A Critical Look at the Abstraction Based on Macro-Operators

Giuliano Armano[1], Giancarlo Cherchi[1], and Eloisa Vargiu[1]

Department of Electrical and Electronical Engineering - University of Cagliari, Italy
{armano, cherchi, vargiu}@diee.unica.it

Abstract. Abstraction can be an effective technique for dealing with the complexity of planning tasks. This paper is aimed at assessing and identifying in which cases abstraction can actually speed-up the overall search. In fact, it is well known that the impact of abstraction on the time spent to search for a solution of a planning problem can be positive or negative, depending on several factors -including the number of objects defined in the domain, the branching factor, and the plan length. Experimental results highlight the role of such aspects on the overall performance of an algorithm that performs the search at the ground-level only, and compares them with the ones obtained by enforcing abstraction.

1 Introduction

Abstraction is known to be an effective speed-up technique for classical planners. Actually, it is often not effective on simple problems, due to the overhead introduced by the need of going back and forth across abstract spaces while performing the search. In other words, enforcing abstraction on simple problems may end up to wasting computational resources. On the other hand, the more planners will be used to solve problems of increasing complexity, like those encountered in real-life applications, the more abstraction techniques will play a central role in the task of reducing the search time.

Let us briefly recall some relevant abstraction techniques proposed in the literature: (i) action-based, (ii) state-based, (iii) Hierarchical Task Networks, and (iv) case-based. The first combines a group of actions to form macro-operators (e.g., [1], [14]). The second exploits representations of the world given at a lower level of detail; its most significant forms rely on (a) relaxed models, obtained by dropping operators' applicability conditions [16], and on (b) reduced models [13], obtained by completely removing certain conditions from the problem space. In the third (e.g., [9]), problem and operators are organized into a set of tasks: a high-level task can be reduced to a set of partially ordered, lower-level, tasks. Reductions allow specifying how to obtain a detailed plan from an abstract one. In the fourth, abstract planning cases are automatically learned from given concrete cases, as done in the PARIS system [7], although the user must provide explicit refinement rules between adjacent levels of the hierarchy.

Being interested in the action-based approach, let us focus on the pros and cons of the cited techniques. The pionieristic work of Korf [14] was not explicitly tailored for abstraction hierarchies -the adoption of macro-operators being limited to the ground level only. This choice was conjectured and shown to be useful on several domains (see also [8]), although -in our opinion- adding macro-operators at the ground level was at least arguable, due to the corresponding negative impact on the average branching factor. On the other hand, in this way, the completeness of the planner was preserved -none of the original ground operators being removed from the domain. Moreover, the soundness was guaranteed as long as macro-operators represent legal sequences of ground operators. As for the HTN-based techniques [9], in a sense, they can be considered as a generalization of Korf's macro-operators, with a greater expressive power due to their capability of actually defining an abstraction hierarchy, together with the ability of allowing partial ordering among operators. The main drawback of this technique appears to be its strict dependence from the domain engineer, which is responsible for defining a (possibly) sound and complete HTN network for the given domain / problem. Furthermore, the amount of actual search strictly depends on the domain engineer's ability of devising high-level tasks with the desirable property of being easily put together to form a solution for the given problem.

This paper addresses the problem of whether or not abstracting macro-operators can be effective for speeding up the search. Section 2 gives some basic definition about abstraction hierarchies; in particular the ones deemed relevant for the task of automatically abstracting macro-operators. Section 3 highlights some important issues related to the task of abstracting macro-operators. Section 4 points to the aspects that allow predicting when abstraction can be potentially useful or not. In section 5 experimental results are reported, aimed at assessing abstraction on macro-operator as compared to the absence of abstraction.

2 Abstraction Hierarchies and Macro-Operators

In general, a planning domain is defined by two kinds of entities: predicates and operators. A particular kind of unary predicates can also be taken into account, giving rise to a third kind of entities -i.e., types- possibly organized according to a suitable "is-a" hierarchy.

