Tool for automatically acquiring control knowledge for planning

Daniel Borrajo **Susana Fernández** **Raquel Fuentetaja** **Juan D. Arias**
Departamento de Informática, Universidad Carlos III de Madrid
Avda. de la Universidad, 30. Leganés (Madrid). Spain
dborrajo@ia.uc3m.es, {sfarregu,rfuentet,jdarias}@inf.uc3m.es

Manuela Veloso
Computer Science Departament, Carnegie Mellon University
Pittsburgh PA 15213-3890, USA
veloso@cs.cmu.edu

Abstract

Current planners show impressive performance in many real world and artificial domains by using planning (either domain dependent or independent) heuristics. But, on one hand, domain dependent planners still outperform domain independent planners by re-defining domain theories, also including control knowledge. On the other hand, these domain dependent planners require a careful and manual refinement of domain theories to incorporate domain and control knowledge. Here, we present a tool that automatically generates domain and control knowledge as a middle ground solution to the definition of efficient quality-based planners.

Introduction

Planning technology has experimented a big advance in a decade. The development of GRAPHPLAN (Blum & Furst 1995) and the definition of domain independent heuristics based on relaxed domains as in HSP (Bonet & Geffner 2001) or FF (Hoffmann & Nebel 2001) has provided a great impact in the field. But, even if these heuristics are very informative, and efficiently computed, they are heuristics after all, so they fail in some problems and domains, specially with respect to computing *good* plans according to different quality metrics. A parallel approach has been domain dependent planners. They rely on the time consuming, error prone manual design of their domain theories. Representative examples are: HTN approaches, as SHOP2 (Nau *et al.* 2003), that require a re-definition of the domain theory from the standard planning STRIPS/ADL-based competition language, PDDL (Fox & Long 2002), into networks of tasks and methods; or temporal logic approaches, such as TLPLAN (Bacchus & Kabanza 2000), that require the definition of new predicates based on others, and search control knowledge expressed in a temporal logic.

Copyright © 2005, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

In this paper, we propose the use of tools that can guide the heuristics of the first type of planners when they lead towards no solution or *bad* solutions, and doing it automatically in opposition to the second type of planners. In the past, these type of tools have usually employed fully automated machine learning technology ranging from macro-operators acquisition (Fikes, Hart, & Nilsson 1972; Korf 1985), case-based reasoning (Veloso 1994; Kambhampati 1989), rewrite rules acquisition (Ambite, Knoblock, & Minton 2000; Upal & Elio 2000), generalized policies (Khardon 1999), deductive approaches of control knowledge learning (Minton 1988; Qu & Kambhampati 1995; Kambhampati 2000), learning domain models (Wang 1994), to inductive approaches (ILP based) (Borrajo & Veloso 1997; Estlin & Mooney 1997; Aler, Borrajo, & Isasi 2002; Huang, Selman, & Kautz 2000) (see (Zimmerman & Kambhampati 2003) for a general overview of the field).

However, we believe that a better knowledge engineering solution for the deployment of this technology in real world applications consists of a mixed-initiative approach to domain and control knowledge acquisition (Aler & Borrajo 2002). More specifically, we present here a tool that is able to learn search control knowledge in the form of control rules, as well as macro-operators, for the re-definition of the domain theory. Control rules learning is based on two approaches: a pure deductive approach, as EBL (Minton 1988; Mitchell, Keller, & Kedar-Cabelli 1986), and an inductive-deductive approach, based on HAMLET (Borrajo & Veloso 1997). Macro-operators learning consists on using a straightforward implementation of STRIPS macro-operators (Fikes, Hart, & Nilsson 1972). This tool works on top of the IPSS integrated planner and scheduler (Rodríguez-Moreno *et al.* 2004b), which is based on QPRODIGY (Borrajo, Vegas, & Veloso 2001) and PRODIGY (Veloso *et al.* 1995).

