Selectively Delaying Linking Commitments in Partial-Order Planners

Mr. R. S. de Silva
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, Pennsylvania 15213-3891
U. S. A.
desilva+@cmu.edu
Phone (412) 268-3620; Fax (412) 268-5576

Abstract. Total-order planners perform backtracking search over all possible total-orderings, but their commitment to a total order gives them a well-defined current state that can be used to guide the search at later choice-points. Partial-order planners handle operator-orderings more efficiently, but lack a current state. In such planners, an open condition can be linked to many steps (both pre-existing and newly introduced), and making a causal link to one is a choice-point. Veloso & Blythe (1994) showed how the lack of a current state to give guidance at these choice-points causes *linkability* difficulties. This paper explores how partial-order planners can maintain an approximation of the current state, and use it to give guidance at linkability choice-points. An inexpensive domain-independent heuristic selectively delays linking commitments in a manner very similar to that in which a total-order planner handles goals that are already satisfied in the current state. The heuristic, implemented in the partial-order planner UCPOP, permits it to avoid linkability difficulties to some extent. In effect, the heuristic embodies some of the strengths of total-order planners without inheriting their weaknesses.

Keywords: planning, non-linear, partial-order, causal-link, total-order, linkability, serializability, heuristic, efficiency

1 Introduction

A planner takes a set of operator templates, an initial state, and a desired goal, and attempts to find a plan (a sequence of operators) that achieves the desired goal when applied from the initial state. There are two broad classes of means-ends non-linear planners, each with its attendant strengths and weaknesses: total-order non-linear planners such as PRODIGY [3], and partial-order planners such as UCPOP [12].

A total-order planner uses a totally-ordered sequence of operators to maintain a current state, which begins at the initial state and is successively modified by the simulated application of operators in the plan being constructed. The current state may be used to guide the planner at choice-points. However, the necessity of committing to a total order means that the planner has to perform backtracking search over all possible operator application orderings.

In contrast, a partial-order planner always maintains the operators in a partial-order. This means that it lacks a current state which may be used to guide the planner

at choice-points. The avoidance of a total order reduces the search-space because the planner does not add unnecessary ordering constraints. However, the partial order makes it more difficult to reason about the plan being constructed. Causal-link planners reduce the difficulty by making explicit commitments as to how goals are to be achieved, either by linking it to a pre-existing step, or by linking it to a newly introduced step. This choice-point as to how the link should be made leads to an increase in the search-space. Veloso & Blythe [14] identified this issue and termed it *linkability*.

This paper explores how total-order planners use the current state to avoid linkability difficulties, and how partial-order planners can handle linkability by mimicking the reasoning done by total-order planners. This may be done by a domain-independent heuristic that selectively delays making linking commitments for some goals, imitating how a total-order planner handles goals that are already satisfied in the current state.

Section 2 briefly describes UCPOP, the partial-order planner that was used for this work. Section 3 defines the concept of linkability, and gives some illustrative domains. Section 4 describes how PRODIGY easily handle these domains, and goes on to show how UCPOP may handle them by mimicking PRODIGY's behaviour to some extent. Section 5 presents the results obtained, and Section 6 the conclusions.

2 Description of UCPOP

UCPOP is a partial-order planner developed by Barrett, Weld *et. al.* [12] from SNLP [9]. Figure 1 shows a simplified version of its algorithm, adapted from Weld [15, page 47]. The input is a domain given by a set of operators (steps) $\mathcal{O}$, an initial state Σ, and a desired final state Ω. The planner maintains a list of incomplete plans PL, ordered by a plan-ranking function. Each such plan is a tuple $\prec S, O, L, B, G \succ$ consisting of the steps S in the plan, the ordering constraints O on the steps, the links L that support the goals, the binding constraints B, and the goals G that are yet unsupported. Initially, PL contains a single plan consisting solely of a dummy step s_0 that adds Σ, and another dummy step s_∞ that needs Ω.

UCPOP proceeds by picking the highest ranked plan off PL. It then selects a flaw in this plan, using a flaw-ranking function. If the flaw selected is an unsupported goal $g \in G$, UCPOP considers all steps that may achieve g, and adds the corresponding plans to PL. If the flaw selected is a threatened link, UCPOP attempts to protect the link by either promotion, demotion, or non-codesignation, and adds the corresponding plans to PL.

The process terminates with success when a plan with no flaws is picked from PL. If PL becomes empty, then the planner has unsuccessfully considered all possible plans to achieve Ω, and returns failure.

UCPOP's algorithm performs best-first search (as determined by the plan-ranking function). PL holds the boundary of the search-space explored so far.

