To appear in the *Journal of Experimental & Theoretical Artificial Intelligence*
Vol. 00, No. 00, Month 20XX, 1–21

ORIGINAL ARTICLE

MEvo: A Framework for Effective Macro Sets Evolution

Mauro Vallati[a*] and Lukáš Chrpa[b,c] and Ivan Serina[d]

[a]*Department of Informatics, University of Huddersfield, Huddersfield, UK*
[b]*Department of Computer Science, Czech Technical University in Prague, Prague Czech Republic*
[c]*Department of Theoretical Computer Science and Mathematical Logic, Charles University in Prague, Prague Czech Republic*
[d]*Dipartimento d'Ingegneria dell'Informazione , Universitá degli Studi di Brescia Brescia, Italy*

(*Received 00 Month 20XX; final version received 00 Month 20XX*)

In Automated Planning, generating macro-operators (macros) is a well-known reformulation approach that is used to speed-up the planning process. Nowadays, given the number of existing techniques, a large number of macros is already available or can be easily extracted. Most of the macro generation techniques aim for using the same set of generated macros for each planner and every problem instance in a given domain. Although they provide "general improvement", the effect of macros might vary a lot for different planners. Moreover, the impact of macros on structurally different problem instances than the training ones can be potentially very detrimental. Evidently, this limits the exploitation of macros in real-world planning applications, where the structure of problem instances can often change as well as the exploited planning engine can change from time to time.

In this paper we propose the Macro sets Evolution (MEvo) approach. MEvo has been designed for overcoming the aforementioned issues in order to improve the performance of domain-independent planners by dynamically selecting promising macros –taken from a given pool– while solving continuous streams of problem instances. Our extensive empirical study, involving more than 1,000 planning problem instances and 8 state-of-the-art planning engines, demonstrates effectiveness and efficiency of MEvo.

Keywords: Automated Planning; Domain Model Reformulation; Sets Evolution

1. Introduction

Macro-operators ("macros" for short) are a well-established problem reformulation technique for encapsulating sequences of planning operators in the same format as original planning operators, and can therefore be used by any planner. Macros can be thus seen as "short-cuts" in the state space: they can reduce the number of steps needed to reach the goal. This property can be useful since, by exploiting them, it is possible to reach the goals in fewer steps. However, the number of instances of macros is often high, as they usually have a large set of parameters, that derives from the parameters of the original operators that are encapsulated. This increases the branching factor in the search space, which can slow down the planning process and, moreover, increase the memory consumption. Therefore, it is important that benefits of macros exploitation outweigh

*Corresponding author. Email: m.vallati@hud.ac.uk

their drawbacks. This problem is known as the *utility problem* (Minton, 1988).

Macro generation techniques mostly rely on an expensive training phase, where solutions of simple problems generated by a given planning engine are analysed; this phase provides useful insights into the structure of plans that can be exploited (in the form of macros) for solving new instances. Furthermore, this approach is able to provide only a static view of the problem; in fact, learnt macros are then exploited without modification on the new instances of the same domain, regardless of their similarity with training instances, and the planning technique used during the search. Indeed, as soon as either structurally different instances have to be solved, or different planning engines are used, the impact of macros on performance becomes unknown. Re-training can be an option under such circumstances, but: (i) it can be computationally very expensive; (ii) some incoming problem instances might be of a different structure, which is typical in planning applications; and (iii) it requires the exploitation of representative training instances of those which will need to be solved in the future: such instances may not be available. The aforementioned issues limit effective and efficient exploitation of macros in real-world planning applications, where the structure of planning problems can considerably vary, as well as the exploited planner. Imagine, for example, a small company exploiting planning for dealing with a version of the well-known `Depots` domain Fox and Long (2003) which combines the two well-known domains `Logistics` and `Blocksworld`. The planning task is about moving crates among locations by trucks, and stacking crates onto pallets using hoists that are available at fixed locations; the loads are all crates that can be stacked and unstacked onto a fixed set of pallets at the locations. The trucks do not hold crates in a particular order, so they can act like a table in the Blocks domain, allowing crates to be reordered. As soon as the company grows, changes in the number and size of trucks, locations of depots, etc. will happen. Such changes have a strong impact on the structure of instances to be solved, as well as on plans. Also, in planning applications. problem instances usually arrive in continuous streams, thus preventing the possibility to perform another expensive offline training.

In this work we propose **MEvo** –**M**acro sets **Evo**lution– that, given a pool of macro-operators and a domain-independent planner, can automatically select useful sets of macros to be used for solving a stream of problem instances from a given domain and, more importantly, can dynamically evolve the composition of such macro sets according to observed performance. Therefore, we can mitigate a possible issue of selecting "peculiar" training problems, deal with situations where problem structure changes, and select macros that "comply" with a given planning engine. The approach proposed is inspired by two major lines of research in AI: portfolios (Kotthoff, 2014) and realtime algorithm configuration (Fitzgerald et al., 2015). From portfolio approaches there comes the idea of combining macros into promising sets, given an available pool of elements; the real-time algorithm configuration area inspired the overtime evolution of the generated sets of macros. MEvo is based on the following pillars: (i) nowadays, multi-core machines are widely available and cheap; (ii) multi-core planning engines are not as well engineered as traditional planners (López et al., 2015; Vallati et al., 2015a); and (iii) given the large number of existing techniques, macros can be collected from various sources, and even hand-coded. The idea is that when a new problem instance arrives, four parallel runs of the solver attempt to solve it. One run is exploiting the original domain model encoding, while each of the other three runs incorporates different sets of macros. As soon as one run solves the instance, all the others are terminated and the usefulness evaluation of each macro is then updated accordingly. The aim of such a process is to evolve the most suitable set of macros for a given planner and a given class of problem instances, and eventually adapt those sets if problem instances change their structure. A large-scale

```
(:action LOAD-TRUCK
  :parameters
   (?obj - obj ?truck - truck ?loc - location)
  :precondition
   (and (at ?truck ?loc) (at ?obj ?loc))
  :effect
   (and (not (at ?obj ?loc)) (in ?obj ?truck))
   )
```

Figure 1. An example of a PDDL encoding of a LOAD-TRUCK operator.

experimental analysis, that involved more than 1,000 planning instances and required more than 6 months of CPU-time, demonstrates the usefulness of MEvo, and provides valuable insights into the structure and size of macro sets whose impact on the performance of particular planning engines is considerable. A preliminary version of MEvo has been presented in a conference paper (Vallati et al., 2017). Here we significantly extend that work by (i) considering a larger and more varied number of planning engines; (ii) analysing the impact of the threshold on the performance of MEvo; and (iii) providing an in-depth assessment of the behaviour of the parallel elements of MEvo.

The remainder of this paper is organised as follows. Section 2 provides the necessary background on AI planning. Related work is introduced and discussed in Section 3. Then, in Section 4, the proposed MEvo approach is introduced. After that, the results of the experimental analysis are presented and discussed. Finally, conclusions are given.

2. Background

Classical planning deals with finding a partially or totally ordered sequence of actions to transform the environment from an initial state to a desired goal state, with a static and fully observable environment and deterministic and instantaneous actions (Ghallab et al., 2004).

In the classical (STRIPS) representation the environment is described by first-order logic predicates. *States* are defined as sets of grounded predicates (atoms). Atoms that are present in a given state are considered to be true in that state while atoms not present in a given state are considered to be false in that state. We say that $o = (name(o), pre(o), \mathit{eff}^-(o), \mathit{eff}^+(o), cost(o))$ is a *planning operator*, where $name(o) = op\_name(x_1, \ldots, x_k)$ (op_name is a unique operator name and $x_1, \ldots x_k$ are variable symbols (arguments) appearing in the operator) and $pre(o)$, $\mathit{eff}^-(o)$ and $\mathit{eff}^+(o)$ are sets of (ungrounded) predicates with variables taken only from $x_1, \ldots x_k$ representing o's precondition, negative and positive effects, and $cost(o)$ is a numerical value representing o's cost[1]. *Actions* are grounded instances of planning operators. An action a is *applicable* in a state s if and only if $pre(a) \subseteq s$. Application of a in s (if possible) results in a state $(s \setminus \mathit{eff}^-(a)) \cup \mathit{eff}^+(a)$.

