Accelerating Partial Order Planners by Improving Plan and Goal Choices*

Lenhart Schubert
Dept. of Computer Science, University of Rochester
Rochester, NY 14627-0226, USA. Tel.: 716-275-8845
Email: schubert@cs.rochester.edu

Alfonso Gerevini
Dip. di Elettronica per l'Automazione, Università di Brescia
via Branze 38, 25123 Brescia, Italy
Email: gerevini@bsing.ing.unibs.it

Abstract
We describe some simple domain-independent improvements to plan-refinement strategies for well-founded partial order planning that promise to bring this style of planning closer to practicality. One suggestion concerns the strategy for selecting plans for refinement among the current (incomplete) candidate plans. We propose an A* heuristic that counts only steps and open conditions, while ignoring "unsafe conditions" (threats). A second suggestion concerns the strategy for selecting open conditions (goals) to be established next in a selected incomplete plan. Here we propose a variant of a strategy suggested by Peot & Smith and studied by Joslin & Pollack; the variant gives top priority to unmatchable open conditions (enabling the elimination of the plan), second-highest priority to goals that can only be achieved uniquely, and otherwise uses LIFO prioritization. The preference for uniquely achievable goals is a "zero-commitment" strategy in the sense that the corresponding plan refinements are a matter of deductive certainty, involving no guesswork. In experiments based on modifications of UCPOP, we have obtained improvements by factors ranging from 5 to more than 1000 for a variety of problems that are nontrivial for the unmodified version. Crucially, the hardest problems give the greatest improvements.

*Please address correspondence to the first author. This paper is an extended and revised version of a paper that appeared in the Proc. of the 7th Int. Conf. on Tools with Artificial Intelligence (ICTAI-95), Nov 5–8, Washington DC. The work of the first author was supported in part by Rome Lab contract F30602-91-C-0010, and both authors had partial support from NATO Collaborative Research Grant CRG951285. The work of the second author was partly carried out when he was working at IRST, 38050 Povo (TN), Italy. The helpful comments and perceptive questions of Marc Friedman, David Joslin, Rao Kambhampati, Colm O'Riain, Martha Pollack, and Mike Williamson are gratefully acknowledged.
1 Introduction

The history of planning research shows at least two major strands, whose respective goals are to achieve practical planning and well-founded planning. Practical planning research seeks to provide planning frameworks and tools that are sufficiently expressive, flexible and efficient to be effectively usable in applications such as planning robot actions, transportation planning, factory scheduling, genetic engineering and conversation planning. Some of the earliest practically motivated planning “formalisms” were the MICROPLANNER implementation of Hewitt’s PLANNER language [16, 30], STRIPS [10], and NOAH [28], and some familiar later examples are NONLIN [31], DEVISER [32], SIPE [33], PRS [11], FORBIN [9], and O-Plan [8]. While STRIPS was a linear planner, noteworthy characteristic of most later planners, starting with NOAH, was their ability to produce nonlinear (partially ordered) plans, when a partial ordering suffices to guarantee achievement of the goals.

The emphasis in well-founded planning is on constructing planners that can be proved to have certain desirable properties, such as soundness and completeness for their intended class of problems, or the ability to find optimal or near-optimal solutions. The first well-founded planner was probably C. Green’s QA3 [15], offering sound and complete linear planning within the expressively quite rich situation calculus. However, it was limited both by the linearity of the plans produced and by impractical performance, attributable in part to the use of frame axioms. These limitations provided some of the impetus behind the development of STRIPS and its descendants. The subsequent quest for more practical planners contributed many valuable ideas to planning theory and practice, but there remained a lingering dissatisfaction in the planning community with the lack of formal foundations and guarantees for the resultant planners (highlighted by troublesome problems such as the Sussman anomaly and the register exchange problem).

This led to a renewal of efforts in the 80’s to find viable approaches to well-founded planning. These efforts produced some novel approaches to linear planning such as BIGRESS [27, 21] (based on dynamic logic) and Bibel’s linear connection method for plan generation [5], as well as the first algorithms for well-founded nonlinear planning, TWEAK [6] (Chapman’s partial-order planner based on his “modal truth criterion”) and SNLP [23] (another systematic partial-order planner using propositional STRIPS operators). Since then there has been an upsurge of activity in well-founded partial-order planning, aimed both at increasing the expressiveness of such planners, and assessing and improving their performance. UCPOP [25] and BURIDAN [22] exemplify the move toward greater expressiveness, and some recent performance studies are reported in [4, 20].

Despite these efforts, it seems fair to say that well-founded planners still perform dismally in practical terms. For example, when we tried to apply the programs evaluated in [20] to the standard UCPOP suite of test problems, we found that none achieve reasonable performance on the 3-disk Towers of Hanoi
(T of H) puzzle (requiring 7 moves for its solution), or on some other simple problems. UCPOP did best on T of H but still took over 3 minutes of CPU time on a SUN 10, generating tens of thousands of partial plans. (This was with the “delay separation” switch on [26];1 with this switch off, performance was typically several times worse.) This is disappointing, since puzzles like T of H are easily solved by inexperienced people, with very little trial and error search; moreover, the very first well-founded planner, C. Green’s QA3, reportedly solved some (carefully formulated) versions of this problem rather easily [15]. It should be noted that such toy problems are not particularly outlandish from a practical perspective; for instance T of H and blocks world problems resemble problems that arise in such areas as connecting railroad cars into trains (with use of sidetracks) and pallet management in automated warehouses. (Some of the other problems in the test suite, such as Stuart Russell’s “tire domain”, are more directly evocative of real-world applications.)

Some recent discussions of partial-order planning strategies (e.g., some of those reported in [34]) could be interpreted as implying that the level of planning performance achieved so far is about the best that is possible for domain-independent planners; any real improvements from this point on will have to come from exploiting domain-specific information (for instance, in the form of higher-level, multi-step actions or alternatively, reactive rules, for particular domains and goals). Our outlook on well-founded, domain-independent planning is more optimistic. In the following, we suggest improved planning strategies based on the one hand on more carefully formulated heuristics for selecting plans for refinement, and on the other on “zero commitment” strategies for choosing subgoals. We describe these two classes of techniques in Sections 2 below, and in Section 3 we report our experimental results based on slightly modified versions of UCPOP. These results suggest that order-of-magnitude improvements in the performance of well-founded planners are possible, bringing them closer to practical usability.

2 Plan Selection and Goal Selection

2.1 UCPOP

We will be basing our discussion and experiments on UCPOP, an algorithm exemplifying the state of the art in well-founded partial-order planning. Thus we begin with a sketch of this algorithm, referring the reader to [3, 25] for details.

UCPOP uses STRIPS-like operators, with positive or negative preconditions and positive or negative effects. The initial state consists of positive predications with constant arguments (if any), and all other ground predications are false by default. Unlike STRIPS, UCPOP also allows conditional

---

1i.e., delaying the use of “promotion” and “demotion” to avert threats until all variables appearing in the conflict conditions are bound; and disabling altogether the use of inequations to block unification of threatening effects with threatened causal links
effects, expressed by 2-part WHEN-clauses specifying a (possibly complex) extra condition needed by that effect and the (possibly complex) effect itself. For instance, an action $\text{PUTON}(x, y, z)$ ("put $x$ on $y$ from $z$") might have conditional effects stating that when $y$ is not the table, it will not be clear at the end of the action, and when $z$ is not the table, it will be clear at the end of the action. The "U" in UCPOP indicates that universally quantified conditions and effects are permitted as well. For instance, it is permissible to have a precondition for a $\text{PICKUP}(x)$ action that says that for all $y$, (not (on $y$ $x$)) holds. Universal statements are handled by explicit substitution of domain constants and need not concern us here.

In essence, UCPOP explores a space of partially specified plans, each paired with an agenda of goals still to be satisfied and threats still to be averted. The initial plan contains a dummy *start* action whose effects are the given initial conditions, and a dummy *end* action whose preconditions are the given goals. Thus goals are uniformly viewed as action preconditions, and are uniformly achieved through the effects of actions, including the *start* action.

