Wizard: Suggesting Macro-Actions Comprehensively

M.A. Hakim Newton, John Levine
Computer and Information Sciences
University of Strathclyde
Glasgow, United Kingdom
e-mail: {newton, johnl}@cis.strath.ac.uk

Abstract

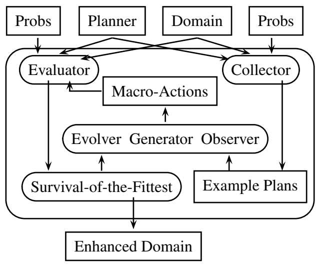
This paper presents Wizard, a generalised framework for learning macro-actions in planning. Wizard suggests macro-actions that can be provided as additional actions for future planning. It enhances a domain for a planner through comprehensive macro suggestions. Wizard learns macro-actions for arbitrary planners or domains without exploiting their structural properties. It not only captures macro-actions that are observable from examples, but also evolves other macro-actions that are not observable. It learns macro-actions that capture various system aspects. It explores both individual macro-actions and their combinations.

Introduction

Planning has achieved significant progress in recent years from planning competitions. The focus of planning research, however, lies mostly on developing planning technologies while the impact of problem formulation on its solution process remains overlooked. Re-engineering a domain by utilising knowledge acquired for a planner paves the way for further research in this direction. Macro-actions, when represented as additional actions, are one relatively convenient way by which to convey such knowledge and achieve domain enhancements. Within current limits of the Planning Domain Definition Language (PDDL), any knowledge can be conveyed only by additional actions and practically only in STRIPS and FLUENTS subsets of the PDDL.

A *macro-action*, or *macro*, is a group of actions selected for application at one time like a single action. One application of a macro leads to planning of several steps at a time. Macros could represent plan fragments that are found with enormous search effort or are frequently used. Macros could capture local search to find better successor nodes especially when the immediate search neighbourhood is not good. Consequently, a goal could be *reached* quickly and problems that are *unsolvable* could become *solvable*[1]. When macros are added into a domain as additional actions, no planner modification is needed; also, the *reachability* of a problem is not affected. But they cause more preprocessing time and incur an extra overhead for the planners adding more branches in the search tree. Furthermore, macros, often lead to increased plan length.

This paper presents Wizard, a *comprehensive framework*[2] (see Figure 1) for learning macros in planning. Given a planner, a domain, and a number of example problems, Wizard suggests macros that can be provided as additional actions for future planning. The framework uses evolutionary and example-based learning approaches for macro generation; for their evaluation, it uses experiential and rewarding methods. Wizard learns macros for arbitrary planners or domains. Planners could have deterministic or stochastic output but their internal architectures could be arbitrary. Although syntactically restricted to STRIPS and FLUENTS, Wizard is not limited by any assumption of particular structures being present in the domain. Wizard explores both individual macros, called *chunks* and their combinations (*i.e.* collections of macros), called *bunches* (henceforth both referred to as macros). Note, a bunch's performance may not be a simple accumulation of the member chunks' performances. Wizard not only captures macros that are observable from examples, but also evolves macros that are not observable. The macro evaluation is based on an weighted average of time gains achieved while solving problems with the original domain and the macro augmented domains. The problems used in evaluation are more *difficult*[3] than those used in capturing macros; however, all of them are solvable with the original domain.



- Planners
  - Deterministic and stochastic
  - Arbitrary planning architecture
- Domains
  - STRIPS and FLUENTS
  - Semantically any structure
- Macros
  - Chunks: observed, evolved
  - Bunches as a combination
  - Capturing various aspects

Figure 1: Wizard's architecture and different dimensions

The rest of the paper, in separate sections, discusses related work, evolutionary algorithms, motivations, implementation, experiment, and conclusion in the order.

[1]By *solvability* we mean, using the original domain, whether the planner can solve the problem within given resource (*e.g.* time, memory, etc.) limits. Whether the goal of a problem can be attained in a given context is discussed under the term *reachability*.

[2]Part of this framework appears in (Newton *et al.* 2007)

[3]By *problem size or difficulty level* we mean, the time required by the given planner to solve the problem with the original domain.

Related Work

Macros are not very new in planning research. STRIPS (Fikes, Hart, & Nilsson 1972) generates macros from unique subsequences of wholly parameterised plans. REFLECT's (Dawson & Siklóssy 1977) macros are based on causal links between actions in the domain. MORRIS (Minton 1985) learns macros from plan fragments that are frequently used or achieve interacting goals. Macro Problem Solver (MPS) (Korf 1985) learns a complete set of macros that totally eliminates the search but only for a particular goal in fixed size problems on domains exhibiting *operator decomposability*. MACLEARN (Iba 1989) learns macros from action sequences that lead the search to reach a peak from another peak in its heuristic profile. It then uses an automated static filter based on domain knowledge and a manual dynamic filter based on usages of macros in plans. It finally adds a small number of macros to the domain like regular actions. MARVIN (Coles & Smith 2004) learns macros from the plan of a reduced version of the given problem after eliminating symmetries and also from the action sequences that help the search escape plateaus in its heuristic profile. It further adopts library management strategies to keep number of macros manageable. Macro-FF (Botea *et al.* 2005) learns macros by using component level abstraction based on static facts of a domain and also by partial-order lifting from plans based on an analysis of causal links. It then evaluates the macros by solving other problems and counting the states explored. It also keeps a small number of macros. Macro-FF's recent extension exploits its FF style search to dynamically build iterative macros *i.e.* chains of the best macros.