A *planning domain model* is specified by a set of predicates and a set of planning operators. A *planning problem* is specified via a domain model, a set of objects, initial state and set of goal predicates. A *solution plan* of a planning problem is a sequence of actions such that their consecutive application starting in the initial state results in a

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

state containing all the goal predicates.

An example of a PDDL operator that can be used to model the loading of a truck, is given in Figure 1. The operator specification includes the list of parameters, i.e. variables that can be involved, preconditions that need to be satisfied in order to apply the operator, and the set of effects that the application of the operator would have on the state of the world.

Macro-operators (macros) are encoded in the same way as ordinary planning operators, but encapsulate sequences of planning operators (Chrpa, 2010). Technically, an instance of a macro is applicable in a state if and only if a corresponding sequence of operators' instances is applicable in that state and the result of the application of the macro's instance is the same as the result of application in the corresponding sequence of operators' instances. Macros can be added to the original domain models, which gives the technique the potential of being *planner independent*.

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

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

Instantiating two or more macro's variables to a same object might make the macro unsound. For example, in the Blocks-World domain, a macro pickup-stack(?x ?y) that has (clear ?x)(ontable ?x)(clear ?y)(handempty) in its precondition can be instantiated into pickup-stack(A A) that is applicable if (clear A)(ontable A)(handempty) is true in a current 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) and hence pickup-stack(?x ?y) is unsound. These situations can be easily identified by checking whether the same object substitution leads to the situation in which the first operator deletes a precondition for a second operator. To make the affected macro sound, *inequality constraints* have to be added to macro's precondition (e.g. (not (= ?x ?y)) is added into pickup-stack(?x ?y)'s precondition). For more details, see (Chrpa, 2010).

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

3. Related Work

The use of macros dates back to 1970s in which the system, called REFLECT (Dawson and Siklóssy, 1977), was developed. REFLECT builds macros from pairs of primitive operators that can be successively applied and share at least one argument. In 1980s, several other macro techniques were developed. MORRIS (Minton, 1988) learns macros from parts of plans appearing frequently (S-macros) or being potentially useful despite having low priority (T-macros). Macro Problem Solver (Korf, 1985) learns macros for particular non-serializable sub-goals (e.g. in Rubik's cube).

Since 1998, the International Planning Competition (IPC) is being organised which is boosting the research in Automated Planning. Importantly, IPC led to development of the PDDL language (Mcdermott et al., 1998) that became the official language of the competition and hence the majority of existing planning engines supports it. Consequently, numerous techniques have been proposed for extracting macros. Well-known

examples include MUM (Chrpa et al., 2014), MacroFF (Botea et al., 2005), Wizard (Newton et al., 2007) and DBMP/S (Hofmann et al., 2017). Such techniques rely on a training phase where training plans, solutions of simple problems generated by a given planning engine, are analysed. Doing so gives useful insights into structure of plans that can be exploited (in the form of macros) for solving harder instances. The number of training plans might considerably vary as, for instance, DBMP/S uses 20–100 training plans while MUM uses only 5–8 training plans. More importantly, if training problem instances are not sufficiently representative (e.g. they are structurally different than "testing" problem instances) generated macros might have detrimental effect (Chrpa and Vallati, 2018). Alhossaini and Beck (2013) address this issue, to some extent, by learning a classification model according to performance of different sets of macros on training problems. However, as before, training-based techniques are very dependent on the selection of training problems, which have to accurately represent the whole class of possible problems. These techniques are usually either planner-independent such as MUM (Chrpa et al., 2014), or planner-specific such as SOL-EP version of MacroFF (Botea et al., 2005). The former approaches are designed to perform well in general regardless of planning techniques, while the latter approaches are tailored to a specific planning technique. Wizard (Newton et al., 2007) is a technique that, to some extent, combines both. Wizard uses a genetic algorithm to extract macros from training plans and cross-validates these macros on a given planner. Although Wizard achieved promising results, it is outperformed by recent techniques (Chrpa et al., 2014). Dulac et al. (2013) propose techniques to estimate the length of the macro-operators, and offline techniques to select from a library of possibilities the macro-actions to be used online for a specific planning problem.

Macro actions have also been investigated in numerical planning. Scala (2014) proposed an approach for identifying useful macro actions that can be exploited for improving the plan repair task.

Most of macro-generation techniques aim at "general improvement", i.e., provide overall improvement to any possible planner on instances from the considered domain. As state-of-the-art planners perform grounding in a pre-processing step (before search), a possibly large number of instances of macros might have detrimental effects on planners – both CPU time wise and memory wise. For this reason, macro generation techniques are rather "defensive" which means that generated macros are often short (encapsulate 2-3 original operators), in order to reduce the number of possible instances. The CA-ED version of MacroFF (Botea et al., 2005) accommodates a "locality rule" for restricting the number of macros' arguments while MUM (Chrpa et al., 2014) exploits "outer entanglements" (Chrpa et al., 2018) as a heuristic for generating macros with limited number of instances. "Instance wise" macros do not increase the branching factor as well as memory requirements that much, which reduces their possibly detrimental effects on planners' performance. On the other hand, such macros are quite conservative and might not capture some useful longer operator sequences. More aggressive techniques, such as BloMa (Chrpa and Siddiqui, 2015) that exploits block deordering (Siddiqui and Haslum, 2012) for generating "macro blocks" from training plans, aims at generating longer macros. Whereas the longer macros have many more instances and thus the branching factor and memory requirements can considerably increase, they can capture meaningful and repetitive non-trivial activities (e.g. shaking a cocktail and cleaning the shaker afterwards). As shown in literature, in some cases (e.g. the Barman domain), BloMa considerably outperforms "instance-wise" techniques such as MUM (Chrpa and Siddiqui, 2015).

Online macro generation techniques such as Marvin (Coles et al., 2007), OMA (Chrpa et al., 2015), or DHG (Armano et al., 2004), can generate specific macros for a given problem. On the one hand, online generation does not require any training instance –thus

Algorithm 1 The MEvo algorithm

procedure MEvo(D,I,p,M,n)
  initialiseMacroScores(M)
  for all i in I **do**
    R = genRandomMacros(n)
    $Tm = \{m \mid m \in M, score(m) > avg\_score(M)\}$
    B = selectTop(Tm,n)
    AB = selectTop(B,j) $\cup$ selectTop($M \setminus B$,l)
    X, Y = solveInstance(D,$\emptyset$,R,B,AB,i,p)
    for all m in X **do**
      $score(m) = score(m) + \frac{100 - score(m)}{|X|} * \left(1 - \frac{|score(m) - avg\_score(M)|}{100}\right)$
    end for
    for all Q in Y **do**
      for all m in Q **do**
        $score(m) = score(m) - \frac{score(m)}{|Q|} * \left(1 - \frac{|score(m) - avg\_score(M)|}{100}\right)$
      end for
    end for
  end for
end procedure

avoiding a possibly long training process. However, limited information is available online and, for the sake of efficiency, the extraction process cannot be too sophisticated and hence training-based techniques are often more efficient than online techniques (Chrpa et al., 2015).

4. The MEvo Approach

The core of the proposed MEvo approach can be abstracted to Algorithm 1, taking five parameters: a domain model D, a stream of problem instances I, a solver p, a pool of macros M and the maximum number of macros n for being considered in a set. Firstly, the *score* for each macro is initialised. It is in the range $[0, 100]$, where 0 indicates that the macro has never improved the performance of the planner, while 100 indicates a very useful macro, that always had a positive impact on the performance of the considered solver. Initially (the *initialiseMacroScores* function), the score can be assigned randomly, or it can rely on some previously observed performance or on experts' knowledge. In our implementation the initial score has been set to 10: this can be seen as a defensive strategy, as it relies on the fact that macros are expected to provide a very limited speed-up to the planning process.