The plans themselves consist of a collection of steps (i.e., actions obtained by instantiating the available operators), along with a set of causal links, a set of binding constraints, and a set of ordering constraints. When an open goal (precondition) is selected from the agenda, it is established (if possible) either by adding a step with an effect that unifies with the goal, or by using an existing step with an effect that unifies with the goal. (In the latter case, it must be consistent with current ordering constraints to place the existing step before the goal, i.e., before the step whose preconditions generated the goal.) When a new or existing step is used to establish a goal in this way, there are several "side effects":

- A causal link $(S_p, Q, S_c)$ is also added, where $S_p$ indicates the step "producing" the goal condition $Q$ and $S_c$ indicates the step "consuming" $Q$. This causal link serves to protect the intended effect of the added (or reused) step from interference by other steps.
- Binding constraints are added, corresponding to the unifier for the action effect in question and the goal (precondition) it achieves.
- An ordering constraint is added, placing the step in question before the step whose precondition it achieves.
- If the action in question is new, its preconditions are added to the agenda as new goals (except that eq/neq conditions are integrated into the binding constraints – see below).
- New threats (called "unsafe conditions") are determined. For a new step and its causal link, other steps threaten the causal link if they have effects unifiable with the condition protected by the causal link (and these effects can occur temporarily during the causal link); and the effects of the new step may similarly threaten other causal links. In either case, new threats are placed on the agenda.
Binding constraints assert the identity (eq) or nonidentity (neq) of two variables or a variable and a constant. Eq-constraints arise from unifying open goals with action effects, and neq-constraints arise from neq-preconditions of newly instantiated actions and from matching negative goals containing variables to the initial state. (We set aside “separation” as a means of averting threats, which also leads to neq-constraints.) Neq-constraints may be disjunctive, but are handled simply by generating separate plans for each disjunct.

The overall control loop of UCPOP consists of selecting a plan from the current list of plans (initially the single plan based on *start* and *end*), selecting a goal or threat from its agenda, and replacing the plan by the corresponding refined plans. If the agenda item is a goal, the refined plans are those corresponding to all ways of establishing the goal using a new or existing step. If the agenda item is a threat to a causal link $(S_p, Q, S_c)$, then with the “delay separation” switch on there are at most three refined plans. Two of these constrain the threatening step to be before step $S_p$ (demotion) or after step $S_c$ (promotion), thus averting the threat. A third possibility arises if the effect threatening $(S_p, Q, S_c)$ is a conditional effect of the threatening action. Such a conditional threat can be averted by creating a goal denying some precondition needed by the conditional effect.

Inconsistencies in binding constraints and ordering constraints are detected when they first occur (as a result of adding a new constraint) and the corresponding plans are eliminated. Planning fails if no plans remain. The success condition is the creation of a plan with consistent binding and ordering constraints and an empty agenda.

The allowance for conditional effects and universal conditions and effects causes only minor perturbations in the operation of UCPOP. For instance, conditional effects can lead to multiple matches against operators for a given goal, each match generating different preconditions. (Of course, there can be multiple matches even without conditional effects, if some predicates occur more than once in the effects.)

The key issues for us are the strategic ones: how plans are selected from the current set of plans (discussed in section 2.2), and how goals are selected for a given plan (discussed in section 2.3).

### 2.2 The trouble with counting unsafe conditions

The choice of the next plan to refine in the UCPOP system is based on an A* best-first search. Recall that A* uses a heuristic estimate $f(p)$ of overall solution cost consisting of a part $g(p) =$ cost of the current partial solution (plan) $p$ and a part $h(p) =$ estimate of the additional cost of the best complete solution that extends $p$. In the current context it is helpful to think of $f(p)$ as a measure of plan complexity, i.e., “good” plans are simple (low-complexity) plans.

There are two points of which the reader should be reminded. First, in order for A* to guarantee discovery of an optimal plan (i.e., the “admissibl-
ity condition), $h(p)$ should not overestimate the remaining solution cost [24]. Second, if the aim is not necessarily to find an optimal solution but to find a satisfactory solution quickly, then $f(p)$ can be augmented to include a term that estimates the remaining cost of finding a solution. One common way of doing this is to add a term proportional to $h(p)$ for this as well, i.e., we “emphasize” the $h$-component of $f$ relative to the $g$-component. This is reasonable to the extent that the plans that are most nearly complete (indicated by a low $h$-value) are likely to take the least effort to complete. Thus we will prefer to pursue a plan $p'$ that seems closer to being complete to a plan $p$ further from completion, even though the overall complexity estimate for $p'$ may be greater than for $p$ [24] (pages 87–88). Alternatively, we could add a heuristic estimate of the remaining cost of finding a solution to $f(p)$ that is more or less independent of the estimate $h(p)$.

With these considerations in mind, we now evaluate the advisability of including the various terms in UCPOP’s function for guiding its $A^*$ search, namely

$$S, OC, CL, \text{ and } UC,$$

where $S$ is the number of steps in the partial plan, $OC$ is the number of open conditions (unsatisfied goals and preconditions), $CL$ is the number of causal links, and $UC$ is the number of unsafe conditions (the number of pairs of steps and causal links where the step threatens the causal link). The default combination used by UCPOP is $S + OC + UC$.

(a) Concerning $S$, the number of steps currently in the plan, this can naturally be viewed as comprising $g(p)$, the plan complexity so far. Intuitively, a plan is complex to the extent that it contains many steps. While in some domains we might want to make distinctions among the costs of different kinds of steps, a simple step count seems like a reasonable generic complexity measure.

(b) Concerning $OC$, the number of open conditions, this can be viewed as playing the role of $h(p)$, since each remaining open condition must be established by some step. The catch is that it may be possible to use existing steps in the plan (including *start*, i.e., the initial conditions) to establish remaining open conditions. Thus $OC$ can overestimate the number of steps still to be added, forfeiting admissibility.

Despite this criticism, several considerations favor retention of the $OC$ term. First, a better estimator of residual plan complexity seems hard to come by. Perhaps one could modify $OC$ by discounting open conditions that are matched by existing actions, but this presumes that all such open conditions can actually be achieved by action re-use, which is improbable if there are remaining threats, or remaining goals requiring new steps.\(^2\) Second, the possibility that $OC$ will overestimate the residual plan complexity will rarely be

---

\(^2\)Note that threats and remaining goals impose constraints that may not be consistent with seemingly possible instances of action re-use. This is clear enough for threats, which
actualized, since typically further steps still need to be added to achieve some of the goals, and those steps will typically introduce further open conditions again requiring new steps. Finally, to the extent that OC does at times overestimate the residual plan complexity, it can be viewed as emphasizing the the $h(p)$ term of $f(p)$, thus promoting faster problem-solving as explained above.

(c) Concerning CL, the number of causal links, one might motivate the inclusion of this term by arguing that numerous causal links are indicative of a complex plan. As such, CL appears to be an alternative to step-counting. However, note that CL is in general larger than S, since every step of a plan establishes at least one open condition and thus introduces at least one causal link. The larger CL is relative to S, the more subgoals are achieved by action re-use. Hence, if we use CL instead of (or in addition to) S in the $g(p)$ term, we would in effect be saying that achieving multiple subgoals with a single step is undesirable; we would tend to search for ways of achieving multiple goals with multiple steps, even when they can be achieved with a single step. This is clearly not a good idea, and justifies the exclusion of CL from $f(p)$.

(d) Concerning UC, the number of unsafe conditions, we note first of all that this is clearly not a $g$-measure. While the number of threats will tend to increase if we establish more and more subgoals without curtailing threats, threats as such are not elements of the plan being constructed and so do not contribute to its complexity. In fact, when the plan is done all threats will be gone.

Can UC then be viewed as an $h$-measure? One argument of sorts for the affirmative is the following. Not all partial plans are expandable into complete plans, and a high value of UC makes it more likely that the partial plan contains irresolvable conflicts. If we regard impossible plans as having infinite cost, then inclusion of a term increasing with UC as part of the $h$-measure is reasonable. This carries a serious risk, though, since in the case where the partial plan does have a consistent completion (despite a high UC-count), inclusion of such a term can greatly overestimate the residual plan complexity.