IPSS planner

IPSS is an integrated tool for planning and scheduling. It uses the quality-based version of the PRODIGY planner, QPRODIGY, integrated with a constraints based scheduler (Cesta, Oddi, & Smith 2002). We have not yet used the learning tools in combination with the

scheduler, so we will focus now only on the planning component of IPSS. The planning component is a nonlinear planning system that follows a means-ends analysis (Veloso *et al.* 1995). The inputs to the planner (domain theory, problem, and control knowledge) are specified in the PDL language (PRODIGY DESCRIPTION LANGUAGE (Minton *et al.* 1989; Carbonell *et al.* 1992)) which is similar in terms of representation power to the ADL version of PDDL2.2 (including axioms, called inference rules in PDL).[1] It provides some language dependent enhancements, such as the possibility of calling LISP functions to specify values of operators variables or operator quality metrics, or definition of functions called handlers that are called by the planner everytime a node is expanded. It also provides planner dependent enhancements as the possibility of having predicates with more than one numeric argument. Also, control knowledge can be declaratively represented and used by the planner by defining control rules.

From a control knowledge acquisition perspective, IPSS planning cycle, involves several decision points. Figure 1 shows an schematic view of a generic search tree with all those decisions. The types of decisions made are:

- *select a goal* from the set of pending goals and subgoals;
- *choose an operator* to achieve a particular goal;
- *choose the bindings* to instantiate the chosen operator;
- *apply* an instantiated operator whose preconditions are satisfied or continue *subgoaling* on another unsolved goal.

Figure 2 shows an example of control knowledge represented as a rule to determine when the `unload-airplane` operator of the well known logistics domain must be selected for achieving the goal of having an object in an airport of another city where it is now. First of all, remember that this is a backward chaining planner, so decisions are made when trying to achieve the goals. The `different-vars-p` meta-predicate in the control rule checks whether all different variables (between < and >) of the rule are all bound to different values. So, it says that if an object has to be at a location, that is of type airport, (goal) and currently is at another location in another city (state), then in order to achieve the goal, the planner should use the operator `unload-airplane`.

The planner does not incorporate yet domain independent heuristics as current planners do, so its performance cannot really be compared against those with respect to efficiency. However, we have shown elsewhere that it can be competitive with respect to state of the art planners if we compare it with respect to its flexibility to incorporate quality-based, scheduling or other types of reasoning (Rodríguez-Moreno *et al.* 2004a).[2]

[1]We have defined translators working both directions, PDL to PDDL and viceversa, covering almost all representation features.

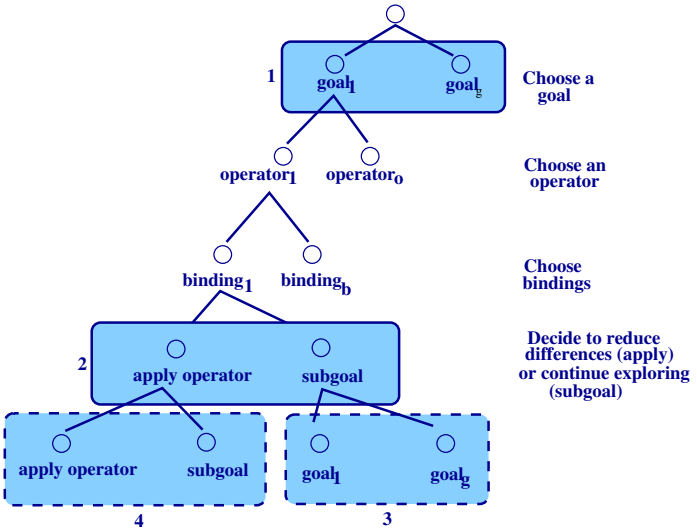

Figure 1: Generic IPSS search tree.

```
(control-rule select-operators-unload-airplane
   (if (current-goal (at <object> <location1>))
       (true-in-state (at <object> <location2>))
       (true-in-state (loc-at <location1> <city1>))
       (true-in-state (loc-at <location2> <city2>))
       (different-vars-p)
       (type-of-object <object> object)
       (type-of-object <location1> airport))
   (then select operator unload-airplane))
```

Figure 2: Example of a hand crafted control rule for selecting the `unload-airplane` operator in the logistics domain.