Solving the stream of problem instances I one by one such that four sets of macros (including the empty set) are generated according to scores of macros ($score(m)$). The generated macro sets are then used for creating four different domain models which are exploited in four parallel runs of the solver p. As soon as one run finishes solving the instance, all of them are terminated and corresponding results are recorded in sets X, Y where X stands for the set of macros present in a domain model that returned a solution while Y stands for the remaining set of macros presented in domain models whose run was terminated. Finally, macro scores are updated according to the performance observed. It is of critical importance to stop all runs as soon as one finds a solution: otherwise,

MEvo would be as slow as its slowest run, thus wasting a lot of valuable CPU-time. As a drawback, this does not allow to exploit any information on the relative performance of the considered sets.

In particular, four different sets of macros are generated ($avg\_score(M)$ is an average score of all the macros in M):

- **Original**: an empty set, i.e. the original domain model;
- **Random**: a randomly selected non-empty set of at most n macros (from M);
- **Best**: at most n macros whose score is higher than avg_score (the Tm set in Algorithm 1);
- **AlmostBest**: a set of macros including top j macros from the best set where j is (strictly) smaller than the size of the best set, and top l macros ($l \geq 1$) that were not included in the best set. The values of j and l are randomly chosen, but $j + l \leq n$ must hold.

The use of the original model guarantees that, in the case in which all the sets of macros have a detrimental impact, the problem will be solved as if running without knowledge. The random set allows to explore the space of macros. The best set exploits the set of macros believed to be the most suitable as the macros have the highest score. Finally, the almost best set includes (with some probability) part of the best set and part of the "nearly best" macros, so it aims at exploring possible useful macros that might not (yet) qualify to the best set.

Hence, four different configurations are run in parallel (the solveInstance function in Algorithm 1) and the "winning" configuration (macro set) is stored in X while the "losing" configurations are stored in Y. If none of the considered configurations managed to solve a given instance i, X and Y are empty (i.e., all macro scores will remain unchanged after this run since no information about the relative performance of macros is available). The score of each macro of the "winning" configuration is increased according to the expression on Line 10 of Algorithm 1. Similarly, the score of each macro of the "losing" configurations is decreased according to the expression on Line 14 of Algorithm 1.

Our score adjustment method relies on three aspects: current macro score, average score of all macros, size of the given macro set. The score value is more incremented (decremented) if it is far from the maximal (minimal) value. The reason is that for high score macros one expects that they will be nearly always a part of the "winning" configurations. Hence the score increment should not be high as the macro performance meets the expectation. On the other hand, if such a macro becomes part of the "losing" configuration, its score decrement should be higher to reflect the discrepancy between the expectation and the performance observed. Furthermore, macros scoring close to the average should get larger score adjustments in order to categorise their potential usefulness early in the process, in other words, to early identify macros that "stand out" either in a positive or negative way. As we cannot determine the performance of individual macros in a given set (unless the set contains a single macro), the score adjustment is evenly distributed to all the macros in a given set. Finally, it is worth reminding that the scoring method cannot rely on information about the relative performance –such as speedup– of the parallel runs, since as soon as one run finds a solution, all the others are terminated. Moreover, given the fact that MEvo is planner-independent, information about the search performed by the different runs –such as visited states or closeness to the goal– is also unavailable.

The proposed MEvo framework can easily accommodate changes to the macros M while solving instances from the stream I: new macros can be added, for instance, because a new macro generating technique has been developed. The initial score for a newly

added macro can be randomly initialised, fixed at the average of other macros or defined according to expectations.

5. Experimental Analysis

Here we focus on a scenario in which the structure and size of instances changes over time, and no assumptions are made on the performed training. The training could have involved different techniques to extract macros, multiple classes of differently-structured planning problems, as well as a number of planners. This scenario is relevant because it gives maximum flexibility in terms of the use of the MEvo framework, and allows to be as inclusive as possible in terms of the macros to be considered.

The main aims of the experimental analysis are: (i) assessing the usefulness of the MEvo approach in selecting useful macros; (ii) assessing the ability of MEvo to evolve the set of macros according to the considered testing instances; (iii) comparing MEvo with a parallel portfolio of planners; and (iv) comparing MEvo with state-of-the-art parallel planning approaches.

5.1. *Experimental Settings*

In this experimental analysis we considered instances from the following domains: Barman, Blocksworld, Depots, Gripper, Matching-BW, Parking, Satellite. Domains were selected due to the large availability of differently structured macros, thus providing an interesting challenge for the MEvo approach. For each of the considered domain models, macros have been generated by BLOMA (Chrpa and Siddiqui, 2015), Wizard (Newton et al., 2007), MUM (Chrpa et al., 2014), MacroFF (Botea et al., 2005) on some disjointed set of training instances. In addition, manually crafted macros have been considered; such macros encode the human expertise (intuition) of the domain. Around 15 different macros have been considered per domain model, and between 100 and 200 problems per domain have been generated using existing randomised generators. Problems have been ordered according to their size (from the simplest to the most complex), in order to simulate an environment in which problem instances evolve over time.

As benchmarking planners we chose Jasper (Xie et al., 2014), LPG-td (Gerevini et al., 2003), Mp (Rintanen, 2012, 2014), Probe (Lipovetzky et al., 2014), Yahsp (Vidal, 2014), Lama (Richter and Westphal, 2010), Mercury (Domshlak et al., 2015) and SIW (Lipovetzky et al., 2014). All the planners competed in International Planning Competitions (IPCs), with remarkable results; moreover they exploit very different techniques for finding satisficing plans. All of them have been used in the configuration for finding one satisficing plan (no incremental search for improving plan quality) and, whether possible, seeds have been fixed.

A runtime cutoff of 900 CPU seconds (15 minutes, as in learning tracks of IPC) was used. IPC score (time metrics), as defined for the learning track of the IPC 2011 has been considered. For a planner $\mathcal{S}$ and a problem p, $score(\mathcal{S}, p)$ is defined as:

$$score(\mathcal{S}, p) = \begin{cases} 0 & \textit{if } p \textit{ is unsolved} \\ \frac{1}{1+\log_{10}\left(\frac{T_p(\mathcal{S})}{T_p^*}\right)} & \textit{otherwise} \end{cases}$$

Set		Domains						
		Bar	Bw	Dep	Grip	MBW	Park	Sat
		150	160	165	110	150	140	140
Jasper	O	94.3	57.2	84.6	109.6	**36.8**	0.0	40.3
	B	113.6	**86.2**	**86.1**	109.6	34.7	0.0	**61.6**
	A	**115.1**	40.3	78.9	41.1	34.3	0.0	53.3
	MEvo	130.0	97.0	120.0	110.0	49.0	0.0	79.0
LPG	O	0.0	70.2	**146.2**	99.8	**24.2**	0.0	116.3
	B	0.0	**158.4**	107.6	**104.8**	22.5	0.0	**129.1**
	A	0.0	98.7	111.6	101.7	23.2	0.0	129.0
	MEvo	0.0	160.0	165.0	110.0	33.0	0.0	135.0
Mp	O	13.5	0.0	147.8	79.2	**1.2**	1.0	74.0
	B	**15.5**	**98.5**	**162.8**	**109.6**	1.1	1.0	101.2
	A	14.5	47.8	159.9	72.5	1.0	0.0	**110.3**
	MEvo	24.0	106.0	165.0	110.0	3.0	1.0	117.0
Probe	O	89.6	106.6	**164.5**	89.5	37.8	2.0	17.1
	B	**98.4**	**140.8**	159.0	**106.5**	**42.2**	2.0	**43.9**
	A	93.5	135.2	157.6	105.3	34.7	0.0	40.3
	MEvo	128.0	144.0	165.0	110.0	64.0	2.0	59.0
Yahsp	O	0.0	0.0	0.0	0.0	6.9	0.0	0.0
	B	0.0	0.0	3.7	**98.4**	7.0	0.0	0.0
	A	0.0	0.0	**8.0**	48.7	7.0	0.0	0.0
	MEvo	0.0	0.0	11.0	110.0	7.0	0.0	0.0
SIW	O	6.6	139.9	90.8	39.1	2.7	57.9	0.13
	B	12.6	**146.0**	131.1	**107.5**	**6.4**	57.9	41.0
	A	**19.6**	67.2	**146.8**	103.3	5.8	0.1	**41.5**
	MEvo	21.0	148	155.0	110.0	8.0	57.9	47.0
Mercury	O	88.0	86.8	21.7	86.2	16.5	17.9	72.4
	B	108.3	**143.5**	49.7	**108.0**	**52.4**	17.9	**134.7**
	A	**110.1**	75.6	**61.9**	98.8	42.4	4.0	127.3
	MEvo	130.0	147.0	67.0	110.0	55.0	19.0	138.0
LAMA	O	86.6	130.0	44.8	108.5	**51.7**	**25.7**	72.8
	B	**107.4**	**138.5**	**141.6**	**108.7**	51.0	22.8	105.9
	A	105.2	114.2	141.5	48.2	51.3	4.6	**111.2**
	MEvo	116.0	152.0	147.0	110.0	57.0	41.0	126.0

