HAL
open science

Learning Useful Macro-actions for Planning with N-Grams

Adrien Dulac, Damien Pellier, Humbert Fiorino, David Janiszek

To cite this version:

Adrien Dulac, Damien Pellier, Humbert Fiorino, David Janiszek. Learning Useful Macro-actions for Planning with N-Grams. IEEE International Conference on Tools with Artificial Intelligence (ICTAI), Nov 2013, Herndon, United States. pp.803-810, 10.1109/ICTAI.2013.123 . hal-00952270

HAL Id: hal-00952270

https://inria.hal.science/hal-00952270

Submitted on 9 Apr 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Learning Useful Macro-actions for Planning with N-Grams

Adrien Dulac*, Damien Pellier†, Humbert Fiorino* and David Janiszek†
* *Univ. Grenoble Alpes, Laboratoire d'Informatique de Grenoble, F-38041 Grenoble, France*
Email: {adrien.dulac, damien.pellier, humbert.fiorino}@imag.fr
† *Univ. Paris Descartes, France*
Email: {david.janiszek}@parisdescartes.fr

***Abstract*—Automated planning has achieved significant breakthroughs in recent years. Nonetheless, attempts to improve search algorithm efficiency remain the primary focus of most research. However, it is also possible to build on previous searches and learn from previously found solutions. Our approach consists in learning macro-actions and adding them into the planner's domain. A *macro-action* is an action sequence selected for application at search time and applied as a single indivisible action. Carefully chosen macros can drastically improve the planning performances by reducing the search space depth. However, macros also increase the branching factor. Therefore, the use of macros entails a *utility problem*: a trade-off has to be addressed between the benefit of adding macros to speed up the goal search and the overhead caused by increasing the branching factor in the search space. In this paper, we propose an online domain and planner-independent approach to learn 'useful' macros, i.e. macros that address the utility problem. These useful macros are obtained by statistical and heuristic filtering of a domain specific macro library. The library is created from the most frequent action sequences derived from an n-gram analysis on successful plans previously computed by the planner. The relevance of this approach is proven by experiments on International Planning Competition domains.**

***Keywords*- automated planning; macro-actions; n-gram analysis; supervised learning.**

I. Introduction

Automated planning is a key component for devising autonomous and intelligent systems able to carry out complex tasks in real environments. Many highly efficient algorithms have been proposed in recent years to speed up plan generation. However, few learning algorithms have been developed which allow the planner to build on its experience. Intuitively, a system capable of using its previous experience should be able to achieve better performance. Consider a vacuum cleaner robot cleaning a house day after day. Each day, the task to be achieved is slightly different, e.g., a usually closed door is opened, an armchair has been moved, etc., but the routine remains the same: the house must be cleaned. How can such a robot discover and exploit information from its daily routines?

Our approach to building on past experiences is to learn macro-actions and add them into the planner's domain. A *macro-action*, or macro in short, is an action sequence selected for application at search time and applied as a single indivisible action. Macros extend the successors of each state with promising states in terms of progression toward the goal to be achieved. Carefully chosen macros can drastically improve planning performance by reducing the search space depth. However, macros also increase the branching factor. Therefore, the use of macros entails a *utility problem*: a trade-off has to be addressed between the benefit of adding macros to speed up the goal search and the overhead caused by increasing the branching factor in the search space.

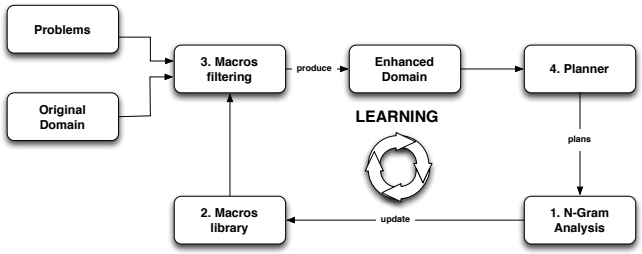

Figure 1. **Learning and planning loop**

In this paper, we propose an online domain and planner independent approach to learn "useful" macros, i.e., macros that address the utility problem. Our approach works with arbitrary planners or domains and does not exploit any structural knowledge about planners or domains. Given a planner, a domain and a set of problems, our approach extracts information from successful plans based an online n-gram analysis to generate macros. Performance improvement is obtained by adding macros in the planning domain. N-gram analysis technique is used in statistical natural language processing tasks such as automatic speech recognition or machine translation. In these tasks, n-grams make it possible to compute the occurrence probability of words given an observed history (i.e. the previous words).

The main contribution of this paper consists of a domain and planner-independent method for learning useful macros by n-gram analysis of action sequences extracted from previously computed solutions, i.e. plans. The successive steps of the approach (see figure 1) are:

1) *Analysis*: Extract statistical information from successful plans based on on-line n-gram analysis.
2) *Generation*: Build a macro library based on previous

extracted information and a generalisation and specialisation process.

