

# The Optop Planner

Drew McDermott

Yale University Computer Science Department  
P.O. Box 208285  
New Haven, CT 06520-8285  
drew.mcdermott@yale.edu

## Introduction

Optop<sup>1</sup> is an *estimated-regression* planner, meaning that it is a “state-space planner” that is guided by a heuristic measure of how close a situation is to satisfying a goal, and how good it is according to an objective function. Research on Optop is focused more on deep reasoning about situations and transitions than on raw performance.

Instead of talking about “state space,” I prefer to characterize the search space of Optop as the set of *plan prefixes*, that is, sequences of actions that are executable starting in the initial state. Such a sequence generates a unique situation, called the *current situation* for that prefix.

## Heuristic Search Using Estimated-Regression Graphs and Objective Functions

Optop decides which plan prefix to work on next using a heuristic inspired by *means-ends analysis* (Ernst & Newell 1969). For each plan prefix, it constructs a *regression-match graph* that is a simplified prediction of how that goal might be achieved starting in the current situation for a given plan prefix. The graph is constructed by *maxmatching* the goal against the current situation, which produces a substitution (called a *maximal match*) that binds the variables in the goal so as to make as many of its conjuncts true as possible. The remaining conjuncts, the *differences* left by the maxmatch, become subgoals. For each literal in differences, Optop finds all the actions, processes, or implications that could make it true. Each has some kind of precondition that is maxmatched against the current situation, giving further differences. As this process is continued, a tripartite graph emerges, each of whose nodes is of one of the following three types:

1. An *L-node*: A literal occurring as differences in a maxmatch.

Copyright © 2004, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

<sup>1</sup>This looks like an acronym for something; ordered? operator? tops? How is it syllabified, as Opt-op or Op-top?

2. An *effort-spec*: An L-node plus numerical constraints on its free variables. Numerical constraints can’t be handled by regression, but must be postponed and satisfied by a special numerical module at the appropriate time.
3. A *reduction*: A record of the application of a “regression method” to an effort-spec. A typical regression method corresponds to an action definition, and specifies sufficient conditions for that action to cause each of its possible effects. (Some of the other kinds are discussed below.) These conditions are maxmatched to derive a set of differences, each of which is an effort-spec in the graph.

Each effort-spec may have several reductions, and each reduction has a set of precondition effort-specs which are sufficient to ensure that the action, process, or implication associated with the reduction will cause the L-node of the effort-spec to be true. (Actually, reductions and maxmatches are cached on L-nodes; when an effort-spec for an L-node, is built, Optop copies the reductions, adds the numerical constraints if any, and verifies that they are satisfiable.)

L-nodes and effort-specs are “uniquified”; that is, if an equal L-node already exists, it is used instead of a new one being created. That means the regression-match graph for a planning problem tends to be much smaller than its situation space.

The graph yields a rough estimate of the difficulty of the problem, obtained by counting the actions in a subtree of the graph that is minimal in a sense explained in (McDermott 1996; 1999). However, many planning problems include a specification of an “objective function” to be minimized. Optop finds linearizations of the regression-match graph that then give rise to *plausible projections* of the rest of the plan. The result is a collection of feasible actions and speculative versions of the final situation that might follow from them, and Optop evaluates the objective function in those situations to produce estimates of the quality of alternative extensions of the current plan prefix (McDermott 2003).

## Expressivity

In addition to actions, Optop can reason about autonomous processes, which run whenever their conditions are true without the need for planner intervention. The planner can plan to bring these into existence by making the condition true, or can take advantage of processes defined as part of the problem.

Optop can handle all of ADL (Pednault 1989), including universally-quantified preconditions. It uses the Screamer system (Siskind & McAllester 1993) to solve numerical constraints, especially those that arise in connection with predicting when processes will cause something to become true.

The reason for Optop’s versatility is that its reasoning is closely tied to complete descriptions of situations, unlike partial-order planners (Weld 1994) and Graphplan-style planners (Blum & Furst 1997), which reason about goal-satisfaction links, mutual-exclusion relations, and the like without tying them to any particular situation. Generating the regression-match graph requires reasoning backward from the goal to the current situation, and can use any reasoning technique, domain-dependent or -independent, without worrying about enough information is known about that situation. (Of course, that is not the only consideration; Optop is no better than other Strips-style planners in doing regression involving geometrical reasoning.)

Once an action is chosen to explore, Optop typically generates a new current situation following that action. However, if autonomous processes are active, the next situation is the one that occurs when those processes cause a discrete change of some kind. Again, just about any computation that projects the sequelae of the current state of affairs is easy to exploit.

In addition to its heuristic evaluator, a planner must also have a search strategy. Optop uses best-first search as long as its heuristic is sharply differentiating among alternative plan prefixes. When too many accumulate that seem to be of about the same quality, it switches to a strategy of “hill climbing with random restarts.” In this mode, it always extends the plan prefix with the action that looks the best locally; that is, if it has to choose among actions it with  $A_1, \dots, A_k$ , it picks an  $A_i$  that dominates  $A_1, \dots, A_n$ , without regard to previously generated possibilities. If it reaches a point where there is no feasible action that leads to a new situation, it makes a random choice among all the plan prefixes it has generated and resumes hill climbing from there.