Table 1. Accumulated IPC score of the considered planners exploiting the Original domain model (O), the Best macro set (B) and the AlmostBest set (A) on the selected domains. The MEvo row reports the overall performance achieved by the planner when exploiting the approach proposed. Bar, Bw, Dep, Grip, MBW, Park and Sat stand for, respectively, Barman, Blocksworld, Depots, Gripper, Matching-BW, Parking and Satellite. The number of testing instances is indicated on the top of each column. Bold indicates the best IPC score among O, B and A sets for a given planner.

where $T_p(\mathcal{S})$ is the CPU time needed by a planner $\mathcal{S}$ to solve the problem p and T_p^* is the CPU-time needed by the best considered planner, otherwise. The total IPC score is the sum the scores achieved on each considered instance.

We fixed the maximum size of macro sets (n in Algorithm 1) to $n = 3$: this has been empirically observed to be a good upper bound (see, e.g., (Botea et al., 2005; Chrpa et al., 2014)). All the experiments were run on a quad-core 3.0 Ghz CPU, with 4GB of RAM made available for each run. Presented results are averaged across three runs, in order to take –to some extent– randomness into account. Overhead due to macro selection and scores update is negligible (few milliseconds).

5.2. *MEvo Performance on Considered Benchmarks*

In this section we investigate the ability of MEvo in identifying suitable macro sets. Table 1 shows the results achieved by the considered planners, in terms of accumulated IPC score, when exploiting the Original domain model (O), the Best macro set –according to MEvo– (B) and the AlmostBest set (A). The results in terms of coverage are presented in Table 2. The overall performance of the proposed approach is shown; clearly, they should

Set		Domains						
		Bar	Bw	Dep	Grip	MBW	Park	Sat
		150	160	165	110	150	140	140
Jasper	O	69.3	36.9	55.2	100.0	26.7	0.0	32.1
	B	80.7	55.0	56.3	100.0	26.0	0.0	45.0
	A	80.7	31.3	52.1	46.4	26.	0.0	38.5
	MEvo	86.7	60.6	72.7	100.0	32.7	0.0	56.4
LPG	O	0.0	76.9	100.0	100.0	17.3	0.0	100.0
	B	0.0	100.0	95.2	100.0	18.7	0.0	100.0
	A	0.0	63.1	100.0	98.2	19.3	0.0	100.0
	MEvo	0.0	100.0	100.0	100.0	22.0	0.0	100.0
Mp	O	10.7	0.0	100.0	100.0	1.3	0.7	60.0
	B	11.3	62.5	100.0	100.0	1.3	0.7	75.7
	A	10.7	31.3	98.8	77.3	0.7	0.0	80.7
	MEvo	16.0	66.3	100.0	100.0	2.0	0.7	83.6
Probe	O	68.0	77.5	100.0	100.0	27.3	1.4	15.0
	B	73.3	88.8	100.0	100.0	32.7	1.4	32.1
	A	71.3	86.3	100.0	100.0	27.3	0.0	29.3
	MEvo	85.3	90.0	100.0	100.0	42.7	1.4	42.1
Yahsp	O	0.0	0.0	0.0	0.0	4.7	0.0	0.0
	B	0.0	0.0	2.4	99.1	4.7	0.0	0.0
	A	0.0	0.0	4.8	68.2	4.7	0.0	0.0
	MEvo	0.0	0.0	6.7	100.0	4.7	0.0	0.0
SIW	O	6.0	92.5	58.7	100.0	3.3	42.1	3.6
	B	10.0	92.5	83.0	100.0	5.3	42.1	32.9
	A	14.0	48.1	93.9	100.0	4.7	0.7	33.5
	MEvo	14.0	92.5	93.9	100.0	5.3	42.1	33.5
Mercury	O	86.7	70.0	20.6	100.0	13.3	13.6	97.2
	B	86.0	91.9	33.4	100.0	36.7	13.6	98.6
	A	80.0	62.5	40.6	91.8	32.0	3.6	97.2
	MEvo	86.7	91.9	40.6	100.0	36.7	13.6	98.6
LAMA	O	74.0	90.6	46.7	100.0	38.7	29.4	73.5
	B	77.3	95.0	89.1	100.0	36.7	27.5	85.7
	A	76.7	87.5	89.1	61.8	38.0	15.6	90.0
	MEvo	77.3	95.0	89.1	100.0	38.7	29.4	90.0

Table 2. Percentage of solved instances of the considered planners exploiting the Original domain model (O), the Best macro set (B) and the AlmostBest set (A) on the selected domains. The MEvo row reports the overall performance achieved by the planner when exploiting the proposed approach. Bar, Bw, Dep, Grip, MBW, Park and Sat stand for, respectively, Barman, Blocksworld, Depots, Gripper, Matching-BW, Parking and Satellite. The number of testing instances is indicated on the top of each column.

not be directly compared with the performance of O, B and A sets, since MEvo subsumes them. The results of planners exploiting the random sets are not shown because they are only used on exploring the space of macros and these sets only occasionally outperform the other sets. It should be noted that –given a stream of instances on a specific domain– Best and AlmostBest sets contain different macros, according to the evolution of scores of considered macros.

As a baseline –not shown in Tables 1 and 2– we run experiments where the domain models were extended using all the available macros; none of the considered planners was able to solve any instance. Expectedly, these results support the need of selecting good quality macros (from the available pool).

From Tables 1 and 2, three main conclusions can be derived. First, the same macros can have a very different impact on planners exploiting different approaches. This behaviour, even though well-known, is confirmed by the variability of the relative performance between the Original and the Best sets, and indicates the need for systems that can automatically identify the best set of macros to exploit –if any–. The second observation we derived from the results is that the best set of macros that outperforms the original encodings is usually established after solving a relatively small number of instances; see, for instance, the performance of Yahsp in the Gripper domain. Third, Table 1 clearly

indicates that the MEvo algorithm can effectively exploit the speed-up given by macros while guaranteeing at least the same performance as for the original models.

With regards to the size of Best macro sets that are identified by MEvo, and to the macros included, it is worth noting that both aspects vary among the planners considered. For instance, Yahsp and Mp usually exploit sets composed by a single macro. Conversely, Probe shows a significant performance improvement when exploiting set composed by 2-3 macros. In terms of the structure of the exploited macros, Mp can better exploit short macros, i.e. those encapsulating a small number of original operators (2-3), while Probe can also exploit macros encapsulating 4-5 operators.

We noticed that in cases where the accumulated IPC score of the Original and Best sets are very similar –with regard to Table 1– this is due to: (i) no macros are improving the performance of the planner; (ii) the macros speed-up is limited; (iii) there is a large number of macros that are, in turn, providing limited improvement on different instances. In principle, the performance of the Best set can be improved by introducing a threshold (currently, only the average value is considered). However, given the fact that the Original set guarantees a lower bound on performance, we preferred to give some freedom to the Best set; this can potentially lead to better overall performance of the MEvo framework.