Another possible affirmative argument is that conditional threats are sometimes resolved by “confrontation”, which introduces a new goal denying a condition required for the threatening conditional effect. This new goal may in turn require new steps for its achievement, adding to the plan complexity. However, this link to complexity is very tenuous. In the first place, many of the UCPOP test domains involve no conditional effects, and threat removal often imply temporal ordering constraints inconsistent with re-use of an action. It is also fairly clear for remaining goals. For instance, in Towers of Hanoi the small disk D1 is initially on the medium disk D2, which in turn is on the big disk D3, and D3 is on peg P1. The goal is to move the tower to the third peg P3, so it seems to UCPOP initially as if (on D1 D2) and (on D2 D3) could be achieved by “re-use” of *start*. However, the 3rd goal (on D3 P3) implies that various actions must be added to the plan which are inconsistent with those two seemingly possible instances of action re-use.
by promotion, demotion or separation adds no steps. Even when conditional effects are present, many unconditional as well as conditional threats are averted by these methods.

Furthermore, UC could swamp all other terms since threats may appear and “expire” in groups of size $O(n)$. For instance, consider a partial plan that involves $n$ back-and-forth trips of a robot $R$ between locations $A$ and $B$, so that $n$ causal links are labeled $at(R, A)$. If some new action is now added with effect $\neg at(R, A)$, initially without temporal ordering relative to the $n$ trips, this will threaten all $n$ causal links labeled $at(R, A)$. If this new action subsequently happens to be demoted so as to precede the first trip (or promoted so as to follow the last), all $n$ threats expire. In the worst case there may be $O(n^2)$ unsafe conditions, destined to expire as a result of $O(n)$ promotions/demotions. Even more rapid expiration of threats may occur as a side effect of parameter binding. For instance, if there are $n/2$ effects $\neg P(x)$ threatening $n/2$ causal links labeled $P(y)$, then if $x$ becomes bound to $A$ and $y$ becomes bound to $B$, all $n^2/4$ threats expire. Note that when expired threats are selected from the agenda by UCPOP, they are recognized as such and discarded without further action.

Our conclusion is that it would be a mistake to include UC in full in a general $h$-measure, though some increasing function of UC that remains small enough not to mask OC may be worth including in $h$.

Finally, can UC be regarded as a measure of the remaining cost of finding a solution? Here, similar arguments to those above apply. On the affirmative side, we can argue that a high value of UC indicates that we may be facing a combinatorially explosive, time-consuming search for a set of promotions and demotions that produce a conflict-free step ordering. In other words, a high value of UC may indicate a high residual problem-solving cost. (And at the end of such a search, we may still lack a solution, if no viable step ordering exists.) On the other hand, we have already noted that unsafe conditions include many possible conflicts which may expire as a result of subsequent partial ordering choices and variable binding choices not specifically aimed at removing these conflicts. So counting unsafe conditions can arbitrarily overestimate the number of genuine refinement steps, and hence the problem-solving effort, still needed to complete the plan.

So UC is in general no more trustworthy as a measure of residual planning cost than as a measure of residual plan cost.

Thus we conclude that the most promising general heuristic measure for plan selection is $S+OC$, possibly augmented with an attenuated form of the UC term that will not dominate the $S+OC$ component. (For instance, one might add a small fraction of the term, such as $UC/10$, or more subtly – to avoid swamping by a quadratic component – a term proportional to $UC^5$, etc.)
2.3 The goal selection strategy

An important opportunity for improving planning performance independently of the domain lies in identifying “forced refinements”, i.e., refinements that can be made deterministically. Specifically, in considering possible refinements of a given partial plan, it makes sense to give top priority to open conditions that cannot be achieved; and then preferring open conditions that can only be achieved in one unique way – either through addition of an action not yet in the plan, or through a unique match against the initial conditions.

The argument for giving top priority to unachievable goals is just that a plan containing such goals can be eliminated at once. Thus we prevent allocation of effort to the refinement of doomed plans, and to the generation and refinement of their doomed successor plans.

The argument for preferring open conditions that can only be achieved uniquely is equally apparent. Since every open condition must eventually be established by some action, it follows that if this action is unique, it must be part of every possible completion of the partial plan under consideration. So, adding the action is a “zero-commitment” refinement, involving no choices or guesswork. At the same time, adding any refinement in general narrows down the search space by adding binding constraints and adding a causal link and further effects that can temporally constrain other threatening or threatened actions. For unique refinements this narrowing-down is monotonic, never needing revocation. In short, the strategy cuts down the search space without loss of access to viable solutions.

Peot and Smith [26] studied the strategy of preferring forced threats to unforced threats, and also suggested possible use of a “least commitment” strategy for handling open conditions. “Least commitment” always selects an open condition which generates the fewest refined plans. Thus it entails the priorities for unachievable and uniquely achievable goals above (while also entailing a certain prioritization of nonuniquely achievable goals). Joslin and Pollack [18] studied the uniform application of such a strategy to both threats and open conditions in UCPOP, terming this strategy “least cost flaw repair” (LCFR).3 Combining this with UCPOP’s default plan selection strategy, they obtained significant search reductions (though less significant running time reductions, mainly for implementation reasons, but also because of the intrinsic overhead of computing the “repair costs”) for a majority of the problems in the UCPOP test suite.

Joslin & Pollack [18] and subsequently Srinivasan & Howe [29] proposed some variants of LCFR designed to reduce the overhead incurred by LCFR for flaw selection. These strategies employ various assumptions about the flaw repair costs, allowing the more arduous forms of cost estimation (requiring look-ahead generation of plans) to be confined to a subset of the flaws in the plan, while for the rest an approximation is used that does not significantly increase the overhead. Both teams obtained quite significant reductions in

---

3 We would find “least commitment flaw repair” more accurate.
overhead costs in many cases, e.g., by factors ranging from about 3 to about 20 for the more difficult problems. However, overall performance was sometimes adversely affected. Joslin and Pollack found that their variant (QLCFR) solved fewer problems than LCFR, because of an increase in the number of plans generated in some cases. Each of Srinivasan & Howe’s four strategies did slightly better than LCFR in some of their 10 problem domains but significantly worse in others. In terms of plans examined during the search, their best overall strategy, which uses similar actions instances for similar flaws, did slightly better on 4 of the domains, slightly worse on 4, and significantly worse on 2 (and in those cases the number of plans examined was also more than a factor of 20 above that of default UCPOP).

In the unmodified form of UCPOP, goals are selected from the agenda according to a LIFO (last-in first-out, i.e., stack) discipline. 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—very much as in Prolog goal chaining—rather than attempting to achieve multiple goals in breadth-first fashion. We have therefore chosen to stay with UCPOP’s LIFO strategy whenever there are no “zero commitment” choices. This has led to very substantial improvements over LCFR in our experiments.

Thus our strategy, which we term “ZLIFO” (“zero-commitment last-in first-out”), assigns highest priority to unachievable and uniquely achievable open conditions on the flaw agenda, and second-highest priority to open conditions most recently added to the agenda. Unresolved threats (which are actual threats, with “delay separation” turned on) retain priority over open conditions and are selected from the agenda according to a LIFO discipline, as in the default UCPOP strategy. Hence the overhead incurred by ZLIFO for flaw selection is limited to the open conditions. In Appendix A we give the pseudo-code of ZLIFO. Very recently this implementation has also been packaged into UCPOP 4.0, a new version of UCPOP which is available by anonymous ftp to cs.washington.edu.

3 Experiments Using UCPOP

3.1 Test problems and experimental settings

In order to test our ideas we modified version 2.0 of UCPOP [3], replacing its default plan-selection strategy (S+OC+UC) and goal-selection strategy (LIFO) to incorporate strategies discussed in the previous sections.

We tested the modified planner on several problems in the UCPOP suite, emphasizing those that had proved most challenging for previous strategies, on some artificial problems due to Kambhampati et al. [20], in the TRAINS

\footnote{The Lisp implementation together with instructions for integration into UCPOP is available by inquiry to the authors.}
transportation domain developed in Rochester [1, 2], and in Joslin & Pollack’s Tile-world domain [18].

The UCPOP problems include Towers of Hanoi (T of H), Fixa, Fix3, Fixit, Tower-Invert4, Test-Ferry, and Sussman-Anomaly. In the case of T of H, we added a 3-operator version to the UCPOP single-operator version, since T of H is a particularly hard problem for UCPOP and its difficulty has long been known to be sensitive to the formalization (e.g., [15]). Fixa is a problem from Dan Weld’s “fridge domain”, in which the compressor in the fridge is to be exchanged, requiring unscrewing several screws, stopping the fridge, removing the backplane, and making the exchange. Fix3 is from Stuart Russell’s “flat tire domain”, where a new wheel is to be mounted and lowered to the ground (the old wheel has been jacked up already and the nuts loosened); this requires unscrewing the nuts holding the old wheel, removing the wheel, putting on the new wheel, screwing on the nuts, jacking down the hub, and tightening the nuts. Fixit is more complicated, as the wheel is not yet jacked up initially and the nuts not yet loosened, the spare tire needs to be inflated, and the jack, wrench and pump all need to be taken out of the trunk and stowed again at the end. Tower-Invert4 is a problem in the blocks world, requiring the topmost block in a stack of four blocks to be made bottom-most. Test-Ferry is a simple problem requiring two cars to be moved from A to B using a one-car ferry, by boarding, sailing, and unboarding for each car.