2.1 Ranking of Flaws

Given a plan, UCPOP uses a flaw-ranking function in Step 3 to select the flaw to handle next. As *all* flaws in the plan have to be eventually resolved to achieve success, the choice of flaw does not affect the completeness of the planner, and the planner does not backtrack over this choice. Nevertheless, the choice may affect the efficiency of the search.

Algorithm: $UCPOP(\mathcal{O}, \Sigma, \Omega)$

1. **Initialization**: Let s_0 be an operator that adds Σ; let s_∞ be an operator whose preconditions are Ω, let $p_{init} = \prec\{s_0, s_\infty\}, \{s_0 \prec s_\infty\}, \emptyset, \emptyset, \Omega\succ$ and let $PL = \{p_{init}\}$.
2. **Choose Plan**: If PL is empty, return failure. Else, remove the highest ranked plan $p = \prec S, O, L, B, G\succ$ from PL. If p has no flaws, return it as the solution.
3. **Choose Flaw**: The flaws in p consist of the unsupported preconditions G, and the links in L that are threatened by an operator in S. Select a flaw to handle.
4. **Goal Selection**: If an unsupported precondition $g \in G$ of a step s_{need} has been selected as the flaw to handle, find all steps $s_{add} \in \mathcal{O}$ (either a new step, or an existing step possibly prior to s_{need}) that can possibly add g. For all such steps, construct a new plan $p_{new} = \prec S', O', L', B', G'\succ$, where $S' = S \cup \{S_{add}\}$, $O' = O \cup \{s_{add} \prec s_{need}\}$, $L' = L \cup \{S_{add} \stackrel{g}{\rightarrow} S_{need}\}$, $B' = B \cup \{$bindings to make S_{add} assert $g\}$, and $G' = (G - g) \cup \{$preconditions of s_{add} (if new)$\}$, and add p_{new} to PL. Go to Step 2.
5. **Threat Handling**: Else, a step $s_k \in S$ threatens a causal link $s_i \stackrel{p}{\rightarrow} s_j \in L$. Ensure that s_k does not threaten the link by either promotion ($s_k \prec s_i$), demotion ($s_j \prec s_k$), or by constraining the relevant effect of s_k to not codesignate with p. For each such method whereby the threat may be handled, construct a new plan p_{new} by adding the appropriate constraints to O or B, and insert p_{new} into PL. Go to Step 2.

Figure 1: UCPOP's Planning Algorithm

The default flaw-ranking function simply prefers to handle threats before open goals. However, its performance can be significantly improved by two simple, inexpensive augmentations, viz., *sibling threats* and *monotonic goals*.

A *sibling threat* occurs when a step necessarily threatens the link supported by a sibling operator, as shown in Figure 2. This is very similar to Hertzberg's *enclosure by dependencies* [6]. Such threats can only be handled by one means, viz., promotion.

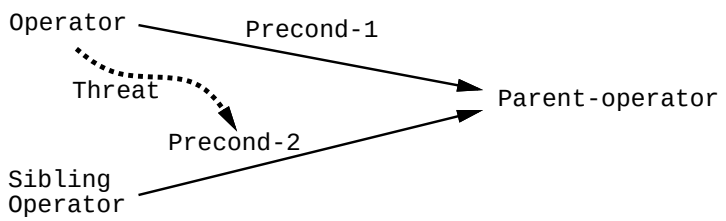

Figure 2: Sibling Threat

A goal is *monotonic* if no operators in the domain achieve it; similarly, a negated goal is *monotonic* if no operators delete it. A monotonic goal must necessarily be linked to the initial state; if that is not possible, then the plan cannot be successfully refined.

Both sibling threats and monotonic goals are flaws that are refinable in at most one way, i.e., the branching factor is zero or unity. Hence handling such flaws first is sensible, and the flaw-ranking function was accordingly modified, significantly improving the UCPOP's performance in most domains.

3 Linkability

Veloso & Blythe [14] introduced the concept of *linkability* in causal-link planners:

> A set of subgoals is *easily linkable* if, independent of the order by which the planner assigns causal links to the subgoals, it never has to backtrack to undo those links. Otherwise it is *laboriously linkable*.

The following subsections present two domains constructed by Veloso & Blythe, as well as one constructed by me, to demonstrate linkability difficulties encountered by causal-link planners. The next section discusses how these difficulties may be overcome.

3.1 Domain link-repeat

The domain *link-repeat* constructed by Veloso & Blythe is shown in Figure 3. The goal g_n is achieved by the operators $A_1, A_2, \ldots, A_n$ in sequence. However, each of these operators requires g_* and deletes it, so A_* is used to re-achieve g_* before each A_i. Veloso & Blythe compared g_* to a resource that may be consumed and replenished, such as fuel in a transportation domain.

