Reviving Partial Order Planning

XuanLong Nguyen & Subbarao Kambhampati*
Department of Computer Science and Engineering
Arizona State University, Tempe AZ 85287-5406
Email: {xuanlong,rao}@asu.edu

Abstract

This paper challenges the prevailing pessimism about the scalability of partial order planning (POP) algorithms by presenting several novel heuristic control techniques that make them competitive with the state of the art plan synthesis algorithms. Our key insight is that the techniques responsible for the efficiency of the currently successful planners–viz., distance based heuristics, reachability analysis and disjunctive constraint handling–can also be adapted to dramatically improve the efficiency of the POP algorithm. We implement our ideas in a variant of UCPOP called REPOP[1]. Our empirical results show that in addition to dominating UCPOP, REPOP also convincingly outperforms Graphplan in several "parallel" domains. The plans generated by REPOP also tend to be better than those generated by Graphplan and state search planners in terms of execution flexibility.

1 Introduction

Most recent strides in scaling up planning have centered around two dominant themes - heuristic state space planners, exemplified by UNPOP[20], HSP-R[3], and CSP-based planners, exemplified by Graphplan[2] and SATPLAN [14] . This is in stark contrast to planning research up to five years ago, when most of the efforts were focused on scaling up partial order planners[19; 27; 15; 23; 11; 13]. Despite such efforts, the partial order planners continue to be extremely slow and are not competitive with the fastest state search-based and CSP-based planners. Indeed, the recent advances in plan synthesis have generally been (mis)interpreted as establishing the supremacy of state space and CSP-based approaches over POP approaches.

Despite its current scale-up problems, partial order planning remains attractive over state space and CSP-based planning for several reasons. The least commitment inherent in partial order planning makes it one of the more open planning frameworks. This is evidenced by the fact that most existing architectures for integrating planning with execution, information gathering, and scheduling are based on partial order planners. In [25], Smith argues that POP-based frameworks offer a more promising approach for handling domains with durative actions, and temporal and resource constraints as compared to other planning approaches. In fact, most of the known implementations of planning systems capable of handling temporal and durative constraints –including IxTET [6] as well as NASA's RAX [10]–are based on the POP algorithms. Even for simpler planning domains, partial order planners search for and output partially ordered plans that offer a higher degree of execution flexibility. In contrast, none of the known state space planners can find parallel plans efficiently [8], and CSP planners such as Graphplan only generate a very restricted class of parallel plans (see Section 5).

The foregoing motivates the need for improving the efficiency of POP algorithms. We show in this paper that the insights and techniques responsible for the advances in plan synthesis made in the recent years in the context of state-based and CSP-based planners are largely adaptable to POP algorithms. In particular, we present novel methods for adapting distance based heuristics, reachability analysis and disjunctive constraint processing techniques to POP algorithms. Distance-based heuristics are used as the basis for ranking partial plans and as flaw selection methods. The other two techniques are used for efficiently enforcing the consistency of the partial plans–by detecting implicit conflicts and resolving them.

Our methods help scale up POP algorithms dramatically–making them competitive with respect to state space planners, while preserving their flexibility. We present empirical studies showing that REPOP, a version of UCPOP [27] enhanced by our ideas, can perform competitively with other existing approaches in many planning domains. In particular, REPOP appears to scale up much better than Graphplan in the parallel domains we tried. More importantly, the solutions REPOP generates are generally shorter in length, and provide significantly more execution flexibility [25].

The paper is organized as follows. In the next section we will briefly review the basics of the POP algorithm. Section 3 describes how distance based heuristics can be adapted to rank partial plans. Section 4 shows how unsafe links flaws can be generalized and resolved efficiently. Section 5 reports empirical evaluations of the techniques that have been described. Section 6 discusses related work, and Section 7 summarizes the contributions of this work.

2 Background on Partial Order Planning

In this paper we consider the simple STRIPS representation of classical planning problems, in which the initial world state I,

*This research is supported in part by the NSF grant IRI-9801676, AFOSR grant F20602-98-1-0182 and the NASA grants NAG2-1461 and NCC-1225. We thank David Smith, Malik Ghallab, Austin Tate, Dan Weld, Terry Zimmerman, Biplav Srivastava, Minh B. Do, and the IJCAI referees for critical comments on the previous drafts of this paper.

