ERA: Extracting planning macro-operators from adjacent and non-adjacent sequences

Sandra Castellanos-Paez, Romain Rombourg, Philippe Lalanda

ERA: Extracting planning macro-operators from adjacent and non-adjacent sequences

Sandra Castellanos-Paez[0000−0002−6241−7974], Romain Rombourg, and Philippe Lalanda

Univ. Grenoble Alpes, CNRS, Grenoble INP**, LIG, 38000 Grenoble, France
sandra.castellanos@univ-grenoble-alpes.fr

Abstract. Intuitively, Automated Planning systems capable of learning from previous experiences should be able to achieve better performance. One way to build on past experiences is to augment domains with macro-operators (i.e. frequent operator sequences). In most existing works, macros are generated from chunks of adjacent operators extracted from a set of plans. Although they provide some interesting results this type of analysis may provide incomplete results. In this paper, we propose ERA, an automatic extraction method for macro-operators from a set of solution plans. Our algorithm is domain and planner independent and can find all macro-operator occurrences even if the operators are non-adjacent. Our method has proven to successfully find macro-operators of different lengths for six different benchmark domains. Also, our experiments highlighted the capital role of considering non-adjacent occurrences in the extraction of macro-operators.

Keywords: Automated Planning · Macro-operators · Learning · Data Mining.

1 Introduction

Planning systems have long been the subject of important research activities in the AI community, be it industrial or academic [8, 12]. They are actually key components of intelligent agents that need to make timely decisions in complex environments, like for instance autonomous robots, smart vehicles, or pervasive systems supporting human beings. Developing such systems however remains challenging for a number of reasons. Performance is one of the main challenges. Indeed, despite remarkable progress in recent years, many planning systems fail to meet timing requirements imposed by demanding domains.

We believe that an interesting approach to improve performance is to use past experiences. With the development of pervasive applications, for instance, large organised collections of plans are available. They can then be inspected, analysed, evaluated and even modified by human experts or automated reasoning systems if needed. Knowledge extracted in those plans repositories can be used to speed up and improve the quality of the planning process in a given domain.

** Institute of Engineering Univ. Grenoble Alpes

In some domains, especially in industry, plans are stored. So, they can be used to extract knowledge about the system. A particular type of knowledge that can be extracted are the routines, i.e. sequences of actions regularly used by the system. Routines are present in real-life applications or closely related systems.

In AI planning, macros can model system routines. A *macro* consists of a sequence of actions that occurs frequently in solution plans. Once learned, they can be re-injected directly into the planning domain. Thus, the domain benefits from the knowledge extracted from previous problem solving. The system applies a macro in the same way that a primitive action. However, macros allow jumping into the search space by building deep and promising states to reach a goal state. Learning macros from previously acquired knowledge has proven to be beneficial for improving a planner's performance [2–4].

Macros have been widely studied to speed-up planning processes [1, 3, 5, 10, 11]. These approaches consist of two main phases: extraction and selection. Extraction consists in identifying sequences of actions that could be potential candidates to augment the domain. The selection phase must find a trade-off between the benefit expected by adding macros and the additional cost induced by the branching factor increase.

Literature about macros presents various techniques to build them, ranging from simple combination of primitive actions and the use of chunks of plans to the use of genetic learning algorithms or statistical analyses based on n-grams.

In this paper, we investigate an automatic extraction of macro-operators from a set of solution plans. This approach should be domain-independent and particularly adapted to the extraction of recurrent patterns (i.e. the routines). Besides, we want to exploit the extraction of sequences of non-adjacent actions. Only a few works have explored this path [2, 3]. Interestingly, they have shown that this allows more routines to be extracted and therefore, would be more profitable for the system.

Precisely, we propose *ERA*, a pattern mining inspired algorithm to mine macro-operators directly from a set of solution plans. This algorithm allows to find all macro-operators satisfying a set frequency threshold even if its actions are non-adjacent in some or all of its occurrences. We see this work as a first step towards a macro-learning method allowing to exploit a large amount of solution plans.

The paper is structured as it follows. First, we introduce the concepts of classical planning, macro-operators and sequential pattern mining. Then, we present the ERA algorithm, describe each of its steps and analyse its complexity. We then present an evaluation of our method quantifying the impact of a gap parameter over six benchmark domains. After, we discuss our results and compare our method to other works. Finally, we provide a conclusion and some perspectives.

2 Background Theory

We are interested in plan synthesis, a particular form of planning which takes a description of the world state, all its known actions and a goal[8]. As a result, we get an organised set of actions whose execution makes it possible to solve the planning task. In this work, we address sequential planning in the STRIPS framework [6].

A state is defined as a set of predicates. A planning task is composed of a *planning domain* and a *planning problem* and the purpose of this task is to find a *plan* to solve this problem. A planning domain describes the world through a set of predicates and a set of planning operators. A planning problem describes the initial state of the world and the goal state to be attained. A planning operator is a triple o = *(name(o), pre(o), effects(o))* where its elements are defined as follows:

- $name(o)$ is in the form $name(x_1, ..., x_n)$ where $x_1, ..., x_n$ are the object variable symbols that appear in o.
- $pre(o)$ is the set of predicates involving the variables of o and that must be satisfied when o is instantiated.
- *effects(o)* is the set of predicates involving the variables of o to be applied to a state when o is instantiated.

