Flaw Selection Strategies for Value-Directed Planning

Mike Williamson
The Robotics Institute
Carnegie Mellon University
mikew@ri.cmu.edu

Steve Hanks
Department of Computer Science and Engineering
University of Washington
hanks@cs.washington.edu

Abstract

A central issue faced by partial-order, causal-link (POCL) planning systems is how to select which flaw to resolve when generating the refinements of a partial plan. Domain-independent flaw selection strategies have been discussed extensively in the recent literature (Peot & Smith 1993; Joslin & Pollack 1994; Schubert & Gerevini 1995).

The PYRRHUS planning system is a decision-theoretic extension to POCL planners that finds optimal plans for a class of goal-directed value functions. Although PYRRHUS uses a branch-and-bound algorithm instead of best-first satisficing search, it is faced with the same flaw selection decision as other POCL planners. This paper explains why popular domain-independent flaw-selection strategies are ineffective within an optimizing framework, and presents two new strategies that exploit the additional value information available to PYRRHUS.

Introduction

Partial-order causal-link (POCL) planning algorithms like SNLP(McAllester & Rosenblitt 1991) and UCPOP(Penberthy & Weld 1992; Barrett *et al.* 1993) are appreciated for their clarity and formal properties, but their use has been limited by performance problems. The heuristic effectiveness of these algorithms depends on two choices made at each stage of the planning process: which partial plan to refine next, and which flaw in that partial plan to repair. Though the latter choice is formally irrelevant (a solution plan can be found regardless of the order in which flaws are addressed), it turns out to be crucially important in practice.

The choice of a flaw determines the offspring of the current partial plan, and so defines the space the planner must search. Experience has shown that the flaw selection strategy can have a dramatic impact on the performance of POCL planning algorithms, and a growing body of work is devoted to finding effective domain-independent techniques for flaw selection (Peot & Smith 1993; Joslin & Pollack 1994; Schubert & Gerevini 1995; Srinivasan & Howe 1995).

Another body of planning research devotes itself to extending the expressive power of the planning algorithms, extending their applicability to broader classes of problems. SNLP, for example, has given rise to a family of more powerful planning systems including UCPOP, CNLP(Peot & Smith 1992), BURIDAN(Kushmerick, Hanks, & Weld 1994; Draper, Hanks, & Weld 1994), Knoblock's parallel execution variant of UCPOP(Knoblock 1994), ZENO(Penberthy & Weld 1994), and Descartes(Joslin & Pollack 1995).

The PYRRHUS planning system (Williamson & Hanks 1994) is a decision-theoretic extension to SNLP. PYRRHUS extends the SNLP action representation to support metric resources (including time), and generates plans that are optimal according to a class of goal-directed value functions. Although PYRRHUS uses branch-and-bound optimization rather than satisficing best-first search, the underlying space of partial plans that it explores is very similar to the space generated by a classical POCL planner, especially in that both must address the plan-ranking and flaw-selection decisions.

This paper investigates whether the flaw selection strategies developed for other POCL planners (typically UCPOP) can be used effectively by PYRRHUS's optimization algorithm. The following next briefly describes PYRRHUS in contrast to other POCL planning systems. Section three describes several flaw selection strategies found in the literature, and makes explicit the aims of and assumptions underlying these strategies. We also point out the significance of the "LIFO" flaw-ordering policy adopted implicitly or explicitly by some of the existing approaches. In section four we discuss how the aims and assumptions underlying flaw selection in an optimizing planner differ from the classical case. We suggest two new flaw selection strategies that exploit additional information provided by the value function. Section five evaluates the effectiveness of the new and existing strategies in guiding the generation of optimal plans.

Value-Directed Causal-Link Planning

Figure 1 shows an abstract version of a classical

Algorithm POCL (*init*, *goal*)
Let *Horizon* = { MakeInitialPlan(*init*, *goal*) }
While *Horizon* is not empty do:
Choose a partial plan P from *Horizon*.
If P has no flaws then
Return P as a solution.
Else
Choose a flaw F from the partial plan P.
Add Refinements(P, F) to *Horizon*.
Return failure.

