1216	1214	1214	1214	15.8	25.8	25.8	25.8	25.8
BFWS	12	29	29	29	29	5431	369	369	367	373	11.0	28.0	28.0	28.0	28.0
FDSS	30	30	30	30	30	1.1	1.2	1.2	1.2	1.1	28.1	27.4	27.4	27.4	27.4
rovers															
ff	1	7	6	8	7	8714	6998	7293	6698	7022	0.9	6.9	5.9	7.8	6.8
lama	30	30	30	29	29	96	95	96	417	405	29.1	28.9	28.9	28.2	28.2
probe	30	29	29	30	30	338	603	617	322	224	29.5	28.5	28.5	29.2	29.2
Mp	16	17	17	16	16	4384	4102	4104	4405	4382	15.5	16.8	16.8	15.6	15.6
mercury	28	29	29	29	28	766	446	491	458	751	26.7	27.5	27.5	28.5	27.5
yahsp	30	30	30	30	30	10	9.5	9.7	10	10	29.2	28.4	28.4	29.7	29.7
BFWS	21	22	22	22	22	2971	2666	2662	2686	2681	20.8	21.1	21.1	21.6	21.6
FDSS	30	30	30	30	30	272	301	295	290	318	29.0	29.2	29.2	28.9	28.9
sokoban															
ff	19	23	26	23	26	3331	2117	1609	2114	1610	17.9	19.6	22.6	19.6	22.6
lama	21	20	22	20	22	2729	3034	2435	3034	2437	19.2	15.8	19.3	15.8	19.3
probe	25	30	1	29	1	1519	65	8729	340	8728	22.7	27.8	0.6	26.8	0.6
Mp	30	30	30	30	30	0.8	2.2	1.4	2.6	1.5	28.5	26.2	25.2	26.2	25.2
mercury	23	24	22	23	22	2158	1878	2482	2151	2484	20.5	20.6	19.3	19.6	19.3
yahsp	26	29	30	29	30	1237	367	48	371	47	18.6	26.9	24.3	26.9	24.3
BFWS	30	30	30	30	30	0.7	1.0	394	1.0	388	28.8	28.6	26.5	28.6	26.5
FDSS	30	30	30	30	30	14	22	11	22	11	27.4	24.8	24.7	24.8	24.7
thoughtful															
ff	17	20	20	20	20	3900	3001	3001	3001	3001	16.4	19.3	19.3	19.3	19.3
lama	25	23	23	23	23	1503	2103	2103	2103	2103	24.4	22.2	22.2	22.2	22.2
probe	21	23	23	23	23	2709	2106	2107	2107	2107	19.1	22.4	22.4	22.4	22.4
Mp	0	0	0	0	0	9000	9000	9000	9000	9000	0.0	0.0	0.0	0.0	0.0
mercury	21	17	17	17	17	2704	3903	3903	3903	3903	20.8	16.6	16.6	16.6	16.6
yahsp	8	6	6	6	6	6603	7218	7218	7228	7228	7.8	5.9	5.9	5.9	5.9
BFWS	30	30	30	30	30	1.6	1.7	1.6	1.8	1.7	29.3	29.4	29.4	29.4	29.4
FDSS	30	30	30	30	30	23	34	35	34	34	28.9	28.4	28.4	28.4	28.4

Table 8: Results comparing the (O)riginal models with CSM enhanced models, namely simple CSMs with (C) and without argument limit (nC). Compound CSMs with (cC) and without argument limit (ncC).