3) *Filtering*: Select the most useful macros to add into the planning domain with respect to the utility problem for a specified planning problem based on statistical information and a heuristic pruning technique.
4) *Planning*: Use the selected macros to improve planning process.

The remainder of the paper is organised as follows: section II discusses related work; section III introduces the general framework and our concepts; section IV details the steps of our approach, and section V presents our experimental results and demonstrates the relevance of learning useful macros to improve planning performances.

II. Related Work

Since the pioneering works [1], [2], [3], [4], [5], most techniques use an off-line learning approach to generate and filter macros before using them in searches or explicitly exploiting properties of the planners and the domains.

More recently, Macro-FF [6] extracts macros in two ways, from solutions computed on training problems, and by the identification of statically connected abstract components. Finally, an off-line filtering technique based on a set of heuristic rules is used to prune the list of macros. Macros are then filtered dynamically with respect to their performances in solving training problems. Only the most effective ones are kept for future searches.

Newton *et al.* [7] propose an off-line technique for learning macros genetically from plans for arbitrarily chosen planners and domains. The macros are randomly chosen from solution plans of simple problems to seed the population. To explore only the macros occurring in the solution plans, genetic operators are restricted to existing macros of adjacent actions in the plan, shrinking a macro by deletion of an action from either end, splitting a macro, and lifting a macro from plans. The ranking method is based on the weighted average of time differences in solving more difficult problems with the original domain augmented with macros. The main limitation of this learning approach is the time needed to converge on the most likely macro-actions.

Botea *et al.* [8] introduce a new off-line method for learning useful macros from a set of training problems. For each problem, a structure called a *solution graph* is built. The nodes of the solution graph are actions of the solution plan and the edges represent causal links between actions. Each sub-graph of the solution graph corresponding to a macro is elicited. Then, each macro is ranked with respect to static rules and filtered at runtime with the RPG (Relaxed Planning Graph) heuristic used in FF [9]. Finally, [10] extend this approach by composing sequences of macro-actions called iterative macros. Iterative macros exhibit both the potential advantages and limitations of classical macros on a much larger scale.

Unlike other approaches, Marvin [11] is a planner based on the EHC (Enforced-Hill-Climbing) search algorithm used by FF [9]. It is also based on an on-line macro learning process. Marvin identifies plateaus in the search space in order to learn plateau-escaping macros and to use them to escape similar plateaus during the search. Macros are learned on-line and the planner must decide which ones are likely to be applicable during the search. However, these macros are not available to solve other problems in the domain. In order to overcome this limitation, a technique built on the Marvin on-line macro inference was proposed by [12]; it consists in building libraries of potentially useful macros for future problems and introducing library management strategies.

In brief, most existing approaches are off-line. They learn macros on a small set of training problems and evaluate them on another set before using them. The learning phase is considered as preprocessing and not as a way to increase and model the planner's experience through repeated problem-solving phases. Only the Marvin planner is based on an on-line learning process. However, its learning approach is planner-dependent, i.e., it exploits the properties of the EHC search algorithm, and cannot be applied to other planners. Therefore, devising a domain and planner-independent macro learning method based on an on-line learning process represents an important contribution to the existing literature on planning.

III. General Framework

This section presents the concepts of our approach.

A. Operators, Actions and Plans

We address sequential planning in the PDDL framework [13]. An *operator* o is a tuple $o = (pre(o), add(o), del(o))$ where $pre(o)$ are the action's *preconditions*, $add(o)$ and $del(o)$ are respectively its positive and negative *effects*. Parameters bound to a constant are *instantiated*, and *free* otherwise. For instance, consider the operator `move` below:

```
(:action move
  :parameters (?thing ?from ?to)
  :precondition (and (road ?from ?to)
     (at ?thing ?from) (mobile ?thing)
     (not (= ?from ?to)))
  :effect (and (at ?thing ?to)
     (not (at ?thing ?from))))
```

An action is a *ground* operator, i.e., an operator with all the parameters instantiated. A *state* s is a set of logical propositions. A state s' is reached from s by applying an action a according to the transition function

$$\gamma(s, a) = (s - del(a)) \cup add(a).$$

The application of a sequence of actions $\pi = \langle a_1, \ldots, a_n \rangle$ to a state s is recursively defined as $\gamma(s, \langle a_1, \ldots, a_n \rangle) = \gamma(\gamma(s, a_1), \langle a_2, \ldots, a_n \rangle)$.