Figure 1: An abstract goal-satisfying planning algorithm

goal-satisfying POCL planning algorithm.[1] The function *MakeInitialPlan* returns a partial plan with two dummy steps encoding the goal and initial state. The function *Refinements* takes a partial plan and a flaw, and returns a set of new partial plans which represent all possible ways of fixing the given flaw. A partial plan can have two kind of flaws:

- *Open conditions* (a.k.a. "goals" or "sub-goals") are preconditions of steps in the plan which the planner has not yet made a commitment to achieving. They are repaired by adding a causal link from some new or existing step. When a new step is added to the plan, its preconditions become new open conditions.
- Threats (a.k.a. "unsafe links") are possible violations of causal links by other steps in the plan. They are repaired by adding step-ordering or variable-binding constraints.

Implementations of this abstract algorithm differ in the way they make the choices of which partial plan to refine and which flaw to repair. The plan to refine is typically chosen best-first according to a weak, domain-independent heuristic measure such as the sum of the number of steps and the number of flaws in the partial plan.[2] The choice of flaw to repair will be discussed in detail below.

The PYRRHUS planning algorithm

Figure 2 gives an abstract description of PYRRHUS.[3] The primary difference between PYRRHUS and a classical planning algorithm is in the formulation of the problem itself. Instead of just being given a formula

Algorithm PYRRHUS (*init*, $\mathcal{V}$)
Let *incumbent* = *status quo* (i.e. do nothing).
Let *Horizon* = { MakeInitialPlan(*init*, $\mathcal{V}$) }
While *Horizon* is not empty do:
Choose a partial plan P from *Horizon*.
If $UB(\mathcal{V}(P)) \leq \mathcal{V}(incumbent)$ then discard P.
Else
If P has no flaws and
$LB(\mathcal{V}(P)) > \mathcal{V}(incumbent)$ then
Let *incumbent* = P.
Else **Choose** a flaw F of the plan P.
Add Refinements(P, F) to *Horizon*.
Return *incumbent*.

Figure 2: An abstract description of the Pyrrhus planning algorithm

describing a goal to achieve, PYRRHUS is given a *goal-directed value function*[4] (Haddawy & Hanks 1993). This value function extends the traditional conception of a goal to include additional information like temporal deadlines, preferences over situations in which the goal is only partially satisfied, and the value associated with satisfying the goal and the cost of the resources required to do so.

PYRRHUS finds an optimal plan with respect to the value function by making tradeoffs between the importance of achieving the goal and the cost of resources consumed in doing so. The value function provides a rich, semantically grounded measure of plan *quality*. Classical planning typically either makes the assumption that all plans satistying the goal are equipreferable, or uses an *ad hoc* and perhaps implicit measure of quality such as the number of steps in the plan. But PYRRHUS is able to draw finer and more meaningful distinctions between plans. In a delivery domain, for example, the value function might lead PYRRHUS to produce a plan which makes a delivery 20 minutes late instead of one which makes the delivery on time but uses more fuel to do so. In some cases, the optimal course of action might be to do nothing at all (maintain the *status quo*), and PYRRHUS can often determine this very quickly.

PYRRHUS can use the value function to construct the initial plan for a plan space including all plans possibly preferable to the *status quo*. But the most important feature of the value function is that it allows the calculation of bounds on the value of a partial plan. If a partial plan has no flaws (is complete) its precise value can be calculated. An incomplete partial plan can be viewed as a compact representation of the (possibly infinite) set of all complete plans that can be

[1] The description of POCL planning herein is necessarily sketchy, omitting precise definitions of partial plans, causal links, threats, etc. The reader is referred to (Weld 1994) for a general introduction to the topic, or (Kambhampati, Knoblock, & Yang 1994) for a detailed discussion of the many possible variations on this basic algorithm.

[2] See (Schubert & Gerevini 1995) for a discussion of some plan ranking heuristics of this kind.

[3] Again a cursory description. For more details see (Williamson & Hanks 1994).

[4] "Value functions" are different from "utility functions" because they directly encode preferences over outcomes rather than preferences over lotteries over outcomes (see (Keeney & Raiffa 1976)). PYRRHUS makes the classical planning assumptions about certainty.

obtained by further refinement, and therefore the value of a partial plan can be represented by an interval containing the values of all possible completions of that plan. In general there will be no lower bound on the value of an incomplete plan, since subsequent refinements might make the plan arbitrarily bad. An upper bound on value can be calculated by considering the resources consumed by the plan so far and the greatest possible degree to which the plan might satisfy the goal. Refinement of a partial plan partitions its possible completions among the offspring, so each offspring must have an upper bound on value no greater than its parent.