	Coverage					PAR10					IPC quality				
Planner	O	AC	AnC	AcC	AncC	O	AC	AnC	AcC	AncC	O	AC	AnC	AcC	AncC
floortile															
ff	2	20	20	20	20	8100	0.1	0.1	0.1	0.1	2.0	19.8	19.8	19.8	19.8
lama	6	20	20	20	20	6367	0.2	0.2	0.2	0.2	6.0	17.9	17.9	17.9	17.9
probe	4	20	20	20	20	7204	0.1	0.1	0.1	0.1	3.9	19.2	19.2	19.2	19.2
Mp	20	20	20	20	20	0.1	0.1	0.1	0.1	0.1	20.0	15.3	15.3	15.3	15.3
mercury	7	20	20	20	20	5889	0.2	0.2	0.2	0.2	7.0	17.9	17.9	17.9	17.9
yahsp	6	20	20	20	20	6347	0.2	0.2	0.2	0.2	5.9	19.8	19.8	19.8	19.8
BFWS	2	20	20	20	20	8102	0.0	0.0	0.0	0.0	2.0	19.6	19.6	19.6	19.6
FDSS	20	20	20	20	20	0.6	0.3	0.2	0.3	0.2	19.9	16.1	16.1	16.1	16.1
Hiking															
ff	9	-	16	-	-	5077	-	1966	-	-	9.0	-	14.9	-	-
lama	19	-	14	-	-	566	-	2789	-	-	18.6	-	9.5	-	-
probe	20	-	20	-	-	13	-	59	-	-	18.3	-	18.5	-	-
Mp	0	-	4	-	-	9000	-	7202	-	-	0.0	-	4.0	-	-
mercury	12	-	18	-	-	3769	-	1007	-	-	11.2	-	16.9	-	-
yahsp	17	-	18	-	-	1457	-	1044	-	-	13.8	-	13.0	-	-
BFWS	11	-	18	-	-	4065	-	909	-	-	11.0	-	12.9	-	-
FDSS	20	-	20	-	-	59	-	99	-	-	19.2	-	15.2	-	-
transport															
ff	0	-	-	-	0	9000	-	-	-	9000	0.0	-	-	-	0.0
lama	17	-	-	-	0	1442	-	-	-	9000	17.0	-	-	-	0.0
probe	16	-	-	-	0	1944	-	-	-	9000	16.0	-	-	-	0.0
Mp	0	-	-	-	0	9000	-	-	-	9000	0.0	-	-	-	0.0
mercury	20	-	-	-	0	13	-	-	-	9000	20.0	-	-	-	0.0
yahsp	20	-	-	-	0	1.5	-	-	-	9000	20.0	-	-	-	0.0
BFWS	20	-	-	-	0	15	-	-	-	9000	20.0	-	-	-	0.0
FDSS	20	-	-	-	0	165	-	-	-	9000	20.0	-	-	-	0.0

Table 9: Results comparing the (O)riginal models with aggressive CSM enhanced models, namely CSMs with (AC) and without argument limit (AnC). Compound CSMs with (AcC) and without argument limit (AncC).

	Coverage					PAR10					IPC quality				
Planner	O	AC	AnC	AcC	AncC	O	AC	AnC	AcC	AncC	O	AC	AnC	AcC	AncC
barman															
ff	0	30	30	30	30	9000	0.3	0.3	0.1	0.1	0.0	30.0	30.0	30.0	30.0
lama	2	30	30	30	30	8418	7.8	7.6	6.8	6.4	1.7	30.0	30.0	30.0	30.0
probe	5	30	30	30	30	7581	74	69	5.8	4.0	3.4	30.0	30.0	30.0	30.0
Mp	0	1	1	30	30	9000	8700	8700	1.3	1.5	0.0	1.0	1.0	30.0	30.0
mercury	26	30	30	30	30	1307	4.9	4.8	6.0	6.2	18.0	30.0	30.0	30.0	30.0
yahsp	0	30	30	30	30	9000	6.1	7.6	0.1	0.1	0.0	30.0	30.0	30.0	30.0
BFWS	0	30	30	30	30	9000	1.1	1.3	0.4	0.3	0.0	30.0	30.0	30.0	30.0
FDSS	24	30	30	30	30	2044	10	10	7.6	7.6	22.0	30.0	30.0	30.0	30.0
bw															
ff	0	28	28	28	28	9000	608	608	607	607	0.0	28.0	28.0	28.0	28.0