A grounded operator is an instance of a planning operator (i.e. a *lifted* operator for which variables are instantiated). A ground operator a is applicable in a state s if and only if all predicates in the preconditions of a belong to s. A state s' is reached from s if a grounded operator can be applied. Finally, a (solution) plan π is an ordered sequence of grounded operators to reach a goal state s_g from an initial state s_i.

Macros are based on the idea of composing a sequence of primitive operators and viewing the sequence as a single operator. For our purposes, we distinguish two related but different terms: grounded macros and lifted macros. A grounded macro is related to a lifted macro as a ground operator is related to a lifted operator. We use these terms according to literature common terminology.

The problem of identifying recurrent sequences of grounded operators in AI planning is analogous to the sequential pattern mining problem. Sequential pattern mining (SPM) is a sub-field of data mining that consists in analysing data, encoded as sequences of symbols, to detect sub-sequences of symbols[7, 9]. SPM is commonly used to detect recurrent sub-sequences.

In the following, we define a set of concepts (borrowed from SPM) necessary for the understanding of this work.

A sequence database C is a set of pairs $< sid, s >$, where sid is a sequence identifier and s is a sequence.

A sequence $S_A = X_1, X_2, \ldots, X_k$, where $X_1, \ldots, X_k$ are ground operators, is a sub-sequence of another sequence $S_B = Y_1, Y_2, \ldots, Y_m$, where $Y_1, \ldots, Y_m$ are ground operators, if and only if there exists integers $1 \leq e_1 < e_2 \cdots < ek \leq m$ such that $X_1 = Y_{e_1}, X_2 = Y_{e_2}, \ldots, X_k = Y_{e_k}$.

The absolute support of a sequence S is the number of sequences S_i, in the sequence database C, where S is a sub-sequence of S_i. The relative support of a sequence S is the absolute support of S divided by the total number of sequences in C. A frequent sequence is a sequence whose relative support satisfies a given relative support threshold. Thereafter, we will call this threshold *minsup*.

Besides, notice that a frequent sequence can occur in several other sequences but not necessarily in a contiguous fashion, as the definition of sub-sequence implies. So, we define a *gap* as the number of operators allowed between two consecutive operators of a sequence and it can take values in $[0, \infty]$. A sub-sequence $S = Y_{e_1}, Y_{e_2}, \ldots, Y_{e_k}$ satisfies a gap constraint g if $\forall i \in [2, k], e_i - e_{i-1} \leq g + 1$.

3 ERA

In the following, we will present ERA, our pattern mining inspired algorithm to mine macro-operators (lifted macros) directly from a set of solution plans. ERA stands for **E**xtraction of **R**ich patterns with **A**ttribute structures.

3.1 Overview

In planning, each plan is composed of an ordered sequence of grounded operators which in turn are composed of parameters (objects). An example of plan is: $\pi = \langle$ `pick-up blockA, put-down blockA, pick-up blockB, stack blockB blockA`$\rangle$.

Mining macro-operators from a set of plans requires an approach which ensures to find the frequent sequences of operators without a loss of information about their characteristics. Then, neither an operator can be dissociated from its objects nor a sequence of operators can disregard the relationship between operators' objects. In other words, the problem is to look for the most frequent sequences of operators with a specific set of object relationships. Thus, it should be noted that for a same sequence of operators, different sets of object relationships lead to the construction of different macro-operators.

Our approach extracts (lifted) macro-operators from a set of sequences of grounded operators (See Figure 1). Besides, it can detect the occurrence of a macro-operator even if the actions composing it are not adjacent.

3.2 ERA Algorithm

ERA mines all macro-operators (regardless of their length or up to a maximum length) satisfying a frequency threshold and under a gap constraint from a set of solution plans. Additionally, for each macro-operator m, this algorithm yields the following characteristics:

- support [`integer`]: the number of plans containing at least one occurrence of m.

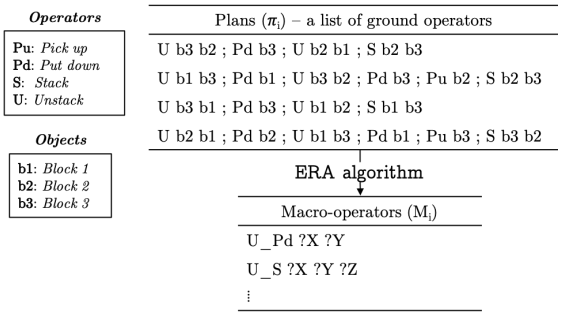


Fig. 1: A sample of extracted macro-operators from a set of plans

– sequence ids [`list`]: the plan identifiers where m appears.
– number of occurrences [`list`]: the number of occurrences of m in each plan.

For processing purposes, plans are encoded into a sequence database in two different ways. First, an action wise encoding where each distinct action is assigned a single number, yielding a dictionary A (for example {1:pick-up b, 2:stack b a, 3:pick-up c, ...}) and second an element wise encoding where every single element is assigned a single number, yielding a dictionary E (for example {1:pick-up, 2:b, 3:a, ...}.

ERA pseudo code is described in Algorithm 1. It takes as input the two previously described encodings A and E of the set of plans, a *minsup* threshold, a gap g, and the maximal length of sequences to extract.