The reasons to choose IPSS are manyfold. Among them, we can highlight the possibility of defining and handling different quality metrics (Borrajo, Vegas, & Veloso 2001), reasoning about multiple criteria, flexibility to easily define new behaviours (through handlers), capability to represent and reason about numeric variables, definition of constrains on variable values in preconditions of operators, explicit definition of control knowledge as well as its automatic acquisition through different machine learning techniques (Aler, Borrajo, & Isasi 2002; Borrajo & Veloso 1997; Veloso *et al.* 1995), and explicit rationale of each problem solving episode through the search tree. This last feature is needed for building learning techniques, since we need to access decisions made and their rationale after search has been performed.

[2]We have interesting results using path planning that have been submitted elsewhere.

HAMLET

HAMLET is an incremental learning method based on EBL (Explanation Based Learning) and inductive refinement of control rules (Borrajo & Veloso 1997). The inputs to HAMLET are a task domain ($\mathcal{D}$), a set of training problems ($\mathcal{P}$), a quality measure (Q) and other learning-related parameters. As any machine learning inductive tool, it needs a set of training problems to be given as input. One way of providing them consists on defining a problem generator, though this solution requires to build one such generator for each domain. A potential solution consists on automating this task (McCluskey & Porteous 1997). We have already done some preliminar experimentation on what types of problems, and domain models, are more appropriate for HAMLET and other related learning systems (Aler, Borrajo, & Isasi 2000), that we would like to extend in the future. Also, we have done some preliminar work on active learning (automatic generation of *good* problems for learning) with no convincing results yet.

The quality metric measures the quality of a plan in terms of number of operators in the plan, execution time (makespan), economic cost of the planning operators in the plan or any other user defined criteria. HAMLET can also use as input an initial set of control rules, so it also can be used for theory refinement. Internally, each call to the learning system receives the search tree expanded by the planner to decide what to learn.

These would be the parameters a non-expert user would set up. For advanced users, there are many other parameters for which the tool provides a reasonable default value that we are not explaining here, but can be played with when using the tool. The output is a set of control rules ($\mathcal{C}$) that potentially guide the planner towards *good* quality solutions. Evidently, for each quality metric, a new set should be learned. HAMLET has two main modules: the Bounded Explanation module, and the Refinement module. Figure 3 shows HAMLET modules and their connection to the planner.

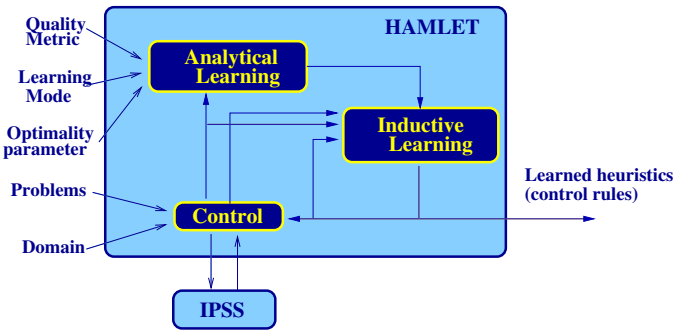

Figure 3: HAMLET high level architecture.

The Bounded Explanation module generates control rules from an IPSS search tree by finding examples of right decisions (lead to a good solution instead of a failure path). Once a right decision is found, a control rule is generated by extracting the meta-state, and performing a goal regression for finding which literals from the state were needed to be true to make this decision (the details can be found in (Borrajo & Veloso 1997)). These EBL like rules might be overly specific or overly general. HAMLET Refinement module solves the problem of being overly specific by generalising rules when analysing new positive examples of decisions of the same type. It also replaces overly general rules with more specific ones when it finds situations in which the learned rules lead to wrong decisions. HAMLET gradually learns and refines control rules, in an attempt to converge to a concise set of correct control rules (i.e., rules that are individually neither overly general, nor overly specific).

Figure 4 shows an example of a rule learned by HAMLET in the logistics domain for selecting the `unload-airplane` operator. This rule would be learned by first learning a rule that would require the airplane to be in another airport from <`location1`> (conditions computed from the goal regression), then learning another similar rule from a problem in which the airplane is at the same <`location1`>, and finally inducing a new more general rule from these other two (and removing the other two more specific rules).

```
(control-rule select-operators-unload-airplane
   (if (current-goal (at <object> <location1>))
       (true-in-state (inside <object> <airplane>))
       (different-vars-p)
       (type-of-object <object> object)
       (type-of-object <airplane> airplane)
       (type-of-object <location1> airport))
   (then select operator unload-airplane))
```

Figure 4: Example of a control rule learned by HAMLET for selecting the `unload-airplane` operator in the logistics domain.