Domain: *link-repeat*: laboriously linkable
Initial: $\{g_0, g_*\}$
Operators:

Action	Precond	Add	Delete
A_i $(i = 1 \ldots n)$	$\{g_*, g_{i-1}\}$	$\{g_i\}$	$\{g_*\}$
A_*	None	$\{g_*\}$	None

Goal: $\{g_*, g_n\}$
Solution: A_1, A_*, A_2, A_*, ..., A_*, A_n, A_*

Figure 3: Domain *link-repeat*

Each A_i in an incomplete plan has the precondition g_*. The planner has to explore the several possible options by which g_* can be achieved: a link to s_0, a link to pre-existing occurrences of A_* in the plan, or a link to a newly introduced occurrence of A_*. Only the last choice leads to a successful plan (see Figure 4), but the several unsuccessful choices for the link make the domain laboriously linkable. The difficulty is exacerbated by the plan-ranking function naturally considering the introduction of a new operator as the worst option.

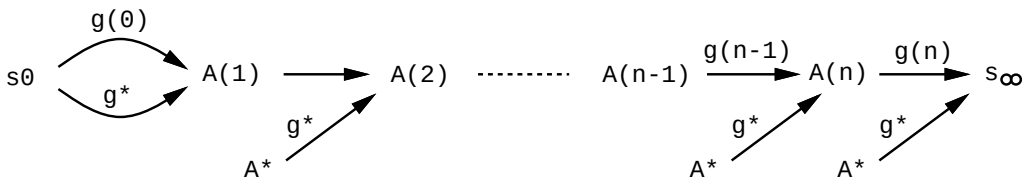

Figure 4: Domain *link-repeat*: Causal links in successful plan

3.2 *Domain* link-chain

Veloso & Blythe created the domain *link-chain* to illustrate the linkability difficulties caused when several *different* operators may achieve a desired goal, and the planner has to choose from among them. This characteristic is found in many real-world domains.

Veloso & Blythe noted that their definition of *link-chain* is defective, as the operators add their own preconditions. However, they also noted that this defect can be corrected by using a combination of operators. Figure 5 shows such a corrected domain, in which each operator in the original definition by Veloso & Blythe is implemented as a combination of two operators, A_i and B_i.

Domain: *link-chain* (corrected): laboriously linkable
Initial: Random permutation of $\{g_j | \forall j \leq n-1\}$
Operators:

Action	Precond	Add	Delete
A_i $(i = 1 \ldots n)$	$\{\neg toggle\} \cup$ $\{g_j \mid \forall j < i\}$	$\{toggle, m_i\}$	$\{g_j \mid \forall j < i\} \cup$ $\{m_j \mid \forall j \leq n, j \neq i\}$
B_i $(i = 1 \ldots n)$	$\{toggle, m_i\}$	$\{g_i\} \cup$ $\{g_j \mid \forall j \leq i-2\}$	$\{toggle, m_i\}$

Goal: Random permutation of $\{g_j | \forall j \leq n\}$
Solution: A_n, B_n, A_{n-1}, B_{n-1}, ..., A_2, B_2, A_1, B_1

Figure 5: Domain *link-chain* (corrected)

A typical pair of operators in this corrected domain is as follows (assuming $n = 4$):

Operator: A_3	Operator: B_3
Precond: $\neg toggle, g_1, g_2$	Precond: $toggle, m_3$
Adds: $toggle, m_3$	Adds: g_1, g_3
Deletes: g_1, g_2, m_1, m_2, m_4	Deletes: $toggle, m_3$

A_3 and B_3 in sequence imitate a single operator with preconditions $\{g_1, g_2\}$ that adds g_3, deletes g_2, and re-asserts g_1. The intermediate marker m_3 and *toggle* tightly tie together A_3 and B_3, so that they must always appear paired with each other.

Figure 6 shows the causal links in a successful plan. An intermediate step-pair such as (A_3, B_3) has preconditions $\{g_1, g_2\}$. These goals can be linked to multiple establishing steps. Determining the correct link is difficult for UCPOP, making the domain laboriously linkable.

3.3 *Domain* D^0S^2e*

Both the above domains had a 'good' order for solving the goals, i.e., if the goals were handled in that order, the planner would find the correct linking commitments without backtracking. Figure 7 shows the domain D^0S^2e* that was constructed so that it has no such 'good' ordering of the goals. Irrespective of the order in which the subgoals are attempted, the planner would backtrack over the linking commitments for $\neg R_i$ (a precondition of A_i^2) and $\neg Q_j$ (a precondition of $Av*$). It has to decide whether to link these preconditions to s_0, or to $Aw*/A_j^1$, thus making the domain laboriously linkable.