First, it searches the set of macro-operators of length two in A by using the procedure *MINE* (line 6, described in Algorithm 2). Second, for each macro-operator of this set, if it does not satisfy the *minsup* threshold it is removed, otherwise the macro-operator and its characteristics are kept (line 11,12,13). Finally, if no macro-operator has been found for the current length, it stops. Otherwise, it increases the length by one (line 14) and it continues to loop until it reaches the maximal length (line 5). It gives as a result a set of frequent macro-operators of different lengths with their respective characteristics.

3.3 Mining procedure

The objective of the mining procedure is to obtain the set of macro-operators of length l and their characteristics from the set of solution plans. To do so, it analyses for each plan all sub-sequences of length l satisfying the gap constraint g and determines if the sub-sequence is a valid occurrence[1] of a macro-operator. If the algorithm finds a valid macro-operator occurrence, it stores this occurrence and updates the characteristics related to this macro-operator. To speed up its computation, it uses previous information obtained when mining length $l-1$.

[1] The operators of the macro can be moved contiguously in the plan without an impact on the final state or without impeding its execution.

The pseudo code of this procedure, called MINE, is described by algorithm 2. It takes as input both sequence databases A and E from the main algorithm, the gap parameter g, the length l to be evaluated and a dictionary M of all found macro-operators (of different lengths less than l) and their support. The purpose of the first loop (line 3) is to go through all combinations of ordered sub-sequences satisfying the gap constraint for each sequence in A (line 7) and for each sub-sequence, determine if it is a valid macro-operator and if it is valid in a number of sequences greater than the *minsup* parameter. To accomplish this, the following steps are performed:

- It moves on to the next sub-sequence,
  - if the sub-sequence of length $l-1$ does not satisfy the *minsup*. For that, the current sub-sequence length should be greater than two in order to be able to build its identifier[2] of length $l-1$. Then, it checks if this identifier is found in the general dictionary of pairs <macro-operator,support> (line 10).
  - if there are not enough plans left to ensure that the sub-sequence is valid in a number of sequences greater than the *minsup* (line 13).
- Otherwise,
  - it removes from the current plan δ the individual grounded operators of the sub-sequence *sp*, it builds a grounded macro-operator from *sp* (line 14) and puts it, each time, at a different position in the plan (line 20,21). It tries δ (line 22) from the calculated initial state S_i for the original plan ρ (line 6). If the result state ϵ is a superset (line 23) of the calculated final state S_g from the original plan ρ (line 6), then it stops trying positions for this sub-sequence. If it finds at least one valid position for the built grounded macro-operator, it stores the modified plan with the lifted macro-operator identifier μ as the key access (line 26) and μ is added to the list of the macro-operators found in the plan (line 27). To analyse new occurrences of an already found macro-operator, the algorithm uses the corresponding modified plan (line 16,17).

Once it has analysed all combinations of sub-sequences of length l from ρ, it moves to the second loop (line 28). The purpose here is to compute and save or update, the characteristics of each found macro-operator. Thus, it updates the set of plans where the macro-operator with identifier μ appeared (denoted K_o) by adding the index of the current plan $indexPlan$ (line 29). Also, it computes and stores the number of occurrences in the plan for the analysed macro-operator (line 30,31). Finally, if the current macro-operator appears in the plan at least once, the support value is incremented by one (line 33) or added with a value of 1 if it did not appear before (line 35).

The mining procedure gives as a result a set of frequent macro-operators of length l with its respective characteristics. They will be filtered, by using the *minsup* parameter in the main algorithm, before being added to the final set of mined macro-operators.

[2] See description in the subsection *Identifier construction*

Algorithm 1 ERA algorithm - Main algorithm

Input A sequence database A of grounded operators, a sequence database E of elements of an operator, a *minsup* parameter, a gap parameter g and a maximal length *maxLength*.

Output A dictionary M of pairs $<m, s>$, m is a macro-operator and s is its support; a dictionary K of pairs $< m, k > k$ is the id of the sequence where m appears; a dictionary J of pairs $<m, j>$, j is the number of occurrences of the macro-operator m for each sequence.

```
1: function MININGMACROS(A, E, minsup, g, maxLength)
2:     Mo, Ko, Jo ← empty dictionaries
3:     M, K, J ← empty dictionaries
4:     stop ← False, l ← 2
5:     while (l ≤ maxLength) ∧ (stop is False) do
6:         Mo, Ko, Jo ← MINE(A,E,M,l)
7:         stop ← True
8:         for each macro-operator m in Mo do
9:             if support(m) ≥ minsup then
10:                stop ← False
11:                add the key m with value s to M
12:                add the pair <m, k> to K from Ko
13:                add the pair <m, j> to J from Jo
14:        increase l by one
15:    return M,K,J
```

Identifier construction The identifier construction procedure takes as input a sub-sequence of grounded operators *sp* and a length l. Only the first l elements of *sp* are kept in this procedure. A string identifier is built as follows. First, each element e is translated by using E. Next, the first sub-element of each e is used together with a character representing the operators. After, we use another character and a incremental number for each other sub-element of e because they represent the parameters. Notice that the incremental number is reset to zero with each new identifier construction and a same parameter will have the same incremental number.