The artificial problems correspond to two parameter settings for ART-#ext-#club, one of the two artificial domains that served as a testbed for Kambhampati et al.’s extensive study of the behavior of various planning strategies as a function of problem parameters [20]. ART-#ext-#club provides two “layers” of 10 operators each, where those in layer 1 achieve the preconditions of those in layer 2, and each operator in layer 2 achieves one of the 10 goals. However, the operators in each layer can establish or “clobber” the preconditions of their neighbors, and this can force operators to be used in a certain order.

The version of the TRAINS domain that we encoded involves four cities (Avon, Bath, Corning, Dansville) connected by four tracks in a diamond pattern, with a fifth city (Elmira) connected to Corning by a fifth track. The available resources, which are located at various cities, consist of a banana warehouse, an orange warehouse, an orange juice factory, three train engines (not coupled to any cars), 4 boxcars (suitable for transporting oranges or bananas), and a tanker car (suitable for transporting orange juice). Goals are typically to deliver oranges, bananas, or orange juice to some city, requiring engine-car coupling, car loading and unloading, engine driving, and possibly OJ-manufacture.

The Tile-world domain consists of a grid on which holes and tiles are scattered. A given tile may or may not fit into a particular hole. The goals are to fill one or more holes by using three possible actions: picking up a tile, going to an x-y location on the grid, and dropping a tile into a hole. The agent can carry at most four tiles at a time.

---

5 This domain was chosen since absolute performance data are provided for it in [20].
Formalizations of several of these domains in terms of UCPOP’s language are provided in Appendix B. This includes the two versions of the T of H, ART-#est-#club, the Rochester TRAINS domain and the Tile-world domain. The formalizations of the other problems from the UCPOP suite are not repeated here.6

The experiments for all problems except Fixit, the TRAINS problems and the Tile-world problems were conducted on a SUN 10 using Lucid Common Lisp 4.0.0, while the rest (tables X–XII in the next subsection) were conducted on a SUN 20 using Allegro Common Lisp 4.2. Among the search control functions provided by UCPOP, we used the default bestf-search when the problem was solvable within the search limit of 40,000 plans generated, while we used the function id-bf-search (an implementation of the linear-space best-first search algorithm given by Korf in [19]), when this limit was exceeded.7

3.2 Experimental results

Tables I–XII show the CPU time (seconds) and the number of plans created/explored by UCPOP on twelve problems in the domains described above: Towers of Hanoi with three disks and either one operator (T-of-H1) or three operators (T-of-H3), “Dan’s fridge” domain (Fixa), Russell’s tire changing domain (Fix3 and Fixit), the blocks world (Tower-Invert4 and Sussman-anomaly), the ferry domain (Test-Ferry), the artificial domain ART-#est-#club (specifically, ART-3-6 and ART-6-3), the Rochester TRAINS domain (Trains1 and Trains2) and the Tile-world domain (tw-1, ..., tw-6). Note that the number of plans is probably more meaningful than the CPU time for evaluating the performance of the strategies examined, since it depends only on the algorithm, not the implementation.8 In fact our implementation of these strategies was committed to not altering UCPOP’s data structures; they could have been implemented more efficiently with modified data structures.

Tables I and II show that for the T of H the plan selection strategy S+OC gives dramatic improvements over the default S+OC+UC strategy. (In these tests the default LIFO goal selection strategy was used.) In fact, UCPOP solved T-of-H1 in 0.97 seconds using S+OC versus 204.5 seconds using S+OC+UC. T-of-H3 proved harder to solve than T-of-H1, requiring 8.5 seconds using S+OC and an unknown time in excess of 600 CPU-second using S+OC+UC.

---

6The formalizations of these domains, which do not include the 3-operator version of the T of H, the artificial domain from [20], or the TRAINS domain, are available along with UCPOP via anonymous FTP from cs.washington.edu. We thank David Joslin who supplied the formalization of the Tile-world domain to us.

7This choice was motivated by the observation that when the problem is relatively easy to solve bestf-search appears to be more efficient than id-bf-search, while for hard problems it can be very inefficient because of the considerable amount of space used at run time and the CPU-time spent on garbage collection, which in some cases made Lisp crash, reporting an internal error.

8It is also worth noting that the number of plans created implicitly takes into account plan size, since addition of a step to a plan is counted as creation of a new plan in UCPOP.
Our ZLIFO goal-selection strategy can significantly accelerate planning compared with the simple LIFO strategy. In particular, when ZLIFO was combined with the S+OC plan-selection strategy in solving T-of-H, it further reduced the number of plans generated by a factor of 3 in T-of-H1 (obtaining an overall reduction by a factor of 636, and decreased the required CPU time from 204.5 to 0.54 seconds), and by a factor of 8 in T-of-H3.

Tables III–VIII provide data for problems that are easier than T-of-H, but still challenging to UCPOP operating with its default strategy, namely Fixa (table III), Fix3 (table IV), Tower-Invert4 (table V), Test-Ferry (table VI) and the artificial domain ART-#_est-#_clob with #_est = 3 and #_clob = 6 (table VII) and with #_est = 6 and #_clob = 3 (table VII). The results show that the combination of S+OC and ZLIFO substantially accelerates UCPOP in comparison with its performance using S+OC+UC and LIFO. The number of plans generated dropped by a factor of 22 for Fixa, by a factor of 5.9 for Fix3, by a factor of 5.7 for Tower-Invert4, by a factor of 5.1 for Test-Ferry, by a factor of 7 for ART-3-6, and by a factor of 17 for ART-6-3.

Concerning ART-#_est-#_clob, note that the performance we obtained with unenhanced UCPOP (624 plans generated for ART-3-6 and 985 for ART-6-3) was much the same as (just marginally better than) reported in [20] for the best planners considered there (700 – 1500 plans generated for ART-3-6, and 1000–2000 for ART-6-3). This is to be expected, since UCPOP is a generalization of the earlier partial-order planners. Relative to standard UCPOP and its predecessors, our “accelerated” planner is thus an order of magnitude faster. Interestingly, the entire improvement here can be ascribed to ZLIFO (rather than S+OC plan selection, which is actually a little worse than S+OC+UC). This is probably due to the unusual arrangement of operators in ART-#_est-
Table III: Performance of plan/goal selection strategies on Fixa

<table>
<thead>
<tr>
<th>goal-selection</th>
<th>plan-selection</th>
<th>CPU-time</th>
<th>plans created/explored</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIFO</td>
<td>S+OC+UC</td>
<td>2.45</td>
<td>2131/1903</td>
</tr>
<tr>
<td>LIFO</td>
<td>S+OC</td>
<td>2.48</td>
<td>2131/1903</td>
</tr>
<tr>
<td>ZLIFO</td>
<td>S+OC+UC</td>
<td>0.33</td>
<td>96/74</td>
</tr>
<tr>
<td>ZLIFO</td>
<td>S+OC</td>
<td>0.33</td>
<td>96/74</td>
</tr>
</tbody>
</table>

Table IV: Performance of plan/goal selection strategies on Fix3

<table>
<thead>
<tr>
<th>goal-selection</th>
<th>plan-selection</th>
<th>CPU-time</th>
<th>plans created/explored</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIFO</td>
<td>S+OC+UC</td>
<td>6.50</td>
<td>3396/2071</td>
</tr>
<tr>
<td>LIFO</td>
<td>S+OC</td>
<td>0.43</td>
<td>351/215</td>
</tr>
<tr>
<td>ZLIFO</td>
<td>S+OC+UC</td>
<td>1.12</td>
<td>357/221</td>
</tr>
<tr>
<td>ZLIFO</td>
<td>S+OC</td>
<td>1.53</td>
<td>574/373</td>
</tr>
</tbody>
</table>

In experimenting with various combinatorially trivial problems that unmodified UCPOP handles with ease, we found that the S+OC and ZLIFO strategy is neither beneficial nor harmful in general; there may be a slight improvement or a slight degradation in performance. Results for the Sussman anomaly in table IX provide an illustrative example.