We have used the same deductive approach for automatically acquiring control rules for other types of planners, such as hybrid HTN-POP planners as HYBIS (Castillo, Fernández-Olivares, & González 2001), though we have not yet implemented its inductive counterpart (Fernández, Aler, & Borrajo 2005). This domain independent planner has been used for industrial manufacturing domains. Within the general framework of learning control knowledge for different types of planners, we are trying to study what is the language for describing this type of knowledge for different planning paradigms, independently of the planner. Our ongoing unpublished work we are applying the same ideas on top of other planners in order to share control knowledge among them with preliminary good results. The main difficulty lies on the fact that each planner generates a different type of search tree, and, therefore, different decision points. However, there is some common knowledge that appears in all searches that we are

trying to capture. In relation to sharing another type of knowledge among planners, in (Fernández, Aler, & Borrajo 2004) we report on the use of FF (Hoffmann & Nebel 2001) solutions for bootstrapping learning in planning in difficult domains for the base planners.

In relation to the use of HAMLET as a mixed-initiative approach or tool, we believe that these control rules are easier to generate, understand, and, therefore, modify by a non planning expert than other kinds of control/domain knowledge coding, such as: the domain independent heuristics of heuristic based planners (Bonet & Geffner 2001; Hoffmann & Nebel 2001), that have the advantage of being domain independent, but they have to be changed by re-programming the planning tool; the temporal logic used in TLPLAN (Bacchus & Kabanza 2000), that has the advantage of a very fast approach, but the user has to understand how temporal logics work; or the generation of HTN models (Nau *et al.* 2003; Currie & Tate 1991). This assumption has been drawn from informal talks with people developing such tools, and our experience in HTN industrial modelling (Fernández, Aler, & Borrajo 2005). However, it has only been empirically validated using a planning expert in (Aler & Borrajo 2002; Fernández, Aler, & Borrajo 2004), but it has not been yet done with non-experts. So, it is just a conjecture for now, which we plan to evaluate in the future.

EBL

In order to also use a pure deductive approach, we built an EBL module by re-using HAMLET's code. We only used its Bounded-Explanation module and learned rules from all decisions that were not the first alternative tried in each node. So, in the previous example, it would learn and keep the two more specific rules, instead of performing the inductive step. Then, a utility analysis was performed for each learned rule. According to the usual equation (Minton 1988),

$$u(r) = s(r) \times p(r) - m(r)$$

where $u(r)$ is the utility of a rule r, $s(r)$ is an estimation of the time that the rule saves when used, $p(r)$ is the probability that the rule would match, and $m(r)$ is the cost of using the rule (matching time). Therefore, we estimated the following variables:

- $s(r)$: given that each rule is learned from a node in a search tree, we estimate $s(r)$ as the number of nodes below the node from where it learned the rule, multiplied by the time IPSS takes to expand a node. A better estimate can be computed by using a second training set of problems as Minton did.
- $p(r)$: we estimated it as the number of times that IPSS tried to use the rule in subsequent problems in the training phase divided by the number of times that it actually fired. As before, a better estimate can be obtained by using a second training set.
- $m(r)$: it is estimated as the total matching time added over all times that it tried to match the rule divided by the number of times that it tried to match it.

After training, the utility $u(r)$ of each rule is computed, and rules whose utility is less than a user-defined threshold can be removed. Better solutions for estimating the control rules utility can be found in other papers (Gratch & DeJong 1992). The tool provides an access to the needed information, so they can potentially be easily implemented here also.

Macro-operators learning

The tool also provides a utility for automatically building macro-operators that are incorporated into the domain theory. This provides some kind of abstraction that can be used by the user to refine the domain theory, creating more efficient versions of it, as it has been shown by MACRO-FF (Botea, Mueller, & Schaeffer 2005). Now, the implementation does not incorporate an utility analysis, as in recent implementations as MACRO-FF or the work reported in (McCluskey & Porteous 1997), but we already have some code for computing utility of control knowledge that can be easily reused for this purpose.