[1] UCPOP [27] → UNPOP [20] → REPOP. REPOP's source code is available from *http://rakaposhi.eas.asu.edu/repop.html.*

goal state G and the set of deterministic actions Ω are given. Each action $a \in \Omega$ has a precondition list and an effect list, denoted respectively as $Prec(a), Eff(a)$. The planning problem involves finding a plan that when executed from the initial state I will achieve the goal G.

A tutorial introduction to POP algorithms can be found in [27]. We will provide a brief review here. Most POP algorithms can be seen as searching in the space of partial plans. A **partial plan** is a five-tuple: $\mathcal{P} = (\mathcal{A}, \mathcal{O}, \mathcal{L}, \mathcal{OC}, \mathcal{UL})$, where $\mathcal{A} \subseteq \Omega$ is a set of (ground) actions,[2] $\mathcal{O}$ is a set of ordering constraints over $\mathcal{A}$, and $\mathcal{L}$ is a set of causal links over $\mathcal{A}$.[3] A causal link is of the form $a_i \xrightarrow{p} a_j$, and denotes a commitment by the planner that the precondition p of action a_j will be supported by an effect of action a_i. $\mathcal{OC}$ is a set of open conditions, and $\mathcal{UL}$ is a set of unsafe links. An **open condition** is of the form (p, a), where $p \in Prec(a)$ and $a \in \mathcal{A}$, and there is *no* causal link $b \xrightarrow{p} a \in \mathcal{L}$. Loosely speaking, the open conditions are preconditions of actions in the partial plan which have not yet been achieved in the current partial plan. A causal link $a_i \xrightarrow{p} a_j$ is called **unsafe** if there exists an action $a_k \in \mathcal{A}$ such that (i) $\neg p \in Eff(a_k)$ *and* (ii) $\mathcal{O} \cup \{a_i \prec a_k \prec a_j\}$ is consistent. In such a case, a_k is also said to **threaten** the causal link $a_i \xrightarrow{p} a_j$. Open conditions and unsafe links are also called **flaws** in the partial plan. Therefore a solution plan can be seen as a partial plan with no flaws (i.e., $\mathcal{OC} = \emptyset$ and $\mathcal{UL} = \emptyset$).

The POP algorithm starts with a null partial plan $\mathcal{P}$ and keeps refining it until a solution plan is found. The **null partial plan** contains two dummy actions $a_0 \prec a_\infty$ where the preconditions of a_∞ correspond to the top level goals of the problem, and the effects of a_0 correspond to the conditions in the initial state. The null plan has no causal links or unsafe link flaws, but has open condition flaws corresponding to the preconditions of a_∞ (top level goals).

A **refinement** step involves selecting a flaw in the partial plan $\mathcal{P}$, and *resolving* it, resulting in a new partial plan. When the flaw chosen is an open condition (p, a), an action b needs to be selected that achieves p. b can be a new action in Ω, or an action that is already in $\mathcal{A}$. The sets $\mathcal{OC}$, $\mathcal{O}$, $\mathcal{L}$ and $\mathcal{UL}$ also need to be updated with respect to b. Secondly, when the flaw chosen is an unsafe link $a_i \xrightarrow{p} a_j$ that is threatened by action a_k, it can be repaired by either **promotion**, i.e adding ordering constraint $a_k \prec a_i$ into $\mathcal{O}$, or **demotion**, i.e adding $a_j \prec a_k$ into $\mathcal{O}$.

The efficiency of POP algorithms depends critically on the way partial plans are selected from the search queue, and the strategies used to select and resolve the flaws. In Section 3 we present several distance-based heuristics for ranking partial plans in the search queue. Section 4 introduces the disjunctive constraint representation for efficiently handling unsafe link flaws, and reachability analysis for generalizing the notion of unsafe links to include implicit conflicts in the plan.

3 Heuristics for ranking partial plans

In choosing a plan from the search queue for further refinement, we are naturally interested in plans that are likely to lead to a solution with a minimum number of refinements (flaw resolutions). As we handle the unsafe links in a significantly different way than standard UCPOP (see Section 4), the only remaining category of flaws to be resolved are open condition flaws. Consequently one way of ranking plans in the search queue is to estimate the minimum number of new actions needed to resolve all the open condition flaws.