PYRRHUS's branch-and-bound optimization algorithm is much like the refinement search procedure used by POCL planners. They both explore a space of partial plans with similar structure.[5] PYRRHUS likewise uses a best-first strategy for selecting plans to refine, but the ranking is based on value rather than simple structural characteristics. Most notably, both are faced with the task of choosing which flaw from a given partial plan to use to refine that plan. The similarity between the choices faced by the two algorithms leads us to hope that flaw selection strategies developed for classical POCL planners might be useful for PYRRHUS as well.

Existing Flaw Selection Strategies

In this section we describe a number of flaw selection strategies found in the literature, formalize the aim of these strategies, and expose certain underlying assumptions. We also discuss *precondition-order sensitivity*, an important aspect of flaw selection strategies that has received little attention.

T/O-LIFO The published SNLP and UCPOP algorithms resolve all threats before working on any open conditions. Formally, the algorithms were described nondeterministically and so were mute on the issue of which particular threat or open condition to resolve. In practice, the most widespread implementations of these algorithms process both threats and open conditions in a LIFO manner. The LIFO aspect of this strategy can have significant performance implications (see below), although we suspect this strategy was originally adopted more by accident than by design. (This strategy has been referred to as "SNLP" and "UCPOP" in the literature. We call it "T/O-LIFO" to avoid confusion with other aspects of those algorithms, and to make the LIFO policy explicit.)

DUnf Peot and Smith (Peot & Smith 1993) examine the question of whether threats should always be resolved before open conditions, both from a theoretical and an experimental standpoint. They consider various strategies for "delaying" certain threats, i.e. choosing to resolve any existing open conditions before the postponed threats. One promising strategy they develop is that of delaying "unforced" threats, that is, those for which refinement leads to more than one offspring.

With respect to open condition selection, they indicate that in their experiments they considered three different strategies: LIFO, FIFO, and a "least-commitment" approach that selects open conditions in order to minimize the number of offspring. However, they do not provide detailed analysis of how these various open condition selection strategies affect performance. We consider two of these variations on the "delay unforced threats" strategy: DUnf-LIFO chooses open conditions in LIFO order, while DUnf-LCOS chooses open conditions to achieve least commitment.

LCFR Joslin and Pollack (Joslin & Pollack 1994) offer a closer examination and generalization of Peot and Smith's least-commitment strategy. Suggesting that the benefit of DUnf derives from minimizing the branching factor of the search space, they propose *always* choosing a flaw that minimizes the number of offspring, regardless of the threat/open condition distinction. They call this strategy "least-cost flaw repair" (LCFR), and compare it experimentally to both Dunf-LIFO and Dunf-LCOS as well as the original T/O-LIFO strategy. They found that LCFR and Dunf-LCOS performed much better than the other strategies on many problems in a variety of domains.

ZLIFO Schubert and Gerevini (Schubert & Gerevini 1995) consider the question of open condition selection, and propose a hybrid strategy. Their technique is to first choose those open conditions which give rise to either zero or one offspring, which can be considered to be "zero-commitment" or deductive decisions. If there are no zero-commitment open conditions, then choose from the remainder in LIFO order. They demonstrate on problems from several domains that this ZLIFO strategy is generally superior to T/O-LIFO.

Their discussion of other strategies is limited, but they compare ZLIFO to LCFR on a few problems. They report that the strategies perform comparably on the easier problems, but that on the harder problems (from the Towers of Hanoi domain), ZLIFO's performance was considerably superior to LCFR.

Flaw selection: aims and assumptions

Why does flaw selection play such an important role in planning efficiency? A common belief is that judicious flaw selection improves planner performance by reducing the *size of the search space*. But this isn't quite right; in most cases, the potential search space is infinitely large, regardless of the flaw selection strategy

[5] In PYRRHUS the definition of open conditions and causal links has been generalized to support metric resources like fuel level as well as logical propositions, but that difference is not relevant here.

employed. Rather, judicious flaw selection improves performance by reducing the *size of the space searched*. This depends not only on the flaw selection strategy, but also on the way the algorithm chooses partial plans to refine.