To improve the performance of a planning algorithm, a domain can be organized into a set of abstraction levels, each of them containing its own set of predicates and operators. Thus, the original search space can be mapped into abstract spaces in which irrelevant details are disregarded at different levels of granularity. In particular, abstracting a domain leads to the definition of an abstraction hierarchy, consisting of a set of predicates and operators, together with a set of mapping functions devised to specify the mapping between two adjacent levels. According to [11], an abstraction is a mapping between representations of a problem. In symbols, an abstraction $f : \Sigma_0 \rightarrow \Sigma_1$ consists of a pair of formal systems (Σ_0, Σ_1) with languages Λ_0 and Λ_1 respectively, and an effective total function $f_0 : \mathit{\Lambda}_0 \rightarrow \mathit{\Lambda}_1$. Extending the definition, an abstraction hierarchy consists of a list of formal systems $(\mathit{\Sigma}_0, \mathit{\Sigma}_1, , \mathit{\Sigma}_{n-1})$ with languages $\mathit{\Lambda}_0, \mathit{\Lambda}_1, , \mathit{\Lambda}_{n-1}$ respectively, and a list of effective total functions $f_k : \mathit{\Lambda}_k \rightarrow \mathit{\Lambda}_{k+1}$, (k=0, 1, , n-2) devised to perform the mapping between adjacent levels of the hierarchy.

Let us consider two abstraction levels, namely ground and abstract (the extension of the definitions to an N-level hierarchy being trivial). A ground operator is characterized by a name, a list of parameters, and the specification of its pre- and post-conditions given in terms of ground predicates. A ground operator can be instantiated by substituting its parameters with objects taken from the given problem, thus giving rise to an instantiated ground operator (i.e., an action). An abstract operator is characterized by a name, a list of parameters, and the specification of its overall pre- and post-conditions given in terms of abstract predicates. A macro-operator is any legal sequence of ground operators, together with the specification of its overall pre- and post-conditions. Formally, let be a sequence of operators (actions), a corresponding macro-operator (macro-action) can be defined by the following formulas:

$$\begin{cases} \gamma^+_\omega = \gamma^+_{\omega_1} \cup (\gamma^+_{\sigma'} \setminus \eta^+_{\omega_1}) \\ \gamma^-_\omega = \gamma^-_{\omega_1} \cup (\gamma^-_{\sigma'} \setminus \eta^-_{\omega_1}) \\ \eta^+_\omega = (\eta^+_{\omega_1} \setminus \eta^-_{\sigma'}) \cup (\eta^+_{\sigma'} \setminus \gamma^+_{\omega_1}) \\ \eta^-_\omega = (\eta^-_{\omega_1} \setminus \eta^+_{\sigma'}) \cup (\eta^-_{\sigma'} \setminus \gamma^-_{\omega_1}) \end{cases} \quad (1)$$

where γ^+, γ^-, η^+, and η^-, represent preconditions, negated preconditions, add-list, and delete-list of the resulting macro-operator, respectively.

Although -in principle- abstraction might be performed on both predicates (including types) and operators, there is no a predefined ordering in the abstraction process. In fact, one may start abstracting types, rather than predicates or operators -although any choice performed on one kind of mapping may impact on subsequent choices. Nevertheless, in this paper we are mainly interested in abstracting operators starting from at least one supporting macro-operator, i.e., macro-operators whose pre- and post-conditions match the one defined for the corresponding abstract operator.

It is worth pointing out in advance that the easiest way to generate an abstract operator from a supporting macro-operator is to consider only the preconditions and the effects of the latter. Thus, in the following, the terms abstract- and macro-operator will be used as synonymous.

3 Generation of Macro-Operators

Macro-operators can be obtained by resorting to "a posteriori" or "a priori" analysis.

An "a posteriori" analysis can be done by processing solutions of previously-solved planning problems, under the assumption that solutions of planning problems often contain recurrent sequences of actions. The application of formula 1 to a sequence generates a macro-action that -by definition- leads to the same state that the given sequence of actions would achieve. In an "a posteriori" analysis, macro-actions must be uninstantiated to obtain macro-operators. To unistantiate a macro-action, the objects involved in all its embedded actions are substituted by typed variables. A system able to perform an "a posteriori" analysis has been described in [2], where an adaptive mechanism that allows discovering relevant sequences from successful plans is proposed. Any such sequence becomes a candidate for generating abstract operators to be embedded into a hierarchical planner. To identify relevant sequences, a "chunking" technique that processes successful plans is exploited. Relevant sequences are identified by a feedforward neural network, fed by a vector of suitable metrics evaluated for each given sequence. A corresponding abstract operator is associated to each sequence, which is made available to the abstract level for any subsequent planning problem to be solved. Due to the dependency between abstract operators and already-solved planning problems, an agent equipped with such algorithm may exhibit an individual adaptation to the given environment.