Example 1. Let us consider the length $l = 2$, the sub-sequence {pick-up b, stack b a, pick-up c} and the element encoding $E = \{1 : pick - up, 2 : b, 3 : stack, 4 : a, 5 : c\}$. We only keep the first two operators since $l = 2$ and by using E, we translate them into {1 2, 3 2 4}. We chose the character 'o' to represent operators and the character 'p' to represent parameters. We obtain the identifier {o1p0o3p0p1}.

3.4 Complexity analysis

The main task of the ERA algorithm is the analysis of a sub-sequence which in the worst case has a complexity $O(l(\rho))$ where $l(\rho)$ is the length of the plan ρ. In the case of an infinite gap, this task is repeated for each sub-sequence in a

Algorithm 2 Mining macro-operators of length l

Input The sequence database A, the sequence database E, a dictionary M of pairs $< m, s >$, m is a macro-operator and s is its support and a length l and a gap parameter g.

Output A dictionary Mo of pairs $< m, s >$, m is a macro-operator of length l and s is its support; a dictionary Ko of pairs $< m, k >$, k is the id of the sequence where m appears; and a dictionary Jo of key $< m, iP >$, iP is the index of the sequence where m appears, and value j, the number of occurrences of the macro-operator m in each sequence.

1: **function** MINE(A, E, M, g, l)
2: *Mo, Ko, Jo* ← empty dictionaries
3: **for each** plan ρ in A **do**
4: $D, macroPlan \leftarrow$ empty dictionaries
5: $idsPlan \leftarrow \{\emptyset\}$
6: $S_i, S_g \leftarrow$ calculate initial and final state from ρ
7: $P \leftarrow$ all combinations of sub-sequences of length l from ρ ▷ †
8: **for each** ordered sub-sequence sp in P satisfying g **do**
9: **if** $l > 2$ **then**
10: **if** $computeId(sp, l-1) \notin M$ **then** skip sp
11: **else** $\mu \leftarrow computeId(sp, l)$
12: **if** len(A)-$indexPlan$<$minsup$−supp(μ) **then** skip sp
13: **else**
14: add the key-value $< k, \{actions(sp)\} >$ to D ▷ $k \in Z^*_-$
15: $i \leftarrow 0$
16: **if** $\mu \in macroPlan$ **then**
17: $\delta \leftarrow macroPlan[\mu]$
18: **else** $\delta \leftarrow \rho$
19: **while** (not ok)∧($i <$ len(δ) − len(sp) + 1) **do**
20: remove sp from δ
21: insert k in δ in position i
22: $\epsilon \leftarrow execute(S_i, \delta)$
23: **if** $S_g \subset \epsilon$ **then** $ok \leftarrow True$
24: reset δ
25: **if** ok **then**
26: add $< \mu, \delta >$ to $macroPlan$
27: add μ to $idsPlan$
28: **for each** identifier μ in $idsPlans$ **do**
29: add the key μ with value $indexPlan$ to Ko
30: $nbA \leftarrow \frac{(\text{len}(\rho) - \text{len}(macroPlan[\mu]))}{(l-1)}$
31: add $< \mu, indexPlan >$ with value nbA to Jo
32: **if** $(nbA > 0) \wedge (\mu$ in $Mo)$ **then**
33: increase support of μ by one
34: **else**
35: **if** $nbA > 0$ **then** add the key μ with value nbA to Mo
36: **return** *Mo,Ko,Jo*

†: the combinations keep the order of appearance in the original plan.

plan, for each plan in the set of solution plans and for each sub-sequence length. In Equation (1), we first compute the number of sub-sequences n_{sp} of length k in a plan ρ of length $l(\rho)$.

$$n_{sp} = \binom{l(\rho)}{k} \tag{1}$$

Then, if n_{MINE} is the number of sub-sequences analysed in a execution of the MINE procedure at length k, we have:

$$n_{MINE} = \sum_{\rho_i \in C} \binom{l(\rho_i)}{k} \tag{2}$$

In the worst case, the ERA algorithm will mine up to the maximal length l_{max}. Considering that all plan lengths are equal to the maximum plan length L, i.e. $L = \max_{\rho_i \in C} l(\rho_i)$, we have N, the total number of sub-sequences analysed given by Equation (3).

$$N = \sum_{\rho_i \in C} \sum_{k=2}^{l_{max}} \binom{L}{k} \tag{3}$$

$$N = O\left(|C| \sum_{k=2}^{l_{max}} O\left(L^k\right)\right) \tag{4}$$

$$N = O\left(|C| L^{l_{max}}\right) \tag{5}$$

From Equation (4), we show in Equation (5) the number of sub-sequences to be analysed. Then, in the worst case with an infinite gap, the complexity is $O\left(|C| L^{l_{max}+1}\right)$, i.e. polynomial in L and linear in the size of the solution plan set C.

In the case of a finite gap g, the analysis of a sub-sequence is repeated for each sub-sequence in a plan satisfying the gap constraint and for each plan and sub-sequence length.