Obviously the ultimate aim of a flaw selection policy is to minimize the expected future planning effort required to find a solution plan. In POCL planners, the amount of effort is directly related to the number of partial plans that must be generated and evaluated before a solution is found. The flaw selection policy determines both the minimum depth (number of refinements) at which a solution plan can be found as well as the total number of partial plans (nodes in the search tree) up to that depth. If this tree were searched breadth-first by number of refinements, all the nodes at the solution depth or above would have to be searched. Lacking the ability to predict how flaw choice affects solution depth, a reasonable heuristic for minimizing the total number of nodes would be to favor the flaw with the least number of direct offspring, thus minimizing the tree's immediate branching factor.

In fact, the best-first search used by many POCL planners can be viewed as "approximately" breadth-first in that the standard ranking functions simply tend to favor plans with fewer commitments, thus explaining the effectiveness of fewest-offspring strategies such as ZLIFO and LCFR.

Precondition-order sensitivity

In light of the above analysis, we were puzzled by the reported dramatic superiority of of ZLIFO to LCFR. Since the two heuristics agree on which flaw to select when the branching factor is less than two, the improvement must derive from cases where the "LIFO" part of ZLIFO selects a flaw with higher branching factor than LCFR would choose. Why is this LIFO flaw selection beneficial?

Regarding LIFO, Schubert and Gerevini suggest that,

> Based on experience with search processes in AI in general, such a strategy has much to recommend it, as a simple default. It will tend to maintain focus on the achievement of a particular higher-level goal by regression ... rather than attempting to achieve multiple goals in breadth first fashion. (Schubert & Gerevini 1995, p. 9)

While this comment about the regression-oriented character of LIFO search strategies is undoubtedly correct, it misses the mark with respect to POCL planning. The true value of LIFO flaw selection[6] is that it allows the introduction of *domain-specific* flaw selection policies into the planning process. The order in which new open conditions are resolved will depend not just on the order that steps are added to the plan, but far more significantly, on the order in which preconditions are specified in the operator description. This *precondition-order sensitivity* (POS) can have an overwhelming effect on performance.

To demonstrate the precondition-order sensitivity of ZLIFO, we created six versions of the single-operator Towers of Hanoi domain (T-of-H1) used by Schubert and Gerevini. These domains differ only in the permutation of the three main preconditions of the `move-disk` operator. Running UCPOP with ZLIFO[7] on the three disk problem in each of these six domains resulted in greatly varying numbers of plans explored, as shown in Table 1.

This experiment clearly shows that the performance of ZLIFO (like any other strategy which employs LIFO flaw selection) depends crucially on the ordering of preconditions in the operator descriptions. The strong performance of ZLIFO (in that domain formulation where its performance *is* strong) is more properly ascribed to the domain-specific flaw selection knowledge encoded in the operator descriptions than to ZLIFO itself. It would be unfair to compare such results to the performance of a flaw selection strategy that did not exhibit such precondition-order sensitivity.

We argue that precondition-order sensitivity is a serious shortcoming of the LIFO-based flaw selection strategies. It severely undermines the process of engineering domain knowledge, since the putatively declarative domain descriptions can have such an impact on tractability. If one *does* believe that search control information should be encoded in the domain description, then it should be done through explicit and well-defined mechanisms such as filter preconditions or other condition types (Currie & Tate 1991; Collins & Pryor 1992; Tate, Drabble, & Dalton 1994).

Eliminating precondition-order sensitivity can be difficult, though, since many strategies (including LCFR) underspecify the choice of flaw. In practice the decision must somehow be made, but the performance of most naive deterministic methods like LIFO will be sensitive to the order in which operator preconditions are declared.

Value-directed Flaw Selection

We now turn to the problem of flaw selection in the context of PYRRHUS's optimizing planning algorithm. As in goal-satisfying planning the overall aim of flaw selection is to minimize the amount of search required to find a solution, but in the optimization case the means to this end are different. Whereas the goal-satisfying algorithm can stop when it finds a successful plan, PYRRHUS must at least implicitly consider the entire search space. On the other hand, the goal-satisfying algorithm cannot eliminate a partial plan from consideration until it is *proven* to be *infeasible*, but the optimizing algorithm can eliminate a partial

[6] At least with respect to the widely available UCPOP implementation.

[7] We also used the S+OC plan selection heuristic.