An "a priori" analysis is performed by processing the given planning domain (and the problem, if needed), without resorting to plans previously found (see for example, [3]). In fact, analyzing the relationships among the operators of the domain, a set of relevant sequences can be identified and used for building suitable macro-operators. Given a sequence of operators, a corresponding macro-operator can be defined that embeds the sequence, and whose preconditions and effects can be evaluated according to formula 1. Since the parameters of a macro-operator are in fact variables, generating pre- and post-conditions of the resulting operator involves a variable-unification process, which may led to semantic inconsistencies. For instance, this problem may occur when dealing with "position predicates", such as *(at ?o - object ?l - location)* taken from the *Logistics* domain. According to its intended semantics, there cannot be two predicates stating that the same object is in two different locations. This condition, not explicitly stated in the domain description, can be expressed through the use of suitable state invariants. A detailed description about how to find state invariants is given in [10], where four kinds of state invariants are defined: identity, state membership, uniqueness of state membership, and fixed resource. The information about the domain, enriched with invariants, allows to discriminate between different alternatives, so that macro-operators' parameters can be correctly unified. A system able to generate macro-operators starting from a static domain analysis (called *DHG*) has been described in [5]. The process consists of finding a set of relevant sequences and then (possibly) promoting them to macro-operators using a graph-oriented technique. A directed graph G, containing information about the dependencies between domain operators, is built. Nodes represent operators, and edges represent relations between effects of the source node and preconditions of the destination node. A relevant sequence of operators may be extracted from each acyclic path. As considering all possible paths would end up to a large amount of sequences, G is pruned through a set of domain-independent heuristics (see [3] for further details). A set of sequences (candidates to generate macro-operators) is then extracted. The sequences deemed relevant are used for generating macro-operators [4], according to formula 1. To avoid semantic inconsistencies, an analysis aimed at finding state invariants is performed using TIM [10].

Let us consider, as an example, the two approaches for the *blocks-world* domain. As for the "a posteriori" analysis, processing plans found by solving planning problems, recurrent sequences of action can be identified. In particular, a *pick-up* action is typically followed by a *stack* action, whereas an *unstack* action is typically followed by a *put-down* action. These actions can be grouped together to form a macro-action. The sequence *(pick-up A);(stack A B)*, where A and B are two objects of type block, after the application of 1, followed by a suitable uninstantiation process, leads to the corresponding macro-operator *(pick-up&stack ?b1 ?b2 - block).* As for the "a priori" analysis, processing the *blocks-world* domain, the sequence *(pick-up ?b1 - block);(stack ?b2 ?b3 - block)* can be identified. The application of (1), while computing the difference between the set of preconditions of *stack* and the set of effects of *pick-up*, requires the unification of the parameters *?b1* and *?b2*, since *pick-up* has *(holding ?b1)* as effect, whereas *stack* has *(holding ?b2)* as precondition (see Figure 1).

```
(:action pick-up
   :parameters (?b1 - block)
   :precondition
      (and (clear ?b1) (ontable ?b1) (handempty))
   :effect
      (and (not (ontable ?b1))(not (clear ?b1))
           (not (handempty))(holding ?b1)))

(:action stack
   :parameters (?b2 - block ?b3 - block)
   :precondition (and (holding ?b2) (clear ?b3))
   :effect (and (not (holding ?b2))(not (clear ?b3))
                (clear ?b2)(handempty)(on ?b2 ?b3)))

(:action pick-up&stack
  :parameters (?b1 - block ?b2 - block)
  :precondition
    (and (clear ?b1)(clear ?b2)(ontable ?b1)(handempty))
   :effect
    (and (not(ontable ?b1))(not(clear ?b2))(on ?b1 ?b2)))
```