We summarize the results of tables I–XI in Figure 5, showing the speedup obtained with the combined ZLIFO goal selection strategy and S+OC plan selection strategy as a function of problem difficulty (as indicated by the number of plans generated by the default LIFO plus S+OC+UC strategy). The trend toward greater speedups for more complex problems (though somewhat dependent on problem type) is quite apparent from the log-log plot.

For direct comparison with Joslin and Pollack’s LCFR strategy, we implemented their strategy and applied it to several problems. It did very well

Table V: Performance of plan/goal selection strategies on Tower-Invert4

<table>
<thead>
<tr>
<th>goal-selection</th>
<th>plan-selection</th>
<th>CPU-time</th>
<th>plans created/explored</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIFO</td>
<td>S+OC+UC</td>
<td>1.35</td>
<td>808/540</td>
</tr>
<tr>
<td>LIFO</td>
<td>S+OC</td>
<td>0.19</td>
<td>148/105</td>
</tr>
<tr>
<td>ZLIFO</td>
<td>S+OC+UC</td>
<td>2.81</td>
<td>571/378</td>
</tr>
<tr>
<td>ZLIFO</td>
<td>S+OC</td>
<td>0.36</td>
<td>142/96</td>
</tr>
</tbody>
</table>
Table VI: Performance of plan/goal selection strategies on Test-Ferry

<table>
<thead>
<tr>
<th>goal-selection</th>
<th>plan-selection</th>
<th>CPU-time</th>
<th>plans created/explored</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIFO S+OC+UC</td>
<td>0.63</td>
<td>718/457</td>
<td></td>
</tr>
<tr>
<td>LIFO S+OC</td>
<td>0.32</td>
<td>441/381</td>
<td></td>
</tr>
<tr>
<td>ZLIFO S+OC+UC</td>
<td>0.24</td>
<td>136/91</td>
<td></td>
</tr>
<tr>
<td>ZLIFO S+OC</td>
<td>0.22</td>
<td>140/93</td>
<td></td>
</tr>
</tbody>
</table>

Table VII: Performance of plan/goal selection strategies on ART-#est-#clob with #est = 3 and #clob = 6 (averaged over 100 problems)

<table>
<thead>
<tr>
<th>goal-selection</th>
<th>plan-selection</th>
<th>CPU-time</th>
<th>plans created/explored</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIFO S+OC+UC</td>
<td>0.67</td>
<td>558/392</td>
<td></td>
</tr>
<tr>
<td>LIFO S+OC</td>
<td>1.36</td>
<td>1299/840</td>
<td></td>
</tr>
<tr>
<td>ZLIFO S+OC+UC</td>
<td>0.16</td>
<td>72/49</td>
<td></td>
</tr>
<tr>
<td>ZLIFO S+OC</td>
<td>0.18</td>
<td>79/54</td>
<td></td>
</tr>
</tbody>
</table>

Table VIII: Performance of plan/goal selection strategies on ART-#est-#clob with #est = 6 and #clob = 3 (averaged over 100 problems)

<table>
<thead>
<tr>
<th>goal-selection</th>
<th>plan-selection</th>
<th>CPU-time</th>
<th>plans created/explored</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIFO S+OC+UC</td>
<td>1.32</td>
<td>985/653</td>
<td></td>
</tr>
<tr>
<td>LIFO S+OC</td>
<td>2.08</td>
<td>1743/1043</td>
<td></td>
</tr>
<tr>
<td>ZLIFO S+OC+UC</td>
<td>0.14</td>
<td>57/37</td>
<td></td>
</tr>
<tr>
<td>ZLIFO S+OC</td>
<td>0.14</td>
<td>57/37</td>
<td></td>
</tr>
</tbody>
</table>

Table IX: Performance of plan/goal selection strategies on Sussman-anomaly

<table>
<thead>
<tr>
<th>goal-selection</th>
<th>plan-selection</th>
<th>CPU-time</th>
<th>plans created/explored</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIFO S+OC+UC</td>
<td>0.06</td>
<td>44/26</td>
<td></td>
</tr>
<tr>
<td>LIFO S+OC</td>
<td>0.04</td>
<td>36/21</td>
<td></td>
</tr>
<tr>
<td>ZLIFO S+OC+UC</td>
<td>0.12</td>
<td>67/43</td>
<td></td>
</tr>
<tr>
<td>ZLIFO S+OC</td>
<td>0.07</td>
<td>41/25</td>
<td></td>
</tr>
</tbody>
</table>
Table X: Performance of the plan selection strategy S+OC in combination with the goal selection strategies ZLIFO and LCFR in solving problems which are very hard for the default strategies of UCPOP. (The CPU-time does not include Lisp garbage collection. The number of plans generated for LCFR does not include those created in service to estimate the repair cost of the flaws.)

<table>
<thead>
<tr>
<th>Problem name</th>
<th>S+OC&amp;ZLIFO</th>
<th>S+OC&amp;LCFR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Plans</td>
<td>CPU-time</td>
</tr>
<tr>
<td>Trains1</td>
<td>4097/2019</td>
<td>15.36</td>
</tr>
<tr>
<td>Trains2</td>
<td>17482/10907</td>
<td>101.98</td>
</tr>
<tr>
<td>Fixit</td>
<td>5885/3685</td>
<td>38.2</td>
</tr>
</tbody>
</table>

Table XI: Performance of the default plan/goal selection strategies of UCPOP for Trains1, Trains2, and Fixit. (The CPU-time does not include Lisp garbage collection)

<table>
<thead>
<tr>
<th>Problem name</th>
<th>CPU time</th>
<th>Plans created/explored</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trains1</td>
<td>3050.15</td>
<td>1071479/432881</td>
</tr>
<tr>
<td>Trains2</td>
<td>&gt;37879</td>
<td>&gt;1000000</td>
</tr>
<tr>
<td>Fixit</td>
<td>27584.9</td>
<td>8090014/4436204</td>
</tr>
</tbody>
</table>

(sometimes better than ZLIFO) for problems on the lower end of the difficulty spectrum, but poorly for harder problems. For T-of-H1 LCFR in combination both with the default S+OC+UC plan selection strategy, and with our S+OC plan strategy did not find a solution within a search limit of 200,000 plans generated (cf. 253 for ZLIFO with S+OC, and 751 for ZLIFO with S+OC+UC!), requiring an unknown CPU time in excess of 4254 seconds with S+OC+UC, and in excess of 4834 seconds with S+OC (cf. 0.54 seconds for ZLIFO with S+OC).9

Table X shows the results for the plan strategy S+OC, with the goal strategies ZLIFO and LCFR, applied to three problems (Trains1, Trains2 and Fixit). These are very hard for the default strategies of UCPOP (see Table XI), but become relatively easy when S+OC is used in combination either with ZLIFO or LCFR. While LCFR did slightly better than ZLIFO for Trains1 (the easiest of these problems), it performed quite poorly for Fixit and Trains2 (the

---

9 This was with UCPOP's delay-separation switch turned off, which is implicit in LCFR [17]. In our experiments we also tested a variant of LCFR, where the switch is forced to be on. The resulting goal strategy in combination with our plan strategy S+OC performed significantly better for T-of-H1, solving the problem generating/exploring 7423/6065 plans, and using 110.45 CPU seconds. Note also that a comparison of our implementation of LCFR and Joelin&Pollack's implementation used for the experiments discussed in [18] showed that our implementation is considerably faster [17].
Problem size
Sussman-anomaly
ART-3-6
T一夜-worner
Fix3
T一夜-of-H1
Fixit

ART-6-3
Fixa
 ART-3-6
Tower-Invert4
T一夜-worner
Sussman-anomaly

Figure 1: Speedup due to ZLIFO and S+OC, relative to the number of plans generated by LIFO and S+OC+UC (log-log scale). The speedups for the problems that UCPOP was unable to solve even with a very high search limit (Trains2 and T of H3) are not included.