Order of preconditions	Nodes explored	Order of preconditions	Nodes explored
(on ?disk ?below-disk) (clear ?disk) (clear ?new-below-disk)	186	(clear ?new-below-disk) (on ?disk ?below-disk) (clear ?disk)	29046
(on ?disk ?below-disk) (clear ?new-below-disk) (clear ?disk)	1834	(clear ?disk) (clear ?new-below-disk) (on ?disk ?below-disk)	100000+
(clear ?disk) (on ?disk ?below-disk) (clear ?new-below-disk)	7816	(clear ?new-below-disk) (clear ?disk) (on ?disk ?below-disk)	100000+

Table 1: Planning difficulty for ZLIFO on variations of the T-of-H1 domain.

plan as soon as the upper bound on its value is lower than the value of the incumbent (see Figure 2).

Thus, PYRRHUS's effectiveness is improved by increasing the rate at which partial plans are pruned, which can be accomplished in two ways:

1. *Quickly find a better incumbent.* This corresponds roughly to goal-satisfying planning, and is achieved by trying to reduce the size of the search space leading to a complete feasible plan. We might thus expect existing strategies such as LCFR to be effective in this regard.
2. *Generate partial plans with tighter upper bounds.* The choice of flaw can improve the quality of value bounds over the offspring, allowing them to be pruned earlier. This approach has no real analogue in classical planning.

The problem is that these two approaches tend to conflict. Often, better bounds on the value of a partial plan are obtained by making exactly the kinds of commitments that a strategy such as LCFR tries to avoid.

As an illustration of this point, consider the simplified `drive` operator from the Truckworld (Hanks, Pollack, & Cohen 1993) domain (Figure 3). The amount of fuel consumed by a `drive` action is determined by the function *fuel-consumed-function*, which is called during the planning process with the current bindings of the parameters of the operator. If any of the parameters are unbound (because of commitments not yet made in the planning process), then this function returns an interval providing upper and lower bounds on the change in fuel level.

Now, imagine that some partial plan contains the partially instantiated step `(drive ?from Paris ?speed)`. Consequently, the plan would also contain at least the two open conditions `(road-connecting ?from Paris)` and `(speed ?speed)`. These could both be closed by making links to the initial state. Resolving the former flaw might cause `?from` to be bound to one of `{Berlin, Vienna, Rome, Madrid, Moscow}`, while resolving the latter would cause `?speed` to be bound to one of `{Slow, Fast}`. LCFR would prefer to resolve the latter flaw, since it generates fewer offspring. But from the standpoint of branch-and-bound optimization, resolving the former flaw is much more important. Knowing where you are driving from allows calculation of a much tighter bound on the amount of fuel consumed, and consequently a better upper bound on the value of each resulting partial plan. In short, the choice of flaw affects not only the branching factor of the search space, but also the amount of knowledge one has about the value of a partial plan.

Strategies for improving value bounds

We would like to develop flaw selection strategies that generate partial plans with better upper bounds on value. But it's not clear what "better" means, since resolving a flaw can result in many offspring, each with its own upper bound on value. Consider Figure 4 which shows a partial plan P1 with two flaws F1 and F2, each giving rise to two offspring. The numbers are the upper bounds on value for each partial plan.

Which flaw will lead to the most future pruning if it is repaired next? We start only with the supposition that the value of the incumbent will increase as the refinement process proceeds and better complete plans are found. One intuition suggests that flaw F1 would be preferable: since the bound on its highest offspring is lower than the bound on F2's highest offspring, all of F1's offspring could be pruned earlier. But another line of reasoning suggests F2: although the offspring with a bound of 0.7 will be pruned later than either of the offspring generated by F1, the other offspring (with a bound of 0.2) will be pruned much earlier. Since deciding between these two alternatives would require additional (and unavailable) information about how the incumbent's bound might improve, we developed the following two flaw selection strategies and compared them empirically.

LUBV If one advantage to refining a partial plan is that doing so can improve upper bounds on the values of the offspring, then an obvious heuristic to consider is choosing the flaw that gives the *most* such im-

```
(operator (drive ?from ?to ?speed)
    :precondition (:and (truck-at ?from)
                        (road-connecting ?from ?to)
                        (speed ?speed))
    :duration duration-function
    :resources ((fuel-level :precondition fuel-required-function
                            :change fuel-consumed-function))
    :effect (:and (:not (truck-at ?from))
                  (truck-at ?to)))
```

Figure 3: The **drive** operator from the Truckworld domain.