lama	28	28	28	28	28	656	610	609	609	608	25.6	24.9	24.9	24.9	24.9
probe	27	30	30	30	30	1101	0.2	0.2	0.2	0.2	24.8	29.2	29.2	29.2	29.2
Mp	0	18	18	18	18	9000	3600	3600	3600	3600	0.0	18.0	18.0	18.0	18.0
mercury	19	11	11	11	11	3327	5734	5735	5734	5733	14.5	11.0	11.0	11.0	11.0
yahsp	28	30	30	30	30	631	0.0	0.0	0.0	0.0	6.8	30.0	30.0	30.0	30.0
BFWS	4	30	30	30	30	7814	36	37	37	36	1.5	30.0	30.0	30.0	30.0
FDSS	25	30	30	30	30	1568	9.8	9.8	9.8	9.8	23.2	24.8	24.8	24.8	24.8
depots															
ff	1	30	30	30	30	8703	0.0	0.0	0.0	0.0	0.6	30.0	30.0	30.0	30.0
lama	1	30	30	30	30	8724	0.2	0.2	0.2	0.2	0.3	30.0	30.0	30.0	30.0
probe	30	30	30	30	30	36	0.0	0.0	0.0	0.0	26.3	29.8	29.8	29.8	29.8
Mp	2	30	30	30	30	8400	0.0	0.0	0.0	0.0	1.3	30.0	30.0	30.0	30.0
mercury	0	30	30	30	30	9000	0.2	0.2	0.2	0.2	0.0	30.0	30.0	30.0	30.0
yahsp	22	30	30	30	30	2479	0.0	0.0	0.0	0.0	2.5	30.0	30.0	30.0	30.0
BFWS	11	30	30	30	30	5759	0.0	0.0	0.0	0.0	6.9	30.0	30.0	30.0	30.0
FDSS	20	30	30	30	30	3563	0.3	0.3	0.3	0.3	19.9	28.5	28.5	28.5	28.5
gripper															
ff	0	30	30	30	30	9000	9.2	9.2	8.3	9.2	0.0	30.0	30.0	30.0	30.0
lama	7	30	30	30	30	7029	2.8	2.7	2.8	2.8	7.0	29.4	29.4	29.4	29.4
probe	0	30	30	30	30	9000	11	11	10	8.3	0.0	29.9	29.9	29.9	29.9
Mp	0	30	30	30	30	9000	0.7	0.7	0.7	0.7	0.0	30.0	30.0	30.0	30.0
mercury	0	30	30	30	30	9000	2.1	2.1	2.2	2.1	0.0	30.0	30.0	30.0	30.0
yahsp	0	30	30	30	30	9000	0.1	0.1	0.1	0.1	0.0	30.0	30.0	30.0	30.0
BFWS	0	30	30	30	30	9000	323	326	325	319	0.0	30.0	30.0	30.0	30.0
FDSS	0	30	30	30	30	9000	4.2	4.2	4.1	4.3	0.0	30.0	30.0	30.0	30.0
matching-bw															
ff	13	30	30	30	30	5160	0.0	0.0	0.0	0.0	10.1	30.0	30.0	30.0	30.0
lama	26	30	30	30	30	1201	0.1	0.1	0.1	0.1	22.0	28.4	28.4	28.4	28.4
probe	13	30	30	30	30	5104	0.0	0.0	0.0	0.0	7.4	30.0	30.0	30.0	30.0
Mp	0	20	20	20	20	9000	3000	3000	3000	3000	0.0	20.0	20.0	20.0	20.0
mercury	10	25	25	25	25	6025	1500	1500	1500	1500	8.1	25.0	25.0	25.0	25.0
yahsp	21	30	30	30	30	2725	0.0	0.0	0.0	0.0	14.8	29.4	29.4	29.4	29.4
BFWS	12	30	30	30	30	5431	0.0	0.0	0.0	0.0	9.9	28.8	28.8	28.8	28.8
FDSS	30	30	30	30	30	1.1	0.1	0.1	0.1	0.1	26.4	28.2	28.2	28.2	28.2

Table 10: Results comparing the (O)riginal models with aggressive CSM enhanced models, namely CSMs with (AC) and without argument limit (AnC). Compound CSMs with (AcC) and without argument limit (AncC).