Figure 2 shows the pattern of the evolution of the accumulated IPC score of the four sets of macros considered by MEvo. Specifically, the example shows the performance of the considered planning systems in a continuous stream of instances taken from the Blocksworld domain. We omitted Yahsp due to the fact that the planner does not solve any instance of the domain.

Let us focus on the graph showing the evolution of the IPC scores of Probe. Initially, the performance of Original and Best sets are exactly the same: there is no available information about promising macros –i.e., no macro has a score which is higher than the average– therefore an empty set is also used by the Best set. As soon as some exploration has been performed by the Random set, and some useful macros have been identified, the performance of the Best set improves and the gap between the Original and Best sets increases. At the same time, the AlmostBest set also shows performance improvement due to the exploitation of a subset of the most promising macros set. At some point, after Probe processed approximately 125 instances, the remaining original problem instances became too large to be solved. Conversely, macros are still very helpful and thus Probe can solve such large instances.

Other considered planners can show significantly different patterns of evolution. With the Mp planner, MEvo is able to identify very quickly the Best macro set, and therefore the gap between this set and any other set is expanding since the very first instances. At instance 100, Mp is not able to solve any additional instance, as it runs out of memory: this is the reason of the plateau that can be observed in the corresponding graph. For LPG and Mercury, Mevo is able to assemble the Best macro set after solving about 25 instances. After that point, the performance gap between the Best and the Original sets increases significantly, and the Almost best set starts to achieve some remarkable performance. The same happens for Jasper, but a larger number of instances has to be solved first. Finally, for the LAMA and SIW planners, macros do not seem to be particularly useful in the Blocksworld domain, and for this reason the performance of Original and Best sets is very similar.

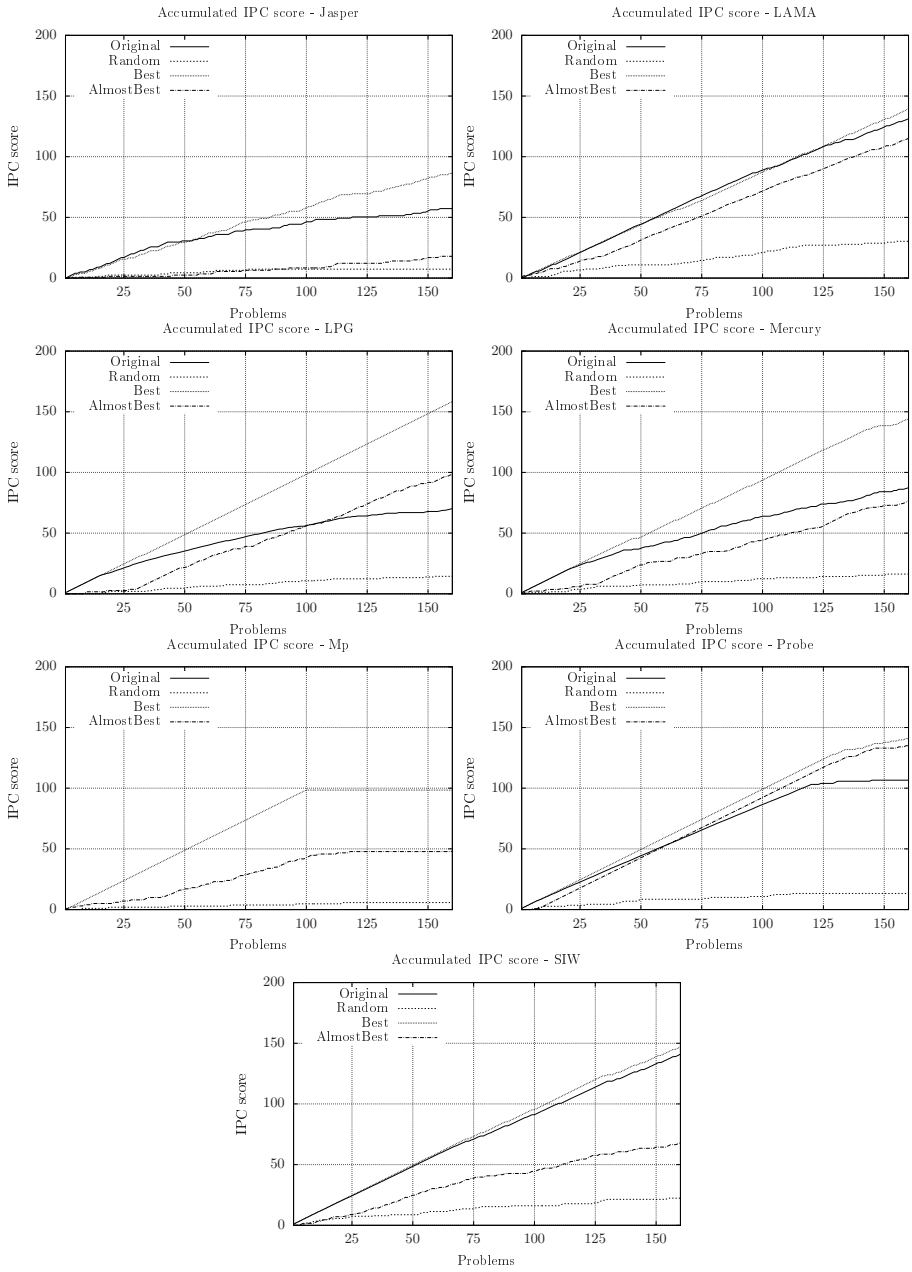

Figure 2. Accumulated IPC score of the considered planners exploiting the four considered macro sets on Blocksworld instances.

5.3. *Comparison Against the Virtual Best Solver*

The results presented in the previous section indicate that MEvo is generally able to quickly identify a suitable set of macros to exploit in order to improve the performance of the considered planner. Since MEvo runs on multiple CPUs at the same time, a natural question that may arise is whether it is worthy to invest resources for extracting macros and selecting the best sets over time, or it would be better (and easier) to exploit available cores by running multiple different planners in parallel. To answer this legitimate question, we compare the performance of the different planners, in the MEvo framework, against a virtual best solver (VBS). The virtual best solver is selected as the sequential

IPC Score									
	VBS	MEvo							
		Jasper	LPG	Mp	Probe	Yahsp	SIW	Mercury	LAMA
Bar	75.8	59.6	0.0	9.2	86.9	0.0	8.4	**120.9**	105.3
Bw	103.3	44.2	139.8	57.1	66.4	0.0	107.5	**140.3**	121.8
Dep	132.7	37.1	**147.9**	132.3	78.9	3.1	104.6	38.0	86.9
Grip	87.5	43.8	95.6	79.2	51.5	47.4	**109.3**	56.5	79.2
MBW	48.3	19.1	22.5	0.8	29.9	2.6	8.2	45.3	**50.2**
Park	57.8	0.0	0.0	0.5	0.9	0.0	**58.0**	10.6	27.4
Sat	115.3	28.0	**134.0**	70.7	20.9	0.0	16.3	90.3	80.2

Coverage									
	VBS	MEvo							
		Jasper	LPG	Mp	Probe	Yahsp	SIW	Mercury	LAMA
Bar	74.0	**86.7**	0.0	16.0	85.3	0.0	14.0	**86.7**	77.3
Bw	92.5	60.6	**100.0**	66.3	90.0	0.0	92.5	91.9	95.0
Dep	**100.0**	72.7	**100.0**	**100.0**	**100.0**	6.7	93.9	40.6	89.1
Grip	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
MBW	38.7	32.7	22.0	2.0	**42.7**	4.7	5.3	36.7	38.7
Park	**42.1**	0.0	0.0	0.7	1.4	0.0	**42.1**	13.6	29.4
Sat	**100.0**	56.4	**100.0**	83.6	42.1	0.0	33.5	98.6	90.0

Table 3. IPC score (top table) and coverage (bottom table) of the virtual best solver (VBS), and of the considered sequential planners when exploiting the MEvo framework. Bar, Bw, Dep, Grip, MBW, Park and Sat stand for, respectively, Barman, Blocksworld, Depots, Gripper, Matching-BW, Parking and Satellite. Bold indicates best results. The results shown are averaged over 3 runs.

planner that delivers the best performance on a given domain when running on the original domain model. In a nutshell, this corresponds to a situation where all the selected planners are run in parallel for solving a given problem, and stopped as soon as the fastest finds a solution. Considering the number of planners included in our experimental analysis, the VBS simulates the use of 8 cores in parallel, while MEvo runs on 4 cores. On the other hand, MEvo can exploit the additional knowledge of the domain –under the form of macro actions– that is not available to the planners used in the VBS.