A planning problem is a tuple (O, s_0, g, C) where O is a set of operators, s_0 is the initial state of the problem, g

Algorithm 1: merge(a_1, a_2)

Input: Two actions a_1 and a_2 to merge
Output: The macro-action m
begin
  $m \leftarrow a_1$;
  foreach *precondition* $p \in \mathrm{Pre}(a_2)$ **do**
    if $p \notin \mathrm{Add}(m) \cup \mathrm{Pre}(m)$ **then**
      $\mathrm{Pre}(m) \leftarrow \mathrm{Pre}(m) \cup \{p\}$;
  foreach *delete effects* $d \in \mathrm{Del}(a_2)$ **do**
    if $d \in \mathrm{Add}(m)$ **then**
      $\mathrm{Add}(m) \leftarrow \mathrm{Add}(m) \backslash \{d\}$;
    else
      $\mathrm{Del}(m) \leftarrow \mathrm{Del}(m) \cup \{d\}$;
  foreach *add effects* $a \in \mathrm{Add}(a_2)$ **do**
    if $a \in \mathrm{Del}(m)$ **then**
      $\mathrm{Del}(m) \leftarrow \mathrm{Del}(m) \backslash \{a\}$;
    $\mathrm{Add}(m) \leftarrow \mathrm{Add}(m) \cup \{a\}$;
  return m;

is the goal to reach (s_0 and g are sets of propositions), and C is a set of constants or objects of the problem possibly classified according to their type. A *plan* is a sequence of actions $\pi = \langle a_1, \ldots, a_n \rangle$ ($a_i \in A$) such that $g \subseteq \gamma(s_0, \pi)$: the planner takes as input a planning problem and outputs a plan if it succeeds to find a solution, otherwise it returns *failure*.

B. N-grams and Macro-operators

N-grams are widely used in statistical natural language processing tasks such as automatic speech recognition or machine translation. Their use is based on a markovian assumption about the underlying process: the occurrence of an event depends on a limited size history. In the context of language processing, n-grams are sub-sequences of n words extracted from large corpora. They are used to compute the probability of occurrence of the n-th word considering its $n-1$ predecessors meaning that they were emitted by a Markov source of order $n-1$. These probabilities can be used to identify the most likely sentences.

In our approach, a set of plans $P = \{\pi_1, ..., \pi_p\}$ computed by the planner to solve past planning problems is the learning corpus for the n-gram analysis, and a n-gram ng (e.g. 2-gram, 3-gram etc.) is any action sequence of length n in the plans of P: $ng = \langle a_1, \ldots, a_n \rangle$. Two actions are merged into a macro-action as defined in Algorithm 1. This merging is extended to the n-grams appearing in plans as defined in Algorithm 2 in order to compute macro-operators. Generating a macro-operator from a n-gram is a *macro generalization*, and $(n-1)$ is the *macro-operator order*. An important property of this algorithm is that two different n-grams can generate the same macro-operator with respect to parameter renaming. This property is the core of the n-gram analysis and the utility criteria introduced in section III-C.

Algorithm 2: MacroGeneralization(ng)

Input: A n-gram $\mathrm{ng} = \langle a_1, \ldots, a_n \rangle$
Output: The generalized macro-operator o
begin
  $Pre(o) \leftarrow \emptyset$, $Add(o) \leftarrow \emptyset$, $Del(o) \leftarrow \emptyset$;
  foreach a_i *in ng* **do**
    $o \leftarrow \mathrm{merge}(o, a_i)$;
  foreach $c \in \mathrm{Pre}(o), \mathrm{Add}(o), \mathrm{Del}(o)$, **do**
    Replace each c by a parameter p;
  return o

C. Utility criteria

Let N be the set of macro-operators m of order n: 2^N is the powerset of N, and card(N) is the number of elements in N.

- *Observation likelihood:* For all $m \in N$, $w(m)$ is the number of occurrences in P of the n-grams used to generate m. The observation likelihood $\mathbb{P}(m)$ of m on the corpus P is defined as:

$$\mathbb{P}(m) = \sum_{n\text{-gram}} \mathbb{P}(a_n \mid a_{n-1}, .., a_0) = \frac{w(m)}{\sum_{m' \in N} w(m')}$$

- *Generalization ratio:* $G(n) = NG/\mathrm{card}(N) \geq 1$ where NG is the number of distinct n-grams in P,
- *Coverage indicator:* given a macro order n, $C(n, p\%)$ measures the minimum number of macro-operators necessary to cover $p\%$ of the n-gram occurrences in P:

$$S(n, p\%) = \{s \in 2^N | \sum_{m \in s} \mathbb{P}(m) \geq p\%\}$$

$$C(n, p\%) = \mathrm{argmin}_{s \in S(n,p\%)} \mathrm{card}(s)$$

- *Good order estimation:* in our approach, the useful macro-operators added into the planning domain have all the same order, the *good order* n_{good}. A good order is defined as a trade-off between the length and the concentration of highly probable n-grams in a n-gram distribution. This concentration is evaluated by a comparison of the generalization ratio G, representing the mean of n-grams generalized into a macro-operator, and the coverage indicator, both homogeneous to a quantity of macros:

$$n_{\text{good}} = \begin{cases} n \text{ such that } G(n) = C(n, p\%) \\ \mathrm{argmax}_n(G(n)) & \text{otherwise.} \end{cases}$$