Tool's description

The current implementation status of the complete planning and learning tool is that the different planning and learning components have already been implemented as described in this and other papers (Borrajo & Veloso 1997; Veloso *et al.* 1995; Aler, Borrajo, & Isasi 2002). We refer to these papers for more evaluation on its learning techniques. The user interface is currently under construction and we plan to show it during the knowledge engineering planning competition. We already have a very simple user interface written in Lisp[3] but we plan to rewrite it using generic user interfaces generation tools. Our goal would be that using this interface a non planning expert would be able to interact with the planning and learning tool in order to generate, or modify control knowledge and macro operators in collaboration with it.

Figure 5 shows a high level view of the tool. The central part would be the interaction with the IPSS planning tool. The input is a problem (potentially provided by a problem generator), a domain description, a set of planning parameters (time bound, node bound, depth bound, how many solutions to generate, ...), and, optionally a set of control rules. The output is a set of solutions and an explicit search tree. The bottom part corresponds to the control rules learning part. The inputs to this learning component are a domain, a set of training problems, a search tree for each problem, and a set of parameters (quality measure, learning eagerness, time bound, ...). The output is a set of control rules that can be refined by the user through the GUI (or by any other learning system, as in (Fernández, Aler,

[3]More specifically, in Lispworks.

& Borrajo 2004)). The top part is the macro-operators learning component. It receives a domain, a set of solutions and some parameters, and generates as output a set of macro-operators. These can be further refined by the user through the GUI.

Acknowledgements

This work has been partially supported by the Spanish MCyT under project TIC2002-04146-C05-05.

References

Aler, R., and Borrajo, D. 2002. On control knowledge acquisition by exploiting human-computer interaction. In Ghallab, M.; Hertzberg, J.; and Traverso, P., eds., *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS-02)*, 112–120. Toulouse (France): AAAI Press.

Aler, R.; Borrajo, D.; and Isasi, P. 2000. Knowledge representation issues in control knowledge learning. In Langley, P., ed., *Proceedings of the Seventeenth International Conference on Machine Learning, ICML'00*, 1–8.

Aler, R.; Borrajo, D.; and Isasi, P. 2002. Using genetic programming to learn and improve control knowledge. *Artificial Intelligence* 141(1-2):29–56.

Ambite, J. L.; Knoblock, C. A.; and Minton, S. 2000. Learning plan rewriting rules. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, 14–17.

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116:123–191.

Blum, A. L., and Furst, M. L. 1995. Fast planning through planning graph analysis. In Mellish, C. S., ed., *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI-95*, volume 2, 1636–1642. Montréal, Canada: Morgan Kaufmann.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1-2):5–33.

Borrajo, D., and Veloso, M. 1997. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review Journal. Special Issue on Lazy Learning* 11(1-5):371–405. Also in the book "Lazy Learning", David Aha (ed.), Kluwer Academic Publishers, May 1997, ISBN 0-7923-4584-3.

Borrajo, D.; Vegas, S.; and Veloso, M. 2001. Quality-based learning for planning. In *Working notes of the IJCAI'01 Workshop on Planning with Resources*, 9–17. Seattle, WA (USA): IJCAI Press.

Botea, A.; Mueller, M.; and Schaeffer, J. 2005. Learning partial-order macros from solutions. In *Proceedings of ICAPS'05*.

Carbonell, J. G.; Blythe, J.; Etzioni, O.; Gil, Y.; Joseph, R.; Kahn, D.; Knoblock, C.; Minton, S.; Pérez, A.; Reilly, S.; Veloso, M. M.; and Wang, X. 1992. PRODIGY4.0: The manual and tutorial. Technical Report CMU-CS-92-150, Department of Computer Science, Carnegie Mellon University.

Castillo, L.; Fernández-Olivares, J.; and González, A. 2001. Mixing expressiveness and efficiency in a manufacturing planner. *Journal of Experimental and Theoretical Artificial Intelligence* 13:141–162.