We recall that a sub-sequence $S_A = Y_{e_1}, \ldots, Y_{e_k}$ of a sequence $S_B = Y_1, \ldots, Y_L$ satisfies a gap constraint g if $\forall i \in [2, k], e_i - e_{i-1} \leq g + 1$. Notice that the sub-sequence S_A can be uniquely identified by the list of indices $\{e_1, \ldots, e_k\}$ of the elements in S_B, but it can also be equivalently represented as a list $\{e_1, e_2 - e_1, \ldots, e_i - e_{i-1}, \ldots, e_k - e_{k-1}\}$. Under this last representation, one can easily see if a given sub-sequence satisfies the gap constraint. Then the maximal number of sub-sequences N_{gap} with first element Y_{e_1} and length k, can be computed as one plus the biggest number in base $g + 1$ with $k - 1$ digits (see Equation (6)).

$$N_{gap} = 1 + g \sum_{i=0}^{k-2} (g+1)^i = (g+1)^{k-1} \tag{6}$$

In the worst case, the ERA algorithm will mine up to the maximal length l_{max}. Similar to the infinite gap case, we consider that all plan lengths are equal to the maximum plan length L.

In Equation (7), we show that an upper bound on the number N of sub-sequences to be analysed can be computed by considering that for each possible first element $Y_i, i \in [1, L-k]$, k the sub-sequence length, the maximal number of sub-sequences N_{gap} will be analysed.

$$N < \sum_{\rho_i \in C} \sum_{k=2}^{l_{max}} (L-k) N_{gap} \tag{7}$$

$$N = O\left(|C|L(g+1)^{l_{max}-1}\right) = O\left(|C|L\right) \tag{8}$$

We show in Equation (8) an upper bound on the number of sub-sequences to be analysed. Then, in the worst case with a finite gap, the complexity is $O\left(|C|L^2\right)$, i.e. quadratic in L and linear in the size of the solution plan set C.

4 Evaluation

In the following, we aim to show the impact of the gap when extracting macro-operators with ERA. For this purpose, we used six benchmarks domains[3] taken from the International Planning competition (IPC-2011): `barman`, `blocksworld`, `depots`, `gripper`, `rover` and `satellite`.

4.1 Experimental setup

To the best of our knowledge, no open plan database is available. Thus, for each benchmark domain, we generated a set of 50 distinct problem instances using the generators[4] from the International Planning Competition. Then, we solved the problems by using the FastDownward planner with an A* search strategy and the FF heuristic as the evaluator for the h-value. As a result, we obtained a set of 50 sub-optimal solution plans, for each benchmark domain, with a length between 20 and 50 ground operators.

To evaluate the gap impact, for each benchmark, we used ERA with a fixed frequency threshold *minSup* of 0.85, an infinite maximum length and a finite gap varying between 0 and 15 plus the infinite gap case. Experiments were performed on an Intel Core i7-4710MQ quad-core CPU clocked at 2.5GHz and with 8GB of RAM.

[3] Further details can be found at http://www.plg.inf.uc3m.es/ipc2011-learning/Domains.html

[4] https://bitbucket.org/planning-tools/pddl-generators

4.2 Results of the ERA algorithm

We present in Figure 2, for each domain, the number of extracted macro-operators as a function of the gap. We observe that the number of extracted macro-operators increase consistently with the gap for all domains. In most domains, a gap of at least 9 was necessary to extract all macro-operators, i.e. the number of extracted macro-operators with an infinite gap. Among the six domains, Barman and Satellite were exceptions. For Barman, all macro-operators were found for all gaps greater or equal to 1. And for Satellite, not all macro-operators were found even with a gap of 15.

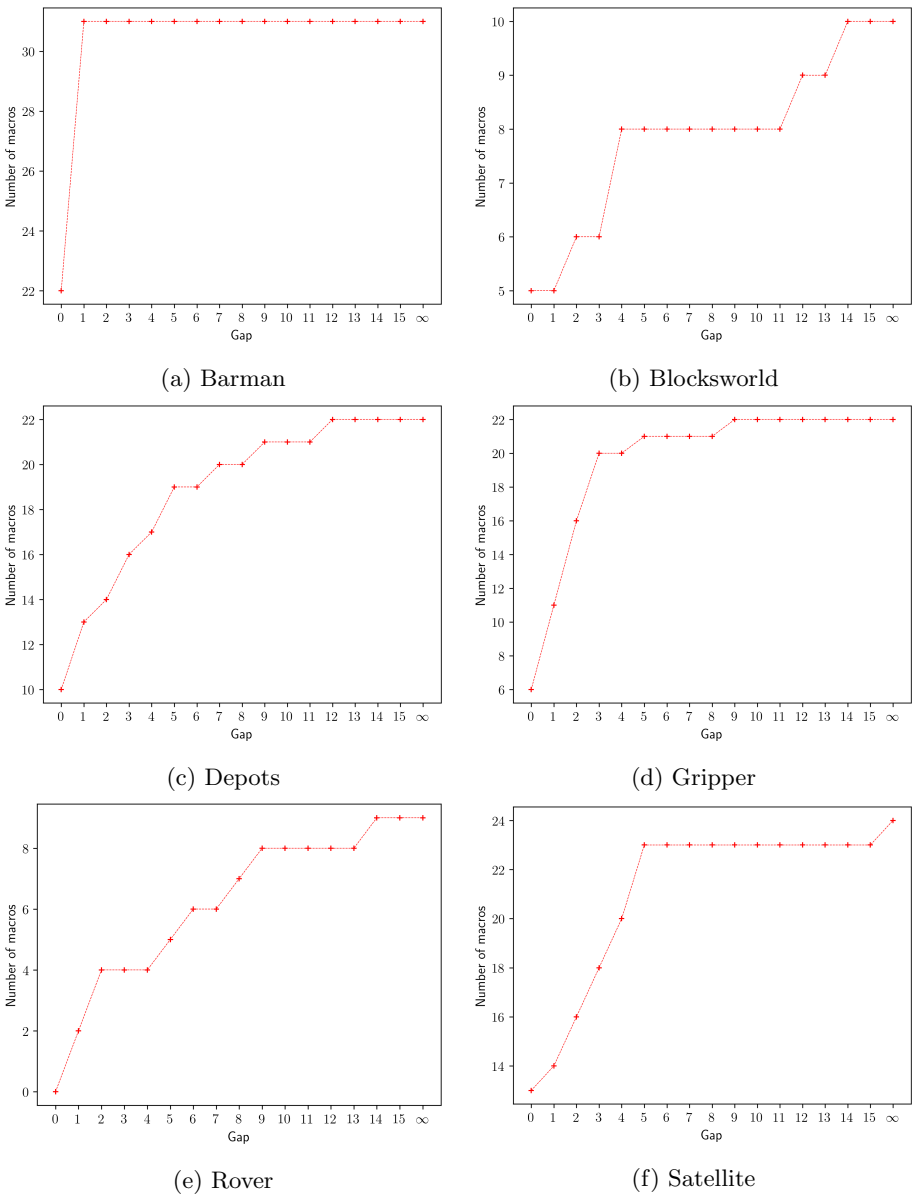
We show in Figure 3, for each domain, the length of the longest macro-operator found as a function of the gap. We observe that our method is able to recover long macros (e.g. a size of 6 for the longest macro in Barman and a size of 5 for the longest macro in Satellite) since we did not have any restriction on macro length. Also, similar to the number of extracted macros, higher gap values allow for longer macros.