- *Computing resources:* Macro usefulness is also determined by the available computing resources. In

this respect, the memory needed to instantiate macro-operators into actions is certainly the most critical computing resource: the branching factor as well as the planning search space are strongly limited by the available memory. As a consequence, we estimate a theoretical upper bound of memory for each macro order n as follows: for each macro-operator $m \in N$ having k free parameters, the number of actions[1] is $card(C)^k$ (C is the set of constants). Let $\Delta(n)$ be the number of actions generated by all the macros m of order n: $\log_2(\Delta(n))$ is an estimation of the bits necessary to encode all the actions. Assuming that we want to use all the available memory M, we define the memory order criterion n_{mem} as:

$$M = \log_2(\Delta(n_{\text{mem}}))$$

- *Best order:* the best order is a tradeoff between n_{good} and n_{mem} defined as:

$$n^* = \min(n_{\text{good}}, n_{\text{mem}}),$$

and the sets s^* of macro-operators eventually inserted into the planning domain are as follows: let N^* be the set of macros of order n^*. Then, with s and $s' \in S(n^*, p\%)$,

$$s^* = \{s \mid \forall s',\ \text{card}(s) < \text{card}(s')\}.$$

IV. Macro Generation and Filtering

The macro-operators are provided by n-gram (i.e. action sequence) generalization. However, this generalization process leads to a *utility problem*: a trade-off has to be found between generating useful macro-operators that speed up planner progression toward goals, and deteriorating planner performances caused by an increase of the branching factor. In this section we present in details the three steps of our approach: (1) the analysis step which consists in extracting statistical information from successful plans based on n-gram analysis, (2) the generation step which consists in generating the macro-operators library and (3) the filtering step which consists in filtering the most useful macros with respect to the utility problem for a specified problem.

A. N-gram Analysis

Our assumption is that macro-operators learned from previously solved problems can improve the resolution time for new ones. Therefore, each successful plan computed by the planner is stored for a future n-gram analysis which is possible only when there is a sufficient collection of plans P; when the n-gram distributions extracted from this collection become stable.

We have observed that the number of distinct n-grams increases along with the number of plans and appears to follow Heap's law [14]. Whatever the n-gram order, its evolution depends on the number of operators, the number of parameters used to instantiate them and the number of plans. Consequently, our observations show that the stability of the number of distinct n-grams is not appropriate to stop plan collection.

However, we have observed that the most frequent n-grams becames stable. So we use this criterion to determine whether the collection of plans is sufficient to compute an n-gram analysis: we stop collecting plans when changes become rare in the list of the most frequent n-grams.

Structural knowledge extracted from previously solved problems and the n-gram analysis is twofold:

- an estimation F of the planning domain branching factor calculated as the ratio of the number of actions to the number of facts (the instantiated predicates in the training problems),
- for each n-gram order, a distribution of n-grams with their occurrence count in P.

B. Macro Library Generation

The generation of the macro-operators library is composed of two steps to deal with the utility problem: the generation of macro-operators based on a generalization process that uses the previously extracted statistical information and a partial specialization process to reduce the number of potentially induced actions.

Macro Generalization

Building macro-operators from n-grams is the core of our learning and generalization process. Macro-operator synthesis is based on a generalization process of n-grams (see Algorithm 2). For each action in a n-gram, preconditions and postconditions are merged (see Algorithm 1).

As the branching factor grows exponentially with the number of macro parameters, the generalization process leads to a significant performance overhead. An estimation of this overhead is obtained with the native domain planning branching factor F. Hence, according to an arbitrary limit for this overhead, a macro specialization is performed.

Macro Specialization

Some of the macro parameters are instantiated with constants. Macro specialization is a trade-off between the reduction of variables on the one hand (and hence the reduction of generated macro-actions at planning time), and on the other hand, the usefulness of macro-operators in terms of the probability of being used at planning time:

- Firstly, we calculate for each macro, and for each macro parameter, the most probable constant mapped to this parameter in the set of n-grams used to generate the macro,
- Then, an arbitrary number of parameters are instantiated with their most probable constant. The instantiated

[1]Practically, we use typed parameters, which allow us to refine this number.

parameters are either those having the k most probable constants or those with a constant probability above an arbitrary threshold (see Algorithm 3).