Evolutionary Algorithms

An evolutionary algorithm keeps a population of good individuals and generates a new population from the current one using a given set of genetic operators. It then replaces inferior current individuals by superior new individuals (if any) to get a better current population, which is again used to repeat the process until the termination condition is met. In a particular problem context, an individual is taken for a solution (chunk or bunch in our case); which means evolutionary algorithms are an optimisation based multi-point search on the solution space. Moreover, newly generated individuals are other possible solutions in the neighbourhood of the currently kept solutions and a richer collection of genetic operators explores more possible solutions.

Evolutionary algorithms have produced promising results in learning control knowledge for domains and some success in generating plans. EvoCK (Aler, Borrajo, & Isasi 2001) evolved heuristics generated by HAMLET (Borrajo & Veloso 1997) for PRODIGY4.0 (Veloso *et al.* 1995) and outperformed both of them. L2Plan (Levine & Humphreys 2003) evolved control knowledge or policies that outperformed hand-coded policies. Spector, using evolutionary algorithms, managed to achieve plans for small problems having a range of initial and goal states (Spector 1994). SINERGY (Muslea 1998) could only solve problems with specific initial and goal states. GenPlan (Westerberg & Levine 2000) showed that genetic algorithms can generate plans; but it is somewhat inferior than the state-of-the-art planners.

Motivations

Conceptual A system achieves better performance when it is assisted with given knowledge. Such knowledge should be acquired from simpler situations, manipulated for further evolution, reinforced in complex but manageable situations, and applied in yet more complex and even unmanageable situations. The learning method should be generic over the systems for which it learns, over the knowledge it acquires for them, and over the way it conveys knowledge. For artificial systems, excessive knowledge becomes overhead; so volume of delivered knowledge should be optimised. Our interest lies with planning systems as we take planning as self thinking before acting. For various reasons, described in Introduction, we choose macros as knowledge conveyors. Modelling a domain and optimising it for a planner are difficult. Our motivation is to enhance a domain through comprehensive macro suggestions.

Technical Most existing methods (see Related Work) exploit specific planner or domain characteristics, or are limited to capturing macro-actions that are observable in examples only, or learn individual macros only. Any specific properties are not likely to be common with many planners or domains. The examples, especially that are used to acquire previous experiences, might not cover many aspects of the system as what problems make a good collection is not easily addressed. Also, the examples might not always reflect that better choices have been made during the search as planners often make hasty moves to avoid overwhelming grounded actions. Furthermore, keeping arbitrary number of best macros in the collection can not be desirable because individual best macros may not collaborate with each other. Wizard's example-based learning captures observable macros, the evolutionary approach explores other non-observable macros, the experiential and rewarding approaches evaluate them. Wizard generates macros using actions from plans of smaller problems, evaluates them against other larger but solvable problems, and demonstrates performance of the suggested macros against yet larger problems that might include unsolvable instances. The use of plans to represent examples is because plans reflect successful choices made during search and provide a unified source of knowledge, bearing the system's structure inherently.

Implementation

Chunks are represented both as sequences of parameterised (and so generalised) constituent actions and as resultant actions composed up by *regression*[4] of the actions in the sequences; bunches are simple combinations (*i.e.* collections or sets). The genetic operators on chunks operate on sequences and the operators on bunches are set operators. The constituent actions of a chunk come from example plans always. However, the operators are designed from clear motivations of exploring wider spaces of chunks and bunches; chunk operators contain both types – generating observ-

[4]Action composition by regression is a binary, associative, and non-commutative operation on actions where the latter action's precondition and effect are subject to the former action's effect, and both actions' parameters are unified. Regression is practically feasible in STRIPS and FLEUNTS only

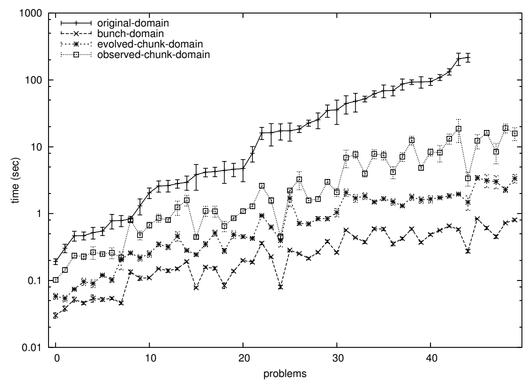
able ones and non-observable ones. Note that, by defining these, the specific knowledge, we give, is actually generic in planning and by no way specific to a planner or a domain. Macro evaluation is based on time gain because explored state or macro usage do not necessarily translate into time efficiency. For a good macro (chunk or bunch), in qualitative terms, *most problems* should be solved taking *less time* in *most cases* when added to the domain. A bad macro, in contrast, causes an overhead that leads to longer solution times or even failures in solving problems within given resource limits. A good macro, however, may not have *high usage* because there could be a less frequent *tricky* macro that saves enormous search time. Furthermore, good macros need not be intuitively natural. Plans containing macros are validated as needed. Various pruning techniques, but not excessive, are used to save effort wastage. Bunches are learnt from the chunks learnt already or even incrementally. Alternatively, a combined procedure explores both chunks and bunches simultaneously focusing more on chunks initially and then shifting focus gradually on bunches.