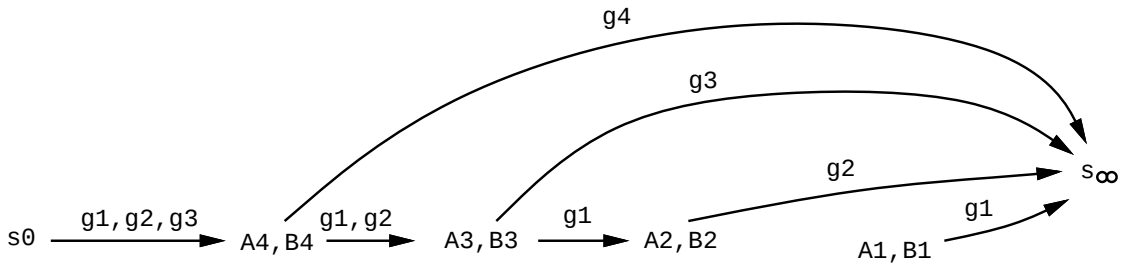

Figure 6: Domain *link-chain*: Causal links in successful plan

Domain: D^0S^2e*: laboriously linkable

Initial: $\{I_j|\forall j \leq n\} \cup \{I*\}$

Operators:

Action	Precond	Add	Delete
$A_i^1 v\ (i = 1 \ldots n)$	$\{I_i\}$	$\{Pv_i, Q_i\}$	$\{I_i\}$
$A_i^1 w\ (i = 1 \ldots n)$	$\{Pv_i\}$	$\{P_i\}$	$\{Q_i, Pv_i\}$
$A_i^2\ (i = 1 \ldots n)$	$\{P_i, \neg R_i\}$	$\{G_i\}$	$\{I*\}$
$Av*$	$\{I*\} \cup$ $\{\neg Q_j|\forall j \leq n\}$	$\{Gv*\} \cup$ $\{R_j|\forall j \leq n\}$	$\{I*\} \cup$ $\{I_j, Pv_j, G_j|\forall j \leq n\}$
$Aw*$	$\{Gv*\} \cup$ $\{\neg Q_j|\forall j \leq n\}$	$\{G*\}$	$\{Gv*\} \cup$ $\{I_j, Pv_j, R_j, G_j|\forall j \leq n\}$

Goal: Random permutation of $(\{G_j|\forall j \leq n\} \cup \{G*\})$

Solution: The operators $A_i^1 v$, $A_i^1 w$, A_i^2 $(i = 1 \ldots n)$, $Av*$ and $Aw*$ in any sequence satisfying the partial order:

$$\begin{array}{ccccc} A_1^1 v \rightarrow A_1^1 w & & & & A_1^2 \\ A_2^1 v \rightarrow A_2^1 w & \searrow & & \nearrow & A_2^2 \\ A_3^1 v \rightarrow A_3^1 w & \longrightarrow & Av* \rightarrow Aw* & \longrightarrow & A_3^2 \\ \vdots & \nearrow & & \searrow & \vdots \\ A_n^1 v \rightarrow A_n^1 w & & & & A_n^2 \end{array}$$

Figure 7: Domain D^0S^2e*

4 Linkability and the Current State

The previous section presented three laboriously linkable domains: *link-repeat*, *link-chain*, and D^0S^2e*. UCPOP finds all three domains difficult to solve because of linking commitments. However, total-order planners such as PRODIGY effectively use the current state to solve the first two domains in polynomial-time, and the size of the search-space explored is linear in the number of subgoals.

4.1 Total-Order Planners' Handling of Linkability

This large difference in performance is due to the manner in which PRODIGY 'links' open preconditions to the current state. Both PRODIGY and UCPOP generate a partially-ordered subgoal-tree containing goals and operators. As shown in Figure 8, branches are terminated when the open preconditions (such as g) are satisfied in the

current state (shown as CS).

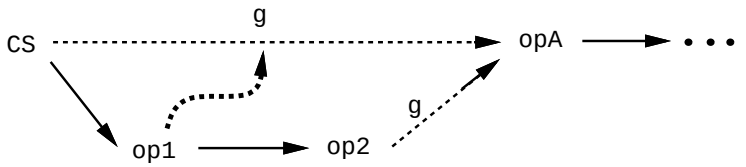

Figure 8: Tentative Links in PRODIGY

PRODIGY need not immediately apply opA (Stone *et. al.* [13]); it may choose to apply other applicable operators such as $\{op_1, op_2\}$ first. These operators may disachieve g, and even re-achieve it, as shown in Figure 8.