Table 3 shows the results of the comparison performed in terms of IPC score and coverage. As it is apparent, from the results shown, macro actions can provide a significant performance improvement: MEvo outperforms the VBS in all the considered domains in terms of IPC. This, of course, does not mean that any planner is able to outperform the VBS when empowered with macros in the MEvo framework. However, a good performing planner is likely to obtain a performance boost that can outperfom the use of multiple different planners in a parallel run on the original domain model. Similar results can be observed in terms of coverage.

Summarising, MEvo is a promising framework for exploiting multiple cores when macros are available, and can in general lead to performances that are better than those achieved by running a set of planners on original domain models.

5.4. *The Impact of the Maximum Number of Macros per Set*

Following the results observed in literature, in our experimental analysis the maximum size of macro sets (n) has been fixed to $n = 3$. It is however of interest to understand how

Set		Domains			
		Blocksworld	Depots	Gripper	Matching-BW
Jasper	1	**112.6** (61.1)	72.5 (55.2)	**109.9** (100.0)	17.2 (23.3)
	3	97.8 (55.0)	**87.3** (56.3)	**109.9** (100.0)	28.4 (26.0)
	5	110.9 (60.6)	65.3 (55.2)	106.3 (100.0)	**30.9** (26.7)
LAMA	1	135.8 (93.1)	137.4 (87.9)	108.9 (100.0)	46.0 (32.7)
	3	**136.3** (95.0)	**139.6** (89.1)	**109.9** (100.0)	**52.1** (36.7)
	5	120.9 (91.3)	135.2 (87.9)	108.5 (100.0)	49.3 (35.3)
LPG	1	145.6 (100.0)	148.9 (95.2)	104.2 (100.0)	**25.6** (19.3)
	3	148.8 (100.0)	150.1 (95.2)	107.0 (100.0)	23.7 (18.7)
	5	**155.8** (100.0)	**150.2** (95.2)	**108.0** (100.0)	22.1 (18.7)
Mercury	1	145.4 (91.9)	39.6 (30.3)	**108.8** (100.0)	36.7 (26.7)
	3	**146.4** (91.9)	**49.2** (33.4)	108.7 (100.0)	**51.4** (36.7)
	5	**146.4** (91.9)	44.4 (30.3)	108.7 (100.0)	50.1 (34.7)
Mpc	1	**105.8** (62.5)	161.3 (100.0)	107.8 (100.0)	1.8 (1.3)
	3	**105.8** (62.5)	160.0 (100.0)	**109.8** (100.0)	**2.0** (1.3)
	5	105.7 (62.5)	**161.8** (100.0)	107.2 (100.0)	**2.0** (1.3)
Probe	1	**141.4** (88.8)	**161.3** (100.0)	**108.8** (100.0)	**46.5** (33.4)
	3	139.5 (88.8)	159.9 (100.0)	107.0 (100.0)	46.1 (32.7)
	5	130.2 (88.8)	159.1 (100.0)	108.0 (100.0)	45.3 (32.7)
SIW	1	**146.6** (92.5)	**130.2** (83.0)	108.9 (100.0)	7.9 (5.3)
	3	145.7 (92.5)	129.4 (83.0)	**109.1** (100.0)	7.9 (5.3)
	5	144.9 (92.5)	128.2 (83.0)	108.4 (100.0)	7.9 (5.3)

Table 4. Accumulated IPC score (percentage of solved instances) of the considered planners exploiting the Best macro sets, with a different maximum number of macros, on the four selected domains. Bold indicates the best IPC score for a given planner.

such value can affect the overall performance of the proposed MEvo approach. Intuitively, using a larger n should allow to better identify synergies among macros, at the expenses of the number of iterations needed to select a stable and high performing Best set. On the other hand, in cases where a very small set of macros (1–2) is needed to boost the performance of the planner considered, it would instead be better to consider smaller values of n.

Table 4 shows the results of the performed analysis on the impact of the n value. Results are presented in terms of IPC score and coverage of the best macro set. We focused on the Best set because it is the set that should be mostly affected by the threshold, and would therefore better reflect the overall impact of the n value on MEvo.

In our investigation, we considered three values of n: 1, 3, and 5. Among those used in our experimental analysis, four domains have been selected: Blocksworld, Depots, Gripper, and Matching-BW. Blocksworld is the domain where, on average, the MEvo Best macro set showed to consistently improve the performance of the planners. On the contrary, Matching-BW is the domain where the Best macro set does not seem to be able to generally identify useful macros for the planners considered. Gripper allows all considered planners to solve the vast majority of considered benchmarks, while Depots is a domain where MEvo provides some "average" improvements for the planners. For our analysis, we omitted Yahsp due to the limited solving capability the planner showed on the selected benchmarks.

The results in Table 4 suggest that there is not a single n value that can work well for all the considered planners and benchmarks, but that instead the best value can vary. Jasper is a striking example: it delivers the best performance with $n = 1$ in Blocksworld, and $n = 5$ in Matching-BW. LAMA is instead the planner that delivers the best performance exactly when $n = 3$. SIW and Probe are two planners that seem to perform better with $n = 1$, while LPG works better for higher values of n.

Summarising, it looks that the best value of n is not directly related to the specific

benchmark domain, but is more a planner-dependent value. Some planners consistently perform better for a given value of n, while for others the actual best value can vary. However, according to the results shown in Table 4 , $n = 3$ is a value that works reasonably well in general, and that can be used when no other information is available.

5.5. *Evaluation of MEvo Scoring Functions*

Equations 1 and 2 are pivotal for the evolution over time of the sets of macro considered by MEvo. Notably, given the limited amount of available information –e.g., no information about speed-up is available, since everything is stopped as soon as one run finds a solution– aforementioned equations exploit the knowledge derived from the size of sets, and the average scores of macros in the pool. In order to understand the importance of these two components of Equations 1 and 2, we designed two modified versions of MEvo:

- **MEvoN**: the *size* of the set is not considered when updating the c_v score;
- **MEvoAv**: the distance of the c_v from the *average* score is ignored.

Both versions were then tested on the complete benchmark set, using the Probe planner, which generally gains the best improvement when exploiting selected macros (see Table 1). Unsurprisingly, we observed that both MEvoN and MEvoAv perform similarly to MEvo when the Best set for the selected planner includes a single macro, and such macro allows to improve the performance of the planner in most of the considered instances. Otherwise, MEvo tends to be better because: (i) it allows a faster convergence to a good set of macros; and (ii) its score update function has been designed for limiting the impact of cases in which macros perform very differently than usual on very few instances. Summarising, the experiments discussed in this section demonstrated that MEvo scoring functions provide robustness against "outliers", yet they allow quick convergence on promising macro sets.

5.6. *Comparison with the State of the Art of Parallel Planning*

As previously discussed, the MEvo approach exploits four cores in parallel. It is therefore important to investigate how it performs when compared with the state of the art of multi-core parallel planning engines. As a baseline for this comparison, we considered the winner of the sequential multi-core track of the deterministic part of the 2014 edition of the International Planning Competition, ArvandHerd-14 (Valenzano et al., 2014). In that competition track, planners were allowed to exploit four cores in parallel for solving a single planning instance, exactly as MEvo does.