Fig. 1. *pick-up* and *stack* operator and the corresponding *pick-up&stack* macro-operator

4 A Critical Look at the Abstraction Based on Macro-Operators

The impact of abstraction on the time spent to search for a solution of a planning problem can be positive or negative, depending on several factors -including the average branching factor, and the plan length. Intuitively, in the worst case, the search time grows with the average branching factor (b) and the plan lenght (l) proportionally to b^l. Let us note that, b is influenced by the number of domain operators, the number of parameters of each operator, and the adopted heuristic function; whereas l is influenced by the problem complexity.

Typically, the abstract domain contains fewer operators than the ground domain; nevertheless, there is a usually-negative impact on the average branching factor, due to the increased complexity of the macro-operators. In fact, a macro-operator has generally a number of parameters greater than the ones belonging to each of its operators. On the other hand, using macro-operators reduces the average plan length. Thus, the time required to search for a solution at the abstract level (T_a) may be significantly lower than the time required at the ground level (T_g).

Let us recall that the abstract level is used to guide the search at the ground level. Given a plan at the abstract level, each abstract operator must be refined, each refinement becoming a planning problem at the ground level. For the sake of simplicity, let us suppose that the time required to solve a problem using a two-level abstraction (T_h) is $T_a + T_r$, where T_r, i.e. the time needed to perform all the refinements, is proportional to $l_a \cdot b_g^{l_g/l_a}$. If T_h is greater than T_g, the impact of abstraction is negative, especially if a large number of refinements occurs. It is worth noting that when b_g is close to 1, T_r becomes greater than T_g. This typically occurs when a planner equipped with a good heuristic function is used to refine the abstract solution, thus nullifying the advantages of abstraction. On the other hand, the more b_g increases, the more abstraction becomes effective. In short, the use of abstraction based on macro-operators is not only influenced by the branching factor and the plan length, but also by the adopted planning algorithm.

To verify whether an abstraction based on macro-operator can improve the performances of the search, we made experiments on some classical benchmarking domains. Abstraction hierarchies have been automatically generated using the *DHG* system, which follows an "a priori" approach. For the sake of simplicity, only two relevant domains have been selected, i.e., *elevator* and *blocks-world*.

Let us consider the *elevator* domain, whose operators are reported in Figure 2. An example of recurrent sequence is *up;board*, and the corresponding macro-operator is:

```
(:action up-board
  :parameters
    (?passenger1 - passenger ?floor2 ?floor1 - floor)
  :precondition
    (and (origin ?passenger1 ?floor2)
(:action board
  :parameters   (?f - floor ?p - passenger)
  :precondition (and (lift-at ?f) (origin ?p ?f))
  :effect       (boarded ?p))

(:action depart
  :parameters   (?f - floor ?p - passenger)
  :precondition (and (lift-at ?f) (destin ?p ?f)(boarded ?p))
  :effect       (and (not (boarded ?p)) (served ?p)))

(:action up
  :parameters   (?f1 - floor ?f2 - floor)
  :precondition (and (lift-at ?f1) (above ?f1 ?f2))
  :effect       (and (lift-at ?f2) (not (lift-at ?f1))))

(:action down
  :parameters   (?f1 - floor ?f2 - floor)
  :precondition (and (lift-at ?f1) (above ?f2 ?f1))
  :effect       (and (lift-at ?f2) (not (lift-at ?f1))))
```

Fig. 2. Operators of the *elevator* domain

```
          (lift-at ?floor1) (above ?floor1 ?floor2))
  :effect
    (and (lift-at ?floor2) (boarded ?passenger1)
          (not (lift-at ?floor1))))
```