When PRODIGY reconsiders applying opA, it re-checks its preconditions, and re-achieves them if necessary. In this case, the link to the original current state CS is no longer viable. Instead, it would tentatively link g to op_2, as this is the new current state after op_1 and op_2 are applied.

Thus, PRODIGY used the *possibility* of the link $CS \xrightarrow{g} opA$ to terminate the expansion at opA. However, it did not *commit* to this link; it later tentatively re-linked g to op_2 when the change in the current state made that link the viable option.

4.2 Influence of Flaw Choice on Planner Efficiency

UCPOP cannot make *tentative* links in this manner, as this would compromise systematicity. In fact, when it decides to handle a flaw (such as the open precondition g of opA), it must immediately commit to how to handle that flaw (as by linking g to CS). Hence the best it can do is to *delay* handling such flaws.

The efficacy of this heuristic may be illustrated in the *link-chain* domain. UCPOP finds the incomplete plan IP_1 shown in Figure 9 without difficulty. IP_1 can be refined to the complete (successful) plan shown in Figure 6.

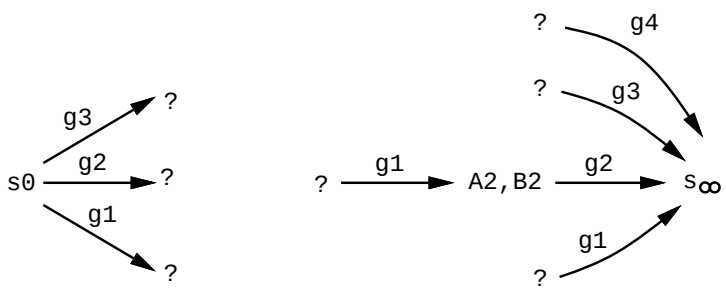

Figure 9: Domain *link-chain*: Incomplete plan IP_1

Suppose that UCPOP chooses to handle the flaw consisting of the open condition g_1 of A_2. There are four possible linking commitments: to s_0, or to a newly introduced step B_1, B_3 or B_4 (B_2 does not assert g_1). UCPOP generates a new incomplete plan for each linking possibility.

Of these, the plan with the link to s_0 is quickly found to lead to a dead-end because of an unresolvable threat to $s_0 \stackrel{g_1}{\rightarrow} A_2$.

The remaining three incomplete plans seem equally good to the plan-ranking function. Actually, only one of these leads to a successful plan, but to determine which, UCPOP has to refine all three further until it becomes obvious that the link $B_3 \stackrel{g_1}{\rightarrow} A_2$ is the correct choice (see Figure 6). However, the same linking difficulty recurs for the remaining subgoals while refining the three possibilities, thus leading to an exponential search-space.

All these difficulties arise because of UCPOP's decision to handle the condition g_1 of A_2. The 'obvious' link to s_0 does not work, and there isn't sufficient information to meaningfully evaluate the remaining three choices.

Instead, if UCPOP deferred handling this flaw in favour of handling other open conditions, it would eventually reach the incomplete plan IP_2 shown in Figure 10. This plan still has the flaw of the open condition g_1 of A_2 (among others). If UCPOP decides to handle this flaw now, there are six linking possibilities (instead of the original four): link to s_0, link to the pre-existing B_3 or B_4 (the link to the pre-existing B_1 is ruled out by ordering constraints not shown in the figure), or link to a newly introduced B_1, B_3 or B_4. However, the links to s_0 and the pre-existing B_4 quickly lead to unresolvable threats. Of the remaining four possibilities (pre-existing B_3 or newly introduced B_1, B_3 or B_4), the correct choice (link to pre-existing B_3) is ranked higher than the others, because it does not necessitate the introduction of a new step. Hence UCPOP would prefer to explore that plan, and would quickly find the complete solution.