<table>
<thead>
<tr>
<th>Problem name</th>
<th>ZLIFO*</th>
<th>LCFR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU-time</td>
<td>Plans</td>
</tr>
<tr>
<td>tw-1</td>
<td>0.09</td>
<td>26/15</td>
</tr>
<tr>
<td>tw-2</td>
<td>0.61</td>
<td>72/39</td>
</tr>
<tr>
<td>tw-3</td>
<td>2.55</td>
<td>138/71</td>
</tr>
<tr>
<td>tw-4</td>
<td>7.80</td>
<td>224/111</td>
</tr>
<tr>
<td>tw-5</td>
<td>19.41</td>
<td>330/159</td>
</tr>
<tr>
<td>tw-6</td>
<td>42.57</td>
<td>456/215</td>
</tr>
</tbody>
</table>

Table XII: Performance of UCPOP in the Tile-world domain using ZLIFO* and LCFR for goal selection, and S+OC+F+0.1UC for plan selection

hardest problems) compared to ZLIFO.

In [18] Joslin and Pollack tested their LCFR strategy on six problems in the Tile-world (tw-1, ..., tw-6), five of which are very hard for default UCPOP, but easy for UCPOP using LCFR.\textsuperscript{10} We tested our ZLIFO strategy in the Tile-world using the same six problems. ZLIFO did well for tw-1-4, but for tw-5 and tw-6 its performance dropped well below that of LCFR. This raised the question whether for these particular problems it is crucial to minimize

\textsuperscript{10}In their experiments tw-2, the easiest among tw-2-6, was not solved by UCPOP even when allowed to run for over eight hours. On the other hand, UCPOP using LCFR solves tw-6, the hardest problem, without ever reaching a dead-end node in the search tree.
“repair cost” in flaw selection uniformly, rather than just in certain special cases (ZLIFO does minimize the repair cost when no threat is on the flaw list, and at least one zero-commitment open condition is present). However, further experiments aimed at answering this question suggested that the poor choices made by ZLIFO for some Tile-world problems were not due to selection of “high cost” over “low cost” flaws. Instead two factors appear to be crucial for improving ZLIFO: (a) emphasizing zero-commitment open conditions by giving them higher priority than threats; (b) when there are no zero-commitment open conditions, resolving threats as soon as they enter the agenda.\textsuperscript{11}

We extended our ZLIFO strategy to include (a) and (b), and we briefly tested the resulting variant of ZLIFO (ZLIFO\textsuperscript{*}). Table XI \textsuperscript{2} shows the results for ZLIFO\textsuperscript{*} together with the plan selection strategy S+OC+0.1UC+F, where as discussed in section 2.3 we included an attenuated form of the UC term (UC/10), and also an “F” term equal to the number of so-called “facts”. The latter are special open conditions (e.g., a numerical relation like (add-one x y)) that are not state-dependent and are established by Lisp functions \textsuperscript{3}. ZLIFO\textsuperscript{*} performed very efficiently for all six Tile-world problems, in fact a little better than LCFR. Note that for these problems ZLIFO\textsuperscript{*} is more efficient than LCFR in terms of the CPU-time, even though the number of plans generated/explored by the two strategies is approximately the same. This is because the overhead of selecting the next flaw to be handled is higher in LCFR than in ZLIFO\textsuperscript{*} (and ZLIFO). In fact, while LCFR needs to compute the “repair cost” of each flaw (including the threats) in the current plan, ZLIFO\textsuperscript{*} (ZLIFO) only needs to check for the presence of zero-commitment open conditions, without processing the threats.

Finally, additional experiments showed that the average performance of ZLIFO\textsuperscript{*} does not differ significantly from that of ZLIFO for the other problems we used in our experiments.\textsuperscript{12}

4 Conclusions and Further Work

We have argued in favor of some simple, domain-independent improvements to partial order planning strategies, based on the one hand on a carefully considered choice of terms in the A* heuristic for plan selection, and on the other on a preference for choosing open conditions that cannot be achieved at all or can only be achieved uniquely (with a default LIFO prioritization of other open conditions). Since the plan refinements corresponding to uniquely achievable goals are logically necessary, we have termed this strategy a “zero-

\textsuperscript{11}We realized the relevance of (b) by observing that the performance of a modified version of LCFR, where the delay separation switch is implicitly forced on, dramatically degraded for tw-6 in a slightly different formulation of the Tile-world.

\textsuperscript{12}For some problems we observed a slight improvement in terms of plans created/explored, and for others a slight degradation. The CPU-time tends to increase since the overhead of computing the goal selection function is higher for ZLIFO\textsuperscript{*} than for ZLIFO, because of the extra agenda-management costs.
commitment” strategy.

Our experiments based on modifications of UCPOP indicate that our strategies can give large improvements in planning performance, especially for problems that are hard for UCPOP (and its “relatives”) to begin with. The best performance was achieved when our strategies for plan selection and goal selection were used in combination. Further, our results indicate that zero-commitment is best supplemented with a LIFO strategy for open conditions achievable in multiple ways, rather than a generalization of zero-commitment favoring goals with the fewest children. In practical terms, we were able to solve nearly every problem we tried from the UCPOP test suite in a fraction of a second (except for Fixit, which required 38.2 seconds), where some of these problems previously required minutes or were unsolvable on the same machine. This included a sufficient variety of problems to indicate that our techniques are of broad potential utility.

One promising direction for further work is to make the zero-commitment strategy apply more often by developing ways of identifying “false options” as early as possible. That is, if a possible action instance (obtained by matching an open condition against available operators as well as against existing actions) is easily recognizable as inconsistent with the current plan, then its elimination may leave us with a single remaining match and hence an opportunity to apply the zero-commitment strategy.

One way of implementing this strategy would be to check at once, before accepting a matched action as a possible way to attain an open condition, whether the temporal constraints on that action force it to violate a causal link, or alternatively, force its causal link to be violated. In that case the action could immediately be eliminated, perhaps leaving only one (or even no) alternative. This could perhaps be made even more effective by broadening the definition of threats so that preconditions as well as effects of actions can threaten causal links, and hence bring to light inconsistencies sooner. Note that if a precondition of an action is inconsistent with a causal link, it will have to be established with another action whose effects violate the causal link; so the precondition really poses a threat from the outset.

Another direction for further work is to apply efficient temporal reasoning methods to the problem of eliminating inconsistent promotion/demotion alternatives for threat elimination, given the set of all (definite) threats and ordering relations in the plan under development. Though this problem is in principle NP-hard, algorithms that are very efficient on average are described in [12, 13]. This could be far more efficient than trying each possible promotion and demotion, checking in isolation for consistency with ordering constraints. A similar idea was previously explored in [35] using arc consistency techniques, but we think further gains are possible with the algorithms mentioned above, which are more general than arc-consistency testing and employ intelligent backtracking for efficient search.

Finally, another direction that seems very promising to us is to precompute certain constraints that must hold throughout the search space of a given
problem, based on the structure of the operators, initial conditions and goal conditions. This often permits some matching actions for open conditions to be immediately eliminated, as they would violate the precomputed constraints. We have already begun exploring one such method, which involves precomputing domains (sets of constants) of operator parameters and using these domains as filters in the planning process [14]. A related idea is to precompute implicative state constraints and single-valuedness constraints that are implicit in the operator structure.

Our conclusion, both from the results we have presented and from the possibilities for further speedups we have mentioned, is that ample opportunities still exist for major improvements in the performance of well-founded, domain-independent planners. These may be sufficient to make such planners competitive with current more pragmatically designed planners. This will be an interesting comparison to pursue in the future.
Appendix A

ZLIFO flaw selection strategy for UCPOP

ZLIFO chooses the next flaw according to the following preferences:

1. an actual threat (Dsep is on), using LIFO to pick among these;
2. an open condition that cannot be established in any way;
3. an open condition that can only be resolved in one way, preferring open conditions that can be established by introducing a new action to those that can be established by using "start";
4. an open condition, using LIFO to pick among these.

2. and 3. are zero-commitment choices. In our experiments the sub-preference in 3. gave improvements in the context of Russell's tire changing domain (in particular with Fix3).