For instance, consider the `move-robot` macro below which is a specialization of `move` (see section III-A). This macro allows to specifically move $robot$ from a location to another and parameter $?thing$ is substituted by constant $robot$:

```
(:action move-robot
  :parameters (?from ?to)
  :precondition (and (road ?from ?to)
     (at robot ?from) (mobile robot)
     (not (= ?from ?to)))
  :effect (and (at robot ?to)
     (not (at robot ?from))))
```

Algorithm 3: MacroSpecialization(m)

Input: A macro m
Output: The specialized macro operator m
begin
  eList $\leftarrow$ empty list;
  foreach *parameter* p *in* m **do**
    $e \leftarrow (p, c)$ s.t. $\forall c' \in C$: $\mathbb{P}(c \mid p) > \mathbb{P}(c' \mid p)$;
    eList $\leftarrow$ concat(eList, e);
  eList $\leftarrow$ Sort (p, c) in eList wrt. $\mathbb{P}(c \mid p)$;
  foreach $(p, c) \in$ *eList* **do**
    if $\mathbb{P}(c \mid p) \geq p_{inf}$ *or* $k > free_{sup}$ **then**
      Substitute p by c in m;
      $k \leftarrow k - 1$;
  return m

C. Macro Filtering

The generalization and partial specialization processes provide a macro library too large to be used at planning time, and they do not take into account the specificity of the problem to solve. Therefore, we propose two filtering methods to select the most useful macros from the macro library. The first one estimates the best macro order of the macros to add into the planning domain; the second one uses a heuristic filter to further refine macro selection with respect to the estimated macro best order.

Statistical Filtering

Our policy for selecting the best macros is based on the idea that useful macros are (i) high order macros allowing significant progressions in the search space at planning time, and (ii) generated with highly probable n-grams in the learning database. These ideas are summarized in the best order n^* (see section III-C): the only macros inserted into the planning domain are n^* order macros.

Heuristic Filtering

Before expanding the search space, we use a heuristic technique to decide which n^* order macros are useful with respect to planning problem (initial state and goal) to solve. Once again, the objective is to reduce as much as possible the branching factor. The technique proposed here comes from the heuristic approach of the FF planner [9], and is based on a structure called *relaxed planning graph*. A relaxed planning graph is a structure that encodes in polynomial time the search space of a relaxed version of a problem by ignoring the delete effects of its actions. The plan that can be extracted (in polynomial time) from this graph is very close to the solution plan of the non-relaxed problem and can be used as an informative estimator. Our approach exploits the properties of this plan. Given an initial state and the goal to be reached, we begin by building the relaxed planning graph to reach the goal with all the actions of the planning problem and the macro-actions from the library. We then extract a solution plan from the graph. Macro-actions identified in the plan are considered to be useful macro-actions and are added to the domain. This heuristic macro filtering technique drastically reduces the overhead due to the addition of non-useful macro-actions by selecting the most useful ones according to problem to solve.

V. Experiments

The objectives of our experiments are: (i) to evaluate the performance of our learning macro approach in terms of plan length and search time and (ii) to show the impact of the macro order on the search time.

A. Experimental framework

Our experiments are based on a large set of benchmarks from the learning track of the International Planning Competition[2] (IPC): Blockworld, Ferry, Depot, Satellite, Mprime Gripper, Grid, Parking and Barman. For each benchmark, 10 000 problems were randomly generated with the problem generators developed for the competition. The distribution of the generated problems is shown in Table I. The parameters used are taken from the IPC Learning challenge. The 1 000 simplest problems were retained for learning macros (experiments show that a set of this size is a good trade-off to obtain stable n-gram distributions), and 30 of the most difficult problems were randomly selected to compare the planning performance of the original domain with respect to the augmented domain (the original domain plus the macros). The planner used for our experiments is a simple forward chaining planner based on A* algorithm and the FF heuristic developed with the library PDDL4J[3].

[2]A complete description of the domain and the problems generators are available at http://ipc.icaps-conference.org

[3]http://sourceforge.net/projects/pdd4j/

All the experiments were carried out on an Intel Xeon Core 2.4 with a maximum of 16 Gbytes of memory and were allocated a CPU time of 2 500 seconds. To take into account the computing resources, we arbitrarily limited the n-gram analysis to order 7 and the number of free parameters allowed in a macro to 4. In addition, the maximum number of macros added to the domain was set to 10. Finally, we set the $p\%$ coverage parameter to 50%.