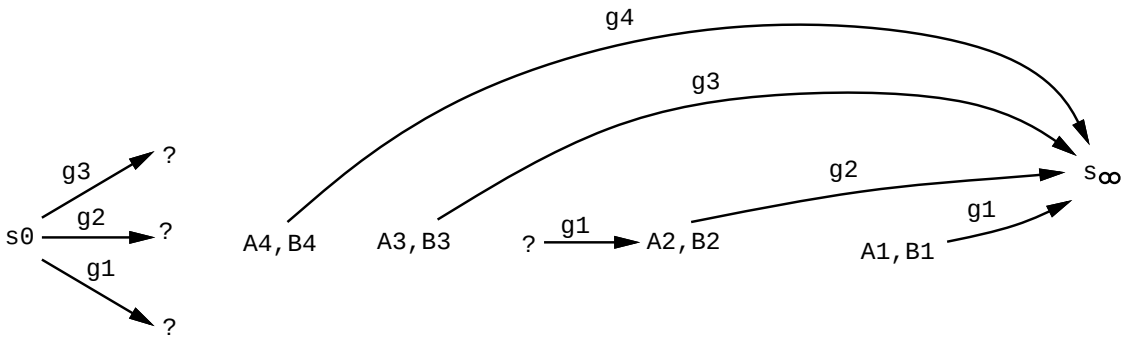

Figure 10: Domain *link-chain*: Incomplete plan IP_2

4.3 Approximating the Current State in UCPOP

UCPOP must eventually handle all flaws in an incomplete plan to obtain a complete plan. Nevertheless, as shown in the previous subsection, the order in which the flaws are handled significantly influences its efficiency, because the planner's ability to evaluate the different commitment possibilities for a given flaw is influenced by the flaws that have *already* been handled.

Figures 9 and 10 showed how an open condition that may be linked to the initial state is a particularly difficult flaw to handle. If that link is correct, then it can be made easily at any time, and deferring doing so does not adversely affect the search. On the other hand, if that link leads to a dead-end, then the planner may have little

information to decide among the remaining linking possibilities. Total-order planners handle this situation by effectively making a 'tentative' link that is subject to later revision. In UCPOP, a similar effect can be obtained by a heuristic that defers such flaws until later in the planning process.

The initial state is merely a special case of the current state. However the current state is not well-defined in UCPOP, as doing such *temporal projections* is NP-complete (see Nebel & Bäckström [11]). Hence the heuristic was generalized to delay handling open conditions that may be linked to a pre-existing step. The effects asserted by pre-existing steps approximate the goals that *may* be true in the current state. The approximation is exact at s_0, but gets progressively worse nearer s_∞. To allow for the resultant uncertainty in the current state, open conditions nearer s_∞ are handled in preference to those further from s_∞. For efficiency, the heuristic, termed DELAY, checks neither ordering constraints nor threats that may invalidate the candidate links. It was implemented by modifying UCPOP's flaw-ranking function, as shown in Figure 11. The flaws are ranked according to the order listed in the table.

UCPOP	UCPOP + DELAY
1. Sibling threats	1. Sibling threats
2. Monotonic open conditions	2. Monotonic open conditions
3. Remaining threats	3. Remaining threats
4. Remaining open conditions	4. Remaining open conditions not linkable to a pre-existing step
	5. Remaining open conditions linkable to a pre-existing step ordered by closeness to s_∞ (closer preferred)

Figure 11: Flaw-Ranking Function

PRODIGY uses the current state to guide operator selection as well, preferring operators that have *fewer* preconditions unmet in the current state. This is a weakened version of the *conspiracy search* described by Elkan [5]. UCPOP may partially imitate this strategy by modifying the plan-ranking function to disregard flaws consisting of open conditions linkable to pre-existing steps.

5 Results

Figures 12, 13 and 14 show the results obtained in the laboriously linkable domains. In each case, the behaviour of UCPOP using the default ranking function (as augmented by the handling of sibling threats and monotonic goals) is compared against the behaviour obtained using DELAY. The size of the search-space explored, and the time taken, are plotted against n, the number of goals. Randomly generated problems were used, and error-bars are shown unless they were insignificant. DELAY significantly improved performance in all three domains.

D^0S^2e* is intractable in UCPOP because of linkability, and in PRODIGY because of operator-orderings. In fact, the domain is *non-serializable* (Barrett & Weld [1, 2]) in both planners. Hence it is remarkable that DELAY combined UCPOP's partial order with an approximation of PRODIGY's current state to yield vastly better performance