Cesta, A.; Oddi, A.; and Smith, S. F. 2002. A Constrained-Based Method for Project Scheduling with Time Windows. *Journal of Heuristics* 8:109–136.

Currie, K., and Tate, A. 1991. O-Plan: the open planning architecture. *Artificial Intelligence* 52(1):49–86.

Estlin, T. A., and Mooney, R. J. 1997. Learning to improve both efficiency and quality of planning. In Pollack, M., ed., *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, 1227–1232. Morgan Kaufmann.

Fernández, S.; Aler, R.; and Borrajo, D. 2004. Using previous experience for learning planning control knowledge. In Barr, V., and Markov, Z., eds., *Proceedings of the Seventeen International Florida Artificial Intelligence (FLAIRS04)*, 713–718. Miami Beach, FL (USA): AAAI Press.

Fernández, S.; Aler, R.; and Borrajo, D. 2005. Machine learning in hybrid hierarchical and partial-order planners for manufacturing domains. *Applied Artificial Intelligence* 19(10). Accepted.

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

Fox, M., and Long, D. 2002. *PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains*. University of Durham, Durham (UK).

Gratch, J., and DeJong, G. 1992. COMPOSER: A probabilistic solution to the utility problem in speed-up learning. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 235–240.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Huang, Y.-C.; Selman, B.; and Kautz, H. 2000. Learning declarative control rules for constraint-based planning. In Langley, P., ed., *Proceedings of the Seventeenth International Conference on Machine Learning, ICML'00*.

Kambhampati, S. 1989. *Flexible Reuse and Modification in Hierarchical Planning: A Validation Structure Based Approach*. Ph.D. Dissertation, Computer Vision Laboratory, Center for Automation Research, University of Maryland, College Park, MD.

Kambhampati, S. 2000. Planning graph as a (dynamic) CSP: Exploiting EBL, DDB and other CSP search techniques in Graphplan. *Journal of Artificial Intelligence Research* 12:1–34.