Finally, we have in Figure 4 the time used by ERA to extract macro-operators for each domain, as a function of the gap. We observe that the time increases with the gap. Also, we find two different behaviours: either polynomial, for barman, blocksworld, depots and satellite or almost linear, for gripper and rover. Notice that this behaviour is unrelated to the complexity shown in Section 3.4, since the complexity was given in terms of plan length and plan set size.

5 Discussion

The ERA algorithm checks every sub-sequence satisfying the gap constraint. Thus, by construction, it is guaranteed to find all macro-operators satisfying the frequency threshold. Also, macro-operator occurrences (even non-adjacent) registered by ERA are valid. Indeed, to be registered, an occurrence must be composed of grounded operators that can be moved contiguously in the plan without an impact on the final state or without impeding its execution.

Intuitively, one could think that high gap values would not impact significantly the mining results since two distant operators should be less related than two close operators. Indeed, that would be true if the planning system was focused on accomplishing sequentially disconnected groups of sub-goals, e.g. preparing and serving a specific cocktail in barman. However, every plan (even optimal) can be reordered in hundreds or even thousands of different ways from which only a few correspond to a configuration where all disconnected goals are accomplished sequentially. Our results clearly show that mining with a non-zero gap allows to extract much more information. Precisely, in five domains out of six, to extract as much information as an infinite gap, a gap value ranging from 9 to 14 was needed. To the best of our knowledge, our work is the only one focused on the extraction of macro-operators from a set of plans by considering adjacent and non-adjacent operators. Other works also included an approach to handle non-adjacent operators to build macro-operators [2, 3]. Botea et al. [2]