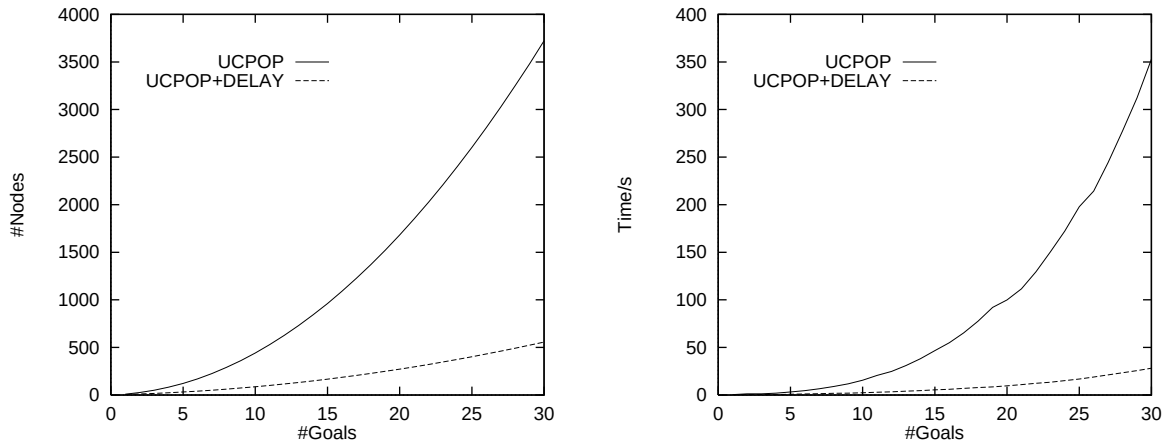

Figure 12: Performance of DELAY on *link-repeat*

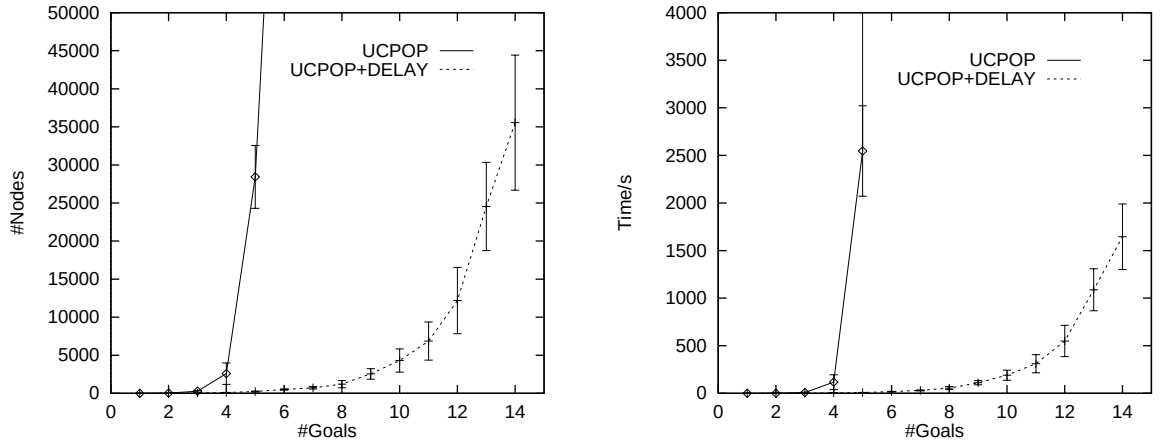

Figure 13: Performance of DELAY on *link-chain*

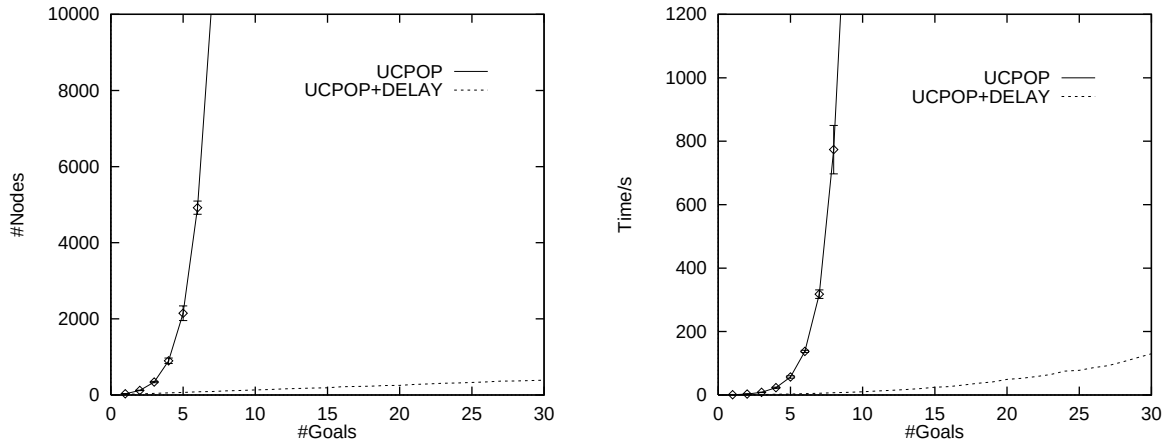

Figure 14: Performance of DELAY on D^0S^2e*

than that obtained by either planner alone.