## Changes for the Competition

To illustrate how easily changes are made to Optop, here’s an account of recent changes to the system.<sup>2</sup>

The ability to handle universally-quantified preconditions was added to Optop for this year’s IPC. An ordinary precondition set such as  $(\text{and } (Q \text{ ?y}) (P \text{ a ?y}))$

<sup>2</sup>Optop is written in Lisp; I can’t imagine how it could evolve so quickly if it were written in any other language.

is handled during maxmatching by finding values for  $?y$  that make either  $(Q \text{ ?y})$  or  $(P \text{ a ?y})$  true. The other precondition, with  $?y$  substituted away, becomes a *difference* to be reduced. Now suppose we have preconditions  $(\text{and } (Q \text{ ?y}) (\text{forall } (z) (\text{if } (R \text{ ?y}) (P \text{ ?y } z))))$ . Suppose  $?y=b$  make  $(Q \text{ ?y})$  true. Then the remaining differences are all the literals whose unprovability produces a counterexample to the universal. A counterexample is an instance of  $(\text{and } (R \text{ } z) (\text{not } (P \text{ b } z)))$ , which can be produced by finding  $z$ ’s such that  $(R \text{ } z)$  is provable and  $(P \text{ b } z)$  is not; each such  $(P \text{ b } z)$  becomes a difference. Writing and plugging in the code for this mechanism was a relatively simple task.

Note that the maxmatcher must find values for  $?y$  before considering the universal. That’s because there is no way to enumerate all the values  $y$  that make  $(\text{forall } (z) (\text{if } (R \text{ } y) (P \text{ } y \text{ } z)))$  provable, or all those that make it unprovable. (Provability is used as a stand-in for truth, because PDDL relies on a closed-world assumption: if a proposition can’t be proved, it is taken to be false.) The deductive system built-in to Optop distinguishes between queries with no answers and queries with an unknown number of answers that might be handled if more of their free variables are bound. This turns out to be a very useful feature with a variety of uses, one of which is to decide how to order preconditions during maximal matching.

For the competition, PDDL was extended in two further ways: with derived predicates and timed initial literals. Optop already had derived predicates, which it used in the following way: Suppose, in the previous example, there was an axiom<sup>3</sup>

```
(forall (x)
  (<- (Q ?y)
    (exists (v)
      (and (R v ?y) (R ?y v)))))
```

The existence of this axiom gives the maxmatcher an extra degree of freedom. Instead of having to classify  $(Q \text{ a})$  as a difference, it can also find a  $v$  such that  $(R \text{ v a})$  and make  $(R \text{ a } v)$  a difference. The term *derived predicate* is just another name for a predicate defined by a single backward-chaining axiom.

Unfortunately, expanding axioms this way is not a good idea unless the axioms are *stratified*, meaning that there is no path from a predicate to itself through the axioms in question. To handle those correctly, we have to cope with the recursion by moving it out to the level of the regression-match graph. That is, an unstratified axiom gives rise to a different kind of regression method, in which the conditions lead immediately to a conclusion with no action or process intervening. For example, the unstratified axiom

```
(forall (x y)
  (<- (above ?x ?y)
    (exists (w)
```

<sup>3</sup>The “<-” indicates that the implication is to be used for backward chaining.

(and (above ?x w)  
(above w ?y))))))

can be used to reduce an L-node (above a e) to (and (above a ?w) (above ?w e)), which, after maxmatching, yields subgoal nodes such as (above b e) (if (above a b) is true in the current situation). An L-node can easily occur as a sub-sub-...-node of itself, but such cycles are simply ignored when the regression-match graph is used to produce and evaluate extensions of the current plan prefix.

## Performance

As shown in (McDermott 1999), although Optop spends more time per search state than other planners, in some domains it explores so few states that its run times are comparable to highly optimized systems. On “well-behaved” domains, its run times grow polynomially with problem size.

There is a price to be paid for Optop’s flexibility. The relaxed search space embodied in the regression-match graph neglects destructive interactions among actions (Bonet, Loerincs, & Geffner 1997; Bonet & Geffner 2001). This neglect makes it difficult to solve problems in domains in which a crucial condition can be irreversibly deleted without that being discovered until several more actions have been added to the plan. (The classic example is the “Rockets” domain of (Blum & Furst 1995).) On the other hand, realistic domains are often more forgiving, and allow problems to be broken into loosely coupled subproblems that can be solved by the sort of hill climbing described above.

## Future Plans

My current research goal is to add hierarchical and contingency planning to Optop. The former requires augmenting search states with information about hierarchical plans (i.e., canned plans from a library) that are in progress. With this addition, the regression-match graph will be built to handle posted but unsatisfied goals from the current hierarchical plan — the *script*. An action that is already in the script will not normally be proposed, unless a new instance is needed in order to achieve a precondition of some other step.

Contingency planning is mainly a matter of running the planner for various alternative scenarios. The mechanics are easy; the hard part is deciding when to stop exploring contingencies.

## References

- Blum, A. L., and Furst, M. L. 1995. Fast planning through planning graph analysis. In *Proc. Ijcai*, volume 14, 1636–1642.
- Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 1–2 90:279–298.
- Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1-2).

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A fast and robust action selection mechanism for planning. In *Proc. AAAI-97*.

Ernst, G. W., and Newell, A. 1969. *GPS: A Case Study in Generality and Problem Solving*. Academic Press.

McDermott, D. 1996. A Heuristic Estimator for Means-ends Analysis in Planning. In *Proc. International Conference on AI Planning Systems*, 142–149.

McDermott, D. 1999. Using Regression-match Graphs to Control Search in Planning. *Artificial Intelligence* 109(1–2):111–159.

McDermott, D. 2003. Reasoning about autonomous processes in an estimated-regression planner. In *Proc. Int’l Conf. on Automated Planning and Scheduling*.

Pednault, E. P. D. 1989. Adl: Exploring the middle ground between Strips and the situation calculus. In *Proc. Conf. on Knowledge Representation and Reasoning*, volume 1, 324–332.

Siskind, J. M., and McAllester, D. A. 1993. Non-deterministic Lisp as a substrate for constraint logic programming. In *Proc. AAAI 1993*, 133–138.

Weld, D. 1994. An introduction to least-commitment planning. *AI Magazine*.