Note that the *up;board* macro-operator has three parameters, whereas both *up* and *board* have two parameters. In this case, the number of macro-actions corresponding to the *up;board* macro-operator is greater than the number of actions corresponding to the *up* operator plus the number of actions corresponding to the *board* operator. In fact, the number of actions grows with respect to the number of objects belonging to the problem. Let n_f be the number of floors and n_p the number of passengers, the corresponding number of *up;board* instances is $n_p \cdot n_f^2$, the number of *up* instances is n_f^2, and the number of *board* instances is $n_p \cdot n_f$. Hence, the number of applicable actions depends on the number of passengers and floors belonging to the problem to be solved. The automatic hierarchy found by *DHG* has an abstract domain composed by four abstract operators (obtained from the macro-operators corresponding to the sequences *up;board*, *up;depart*, *down;board*, and *down;depart*). Each abstract operator has three parameters (two floors and one passenger), being $4 \cdot n_f^2 \cdot n_p$ the number of applicable actions at the abstract level. On the other hand, the number of applicable actions at the ground level is $2 \cdot n_f^2 + 2 \cdot n_p n_f$. Comparing the two expressions, it is clear that the branching factor at the abstract level is greater than the one at the ground level.

As for the *blocks-world* domain, the automatic hierarchy found by *DHG* has an abstract domain composed by two abstract operators (obtained from the macro-operators corresponding to the sequences *pick-up;stack* and *unstack;put-down*). Each abstract operator has two parameters, being $2 \cdot n_b^2$ the number of applicable actions at the abstract level, where n_b is the number of blocks. On the other hand, the number of applicable actions at the ground level is $2 \cdot n_b + 2 \cdot n_b^2$. Comparing the two expressions, it can be noted that the branching factor at the abstract level is always lower than the one at the ground level.

It is now clear that different behaviors hold, depending on the characteristics of the domain taken into account. In particular, two relevant and different cases have been briefly discussed, pointing to the theoretical and actual branching factor. Roughly speaking, we expect that a hierarchical planner based on macro-operators performs better in the *blocks-world* than in the *elevator* domain.

5 Experimental Results

To assess the performance of the abstraction based on macro-operators, we compared the results obtained without abstraction with the domain hierarchies automatically generated (based on macro-operators only). A set of benchmarking domains, taken from the planning competitions ([15], [6]), including *blocks-world* and *elevator (simple miconic)*, has been chosen to generate the abstraction hierarchies. The domain hierarchies have been used as input for the *HW[]* system ([3]) that can embed any external PDDL-compliant planner to search for solutions at any required level of abstraction. Experiments have been performed using *FF* ([12]) as external planner, being *HW[FF]* the resulting system.

Table 1 summarizes the results obtained for the selected domains. The column labeled T_g reports the time (expressed in milliseconds) needed to find the solution without resorting to abstraction techniques; the corresponding columns labeled *steps* reports the required number of steps. The column labeled T_a reports the time (expressed in milliseconds) needed to find the solution using the abstraction based on macro-operators; the corresponding columns labeled *steps* reports the required number of steps. The column labeled T_h reports the total time (expressed in milliseconds) needed to find the solution using abstraction hierarchies; the corresponding columns labeled *steps* reports the required number of steps.

In the *elevator* domain, results show that T_a is generally lower than T_g, since the average abstract plan length (l_a) is lower than the average ground plan length (l_g), although the average branching factor b_a is greater than b_g. The average time needed to perform a refinement is comparable to T_g, and it can be induced from experimental data that b_g is actually close to 1. In this particular case, l_a negatively affects T_r, which becomes greater than T_g. Hence, resorting to abstraction based on macro-operators becomes ineffective.

In the *blocks-world* domain, results show that, T_a is always lower than T_g, since the average abstract plan length (l_a) is lower than the average ground plan length (l_g), and the average branching factor b_g is always greater than b_a. In

Table 1. Performances comparison between FF without abstraction and HW[FF] with an abstraction based on macro-operators.