Definition 1 (h*) *Given a partial plan $\mathcal{P}$, let $h^*(\mathcal{P})$ denote the minimum number of new actions that need to be added to $\mathcal{P}$ to make it a solution plan.*

$h^*(\mathcal{P})$ can be seen as the number of actions that, when executed from the initial state I in some order, will achieve the set of subgoals $S = \{p | (p, a) \in \mathcal{OC}\}$. In this sense, this is similar to estimating the number of actions needed to achieve a state from the given initial state in state search planners [3; 21], but for two significant differences: (i) the propositions in S are not necessarily in the same world state and (ii) the set of actions that achieve S cannot conflict with the set of actions and causal links already present in $\mathcal{P}$.

A well-known heuristic for estimating h^* involves simply counting the number of open conditions in the partial plan [13].

Heuristic 1 (Open conditions heuristic) $h_{oc}(\mathcal{P}) = |\mathcal{OC}|$

This estimate is neither admissible nor informed in many domains, because it treats every open condition equally. In particular, it is ineffective when some open conditions require more actions to achieve than others.

We would like to have a closer estimate of h^* function without insisting on admissibility. To do this, we need to take better account of subgoal interactions[21]. Accounting for the negative interactions in estimating h^* can be very tricky, and is complicated by the fact that the subgoals in S may not be in the same state. Thus we will start by ignoring the negative interactions. This has three immediate consequences: (i) the set of unsafe links $\mathcal{UL}$ becomes empty. (ii) the actions needed in achieving a set of subgoals S will have no conflicts with the set of actions $\mathcal{A}$ and and the causal links $\mathcal{L}$ already present in $\mathcal{P}$. and (iii) a subgoal p once achieved from the initial state can never become untrue. Given these consequences, it does not matter much that the subgoals in S are not necessarily present in the same world state, since the minimum number of actions needed for achieving such a set of subgoals in any given temporal ordering is the same as the minimum cost of achieving a state comprising all those subgoals.

The foregoing justifies the adaptation of many heuristic estimators for ranking the goodness of states in state search planners. Most of the early heuristic estimators used in state search not only ignore negative interactions, but also make the stronger assumption of subgoal independence [3; 20]. A few of the recent ones, [21; 9] however account for the positive interactions among subgoals (while still ignoring the negative interactions). It is this latter class of heuristics that we focus on for use in partial order planning. Specifically, to account for the positive interactions, we exploit the ideas for estimating the cost of achieving a set of subgoals S using a serial planning graph.[4]

Specifically, we build a planning graph starting from the initial state I. Let $lev(p)$ be the index of the level in the planning graph that a proposition p first appears, and $lev(S)$ be the index of the first level at which all propositions in S appear. Let p_S be the proposition in S such that $lev(p_S) = \max_{p_i \in S} lev(p_i)$.

[2] Although partial order planners are capable of handling partially instantiated action instances, we restrict our attention to ground action instances.

[3] Strictly speaking $\mathcal{A}$ should be seen as a set of "steps", where each step is mapped to an action instance [15].

[4] We assume that the readers are familiar with the planning graph data structure, which is used in Graphplan algorithm[2].

p_S will *possibly* be the last proposition in S that is achieved during execution. Let a_S be an action in the planning graph that achieves p_S in the level $lev(p_S)$. We can achieve p_S by adding a_S to the plan. Introduction of a_S changes the set of goals to be achieved to $S' = S + Prec(a_S) - Eff(a_S)$. We can express the cost of S in terms of the cost of a_S and S':

$$cost(S) := cost(a_S) + cost(S + Prec(a_S) - Eff(a_S)) \quad (1)$$

where $cost(a_S) = 1$ if $a_S \notin \mathcal{A}$ and 0 otherwise. Since $lev(Prec(a_S))$ is strictly smaller than $lev(p_S)$, recursively applying Equation 1 to its right hand side will eventually express $cost(S)$ in terms of $cost(I)$ (which is zero), and the costs of actions a_S. The process is quite efficient as the number of applications is bounded by $lev(S)$.

Heuristic 2 (Relax heuristic) *$h_{relax}(\mathcal{P}) = cost(S)$, where $S = \{p|(p,a) \in \mathcal{OC}\}$, and $cost(S)$ is computed using the recurrence relation 1.*

Given such a heuristic estimate, plans in the search queue are ranked with the evaluation function: $f(\mathcal{P}) = |\mathcal{A}| + w * h(\mathcal{P})$. The parameter w is used to increase the greediness of the heuristic search and is set to 5 by default.