	Coverage					PAR10					IPC quality				
Planner	O	M	B	CS	ACS	O	M	B	CS	ACS	O	M	B	CS	ACS
elevators															
ff	0	-	17	20	-	9000	-	1486	22	-	0.0	-	13.2	14.7	-
lama	20	-	20	20	-	36	-	134	10	-	19.7	-	15.4	14.1	-
probe	19	-	20	20	-	602	-	141	25	-	16.0	-	19.7	16.0	-
Mp	15	-	6	15	-	2256	-	6302	2258	-	14.4	-	4.6	13.3	-
mercury	20	-	20	20	-	4.1	-	54	13	-	19.2	-	17.5	15.2	-
yahsp	20	-	11	20	-	23	-	4052	41	-	11.1	-	10.3	18.5	-
BFWS	20	-	20	20	-	7.0	-	8.5	5.3	-	18.3	-	19.1	15.4	-
FDSS	20	-	17	20	-	30	-	1405	19	-	18.6	-	15.6	15.5	-
floortile															
ff	2	2	-	2	20	8100	8100	-	8107	0.1	2.0	1.7	-	1.7	19.8
lama	6	7	-	7	20	6367	5905	-	5897	0.2	5.8	6.4	-	4.4	17.5
probe	4	5	-	4	20	7204	6751	-	7237	0.1	3.8	4.6	-	2.7	19.2
Mp	20	20	-	20	20	0.1	0.1	-	0.1	0.1	19.8	18.6	-	14.2	15.2
mercury	7	7	-	7	20	5889	5890	-	5874	0.2	6.6	6.4	-	4.4	17.6
yahsp	6	7	-	6	20	6347	5905	-	6322	0.2	5.7	6.4	-	5.0	19.6
BFWS	2	2	-	2	20	8102	8100	-	8100	0.0	2.0	1.8	-	1.8	19.6
FDSS	20	20	-	20	20	0.6	0.3	-	0.3	0.3	19.3	18.7	-	15.4	15.5
GED															
ff	0	1	2	6	-	9000	8550	8106	6306	-	0.0	0.8	1.8	6.0	-
lama	20	20	20	20	-	3.0	2.7	11	6.2	-	17.5	18.5	13.1	18.8	-
probe	20	20	18	20	-	55	97	1043	73	-	13.6	16.7	10.1	15.0	-
Mp	2	4	1	2	-	8101	7201	8551	8100	-	2.0	4.0	0.9	2.0	-
mercury	20	0	20	20	-	3.8	9000	15	5.9	-	19.1	0.0	8.0	18.2	-
yahsp	11	20	20	20	-	4113	6.9	73	6.6	-	8.1	15.0	14.1	15.0	-
BFWS	17	19	19	17	-	1359	520	452	1363	-	14.7	15.9	16.6	15.0	-
FDSS	20	20	20	20	-	6.1	4.5	27	12	-	17.9	17.6	13.1	19.2	-
Hiking															
ff	9	-	0	17	16	5077	-	9000	1378	1966	9.0	-	0.0	10.9	14.9
lama	19	-	0	20	14	566	-	9000	57	2789	18.6	-	0.0	11.3	9.5
probe	20	-	0	20	20	13	-	9000	1.9	59	18.3	-	0.0	10.0	18.5
Mp	0	-	0	10	4	9000	-	9000	4501	7202	0.0	-	0.0	9.1	3.8
mercury	12	-	1	18	18	3769	-	8550	936	1007	11.0	-	1.0	13.1	16.4