In our experimental analysis, ArvandHerd-14 was run on the same benchmark instances considered in Tables 1 and 2, with a cutoff of 900 wall-clock CPU seconds. Since we are evaluating the runtime of planners, ArvandHerd-14 was stopped as soon as the first solution was found. It is worthy to remark that the MEvo framework, when running a given domain-independent planner, exploits some additional domain-specific knowledge, under the form of macros. Therefore, we expect the planners empowered by the MEvo system to achieve better performance, in terms of runtime, than ArvandHerd-14.

Table 5 shows the IPC scores achieved on the benchmark instances by ArvandHerd-14 and the planners considered in this experimental analysis, exploiting the MEvo framework. The overall results (last row of Table 5) indicate that all the planners considered, except Yahsp, outperfom ArvandHerd-14 on the considered benchmarks domains. As observed in Table 2, Yahsp is generally unable to solve a vast majority of the testing

	ArHerd	MEvo							
		Jasper	LPG	Mp	Probe	Yahsp	SIW	Mercury	LAMA
Bar	36.3	59.6	0.0	9.2	86.9	0.0	8.4	**120.9**	105.3
Bw	38.3	44.2	139.8	57.1	66.4	0.0	107.5	**140.3**	121.8
Dep	22.7	37.0	**147.9**	132.3	78.9	3.1	104.6	38.0	84.9
Grip	41.8	43.8	95.6	79.2	51.5	47.4	**109.3**	56.5	79.2
MBW	**65.0**	19.1	22.1	0.8	28.3	2.5	8.2	44.2	49.1
Park	0.0	0.0	0.0	0.5	0.9	0.0	**58.0**	10.6	27.4
Sat	16.2	28.0	**134.0**	70.6	21.0	0.0	16.3	90.3	80.2
Total	220.3	231.7	539.4	349.7	333.9	53.0	412.3	500.8	**547.9**

Table 5. IPC score of the parallel planner ArvandHerd-14 (ArHerd), and of the considered sequential planners when exploiting the MEvo framework. Bar, Bw, Dep, Grip, MBW, Park and Sat stand for, respectively, Barman, Blocksworld, Depots, Gripper, Matching-BW, Parking and Satellite. Bold indicates best IPC score. Results shown for the MEvo approach are averaged over 3 runs.

instances.

From a domain perspective, results in Table 5 show that in most of the considered domains, all the planners (ignoring Yahsp) running on the MEvo framework are able to achieve considerably better results than ArvandHerd-14. In Matching-BW, ArvandHerd-14 clearly outperforms MEvo. This is because macros are helpful only on smaller problems and considered planners are not very efficient in this domain. As we recognise that MEvo is also exploiting some additional domain-specific knowledge, which is not available for ArvandHerd-14, we believe that the results of the performed comparison suggest that running differently extended domain models –according to identified promising macros– can be a fruitful way for exploiting multiple cores. This is particularly true if we take into account that, out of the four cores used by MEvo, only two are actually exploiting "smartly" the additional knowledge: Original and Random sets are, respectively, used for backup and exploration of macro sets purposes. This observation is of particular interest for the future development of the area. Specifically, in the light of the fact that most of the works in parallel planning focused mainly on the "reasoning" side, by running different algorithms, or the same algorithm investigating different areas of the search space, on each available core.

6. Discussion

In this section we discuss how sudden changes in the structure of planning problems can be handled by MEvo, and we briefly comment on the well-known downside of using macros, i.e. the potentially significant reduction of plans' quality.

6.1. *Sudden Changes of Problem Structure*

In the previous section we considered scenarios in which there are smooth structural changes of the problem instances over time: this has been done by considering instances with an increasing number of objects involved. Here we are interested in investigating the capability of MEvo to cope with sudden structural changes, which might occur in real-world applications. For instance, in a Depots-like scenario, it can be the case that a company extends its area of influence through the acquisition of an overseas competitor.

For this analysis, we considered the well-known Gripper domain. In such a domain there is a robot, with two grippers, which has to move a number of balls from one room

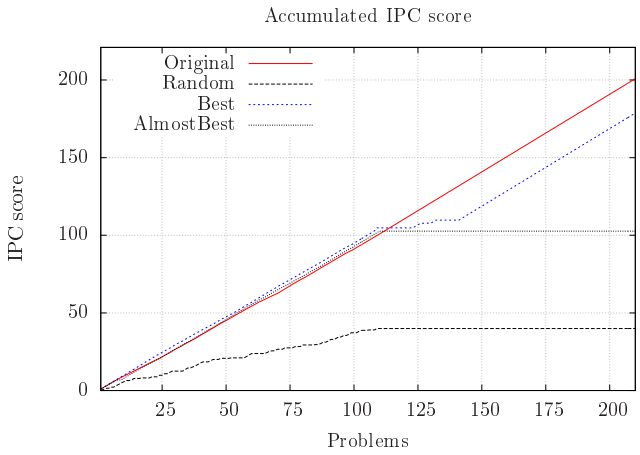

Figure 3. Accumulated IPC score of LPG exploiting the four considered macro sets on Gripper instances. The structure of instances suddenly changes at number 110.

to another. Using the available problem generators, we created 220 instances. The first half has the usual two-rooms structure, i.e. all the balls are in one room and have to be moved to the other one. The remaining 110 planning instances have an increasing number of rooms, from 10 to 20. Every room is connected with all the others, and balls are initially distributed among available rooms. The Gripper domain has been selected due to the significant impact that this structural difference has on planners' performance. In fact, macros extracted for the first scenario do not perform well in the second case. Specifically, the use of any macro results in planners not being able to solve any problem. In our evaluation we considered the case in which macros were extracted considering the problems available at the start, i.e. two-rooms case.

Figure 3 (coloured) shows the results of the analysis, running the LPG planner. Similar behaviours have been observed on other planning systems. As expected, the Best set of macros is identified very quickly by MEvo for the two-rooms structured instances. As instances are solved in less than a second by LPG exploiting the original domain model, the speedup given by macros is not very large. When the structure of instances changes, the macros that work well on the two-rooms structure are no longer useful –see after 110 problems–; in fact, no macro has been seen as useful with a large number of rooms. On average, as the results shown are the average over three runs, it takes about 20 instances to MEvo to discover that no macro is useful. After that, the Best set is empty (and the performance is the same as in the Original set). AlmostBest and Random sets are not able to solve any instance after 110, due to the fact that available macros are useless on those instances.

Remarkably, this experiment highlights the importance of evolving the set of macros to use accordingly to the stream of problem instances. It should be mentioned that in this case macros "favoured" 2 rooms scenarios. However, it is also important to mention that a typical approach (considering the same set of macros for each problem) would fail unless the macro learning phase is manually re-invoked. The typical train-once approach would have not allowed to quickly change the set of macros to exploit. As a result, instances with more than 2 rooms would not have been solved at all.

6.2. *Impact of Macros on the Quality of Generated Plans*

It is well-known that the use of macros can have detrimental effects on the quality of generated plans; see, for instance the works of Botea et al. (2005); Chrpa (2010); Chrpa et al. (2015); Coles et al. (2007). It is however hard to clearly assess the actual impact due to a number of factors that can play a role on this matter, for instance:

- the length of the macro, in terms of the number of operators that are incorporated;
- the number of macros included in the domain model (or in a specific set used by MEvo);
- the usefulness of the macro for the planner considered, which is also related to the search techniques exploited by the planner;
- the usefulness of the macros for a specific problem or class of problems;
- the fact that the macro represents activities that must be performed in order to achieve the goals of the problem at hand.

Given the fact that the aim of MEvo is to provide a framework for automatically selecting and evolving sets of macros, focusing on the speedup provided by the sets, it is out of the scope of this work to assess the impact of macros on the quality of plans generated by the considered planners in detail. As a very general overview, we observed that taking into account only instances solved by both the Original model and the Best set, in approximately 55% of the cases generated solutions are of the same quality. In few cases, approximately 12%, macros lead to better solutions than the Original set. In all the remaining cases the use of the Original model can provide better quality results.