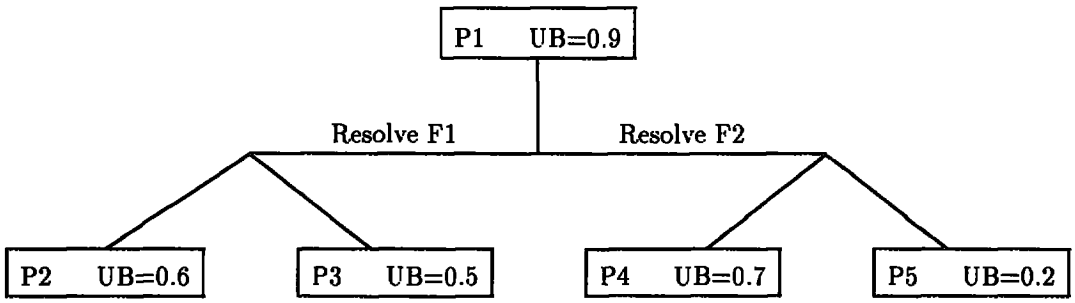

Figure 4: The effect of flaw choice on upper bounds on value

provement. The "least upper bound on value" (LUBV) strategy chooses a flaw such that the maximum upper bound on value over its offspring is minimized. This stragegy would choose flaw F1 in the above example. One interpretation of this strategy is that it favors the flaw whose repair gives the greatest increase in knowledge of the value of the partial plan.

SUBV LCFR and LUBV can be seen as two extremes. While the former tries only to minimize branching factor, the latter tries only to maximize improvement of value bounds. The "sum of upper bounds on value" (SUBV) strategy strikes a balance between these two extremes, choosing the flaw that minimizes the *sum* of the upper bounds on value over the plan's offspring. Whereas LUBV considers only the single offspring with maximum value bound, and LCFR considers only the number of offspring, SUBV can be interpreted as a count of the number of offspring weighted by their value bounds. SUBV would favor flaw F2 in the above example.

Empirical evaluation

We implemented the five existing and two new flaw selection strategies described above in PYRRHUS and evaluated them on a collection of problems drawn from a transportation planning domain inspired by the Truckworld simulator(Hanks, Pollack, & Cohen 1993). These problems involved delivering cargo between multiple origins and destinations with varying partially-satisfiable (soft) deadlines. There were four operators in the domain, and the world consisted of six locations interconnected by seven roads. The planner had to reason about where fuel was available, which routes to take, how fast to drive, and even which deliveries were worth making. Tradeoffs existed between the amount of fuel consumed, the amount of money spent (on toll roads), and the degree to which the deadlines were met. Solution plans varied in length from no steps (in cases where the *status quo* was optimal), to 8 steps.

Our first experiment tested the existing strategies on some classical goal-satisfying problems in this domain.[8] As researchers have found in other domains, LCFR, ZLIFO, and DUnf-LCOS all strongly outperformed T/O-LIFO, with LCFR providing the best overall performance. This experiment suggested that LCFR is a good general heuristic for goal-satisfying planning in this domain.

Our main experiment was to evaluate all seven strategies on a suite of 20 optimization problems. A search limit of 10000 partial plans was imposed. Figure 5 shows the number of partial plans explored by five of the flaw selection strategies on each problem.[9] The problems are shown ordered roughly according to

[8] PYRRHUS can perform both goal-satisfying and value-optimizing planning. In the former role it functions much like UCPOP, but with extensions to handle time and metric resources.

[9] The performance of DUnf-LCOS was uniformly very close to LCFR, and the performance of DUnf-LIFO was similarly close to T/O-LIFO. The two DUnf strategies are therefore omitted.

difficulty for the SUBV strategy. Note that the horizontal axis of the graph contains 20 discrete problems rather than a continuous progression; the lines in the graph are only an aid to visualization.

The strategies that worked hardest to reduce commitment (i.e. minimize branching factor), LCFR and DUnf-LCOS, performed the worst on most problems. The strongly LIFO oriented strategies, T/O-LIFO, ZLIFO, and DUnf-LIFO exhibited better performance on most problems, but as described above, this is due to flaw selection knowledge encoded in the operator descriptions (which had been hand-tuned for acceptable performance under T/O-LIFO during their development). The best overall performance was obtained by SUBV. By explicitly working both to minimize the branching factor and to maximize improvement of the bounds on partial plan value, SUBV dominated all other strategies.