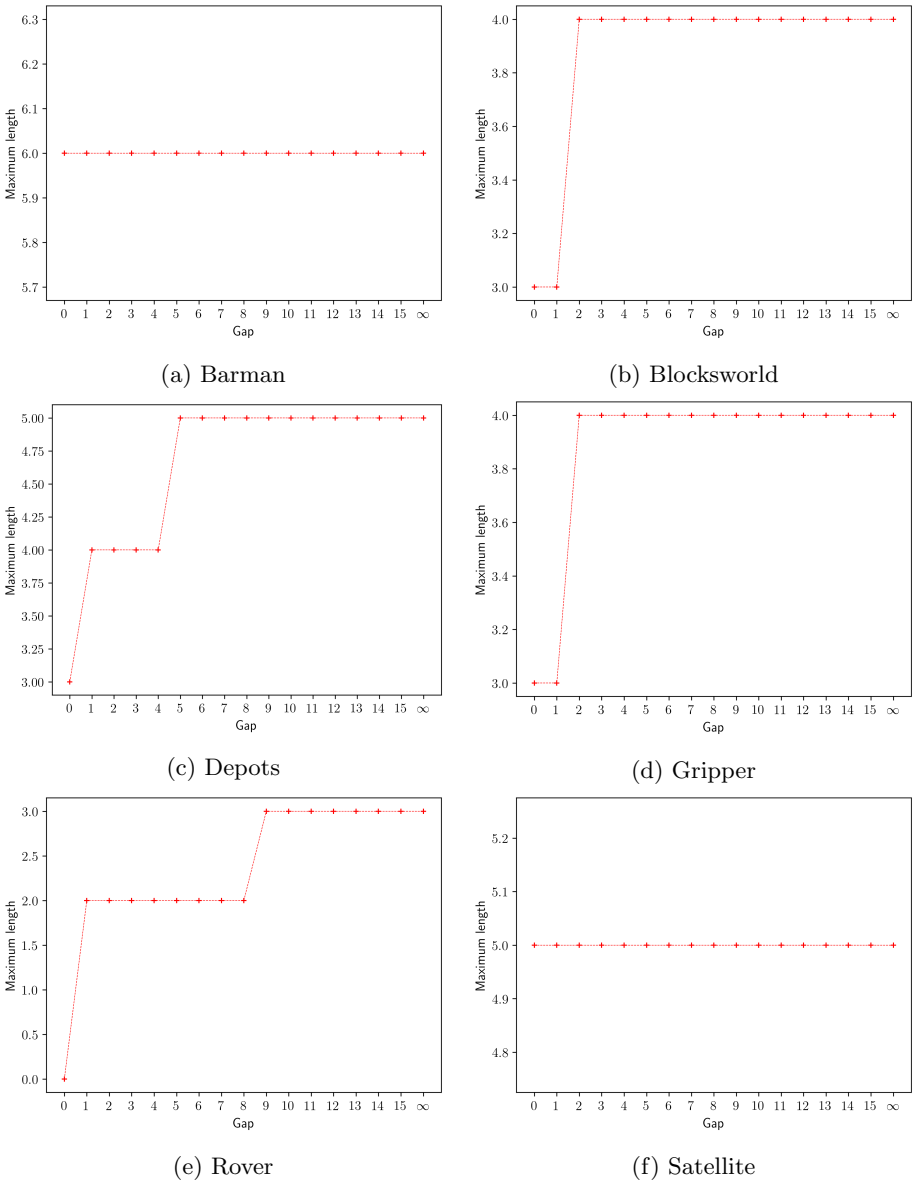


(a) Barman (b) Blocksworld

(c) Depots (d) Gripper

(e) Rover (f) Satellite

Fig. 2: Number of macro-operators extracted for each benchmark domain.

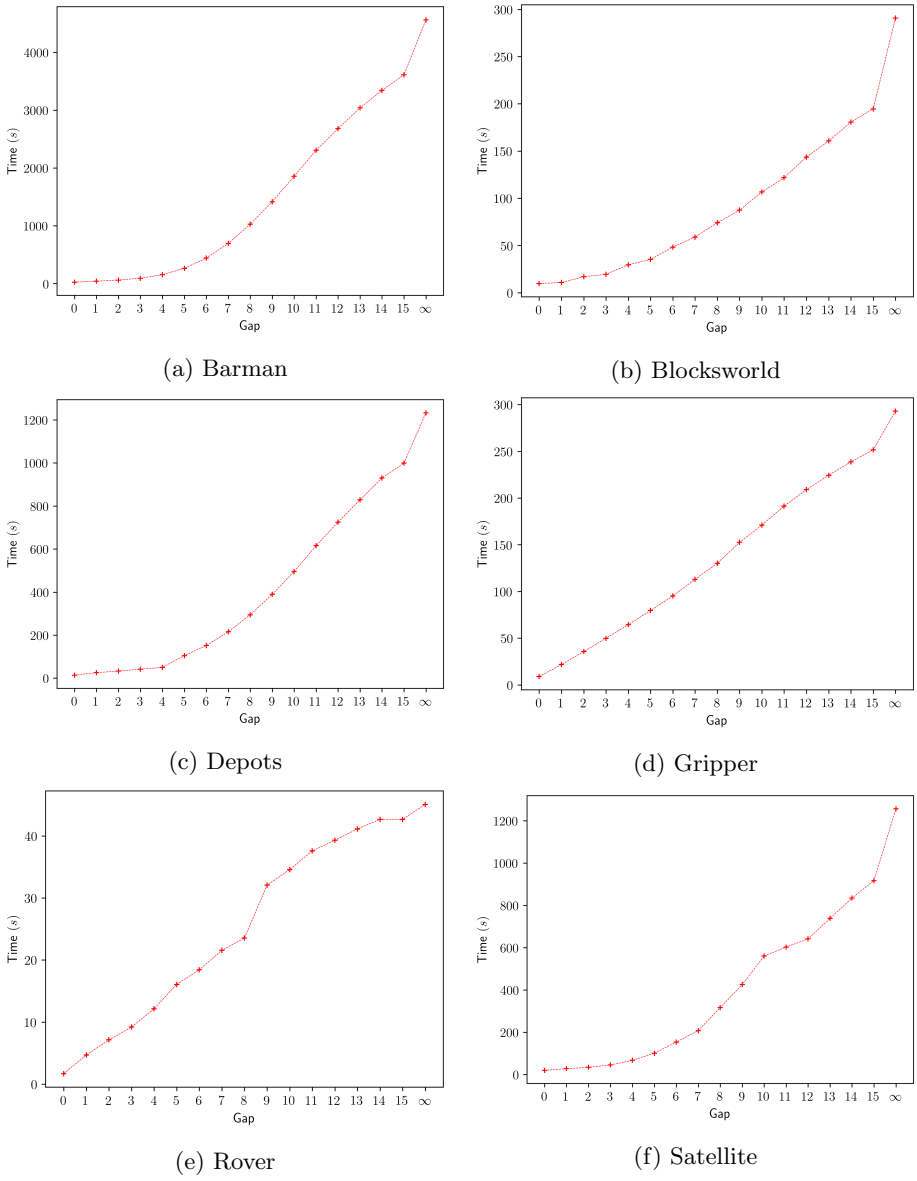


(a) Barman (b) Blocksworld

(c) Depots (d) Gripper

(e) Rover (f) Satellite

Fig. 3: Maximum length of macro-operators extracted for each benchmark domain.