Blocksworld		**Ferry**		**Parking**	
Parameters	*Range*	*Parameters*	*Range*	*Parameters*	*Range*
blocks	3-30	cars	1-200	curbs	2-8
		locations	1-8	cars	1-15
Gripper		**Satellite**		**Depot**	
Parameters	*Range*	*Parameters*	*Range*	*Parameters*	*Range*
robots	1-8	satellites	1-6	depots	1-5
rooms	1-8	instruments	1-2	distributors	1-3
balls	1-75	modes	1-8	trucks	1-4
		targets	1-2	pallets	1-3
		observation	1-20	hoists	1-3
				crates	1-20
Grid		**Mprime**		**Barman**	
Parameters	*Range*	*Parameters*	*Range*	*Parameters*	*Range*
x-scale	3-8	locations	2-8	cocktails	1-30
y-scale	3-8	fuel	1-8	ingredients	1-13
key+lock	1-8	spaces	1-8	shots	1-30
keys	1-8	vehicles	1-8		
locks	1-8	cargos	1-8		
goal proba.	50-100				

Table I
PROBLEM DISTRIBUTION

B. Performance Review

Figure 2 uses a logarithmic scale to show the search time of our baseline planner ('no-macro'), the '2-gram' augmented domain, and the best n-gram order domain ('best estimate'). The 2-gram domain (the lowest macro order) is given as a reference point for the best estimate order domain. The results of Figure 2 are summarised in Table II as relative gains with respect to the baseline planner. For each considered domain, and for each augmented domains, we give the average percentage of gain in terms of search time P and plan length Q: p and q are respectively the search time dispersion and the plan length dispersion.

For each domain, the augmented domains surpass by far the 'no-macro' domain. Regarding the search time, the gain is from 60% for Mprime to 95% for Ferry with the 2-gram domain, and from 77% to 98% for Ferry for the best order domain. Moreover, the best order domain compared with the 2-gram domain improves all the tested IPC domain results, and it is worth noting that the gains increase with the problem difficulty.

C. Impact of N-Gram Order

We have investigated Time Performance (P%), i.e., the percentage of gain in terms of search time P, for different macro orders to (i) evaluate the quality of our good order estimation and (ii) observe if performance according to order was dependent on the domain.

Domain	Macro	P ± p	Q ± q
Blocksworld	2	66 ± 21	-7 ± 0
Blocksworld	3	77 ± 26	-11 ± 1
Ferry	2	95 ± 0	-8 ± 0
Ferry	4	98 ± 0	-22 ± 3
Satellite	2	65 ± 24	-2 ± 0
Satellite	3	82 ± 5	-7 ± 0
gripper*	3	68 ±12	-7 ± 0
gripper*	2	96 ± 0	-9 ± 1
Grid	2	93 ± 1	-4 ± 0
Grid	5	97 ± 0	-42 ± 5
Depot	2	83 ± 4	-4 ± 1
Depot	4	92 ± 1	-16 ± 1
Mprime	2	60 ± 34	-3 ± 0
Mprime	3	86 ± 5	-4 ± 0
parking*	2	74 ± 10	2 ± 2
parking*	3	79 ± 6	0 ± 1
barman	2	14 ± 3	-2 ± 0
barman	6	80 ± 11	-9 ± 1

* Domain with specialized macros

Table II
SUMMARISED EXPERIMENTAL RESULTS

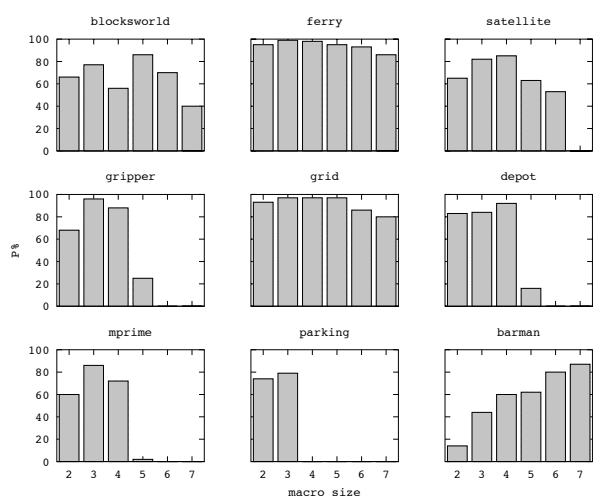

Figure 3. Time Performance (P%) with respect to the macro order from (2-gram to 7-gram)

This experiment shows that our best order estimation provides a relevant indicator for macro order selection. Concerning the domains with specialized macros, Gripper and Parking, both have a branching factor estimation F (actions/facts, see IV-A) greater than 10 (respectively 11 and 17) while in the other domains F is around 4. Gripper and Parking were unable to work without macro specialization because of this high branching factor. In this instance, macro specialization made it possible to decrease the branching factor and solve the domains.