Some final experimentation was performed to evaluate the precondition-order sensitivity of the SUBV strategy, since it uses LIFO to break (infrequent) ties between flaws. We developed an alternative version, SUBV-*R*, which chooses randomly between tied flaws and so completely avoids precondition-order sensitivity. SUBV-*R* was used to solve each problem ten times. It exhibited little variance between runs on each problem. In no case was the performance of SUBV significantly different from the mean performance of SUBV-*R*, thus indicating that SUBV is deriving no benefit from precondition ordering.

Absolute performance In practice, the benefit of more sophisticated flaw selection strategies is diminished by the additional computation required in selecting flaws. In our experiments SUBV required about 40 times as much CPU time per node as T/O-LIFO.[10] This overhead is related linearly to the average number of flaws per plan. But as Peot and Smith point out, on harder problems this cost will be dominated by the reduction in the number of plans explored. Our experiments reinforce that conclusion, with SUBV effectively solving problems that were beyond the reach of other strategies. Joslin and Pollack present an approximation to their LCFR strategy which obtains comparable results at a much lower per node overhead. A similar approximation technique might be used for SUBV.

Conclusion

This paper has investigated the problem of flaw selection as it arises in PYRRHUS, an optimizing planner, in contrast to goal-satisfying POCL planning algorithms. We have described how the optimization algorithm places different demands on a flaw selection strategy. This analysis led to the development of SUBV, a powerful new flaw selection strategy for value-directed planning. We have also pointed out that the performance of many existing flaw-selection strategies is highly sensitive to the ordering of operator preconditions; the absence of a domain-independent theory of how to order these preconditions compromises the claim that these strageties are in fact domain independent.

[10] Which is approximately the same computational overhead noted by Joslin and Pollack (Joslin & Pollack 1994) for LCFR.

Acknowledgements

This work was funded in part by a NASA Graduate Research Fellowship and by NSF grant IRI-9008670. Many thanks to Dan Weld, Marc Friedman and Neal Lesh for constructive feedback on this work.

References

Barrett, A.; Golden, K.; Penberthy, J.; and Weld, D. 1993. UCPOP user's manual, (version 2.0). Technical Report 93-09-06, University of Washington, Department of Computer Science and Engineering. Available via FTP from `pub/ai/` at `ftp.cs.washington.edu`.

Collins, G., and Pryor, L. 1992. Achieving the functionality of filter conditions in a partial order planner. In *Proc. 10th Nat. Conf. on A.I.*

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

Draper, D.; Hanks, S.; and Weld, D. 1994. Probabilistic planning with information gathering and contingent execution. In *Proc. 2nd Intl. Conf. on A.I. Planning Systems.*

Haddawy, P., and Hanks, S. 1993. Utility Models for Goal-Directed Decision-Theoretic Planners. Technical Report 93–06–04, Univ. of Washington, Dept. of Computer Science and Engineering. Submitted to *Artificial Intelligence*. Available via FTP from `pub/ai/` at `ftp.cs.washington.edu`.

Hanks, S.; Pollack, M. E.; and Cohen, P. R. 1993. Benchmarks, testbeds, controlled experimentation, and the design of agent architectures. *AI Magazine* 14(4).

Joslin, D., and Pollack, M. 1994. Least-cost flaw repair: A plan refinement strategy for partial-order planning. In *Proc. 12th Nat. Conf. on A.I.*

Joslin, D., and Pollack, M. 1995. Passive and active decision postponement in plan generation. In *Proceedings of the Third European Workshop on Planning.*

Kambhampati, S.; Knoblock, C.; and Yang, Q. 1994. Planning as refinement search: A unified framework for evaluating design tradeoffs in partial order planning. Department of Computer Science and Engineering TR-94-002, Arizona State University. To appear in Artificial Intelligence Special Issue on Planning and Scheduling.