Experiments

Figure 2 shows performances of some macros for Planner LPG on Domain Blocks. The learning time shown gives an idea and can be improved by tuning different parameters which are currently chosen intuitively. We have other results for a number of planners on several domains; further experiments are underway. The planners are the current state-of-the-art ones from different base architectures (*e.g.* FF, LPG, VHPOP, SATPLAN, SGPLAN, etc.). The domains are bench mark ones (propositional and numerical) used in planning research (*e.g.* blocks, gripper, satellite, ferry, reduced settlers, reduced rovers, etc.) and also some newly created domains. The analysis of the results is based on how significantly the learnt macros improve planners' performances on domains and how effectively differently characterised macros are learnt. The characterisations are on whether macros are chunks or bunches, observable or non-observable, reagent (*i.e.* applicable) or catalytic (*i.e.* not-applicable but help speedup search), domain specific (*i.e.* planner independent) or planner specific, plateau escaping, obtaining interacting subgoals, improving plan length (*e.g.* see examples in Figure 2), etcetera.

Conclusion

This paper presents Wizard, a comprehensive framework for learning macro-actions in planning. The framework uses evolutionary and example-based learning approaches to generate macro-actions; for their evaluation, it uses experiential and rewarding methods. Wizard suggests macro-actions that can be added to the domain permanently as additional actions. Wizard optimises a domain for a planner by suggesting macro-actions. The achievements of Wizard over existing work are manifold. It readily works with arbitrary planners or domains without exploiting any explicitly specific knowledge about them. It learns both individual macro-actions and their combinations. It explores macro-actions that are observable from examples and also that are not. It can learn various types of macro-actions that cover different system aspects. Future work includes experimenting on more domains, speeding up learning by tuning different parameters, and learning macro-actions from simple domains to apply on complex domains.



⋆ S% problems are solved only with the augmented domain and s% only with the original domain.
⋆ T% problems take less time with the augmented domain and t% with the original domain.
⋆ L% problems have less plan length with the augmented domain and l% with the original domain.
⋆ (P%, p%) is (mean, dispersion) of plan time (T) performance $(T_{\text{Orig}} - T_{\text{Aug}})/T_{\text{Orig}}$
⋆ (Q%, q%) is (mean, dispersion) of plan length (L) quality $(L_{\text{Orig}} - L_{\text{Aug}})/L_{\text{Orig}}$

macros	+S -s	+T -t	P ± p	+L -l	Q ± q
Bunch of 2 chunks	+10 -0	+100 -0	64 ± 1	+100 -0	53 ± 1
Chunk-Evolved	+10 -0	+98 -0	60 ± 1	+96 -0	30 ± 1
Chunk-Observed	+10 -0	+90 -0	47 ± 2	+88 -0	21 ± 1

Total Learning Time: 62 mins

Figure 2: Macro-Actions: Planner - LPG, Domain - Blocks

Acknowledgement

This research is supported by the Commonwealth Scholarship Commission in the United Kingdom.

References

Aler, R.; Borrajo, D.; and Isasi, P. 2001. Learning to solve problems efficiently by means of genetic programming. *Evolutionary Computation* 9(4):387–420.

Borrajo, D., and Veloso, M. 1997. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review* 11(1–5):371–405.

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-FF: Improving AI planning with automatically learned macro-operators. *JAIR* 24:581–621.

Coles, A., and Smith, A. 2004. MARVIN: Macro-actions from reduced versions of the instance. In *IPC4 Booklet*. ICAPS.

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

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3(4):251–288.

Iba, G. A. 1989. A heuristic approach to the discovery of macro-operators. *Machine Learning* 3:285–317.

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

Levine, J., and Humphreys, D. 2003. Learning action strategies for planning domains using genetic programming. In *Applications of Evolutionary Computing, EvoWorkshops2003*, volume 2611, 684–695.

Minton, S. 1985. Selectively generalising plans for problem-solving. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Muslea, I. 1998. A general purpose AI planning system based on the genetic programming paradigm. In *Proceedings of the World Automation Congress*.

Newton, M. A. H.; Levine, J.; Fox, M.; and Long, D. 2007. Learning macro-actions for arbitrary planner and domains. In *Proceedings of the ICAPS*.

Spector, L. 1994. Genetic programming and AI planning system. In *Proceedings of the Twelfth National Conference on Artificial Intelligence, AAAI-94*, 1329–1334.

Veloso, M.; Carbonell, J.; Perez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence* 7:81–120.

Westerberg, C. H., and Levine, J. 2000. GenPlan: Combining genetic programming and planning. In *19th Workshop of the UK PLANSIG*.