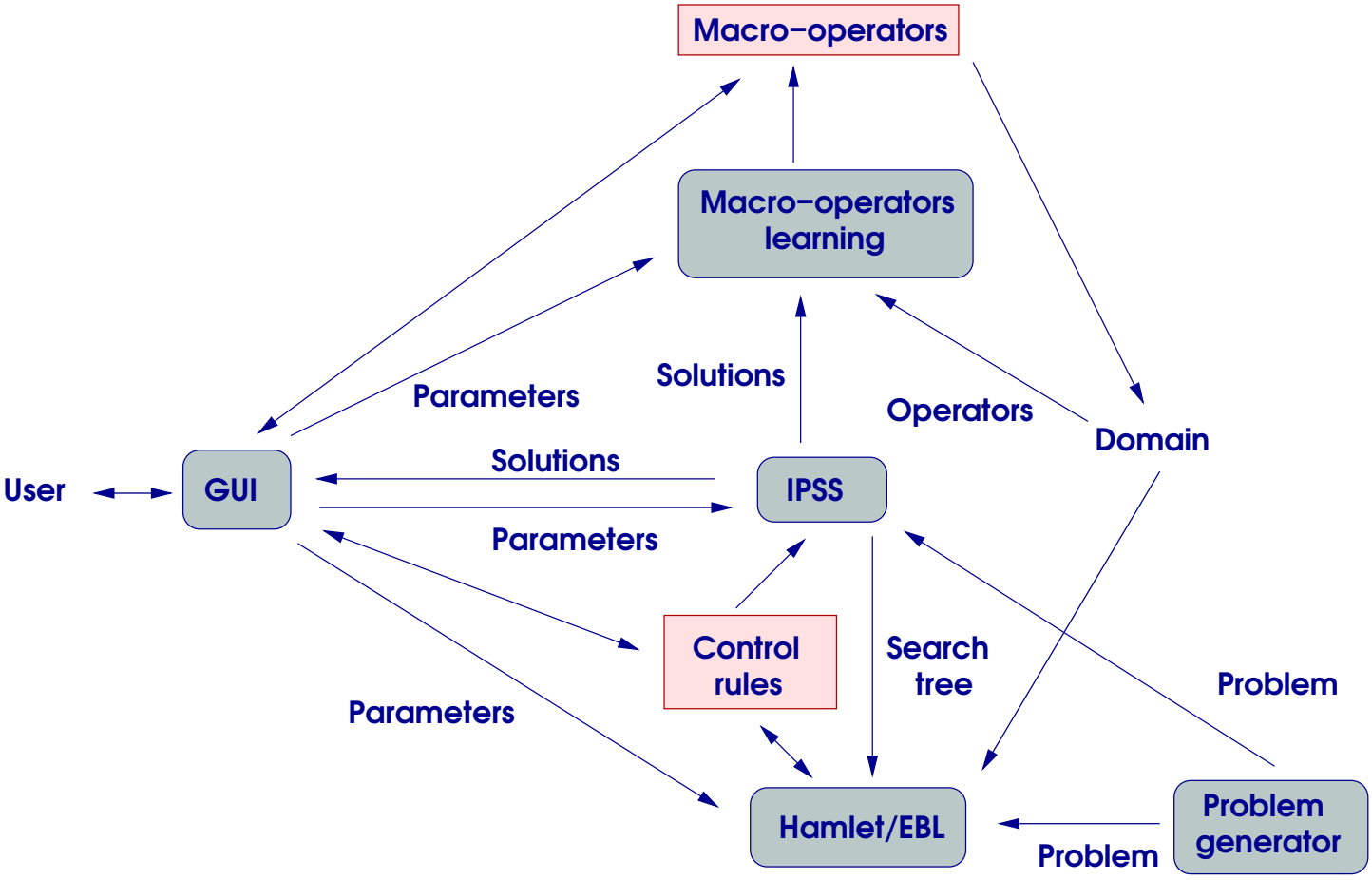

Figure 5: High level architecture of the planning and learning tool.

Khardon, R. 1999. Learning action strategies for planning domains. *Artificial Intelligence* 113(1-2):125–148.

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

McCluskey, T. L., and Porteous, J. M. 1997. Engineering and compiling planning domain models to promote validity and efficiency. *Artificial Intelligence* 95(1):1–65.

Minton, S.; Knoblock, C. A.; Kuokka, D. R.; Gil, Y.; Joseph, R. L.; and Carbonell, J. G. 1989. PRODIGY 2.0: The manual and tutorial. Technical Report CMU-CS-89-146, School of Computer Science, Carnegie Mellon University.

Minton, S. 1988. *Learning Effective Search Control Knowledge: An Explanation-Based Approach*. Boston, MA: Kluwer Academic Publishers.

Mitchell, T. M.; Keller, R. M.; and Kedar-Cabelli, S. T. 1986. Explanation-based generalization: A unifying view. *Machine Learning* 1:47–80.

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Mur, J. W.; and Wu, D. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Qu, Y., and Kambhampati, S. 1995. Learning search control rules for plan-space planners: Factors affecting the performance. Technical report, Arizona State University.

Rodríguez-Moreno, M. D.; Borrajo, D.; Oddi, A.; Cesta, A.; and Meziat, D. 2004a. IPSS: A problem solver that integrates P&S. In *Third Italian Workshop on Planning and Scheduling*.

Rodríguez-Moreno, M. D.; Oddi, A.; Borrajo, D.; Cesta, A.; and Meziat, D. 2004b. IPSS: A hybrid reasoner for planning and scheduling. In de Mántaras, R. L., and Saitta, L., eds., *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004)*, 1065–1066. Valencia (Spain): IOS Press.

Upal, M. A., and Elio, R. 2000. Learning search control rules versus rewrite rules to improve plan quality. In *Proceedings of the Thirteenth Canadian Conference on Artificial Intelligence*, 240–253. New York: Springer-Verlag.

Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink,

E.; and Blythe, J. 1995. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical AI* 7:81–120.

Veloso, M. 1994. *Planning and Learning by Analogical Reasoning*. Springer Verlag.

Wang, X. 1994. Learning planning operators by observation and practice. In *Proceedings of the Second International Conference on AI Planning Systems, AIPS-94*, 335–340. Chicago, IL: AAAI Press, CA.

Zimmerman, T., and Kambhampati, S. 2003. Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine* 24(2):73–96.