yahsp	17	-	0	20	18	1457	-	9000	26	1044	12.1	-	0.0	17.7	8.7
BFWS	11	-	0	17	18	4065	-	9000	1350	909	11.0	-	0.0	9.3	12.9
FDSS	20	-	1	20	20	59	-	8550	23	99	19.2	-	0.5	11.1	15.1
storage															
ff	18	18	19	19	-	5403	5404	4950	4967	-	17.3	16.3	16.9	17.3	-
lama	19	24	21	21	-	4957	2706	4050	4101	-	15.8	22.2	19.9	18.7	-
probe	21	21	28	29	-	4050	4052	930	476	-	20.8	17.9	25.1	26.8	-
Mp	29	25	26	18	-	454	2253	1801	5400	-	22.6	20.9	22.4	17.4	-
mercury	20	24	21	21	-	4515	2743	4051	4066	-	18.8	22.0	19.9	19.5	-
yahsp	22	22	21	23	-	3612	3642	4051	3172	-	14.7	19.6	19.5	20.8	-
BFWS	30	29	30	29	-	1.6	451	2.0	476	-	26.9	27.0	28.3	26.9	-
FDSS	23	25	23	25	-	3187	2297	3190	2349	-	17.5	22.6	21.1	21.5	-
termes															
ff	0	1	1	0	-	9000	8572	8568	9000	-	0.0	1.0	1.0	0.0	-
lama	14	11	14	14	-	2747	4131	2804	2784	-	12.1	7.1	10.3	12.3	-
probe	7	1	6	6	-	5900	8564	6331	6363	-	6.4	0.3	5.1	5.2	-
Mp	0	0	0	0	-	9000	9000	9000	9000	-	0.0	0.0	0.0	0.0	-
mercury	13	8	12	9	-	3196	5444	3674	5044	-	12.5	4.4	9.4	6.9	-
yahsp	0	0	0	0	-	9000	9000	9000	9000	-	0.0	0.0	0.0	0.0	-
BFWS	2	2	2	3	-	8100	8109	8101	7655	-	1.7	1.8	1.7	2.9	-
FDSS	12	6	8	10	-	3619	6340	5407	4509	-	10.9	3.8	5.9	9.3	-
transport															
ff	0	-	11	4	0	9000	-	4157	7270	9000	0.0	-	10.5	3.7	0.0
lama	17	-	18	19	0	1442	-	1001	558	9000	16.1	-	15.6	17.5	0.0
probe	16	-	20	18	0	1944	-	114	1165	9000	14.4	-	17.4	17.2	0.0
Mp	0	-	5	0	0	9000	-	6751	9000	9000	0.0	-	5.0	0.0	0.0
mercury	20	-	0	9	0	13	-	9000	4984	9000	20.0	-	0.0	4.6	0.0
yahsp	20	-	10	3	0	1.5	-	4636	7700	9000	16.4	-	8.2	2.7	0.0
BFWS	20	-	20	20	0	15	-	16	23	9000	19.1	-	17.3	18.3	0.0
FDSS	20	-	20	20	0	165	-	110	165	9000	18.0	-	17.1	18.7	0.0

Table 11: Results comparing the (O)riginal models with macro enhanced models, namely (M)UM, (B)loMa, CSMs (CS) and the Aggressive CSMs (ACS).