4 Enforcing consistency of partial plans

The consistency of a partial plan is ensured through the handling of its unsafe links. In this section we describe two ways of improving this phase. The first involves posting disjunctive constraints to resolve unsafe links. The second involves detecting implicit conflicts (unsafe links) using reachability analysis.

4.1 Disjunctive representation of ordering constraints

Normally, an unsafe link $a_i \xrightarrow{p} a_j$ that is in conflict with action a_k is resolved by either promotion or demotion, that is, splitting the current partial plan into two partial plans, one with the constraint $a_k \prec a_i$, and the other with the constraint $a_j \prec a_k$. A problem with this premature splitting is that a single failing plan gets unnecessarily multiplied into many descendant plans poisoning the search queue significantly. A much better idea, first proposed in [16], is to resolve the unsafe link by posting a disjunctive ordering constraint that captures both the promotion and demotion possibilities, and incrementally simplify these constraints by propagation techniques. This way, we can detect many failing plans before they get selected for refinement.

Specifically, an unsafe causal link $a_i \xrightarrow{p} a_j$ that is in conflict with action a_k can be resolved by simply adding a disjunctive ordering constraint $(a_k \prec a_i) \vee (a_j \prec a_k)$ to the plan.

We use the following procedure for simplifying the disjunctive orderings. Whenever an open condition (p, a) is selected and resolved by either adding a new action or reusing an action b in the partial plan, we add a new ordering constraint $b \prec a$ to $\mathcal{O}$, followed by repeated application of the constraint propagation rules below:

- $(a_1 \prec a_2) \in \mathcal{O} \wedge (a_2 \prec a_3) \in \mathcal{O} \Rightarrow \mathcal{O} \leftarrow \mathcal{O} \cup (a_1 \prec a_3)$
- $(a_1 \prec a_2) \in \mathcal{O} \wedge (a_2 \prec a_1) \in \mathcal{O} \Rightarrow$ False
- $(a_1 \prec a_2) \in \mathcal{O} \wedge (a_2 \prec a_1 \vee a_3 \prec a_4) \in \mathcal{O} \Rightarrow$
  $\mathcal{O} \leftarrow \mathcal{O} \cup (a_3 \prec a_4)$
  $\mathcal{O} \leftarrow \mathcal{O} - (a_2 \prec a_1 \vee a_3 \prec a_4)$

The first two propagation rules are already done as part of POP algorithm to ensure the transitive consistency of ordering constraints. The third rule is a unit propagation rule over ordering constraints. This propagation both reduces the disjunction and detects infeasible plans ahead of time. When all the open conditions have already been established and there are still disjunctive constraints left in the plan, the remaining disjunctive constraints are then split into the search space [16].

4.2 Detecting and Resolving implicit conflicts through reachability analysis

Although the unsafe link detection and resolution steps in the POP algorithm are meant to enforce consistency of the partial plan, often times they are too weak to detect implicit inconsistencies. In particular, the procedure assumes that a link $a_i \xrightarrow{p} a_j$ is threatened by an action a only if a has an effect $\neg p$. Often a might have an effect q (or precondition r) such that no legal state can have p and q (or p and r) true together. Detecting and resolving such implicit interactions can be quite helpful in weeding out inconsistent partial plans from the search space.

In order to do implicit conflict detection as described above, we need to have (partial) information about the properties of reachable states. Interestingly, such reachability information has played a significant role in the scale-up of state space planners, motivating the development of procedures for identifying mutex constraints, state invariants and memos etc. [2; 7; 5] (we shall henceforth use the term mutex to denote all these types of reachability information). One simple way of producing reachability information is to expand Graphplan's planning graph structure, armed with mutex propagation procedure [2]. The mutexes present at the level where the graph levels off are state invariants [21].

Exploiting the reachability information to check consistency of partial plans requires identifying the feasibility of the world states that any eventual execution of the partial plan must pass through. Although partial order plans normally do not have explicit state information associated with them, it is nevertheless possible to provide partial characterization of the states their execution must pass through. Specifically, we define the general notion of cutsets as follows:

Definition 2 (Cutsets) *Pre- and post- cutsets, C^- and C^+ of an action a_k in a plan $\mathcal{P}$ are defined as $C^-(a_k) = Prec(a_k) \cup L(a_k)$, and $C^+(a_k) = Eff(a_k) \cup L(a_k)$, where $L(a_k)$ is the set of all conditions p such that there exists a link $a_i \xrightarrow{p} a_j$ where a_i is necessarily before a_k, and a_j is necessarily after a_k*

The pre- and post-cutsets of an action can be seen as partial description of world states that *must* hold before and after the action a_k. If these partial descriptions violate the properties of the reachable states, then clearly the partial plan cannot be refined into an executable solution.