(a) Barman

(b) Blocksworld

(c) Depots

(d) Gripper

(e) Rover

(f) Satellite

Fig. 4: Time to extract macro-operators using ERA for each benchmark domain.

extract macros from solution graphs of training problems. They enumerate and select sub-graphs from the solution graphs and build one macro for each selected sub-graph. They introduce a k parameter as the maximal number of operators that can be skipped between the first and the last element of the sub-graph (we can roughly see a sub-graph as a sub-sequence). Their work differs from ours because their handling of the gap does not allow an exhaustive search and can then miss many macro-operator occurrences. Also, they set a hard limit on the upper bound of the maximal length of the extracted macro-operators. Chrpa et al.[3] extract macros by iteratively combining operators (even non-adjacent) in plans that share some parameters while keeping the plan valid. However, their technique is not guaranteed to find all macro-operator occurrences.

The macro-operators extracted by ERA can be composed either by repeated operator sequences, e.g. in Depots a macro-operator of length 4 is composed of a repeated sequence of `lift` and `load` operators, or by different operators, e.g. in Barman a macro-operator of length 5 is composed of five different operators.

6 Conclusion

We provided an algorithm to analyse sequences of operators (even non-adjacent) from past solution plans. Precisely, ERA algorithm extracts all macro-operators, satisfying a frequency threshold parameter and under a gap constraint, from existing plans. Our algorithm has a quadratic complexity with a finite gap and a polynomial complexity with an infinite gap. Below, we highlight the advantages of our work:

- *Planner independent*, our algorithm can be used on existing plans from a given domain regardless any domain characteristic or the planner used to obtain those plans.
- *Domain-independent*, our method does not need *a priori* knowledge on the domain.
- *Non-adjacent operators*, we can identify macros from a sequence of adjacent and non-adjacent operators.
- *Completeness*, with an infinite gap our algorithm is guaranteed to find all occurrences of any macro-operator.

Our method has proven to successfully find macro-operators of different lengths for six different benchmark domains. Also, our experiments highlighted the capital role that plays the gap in the extraction of macro-operators.

We see this work as a first step towards a macro-learning method allowing to exploit solution plans. As a future work, we would like to analyse the link between the gap and (1) the correctness of the support and (2) the number of occurrences. More precisely, we would like to investigate if a finite gap could ensure in most cases that all occurrences were found. Another interesting perspective would be to analyse the effect of the gap with plans obtained through different search strategies. Finally, we would like to include ERA in a full macro-learning method including a macro selection phase.

References

1. Botea, A., Enzenberger, M., Müller, M., Schaeffer, J.: Macro-FF: Improving AI planning with automatically learned macro-operators. Journal of Artificial Intelligence Research **24**, 581–621 (2005)
2. Botea, A., Müller, M., Schaeffer, J.: Learning partial-order macros from solutions. In: Proceedings of the Fifteenth International Conference on International Conference on Automated Planning and Scheduling. pp. 231–240. AAAI Press (2005)
3. Chrpa, L., Vallati, M., McCluskey, T.L.: MUM: A technique for maximising the utility of macro-operators by constrained generation and use. In: Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (2014)
4. Coles, A., Smith, A.: Marvin: A heuristic search planner with online macro-action learning. Journal of Artificial Intelligence Research **28**, 119–156 (2007)
5. Dulac, A., Pellier, D., Fiorino, H., Janiszek, D.: Learning useful macro-actions for planning with n-grams. In: 2013 IEEE 25th International Conference on Tools with Artificial Intelligence. pp. 803–810 (11 2013). https://doi.org/10.1109/ICTAI.2013.123
6. Fikes, R., Nilsson, N.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence **3-4**(2), 189–208 (1971)
7. Fournier-Viger, P., Lin, J.C.W., Kiran, R.U., Koh, Y.S., Thomas, R.: A survey of sequential pattern mining. Data Science and Pattern Recognition **1**(1), 54–77 (2017)
8. Ghallab, M., Nau, D., Traverso, P.: Automated planning: theory and practice (2004)
9. Han, J., Pei, J., Kamber, M.: Data mining: concepts and techniques. Elsevier (2011)
10. Hofmann, T., Niemueller, T., Lakemeyer, G.: Initial results on generating macro actions from a plan database for planning on autonomous mobile robots. In: Twenty-Seventh International Conference on Automated Planning and Scheduling (2017)
11. Newton, M.A.H., Levine, J.: Implicit learning of macro-actions for planning. In: Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010) (2010)
12. Vukovic, M., Gerard, S., Hull, R., Katz, M., Shwartz, L., Sohrabi, S., Muise, C., Rofrano, J., Kalia, A., Hwang, J., et al.: Towards automated planning for enterprise services: Opportunities and challenges. In: International Conference on Service-Oriented Computing. pp. 64–68. Springer (2019)