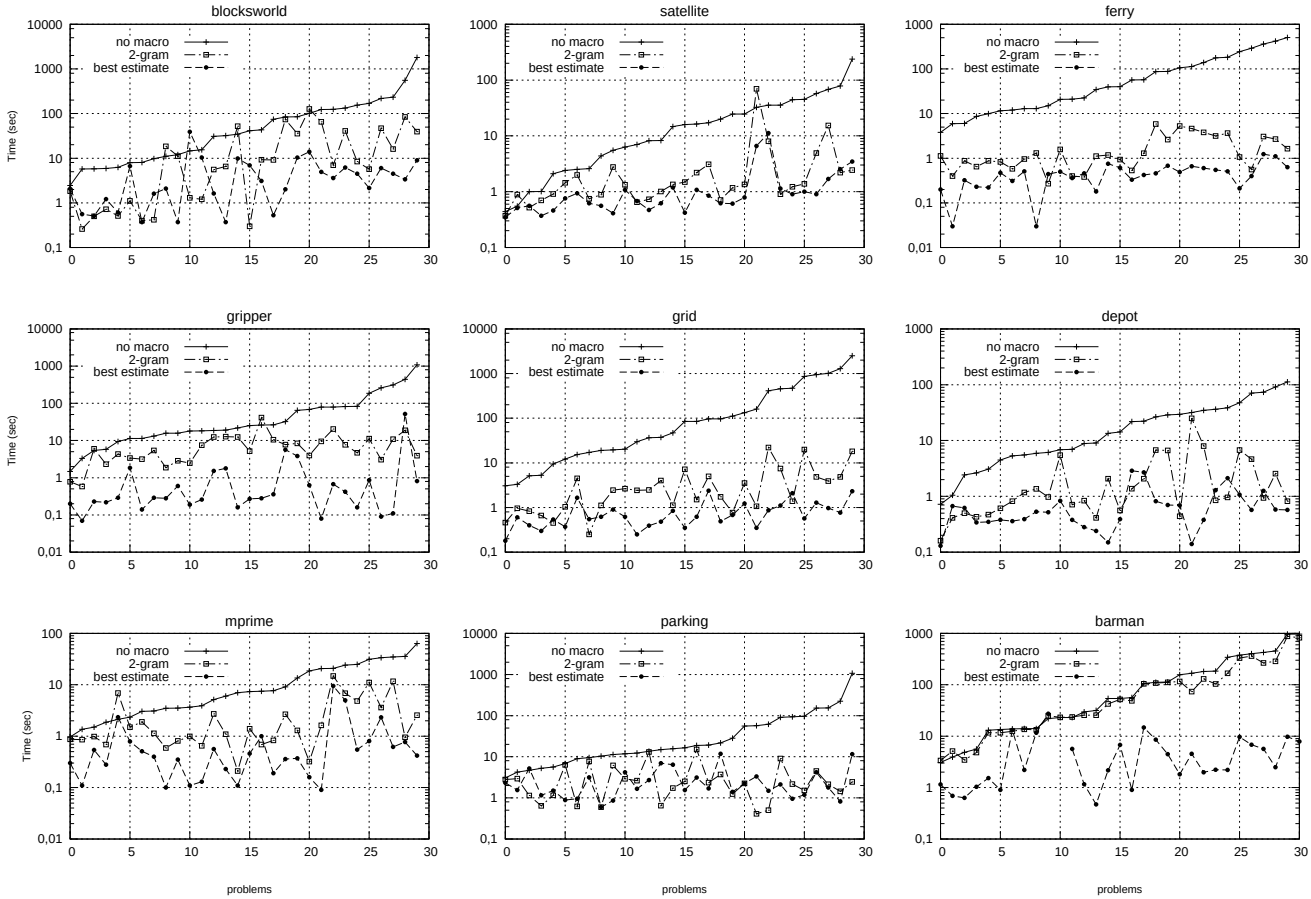

Figure 2. Time performance for the best order and a typical bi-gram: problems are ordered with respect to their difficulty (time resolution with the "no-macro" domain).

It is interesting to observe that some domains lend themselves to the use of multiple macro orders. Ferry and Grid obtain important performance improvements from macro orders 2 to 7. While most of the IPC domains undergo performance decreases up to 3, Barman is an exception. Its performance increases significantly with the macro order. Moreover, Barman is the domain which has the highest number of ground actions and parameters. This suggests that planning domains could be classified with respect to their macro order performance.