	Coverage					PAR10					IPC quality				
Planner	O	M	B	CS	ACS	O	M	B	CS	ACS	O	M	B	CS	ACS
barman															
ff	0	-	0	30	30	9000	-	9000	0.1	0.1	0.0	-	0.0	30.0	30.0
lama	2	-	20	30	30	8418	-	3233	12	6.8	1.7	-	18.4	30.0	30.0
probe	5	-	11	30	30	7581	-	5737	47	5.8	3.4	-	8.7	30.0	30.0
Mp	0	-	0	30	30	9000	-	9000	145	1.3	0.0	-	0.0	30.0	30.0
mercury	26	-	14	30	30	1307	-	4976	14	6.0	18.0	-	9.9	30.0	30.0
yahsp	0	-	0	30	30	9000	-	9000	0.3	0.1	0.0	-	0.0	30.0	30.0
BFWS	0	-	4	30	30	9000	-	7824	3.0	0.4	0.0	-	3.4	30.0	30.0
FDSS	24	-	30	30	30	2044	-	588	160	7.6	22.0	-	28.0	29.9	29.9
bw															
ff	0	-	1	8	28	9000	-	8700	6602	608	0.0	-	0.9	7.7	28.0
lama	28	-	29	29	28	656	-	356	345	610	24.3	-	21.7	23.6	23.4
probe	27	-	30	30	30	1101	-	182	83	0.2	24.1	-	29.5	28.1	28.3
Mp	0	-	0	29	18	9000	-	9000	435	3600	0.0	-	0.0	27.7	16.7
mercury	19	-	11	30	11	3327	-	5777	19	5734	8.2	-	10.0	30.0	9.2
yahsp	28	-	24	27	30	631	-	1834	933	0.0	6.1	-	17.4	23.4	27.2
BFWS	4	-	2	3	30	7814	-	8427	8109	36	1.5	-	0.9	1.4	30.0
FDSS	25	-	22	25	30	1568	-	2434	1563	9.8	22.4	-	15.7	18.3	23.7
depots															
ff	1	9	11	11	30	8703	6386	5758	5789	0.0	0.6	6.0	6.9	7.4	30.0
lama	1	0	0	0	30	8724	9000	9000	9000	0.2	0.3	0.0	0.0	0.0	30.0
probe	30	30	30	30	30	36	26	26	14	0.0	26.2	26.4	27.0	27.5	29.7
Mp	2	2	2	2	30	8400	8400	8400	8400	0.0	1.3	1.6	1.6	1.6	30.0
mercury	0	0	0	0	30	9000	9000	9000	9000	0.2	0.0	0.0	0.0	0.0	30.0
yahsp	22	20	20	20	30	2479	3032	3031	3036	0.0	2.5	3.2	3.2	3.2	30.0
BFWS	11	18	19	16	30	5759	3762	3450	4309	0.0	6.8	14.3	14.0	11.8	29.6
FDSS	20	15	15	15	30	3563	4732	4733	4732	0.3	19.9	13.8	13.8	13.8	28.3
gold															
ff	30	22	30	29	-	11	2402	0.2	331	-	30.0	20.6	23.3	22.8	-
lama	30	30	30	28	-	0.1	17	0.3	600	-	23.3	20.9	20.4	24.9	-
probe	30	30	30	30	-	0.0	1.8	0.0	0.0	-	29.0	26.1	25.0	21.2	-
Mp	30	30	30	30	-	0.0	0.1	0.0	0.0	-	28.3	25.5	18.8	21.4	-
mercury	30	29	30	30	-	1.3	301	0.3	0.4	-	23.0	23.1	20.5	20.0	-
yahsp	28	27	30	29	-	610	965	0.0	332	-	17.2	23.1	16.3	23.4	-
BFWS	30	18	30	30	-	0.0	3600	0.1	0.0	-	30.0	12.8	19.6	22.9	-
FDSS	30	30	30	30	-	0.1	0.3	0.6	0.2	-	27.1	28.4	23.1	23.3	-
gripper															
ff	0	28	28	28	30	9000	718	708	686	9.2	0.0	28.0	28.0	28.0	30.0
lama	7	30	30	30	30	7029	68	74	69	2.8	6.5	30.0	30.0	30.0	25.8
probe	0	0	0	0	30	9000	9000	9000	9000	11	0.0	0.0	0.0	0.0	30.0
Mp	0	0	0	0	30	9000	9000	9000	9000	0.7	0.0	0.0	0.0	0.0	30.0