ZLIFO selection of open conditions (steps 2-4)

Sketch of the main algorithm:

0. Z-OC := nil; L-OC := nil; I-MATCHES := 0; S-MATCHES := 0; O-MATCHES := 0;
   {Z-OC and L-OC are variables that will be used to store selected open conditions. In particular, Z-OC contains an open condition such that when this condition is processed it generates a zero-commitment refinement; L-OC contains an open condition selected using the LIFO strategy. The variables I-MATCHES, S-MATCHES, O-MATCHES, will be used to count the number of different ways of resolving an open condition};
1. FOR EACH open cond. OC taken in LIFO-order from the list of flaws DO
   BEGIN
2.   I-MATCHES := number of (positive) conditions in the initial state which match OC given the current set of binding constraints
   {if OC is a negated condition and it can be established by using the closed world assumption, then count 1 match};
3.   S-MATCHES := number of effects of steps in the current plan (excluding *start*) which match OC given the current set of bindings constraints, and which can possibly be before the step of OC;
4.   O-MATCHES := number of effects of operators matching OC
   {Note that when S-MATCHES=1 S-MATCHES <= O-MATCHES}
5.   TOT-MATCHES := I-MATCHES + S-MATCHES + O-MATCHES;
6.   IF TOT-MATCHES = 0 THEN RETURN OC
   {we have found an open condition with zero matches, i.e. that
cannot be established in any way. When this (zero-commitment) open condition is handled, the corresponding plan is pruned from the plan queue);

7. IF TOP-MATCHES = 1 THEN

8. IF Z-OC = nil THEN Z-OC := OC
   {we have found a zero-commitment open condition but we don’t immediately return it because we could still find another better condition (i.e., a condition with zero matches)}

9. ELSE {sub-preference among open conds with only one match}

10. IF Z-OC is an open condition that can be established only by the initial state (initial action *start*), and OC can be established only by using an operator (introducing a new action into the current plan)
    THEN Z-OC := OC;

11. ELSE {OC is not zero-commitment (TOT-MATCHES > 1)}

12. IF L-OC = nil THEN L-OC := OC
    {L-OC is bound to the open condition in the list of the flaws satisfying the LIFO ordering}

END{FOR}

13. IF Z-OC = nil THEN RETURN L-OC ELSE RETURN Z-OC.
    {if we have found an OC with only one possible way of being resolved, then this is returned; otherwise an open condition taken in LIFO order from the flaws list is returned.}

Optimization notes and extensions

The current implementation of ZLIFO can be improved in some ways. In particular, there is no need to count all the initial conditions matching the open condition (I-MATCHES). When we have found two matching initial conditions, we can stop because we already know that the current OC is not a zero-commitment choice. Furthermore, there is no need to compute S-MATCHES (steps 3) and O-MATCHES (step 4).

Moreover, we need to find at most 1 matching effect of steps in the plan (S-MATCHES). In fact, if S-MATCHES = 1 TOT-MATCHES is greater than 1, since O-MATCHES ≥ S-MATCHES. So, step 2 (S-MATCHES) needs to be computed only if I-MATCHES < 2, and step 4 (O-MATCHES) needs to be computed only if I-MATCHES < 2 and S-MATCHES = 0.

These refinements of the current implementation of ZLIFO will reduce the computational overhead for flaw selection, obtaining better performance in terms of CPU-time.
Appendix B

Formalization of T-of-H1

\[
\begin{align*}
&(\text{define (operator move-disk)} \\
&\quad :\text{parameters } ((\text{disk } ?\text{disk}) \ ?\text{below-disk} \ ?\text{new-below-disk}) \\
&\quad :\text{precondition } (:\text{and } (\text{smaller } ?\text{disk} \ ?\text{new-below-disk}) \ ;\text{handles pegs} \\
&\quad \quad (\neg \text{eq } ?\text{new-below-disk} \ ?\text{below-disk}) \\
&\quad \quad (\neg \text{eq } ?\text{new-below-disk} \ ?\text{disk}) \\
&\quad \quad (\text{on } ?\text{disk} \ ?\text{below-disk}) \\
&\quad \quad (\text{clear } ?\text{disk}) \\
&\quad \quad (\text{clear } ?\text{new-below-disk}))) \\
&\quad :\text{effect } (:\text{and } (\text{clear } ?\text{below-disk}) \\
&\quad \quad (\text{on } ?\text{disk} ?\text{new-below-disk}) \\
&\quad \quad (\neg \text{on } ?\text{disk} ?\text{below-disk}) \\
&\quad \quad (\neg \text{clear } ?\text{new-below-disk})))
\end{align*}
\]

Initial state: 
\[
\begin{align*}
&((\text{smaller } D1 \ P1) \ (\text{smaller } D2 \ P1) \ (\text{smaller } D3 \ P1) \ (\text{smaller } D1 \ P2) \\
&\quad (\text{smaller } D2 \ P2) \ (\text{smaller } D3 \ P2) \ (\text{smaller } D1 \ P3) \ (\text{smaller } D2 \ P3) \\
&\quad (\text{smaller } D3 \ P3) \ (\text{smaller } D1 \ D2) \ (\text{smaller } D1 \ D3) \ (\text{smaller } D2 \ D3) \\
&\quad (\text{clear } P2) \ (\text{clear } P3) \ (\text{clear } D1) \ (\text{disk } D1) \ (\text{disk } D2) \ (\text{disk } D3) \\
&\quad (\text{on } D1 \ D2) \ (\text{on } D2 \ D3) \ (\text{on } D3 \ P1))
\end{align*}
\]

Goal state: 
\[
\begin{align*}
&(\text{and } (\text{on } D1 \ D2) \ (\text{on } D2 \ D3) \ (\text{on } D3 \ P3))
\end{align*}
\]

Formalization of T-of-H3

\[
\begin{align*}
&(\text{define (operator MOVE-D1)} \\
&\quad :\text{parameters } ((\text{thing } ?\text{from}) \ ?\text{to}) \\
&\quad :\text{precondition } (:\text{and } (\text{on } D1 \ ?\text{from}) \\
&\quad \quad (\text{clear } ?\text{to}) \\
&\quad \quad (\neg \text{on } D1 \ ?\text{to}) \\
&\quad \quad (\neg \text{clear } ?\text{from})) \\
&\quad :\text{effect } (:\text{and } (\text{on } D1 \ ?\text{to}) \\
&\quad \quad (\neg \text{on } D1 \ ?\text{from}) \\
&\quad \quad (\neg \text{clear } ?\text{from}))
\end{align*}
\]

\[
\begin{align*}
&(\text{define (operator MOVE-D2)} \\
&\quad :\text{parameters } ((\text{thing } ?\text{from}) \ ?\text{to}) \\
&\quad :\text{precondition } (:\text{and } (\text{on } D2 \ ?\text{from}) \\
&\quad \quad (\text{clear } ?\text{to}) \\
&\quad \quad (\neg \text{on } D2 \ ?\text{to}) \\
&\quad \quad (\neg \text{clear } ?\text{from})) \\
&\quad :\text{effect } (:\text{and } (\text{on } D2 \ ?\text{to}) \\
&\quad \quad (\neg \text{on } D2 \ ?\text{from}) \\
&\quad \quad (\neg \text{clear } ?\text{from}))
\end{align*}
\]

\[
\begin{align*}
&(\text{define (operator MOVE-D3)} \\
&\quad :\text{parameters } ((\text{thing } ?\text{from}) \ ?\text{to}) \\
&\quad :\text{precondition } (:\text{and } (\text{on } D3 \ ?\text{from}) \\
&\quad \quad (\text{clear } ?\text{to}) \\
&\quad \quad (\neg \text{on } D3 \ ?\text{to}) \\
&\quad \quad (\neg \text{clear } ?\text{from})) \\
&\quad :\text{effect } (:\text{and } (\text{on } D3 \ ?\text{to}) \\
&\quad \quad (\neg \text{on } D3 \ ?\text{from}) \\
&\quad \quad (\neg \text{clear } ?\text{from}))
\end{align*}
\]

Initial state: 
\[
\begin{align*}
&(\text{on } D1 \ D2) \ (\text{on } D2 \ D3) \\
&\quad (\text{on } D3 \ P1) \ (\text{clear } D1) \\
&\quad (\text{thing } D1) \ (\text{thing } D3) \\
&\quad (\text{thing } P1) \ (\text{thing } P3) \\
&\quad (\text{clear } D1) \ (\text{clear } D3) \\
&\quad (\text{clear } P1) \ (\text{clear } P3)
\end{align*}
\]

Goal state: 
\[
\begin{align*}
&(\text{and } (\text{on } D1 \ D2) \ (\text{on } D2 \ D3) \\
&\quad (\text{on } D3 \ P3))
\end{align*}
\]
Formalization of ART.  The square brackets (not part of the syntax) indicate parts to be included only for $i < n_+$ ($\#_{est}$); the braces (not part of the syntax) indicate parts to be included only for $0 < i < n_-$ ($\#_{clob}$).