	No Abstraction		**Abstraction Based on Macro-Operators**			
Objects	T_g	***steps***	T_a	***steps***	T_h	***steps***
Elevator						
$n_p = 1 n_f = 2$	11.9	4	10.94	2	38.6	4
$n_p = 2 n_f = 4$	13.2	7	15.57	4	72.1	8
$n_p = 3 n_f = 6$	17.1	10	21.44	7	136.6	14
$n_p = 4 n_f = 8$	22.4	14	19.13	8	176.7	16
$n_p = 5 n_f = 10$	27.8	17	23.92	10	266	20
$n_p = 6 n_f = 12$	28.8	19	24.34	13	467.6	26
$n_p = 7 n_f = 14$	38.6	23	26.27	14	641.3	28
$n_p = 8 n_f = 16$	43.3	27	27.26	18	1044	36
$n_p = 9 n_f = 18$	49.4	31	35.18	20	1057	40
Blocks-World						
$n_b = 4$	21.2	10	16.43	3	60.9	6
$n_b = 5$	24.4	12	20.79	7	118.9	14
$n_b = 6$	29.7	12	20.65	9	146.9	18
$n_b = 7$	404.0	36	20.89	12	203.2	24
$n_b = 8$	398.1	30	23.57	11	189.9	22
$n_b = 9$	6493	52	25.8	15	273.1	30

this particular case, abstraction based on macro-operators becomes effective as the problem complexity increases.

6 Conclusions and Future Work

Abstraction is one of the most useful techniques to improve the performances of classical planners. In this paper, abstraction based on macro-operators has been critically analyzed. Experimental results show that there are cases in which abstraction can be effective or not. This work represents a first step towards the comprehension of the underling mechanisms.

As for the future work, we are currently devising a method aimed at deciding whether using abstraction based on macro-operators may be effective or not.

References

1. Amarel, S. On the Representation of Problems of Reasoning about Actions. In D.Michie (ed.), Machine Intelligence, New York (1968)
2. Armano, G., and Vargiu, E. An adaptive Approach for Planning in Dynamic Environments. Proceedings of the International Conference on Artificial Intelligence (IC-AI 2001), Las Vegas (Nevada) (2001) 987–993.

3. Armano, G., Cherchi, G., and Vargiu, E. A Parametric Hierarchical Planner for Experimenting Abstraction Techniques. Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03), Acapulco (Mexico) (2003) 936–941.
4. Armano, G., Cherchi, G., and Vargiu, E. Generating Abstractions from Static Domain Analysis. Workshop dagli Oggetti agli Agenti - Sistemi Intelligenti e Computazione Pervasiva (WOA'03), Cagliari (Italy) (2003)
5. Armano, G., Cherchi, G., and Vargiu, E. Automatic Generation of Macro-Operators from Static Domain Analysis. European Conference on Artificial Intelligence (ECAI'04), Valencia (Spain) (2004)
6. Bacchus, F. Results of the aips 2000 planning competition, 2000. Url: http://www.cs.toronto.edu/aips200
7. Bergmann, R., and Wilke, W. Building and refining abstract planning cases by change of representation language. Journal of Artificial Intelligence Research (JAIR) (1995) **3**, 53–118.
8. Botea A., Mueller M., and Schaeffer J. Using Component Abstraction for Automatic Generation of Macro-Actions. Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-04), Whistler, British Columbia, Canada (2004).
9. Erol, K., Hendler, J., and Nau, D.S. HTN Planning: Complexity and Expressivity Proceedings of the Twelveth National Conference on Artificial Intelligence (AAAI-94), AAAI Press / MT Press, Seattle, WA (1994) 1123–1128.
10. Fox, M., and Long, D. The Automatic Inference of State Invariants in TIM Journal of Artificial Intelligence Research (JAIR) (1998) **9**, 367–421.
11. Giunchiglia, F., and Walsh, T. A Theory of Abstraction. Technical Report 9001-14, IRST, Trento, Italy (1990)
12. Hoffmann, J., and Nebel, B. The ff Planning System: Fast Plan Generation through Heuristic Search Journal of Artificial Intelligence Research (JAIR) (2001) **14**, 253–302.
13. Knoblock, C.A. Automatically Generating Abstractions for Planning. Artificial Intelligence(1994) **68(2)**, 243–302.
14. Korf, R.E. Planning as Search: A Quantitative Approach. Artificial Intelligence (1987) **33(1)**, 65–88.
15. Long, D. Results of the aips 2002 planning competition, 2002. Url: http://www.dur.ac.uk/d.p.long/competition.html.
16. Sacerdoti, E.D. Planning in a hierarchy of abstraction spaces Artificial Intelligence (1974) **5**, 115–135.