mercury	0	25	25	25	30	9000	1671	1671	1653	2.1	0.0	24.9	24.9	24.9	29.9
yahsp	0	0	0	0	30	9000	9000	9000	9000	0.1	0.0	0.0	0.0	0.0	30.0
BFWS	0	5	5	5	30	9000	7518	7519	7516	323	0.0	4.9	4.9	4.9	30.0
FDSS	0	0	0	1	30	9000	9000	9000	8726	4.2	0.0	0.0	0.0	1.0	29.9
matching-bw															
ff	13	9	0	27	30	5160	6308	9000	900	0.0	10.0	7.6	0.0	26.8	29.7
lama	26	26	0	29	30	1201	1201	9000	301	0.1	21.5	21.0	0.0	25.0	27.7
probe	13	17	0	30	30	5104	3904	9000	0.3	0.0	7.3	11.1	0.0	29.5	29.5
Mp	0	0	0	23	20	9000	9000	9000	2105	3000	0.0	0.0	0.0	17.0	19.9
mercury	10	14	22	22	25	6025	4804	2405	2425	1500	7.5	11.9	17.7	21.1	24.1
yahsp	21	28	0	26	30	2725	622	9000	1216	0.0	13.4	20.9	0.0	22.7	27.7
BFWS	12	16	0	29	30	5431	4273	9000	369	0.0	9.7	13.3	0.0	25.3	28.2
FDSS	30	30	15	30	30	1.1	1.1	4503	1.2	0.1	24.9	25.4	10.7	24.2	26.6
rovers															
ff	1	0	2	8	-	8714	9000	8423	6698	-	0.9	0.0	1.9	7.9	-
lama	30	30	30	29	-	96	88	94	417	-	29.2	24.6	28.9	28.3	-
probe	30	24	25	30	-	338	2151	1868	322	-	29.4	21.8	24.2	29.2	-
Mp	16	10	10	16	-	4384	6125	6107	4405	-	15.3	7.7	9.9	15.3	-
mercury	28	27	29	29	-	766	1026	408	458	-	26.5	22.4	27.8	28.3	-
yahsp	30	30	30	30	-	10	10	10	10	-	28.9	29.4	28.8	29.4	-
BFWS	21	16	7	22	-	2971	4446	7034	2686	-	20.8	13.2	6.9	21.6	-
FDSS	30	30	30	30	-	272	183	279	290	-	29.1	25.5	29.0	29.1	-
sokoban															
ff	19	-	19	23	-	3331	-	3327	2117	-	17.9	-	16.7	19.8	-
lama	21	-	14	20	-	2729	-	4861	3034	-	19.6	-	11.8	15.9	-
probe	25	-	28	30	-	1519	-	631	65	-	20.9	-	24.3	25.8	-
Mp	30	-	30	30	-	0.8	-	0.6	2.2	-	27.7	-	28.4	25.4	-
mercury	23	-	23	24	-	2158	-	2152	1878	-	21.3	-	18.9	21.3	-
yahsp	26	-	26	29	-	1237	-	1251	367	-	19.6	-	14.9	28.4	-
BFWS	30	-	30	30	-	0.7	-	1.3	1.0	-	28.7	-	26.5	28.5	-
FDSS	30	-	30	30	-	14	-	10	22	-	27.9	-	21.3	25.3	-
thoughtful															
ff	17	19	20	20	-	3900	3301	3001	3001	-	16.4	17.8	19.0	19.0	-
lama	25	22	23	23	-	1503	2403	2103	2103	-	24.1	21.0	21.9	21.9	-
probe	21	26	24	23	-	2709	1207	1808	2106	-	18.8	23.7	22.6	21.5	-
Mp	0	0	0	0	-	9000	9000	9000	9000	-	0.0	0.0	0.0	0.0	-
mercury	21	20	17	17	-	2704	3009	3903	3903	-	20.5	19.4	16.4	16.4	-
yahsp	8	7	6	6	-	6603	6904	7224	7218	-	7.7	6.4	5.9	5.9	-
BFWS	30	30	30	30	-	1.6	1.7	1.7	1.7	-	29.1	28.7	29.3	29.2	-
FDSS	30	30	30	30	-	23	35	34	34	-	28.2	27.8	27.7	27.7	-

Table 12: Results comparing the (O)riginal models with macro enhanced models, namely (M)UM, (B)loMa, CSMs (CS) and the Aggressive CSMs (ACS).