D. Discussion

Some lessons can be learned from the experimental results:

- *The generalization of highly probable action sequences into macro-operators can improve significantly the performance of planners*: augmented domains with filtered macro-operators obtain better results than the original domains. However, the use of macros often generates slightly longer plans. This trend increases with the macro order (see figure II).
- *Learning on simple problems improves planning performance for more difficult problems*. Moreover, performance improvements with augmented domains (see Figure 2) are more important when the problems are more difficult. Indeed, in this case macros are used more frequently.
- *N-gram models identify good macro orders for planning domains*: different macro orders enhance domain performances (Figure 3). However, time performances evolve with respect to macro orders. This evolution can be estimated with our utility criteria and the adequate macro order can be chosen to improve planner performance (see Table II).

VI. Conclusion

In this paper we have proposed a domain-independent learning approach in order to enhance planner performance.

This method makes it possible to build on the planner experience – solution plans – and to generate knowledge based on n-gram analysis. The planner's experience is encoded as macros and the domain is re-engineered. Our approach learns good macros maximising the probability of short-cutting the search tree at planning time while controlling the utility problem. Beyond speeding up the search time, it provides a new way to understand the topology and the structure of a domain. From our point of view, this learning approach opens up promising ways in the field of learning and knowledge representation for automated planning.

Our approach can be improved in many ways. As a sequel to the n-gram analysis, the analysis of the syntactic relations of plans seems promising. For example, operator permutations in the macros are sometimes possible without affecting the macro results; this will lead to a reevaluation of the macros' utility, and a definition of macro equivalence relations. Likewise, the combination of different order macros should be investigated. Finally, our syntactic analysis could be completed by a semantic analysis based on precondition and effect relations in actions, and the impact of these relations on macro structures.

More generally, our approach is a closed-loop learning process: supervising techniques such as cross validation could be used to help planners to cluster macros according to their impact during the search phase, for example, by identifying escaping-plateau macros, iterative macros, or bad macros in order to refine the macro library. Although our work is related to state space search, it can be extended to other forms of planning. In particular, we are interested in plan space and HTN planning. The structural information carried by macros could provide operator constraints for plan space planners and patterns for HTN planners.

Another issue in automated planning is domain and problem classification: the features impacting resolution hardness are not clearly understood. We believe that structural information based on n-gram distribution for a domain can be used to determine similarities and differences among domains. Moreover, problem classification could be addressed by the n-gram distribution variations according to different learning databases.

To conclude, our approach to control the utility problem is important for planning under real-time and memory restrictions, such as in embedded systems, as it allows the planner to adapt its decisions according to these restrictions.

References

[1] R. Fikes, P. Hart, and N. Nilsson, "Learning and executing generalized robot plans," *Artificial Intelligence*, vol. 3, no. 4, pp. 251–288, 1972.

[2] C. Dawson and L. Siklóssy, "The role of preprocessing in problem-solving systems," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 1977, pp. 465–471.

[3] R. Korf, "Macro-operators: A weak method for learning," *Artificial Intelligence*, vol. 26, no. 1, pp. 35–77, 1985.

[4] S. Minton, "Selectively generalizing plans for problem-solving," in *Proceedings of International Joint Conference on Artificial Intelligence*, 1985, pp. 596–599.

[5] G. Iba, "A heuristic approach to the discovery of macro-operators," *Machine Learning*, vol. 3, no. 4, pp. 285–317, 1989.

[6] A. Botea, M. Enzenberger, M. Müller, and J. Schaeffer, "Macro-FF: Improving AI planning with automatically learned macro-operators," *Journal of Artificial Intelligence Research*, vol. 24, pp. 581–621, 2005.

[7] M. Newton, J. Levine, M. Fox, and D. Long, "Learning macro-actions for arbitrary planners and domains," in *Proceedings of the International Conference on Automated Planning and Scheduling*, 2007, pp. 256–263.

[8] A. Botea, M. Müller, and J. Schaeffer, "Learning Partial-Order Macros from Solutions," in *Proceedings of the International Conference on Planning and Scheduling*, 2005, pp. 231–240.

[9] J. Hoffmann and B. Nebel, "The FF Planning System: Fast Plan Generation Through Heuristic Search," *Journal of Artificial Intelligence Research*, vol. 14, no. 1, pp. 253–302, 2001.

[10] A. Botea, M. Müller, and J. Schaeffer, "Fast Planning with iterative Macros," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2007, pp. 1828–1833.

[11] A. Coles and A. Smith, "Marvin: A heuristic search planner with online macro-action learning," *Journal of Artificial Intelligence Research*, vol. 28, no. 1, 2007.

[12] A. Coles, M. Fox, and A. Smith, "Online identification of useful macro-actions for planning," in *Proceedings of the International Conference on Automated Planning and Scheduling*, 2007, pp. 97–104.

[13] D. Nau, M. Ghallab, and P. Traverso, *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[14] H. S. Heaps, "Information retrieval: computational and theoretical aspects." Academic Press, 1978, pp. 206–208.