Keeney, R. L., and Raiffa, H. 1976. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs.*

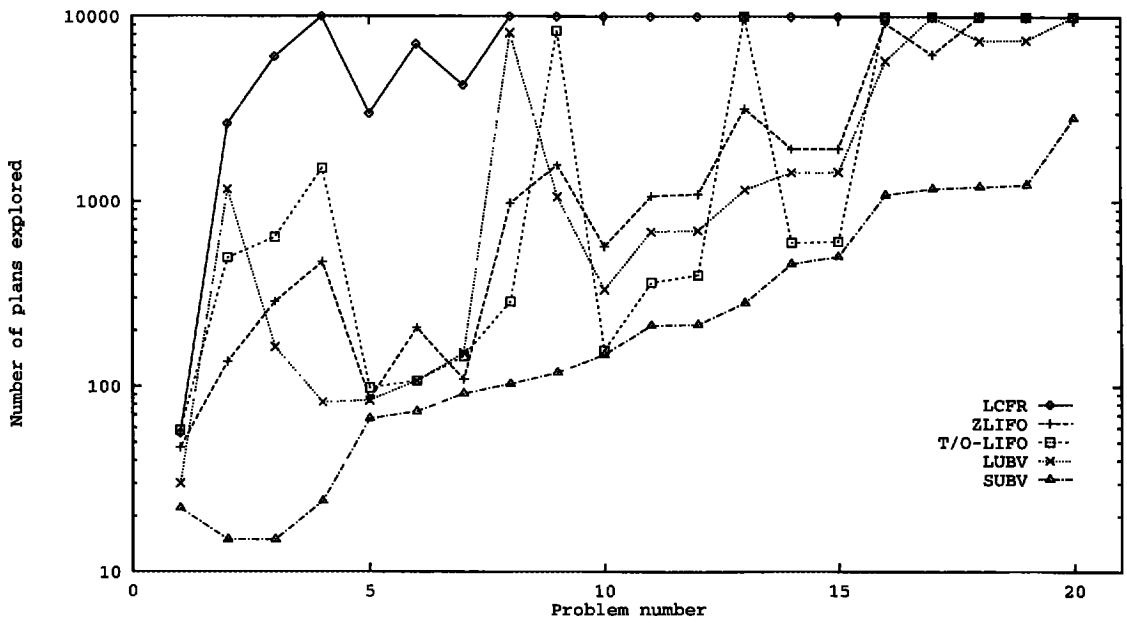

Figure 5: Performance of various strategies on optimization problems (search truncated at 10000 partial plans explored)

John Wiley and Sons. Republished in 1993 by Cambridge University Press.

Knoblock, C. 1994. Generating parallel execution plans with a partial-order planner. In *Proc. 2nd Intl. Conf. on A.I. Planning Systems*, 98–103.

Kushmerick, N.; Hanks, S.; and Weld, D. 1994. An Algorithm for Probabilistic Least-Commitment Planning. In *Proc. 12th Nat. Conf. on A.I.*

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proc. 9th Nat. Conf. on A.I.*, 634–639.

Penberthy, J., and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, 103–114. Available via FTP from `pub/ai/` at `ftp.cs.washington.edu`.

Penberthy, J., and Weld, D. 1994. Temporal planning with continuous change. In *Proc. 12th Nat. Conf. on A.I.*

Peot, M., and Smith, D. 1992. Conditional Nonlinear Planning. In *Proc. 1st Intl. Conf. on A.I. Planning Systems*, 189–197.

Peot, M., and Smith, D. 1993. Threat-removal strategies for partial-order planning. In *Proc. 11th Nat. Conf. on A.I.*, 492–499.

Schubert, L., and Gerevini, A. 1995. Accelerating partial order planners by improving plan and goal choices. Technical Report 570, University of Rochester, Dept. of Computer Science.

Srinivasan, R., and Howe, A. 1995. Comparison of methods for improving search efficiency in a partial-order planner. In *Proceedings of the 14th International Conference on AI.*

Tate, A.; Drabble, B.; and Dalton, J. 1994. The use of condition types to restrict search in an ai planner. In *Proceedings of the Twelfth National Conference on AI.*

Weld, D. 1994. An introduction to least-commitment planning. *AI Magazine* 27–61. Available via FTP from `pub/ai/` at `ftp.cs.washington.edu`.

Williamson, M., and Hanks, S. 1994. Optimal planning with a goal-directed utility model. In *Proc. 2nd Intl. Conf. on A.I. Planning Systems.*