Notably, as MEvo focuses on improving runtime and is agnostic with regards to the planner, to the techniques used to generate considered macros, and to the pool of macros made available, the results can vary to a significant extent. It is however important to highlight this possible downside of using macros (and therefore of MEvo).

7. Conclusion

In this paper we proposed MEvo, an approach for improving the performance of domain-independent planners by dynamically combining and evolving suitable sets of macros, given a pool of macros and a continuous stream of problem instances. MEvo can cope with one of the major drawbacks of existing macros extraction approaches: the fact that the impact of extracted macros, when used on instances which are very different from those used for training or with different planners, is unknown. Remarkably, the pool of macros can be modified while instances are solved: newly-identified macros can be immediately exploited by MEvo.

Our large experimental analysis: (i) demonstrates the usefulness of MEvo as a framework for improving the performance of domain-independent planners; (ii) confirms the ability of MEvo in evolving suitable macro sets, also in case of sudden structural changes of the stream of instances; (iii) indicates that MEvo outperforms the state of the art of multicore planning; and (iv) provides insights into the size of macro sets and the structure of macros, which allows to improve the performance of different planners. Planner-specific knowledge about macros' structure, gained by using MEvo, can be useful for improving macro generation processes.

The results of the comparison between sequential planners exploiting the MEvo tool and the winner of the sequential multicore track of last IPC, are of particular interest. Specifically, in the light of the fact that most of the works in parallel planning focused

mainly on the "reasoning" side, by running different algorithms, or the same algorithm investigating different areas of the search space, on each available core. Our results suggest that also considering different domain model formulations, wisely extended with promising macros, can be a fruitful way for exploiting additional cores.

We see several perspectives for future work. We are interested in performing an analysis of smarter ways for generating the Random set of MEvo, and the use of online macro-generation approaches (e.g., (Armano et al., 2004; Coles et al., 2007)) for increasing the pool of considered macros over time. We plan to investigate techniques for estimating the appropriate size of the different macro sets, instead of having a predefined maximum value of $n = 3$. We also envisage to explore more aggressive techniques for allowing MEvo to incorporate macros into domain models, for instance by removing original operators that macros incorporate: this can improve the performance by reducing the branching factor of problems, at the cost of the potential unsolvability of problems. We are also interested in the investigation of features-based approaches (see e.g., (Cenamor et al., 2013; Fawcett et al., 2014)) in order to extract more data for updating macro scores. Furthermore, we plan to investigate how the MEvo approach can be combined with approaches that aim at configuring the order in which macros and original operators are listed in the domain model (Vallati et al., 2015b). Finally, we plan to combine MEvo with approaches, such as ASAP or PbP (Gerevini et al., 2014; Vallati et al., 2014), that are also able to combine planners into a domain-specific portfolio. That would allow to select suitable planners, and evolve the domain model accordingly.

Acknowledgements

Research was partially 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".

References

Alhossaini, M. A. and Beck, J. C. (2013). Instance-specific remodelling of planning domains by adding macros and removing operators. In *Proceedings of the Tenth Symposium of Abstraction, Reformulation, and Approximation (SARA)*.

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

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

Cenamor, I., de la Rosa, T., and Fernández, F. (2013). Learning predictive models to configure planning portfolios. In *Proceedings of the 4th workshop on Planning and Learning (ICAPS-PAL 2013)*, pages 14–22.

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

Chrpa, L. and Siddiqui, F. H. (2015). Exploiting block deordering for improving planners efficiency. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, pages 1537–1543.

Chrpa, L. and Vallati, M. (2018). Determining representativeness of training plans: A case of macro-operators. In *IEEE 30th International Conference on Tools with Artificial Intelligence, ICTAI 2018, 5-7 November 2018, Volos, Greece.*, pages 488–492.

Chrpa, L., Vallati, M., and McCluskey, T. L. (2014). Mum: A technique for maximising the utility of macro-operators by constrained generation and use. In *Proceedings of the International Conference on Automated Planning and Scheduling, ICAPS*, pages 65–73.

Chrpa, L., Vallati, M., and McCluskey, T. L. (2015). On the online generation of effective macro-operators. In *Proceedings of IJCAI*, pages 1544–1550.

Chrpa, L., Vallati, M., and 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., and Smith, A. (2007). Online identification of useful macro-actions for planning. In *Proceedings of ICAPS*, pages 97–104.

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

Domshlak, C., Hoffmann, J., and Katz, M. (2015). Red-black planning: A new systematic approach to partial delete relaxation. *Artif. Intell.*, 221:73–114.

Dulac, A., Pellier, D., Fiorino, H., and Janiszek, D. (2013). Learning useful macro-actions for planning with n-grams. In *25th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2013, Herndon, VA, USA, November 4-6, 2013*, pages 803–810.

Fawcett, C., Vallati, M., Hutter, F., Hoffmann, J., Hoos, H. H., and Leyton-Brown, K. (2014). Improved features for runtime prediction of domain-independent planners. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*.

Fitzgerald, T., Malitsky, Y., and O'Sullivan, B. (2015). Reactr: Realtime algorithm configuration through tournament rankings. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, pages 304–310.

Fox, M. and Long, D. (2003). PDDL2.1: an extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)*, 20:61–124.

Gerevini, A., Saetti, A., and Vallati, M. (2014). Planning through automatic portfolio configuration: The pbp approach. *J. Artif. Intell. Res.*, 50:639–696.

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

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

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

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

Kotthoff, L. (2014). Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 35(3):48–60.

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

López, C. L., Celorrio, S. J., and Olaya, Á. G. (2015). The deterministic part of the seventh international planning competition. *Artificial Intelligence*, 223:82–119.

Mcdermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL - The Planning Domain Definition Language. Technical Report 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*, pages 564–569.

Newton, M. A. H., Levine, J., Fox, M., and Long, D. (2007). Learning macro-actions for arbitrary planners and domains. In *Proceedings of the International Conference on Automated Planning and Scheduling, ICAPS*, pages 256–263.

Richter, S. and Westphal, M. (2010). The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. Artif. Intell. Res.*, 39:127–177.

Rintanen, J. (2012). Engineering efficient planners with SAT. In *Proceedings of ECAI*, pages

684–689.

Rintanen, J. (2014). Madagascar: Scalable planning with sat. In *The Eighth International Planning Competition. Description of Participant Planners of the Deterministic Track*, pages 66–70.

Scala, E. (2014). Plan repair for resource constrained tasks via numeric macro actions. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*.

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

Valenzano, R., Nakhost, H., Müller, M., Schaeffer, J., and Sturtevant, N. (2014). Arvandherd 2014. In *The Eighth International Planning Competition. Description of Participant Planners of the Deterministic Track*.

Vallati, M., Chrpa, L., Grzes, M., McCluskey, T. L., Roberts, M., and Sanner, S. (2015a). The 2014 international planning competition: Progress and trends. *AI Magazine*, 36(3):90–98.

Vallati, M., Chrpa, L., and Kitchin, D. E. (2014). ASAP: an automatic algorithm selection approach for planning. *International Journal on Artificial Intelligence Tools*, 23(6).

Vallati, M., Chrpa, L., and Serina, I. (2017). On the evolution of planner-specific macro sets. In *AI*IA 2017 Advances in Artificial Intelligence - XVIth International Conference of the Italian Association for Artificial Intelligence, Bari, Italy, November 14-17, 2017, Proceedings*, pages 443–454.

Vallati, M., Hutter, F., Chrpa, L., and McCluskey, T. L. (2015b). On the effective configuration of planning domain models. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, pages 1704–1711.

Vidal, V. (2014). Yahsp3 and yahsp3-mt in the 8th international planning competition. In *Proceedings of the 8th International Planning Competition (IPC-2014)*, pages 64–65.

Xie, F., Müller, M., and Holte, R. (2014). Jasper: the art of exploration in greedy best first search. In *Proceedings of the 8th International Planning Competition (IPC-2014)*.