: Replace $i$ by 0, ..., 9 in the following two operators:

```
(define (operator A11)  (define (operator A12)
 :parameters ()         :parameters ()
 :precondition ((Ii1))  :precondition ((Pi))
 :effect (:and (Pi) [[Ii+1]]) :effect (:and (Gi) [[Pi+1]])
                  {(:not (Ii-1))})
                  {(:not (Pi-1))}))
```

Goal state:  (and (G0) (G1) (G2) (G3) (G4) (G5) (G6) (G7) (G8) (G9))

Formalization of Rochester TRAINS transportation domain.
The initial state of Trains2 is the same as that of Trains1 except that oj-fac1 and e3 are at Corning instead of Elmira, and that the connections from Corning to Bath and from Dansville to Corning are disabled (say, for maintenance). vspace0.2cm

```
(define (operator mv-engine)
  :parameters (?eng ?city1 ?city2 ?track ?car); ?car is a "hidden" parameter
  :effect (:and (?eng ?city1) (:not (?eng ?city2))
          (when (coupled ?eng ?car) (:and (?eng ?city2) (:not (?eng ?city1)))))
)

(define (operator ld-oranges)
  :parameters (?ors ?car ?city)
)

(define (operator ld-bananas)
  :parameters (?bas ?car ?city)
)

(define (operator ld-oj)
  :parameters (?oj ?car ?city)
)

(define (operator make-oj)
  :parameters (?o ?fac ?city)
  :effect (:and (oj ?o) (:not (oranges ?o))))
)

(define (operator unload)
  :parameters (?comm ?car ?city)
)

(define (operator couple)
  :parameters (?eng ?car ?city)
  :effect (:and (:coupled ?eng ?car) (:not (:loose ?car))))
)

```

24
(define (operator uncouple)
  :parameters (?eng ?car)
  :precondition (coupled ?eng ?car)
  :effect (and (loose ?car) (not (coupled ?eng ?car))))

Initial state for Trains1:

(city avon) (city bath) (city corning) (city dansville) (city elmira)
(track tr1) (track tr2) (track tr3) (track tr4) (track tr5) (connects tr1 avon bath)
(connects tr1 bath avon) (connects tr2 bath corning) (connects tr2 corning bath)
(connects tr3 avon dansville) (connects tr3 dansville avon) (connects tr4 dansville corning)
(connects tr4 corning dansville) (connects tr5 corning elmira) (connects tr5 elmira corning)
(engine e1) (engine e2) (engine e3) (car bc1) (car bc2) (car bc3) (car bc4) (car tc1)
(boxcar bc1) (boxcar bc2) (boxcar bc3) (boxcar bc4) (tanker-car tc1) (oranges ors1)
(bananas bas1) (ej-fac ej-fac1) (empty bc1) (empty bc2) (empty bc3) (empty bc4)
(loose bc1) (loose bc2) (loose bc3) (loose bc4) (loose tc1) (at e1 avon) (at bas1 avon)
(at bc1 bath) (at bc2 bath) (at bc3 dansville) (at tc1 corning) (at ors1 corning)
(at e1 elmira) (at bas1 elmira) (at bc4 elmira) (at ej-fac1 elmira)

Goal of Trains1: (:exists (oranges ?x) (at ?x bath))
Goal of Trains2 and Trains3: (:exists (ej ?x) (at ?x dansville))

Formalization of the Tile-world domain (written by David Joslin)

(defvar +tw-carry-max* 4)

(defun tw ()
  ;; purge old domain prior to defining a new domain
  (reset-domain)
  ;; y is (+ ?x 1), but only within [0,+tw-carry-max*]
  (define (fact (add-one ?x ?y))
    (let ((ret
      (cond ((and (variable? ?x) (variable? ?y))
        ; no-match-attempted)
        ((and (not (variable? ?x)) (not (variable? ?y)))
          (if (= ?y (+ ?x 1))
            '(nil) ;; already satisfied nil)) ;; impossible
        ((variable? ?x)
          (if (< ?y 0)
            nil ;; x would be negative
            (list (setb ?x (- ?y 1))))))
        ((variable? ?y)
          (if (> ?x (meta-value +tw-carry-max*))
            nil ;; y would be over the limit
            (list (setb ?y (+ ?x 1))))))
      (t (break "cannot happen"))))
    ;; (format t "add-one ~a ~a returning ~a% ~a /?x /?y ret) ret ))

  ;; when a tile is dropped into a hole, both disappear
  (define (operator drop-tile-in-hole)
    :precondition
      (and (hole-info ?hole ?row ?col ?tile)
        (agent-loc ?row ?col)
        (carrying ?tile)
        (add-one ?new-count ?old-count) ;; subtract
        (carry-count ?old-count))
    :effects ((effect (:and (filled-hole ?hole)
                  (:not (carrying ?tile))))

25
(not (carry-count old-count))
(carry-count new-count)))))

;; when we pick up a tile, delete tile-info so it can't be
;; picked up more than once
(define (operator pickup-tile)
:parameters (?tile ?row ?col old-count new-count max)
:precondition (and (tile-info ?tile ?row ?col)
(agent-loc ?row ?col)
(add-one old-count new-count)
(carry-count old-count)
(carry-max max))
:effects ((:effect (:and (:not (tile-info ?tile ?row ?col))
(carrying ?tile)
(not (carry-count old-count))
(carry-count new-count))))

(define (operator go-to)
:precondition (agent-loc ?oldrow ?oldcol)
:effects ((:effect (:and (:not (agent-loc ?oldrow ?oldcol))
(agent-loc ?row ?col))))

(defvar *tw-init*
'((hole-info H1 2 2 T1)
(hole-info H2 2 3 T2)
(hole-info H3 2 4 T3)
(hole-info H4 1 7 T4)
(hole-info H5 1 8 T5)
(hole-info H6 1 1 T6)

;; for each tile not being carried, generate:
;; (tile-info <tile-name> <row> <col>)
(tile-info T1 7 8)
(tile-info T2 8 4)
(tile-info T3 17 7)
(tile-info T4 7 9)
(tile-info T5 3 14)
(tile-info T6 11 9)
(tile-info T7 12 2)
(tile-info T8 5 8)

;; for each tile being carried, generate:
;; (carrying <tile-name>)
(carry-count 0)
(carry-max 4)
(agent-loc 10 10))

(push (make-problem
:name 'tw-ex
:domain #tw
:inits *(hole-info H1 10 10 T1)
(carrying T1)
(carry-count 1)
(carry-max 1)
(agent-loc 10 10))
:goal *(filled-hole H1))
*tests*)

(push (make-problem
:name 'tw-1
:domain #tw
:inits *tw-init*
:goal *(filled-hole H1))
*tests*)

(push (make-problem
:name 'tw-2
:domain #tw
:inits *tw-init*
:goal *(filled-hole H1))
*tests*)

26
(push (make-problem
  :name 'tw-3
  :domain '#tw
  :inits *tw-init*
  :goal (and (filled-hole H1) (filled-hole H2)))
 *tests*)

(push (make-problem
  :name 'tw-4
  :domain '#tw
  :inits *tw-init*
  :goal (and (filled-hole H1) (filled-hole H2) (filled-hole H3) (filled-hole H4)))
 *tests*)

(push (make-problem
  :name 'tw-5
  :domain '#tw
  :inits *tw-init*
  :goal (and (filled-hole H1)
 (filled-hole H2)
 (filled-hole H3)
 (filled-hole H4)
 (filled-hole H5)))
 *tests*)

(push (make-problem
  :name 'tw-6
  :domain '#tw
  :inits *tw-init*
  :goal (and (filled-hole H1)
 (filled-hole H2)
 (filled-hole H3)
 (filled-hole H4)
 (filled-hole H5)
 (filled-hole H6)))
 *tests*)
References


Also available as: IRST Tech. Rep. 9307-44, Istituto per la Ricerca Scientifica e Tecnologica, 38050 Povo, Trento Italy; Tech. Rep. 496, Computer Science Dept., University of Rochester, Rochester, NY 14627, USA.