6 Conclusion

Both the use of a partial order and the maintenance of a current state are useful in a non-linear, means-ends planner. Unfortunately, the two are mutually incompatible. DELAY attempts to explore an intermediate possibility: usage of an approximation of the current state without sacrificing the partial order.

Naturally, DELAY need not be the only, or even the best, such compromise. For example, it ignores operator ordering constraints in the interests of efficiency. Perhaps a better approximation to the current state can be maintained at the cost of doing more work at each node (but see [11]). The ideal point on this tradeoff would probably vary according to the domain being handled. It will probably be easier to modify a partial-order planner to be more like a total-order planner than the converse (see [7]).

By default, UCPOP handles open conditions in a random order. PRODIGY overcomes linkability by using the current state to selectively order the handling of open conditions. DELAY attempts to similarly overcome linkability by using an approximation of the current state. The approximation cannot give as much guidance as the actual current state, and hence the ordering of open conditions is not as good as PRODIGY's, in general. It is difficult to conceive of a domain, however, in which the ordering suggested by the approximation is *worse* than a random ordering.

Reasoning about causal-links is easy, but they constitute a commitment by the planner. This is avoided by planners such as TWEAK [4] that make no causal links [8, 10]. The penalty paid by such planners is the increased cost and complexity of using Chapman's Modal Truth Criterion (MTC) [4] to handle interactions between subplans. Knoblock & Yang [8] argued that causal-link planners perform better when there are many negative threats, whereas MTC planners perform better when there are many positive threats (many operators that assert the same goal in incomplete plans). As the negative threats alone in the three domains *link-repeat*, *link-chain* and D^0S^2e* make them laboriously linkable, MTC planners should find these domains difficult to handle. In that case, such planners may benefit as well from a heuristic such as DELAY to order the handling of flaws.

In summary, DELAY merges the strengths of partial-order and total-order planners to some extent without compounding their weaknesses.

References

[1] Anthony Barrett and Daniel S. Weld. Characterizing subgoal interactions for planning. In *Proceedings of 13th IJCAI*, pages 1388–1393, August 1993.

[2] Anthony Barrett and Daniel S. Weld. Partial-order planning: evaluating possible efficiency gains. *Artificial Intelligence*, 67(1):71–112, May 1994.

[3] Jaime G. Carbonell, Jim Blythe, Oren Etzioni, Yolanda Gil, Robert Joseph, Dan Kahn, Craig Knoblock, Steven Minton, Alicia Pérez, Scott Reilly, Manuela Veloso, and Xuemei Wang. Prodigy4.0: The manual and tutorial. Manual CMU-CS-92-150, Carnegie Mellon University, 1992.

[4] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[5] Charles Elkan. Conspiracy numbers and caching for searching and/or trees and theorem-proving. In N. S. Sridharan, editor, *Proceedings of 11th IJCAI*, volume 1, pages 341–346. Morgan Kaufmann, August 1989.

[6] Joachim Hertzberg and Alexander Horz. Towards a theory of conflict detection and resolution in nonlinear plans. In N. S. Sridharan, editor, *Proceedings of 11th IJCAI*, volume 2, pages 937–942. Morgan Kaufmann, August 1989.

[7] S. Kambhampati and B. Srivastava. Universal classical planner: An algorithm for unifying state-space and plan-space planning. In *Proceedings of Third European Workshop on Planning*, 1995.

[8] Craig A. Knoblock and Qiang Yang. A comparison of SNLP and TWEAK planning algorithms. In *Working Notes of the AAAI Spring Symposium on Foundations of Automatic Planning: The Classical Approach and Beyond*, pages 73–77, 1993.

[9] David McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, volume 2, pages 634–639, July 1991.

[10] Steven Minton, John Bresina, and Mark Drummond. Commitment strategies in planning: A comparative analysis. In *Proceedings of 12th IJCAI*, volume 1, pages 259–265. Morgan Kaufmann, August 1991.

[11] Bernhard Nebel and Christer Bäckström. On the computational complexity of temporal projection, planning and plan validation. *Artificial Intelligence*, 66(1):125–160, March 1994.

[12] J. S. Penberthy and D. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of 3rd International Conference on Principles of Knowledge Representation and Reasoning*, pages 103–114, October 1992.

[13] Peter Stone, Manuela Veloso, and Jim Blythe. The need for different domain-independent heuristics. In *Proceedings of Second International Conference on Artificial Intelligence Planning Systems*, pages 164–169, June 1994.

[14] Manuela Veloso and Jim Blythe. Linkability: Examining causal link commitments in partial-order planning. In *Proceedings of Second International Conference on Artificial Intelligence Planning Systems*, pages 170–175, June 1994.

[15] Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.