Proposition 1 *If there exists a cutset that contains a mutex, then the partial plan is provably invalid and can be pruned from the search queue.*

While this proposition allows us to detect and prune inconsistent plans, it is often inefficient to wait until the plan becomes inconsistent. Detecting and resolving implicit conflicts is essentially a more active approach that *prevents* a partial plan from becoming inconsistent by this proposition. Specifically, we generalize the notion of unsafe links as follows:

Definition 3 *An action a_k is said to have a* **conflict** *with a causal link $a_i \xrightarrow{p} a_j$ if (i) $\mathcal{O} \cup \{a_i \prec a_k \prec a_j\}$ is consistent and (ii) either $Prec(a_k) \cup \{p\}$ or $Eff(a_k) \cup \{p\}$ contains a mutex. A causal link $a_i \xrightarrow{p} a_j$ is* **unsafe** *if it has a conflict with some action in the partial plan.*

These notions of conflict and unsafe link subsume the original notions of *threat* and *unsafe link* introduced in Section 2, because $\neg p \in Eff(a_k)$ also implies that $Eff(a_k) \cup \{p\}$ is a mutex. Therefore the generalized notion of unsafe links result in detecting a larger number of (implicit) conflicts (unsafe links) present in a partial plan.

Once the implicit conflicts are detected, they are resolved by posting disjunctive orderings as described in the previous subsection. As we shall see later, the combination of disjunctive constraints and detection of implicit conflicts through reachability information leads to quite robust improvements in planning performance.

5 Empirical Evaluation

We have implemented the techniques introduced in this paper on top of UCPOP[27], a popular partial order planning algorithm. We call the resulting planner REPOP. As mentioned in Section 2, both UCPOP and REPOP are given ground action instances, and thus neither of them have to deal with variable binding constraints. Both UCPOP and REPOP use the LIFO as the order in which open condition flaws are selected for resolution. Our empirical studies compare REPOP to UCPOP as well as Graphplan[2] and AltAlt[21], which represent two currently popular approaches (CSP search and state space search) in plan synthesis. All these planners are written in Lisp. In the case of Graphplan, we used the Lisp implementation of the original algorithm, enhanced with EBL and DDB capabilities [17]. AltAlt [22] is a state-of-the-art heuristic regression state search planner, that has been shown to be significantly faster than HSP-R [3]. The empirical studies are conducted on a 500 MHz Pentium-III with 256MB RAM, running Linux. The test suite of problems were taken from several benchmark planning domains from the literature. Some of these, including gripper, rocket world, blocks world and logistics are "parallel" domains which admit solutions with loosely ordered steps, while others, such as grid world and travel world admit only serial solutions.

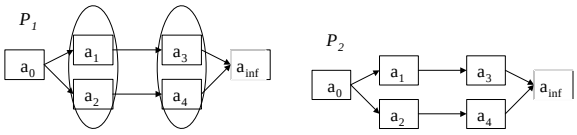
Efficiency of Synthesis: In Table 1, we report the total running times for the REPOP algorithm, including the preprocessing time for computing the mutex constraints (using bilevel planning graph structures [18]). Table 1 shows that REPOP exhibits dramatic improvements from its base planner, UCPOP, in gripper, logistics and rocket domains–all of which are "parallel domains." For instance, REPOP is able to comfortably generate plans with up to 70 actions in logistics and gripper domains, a feat that has hither-to been significantly beyond the reach of partial order planners. More interesting is the comparison between REPOP and the non-partial order planners. In the parallel domains, REPOP manages to outperform Graphplan. Although REPOP still trails state search planners such as AltAlt, these latter planners can only generate serial plans.

Despite the impressive performance of the REPOP over parallel domains, it remains ineffective in "serial" domains including the grid, 8-puzzle and travel world, which admit only totally ordered plan solutions. We suspect that part of the reason for this may be the inability of our heuristics to adequately account for negative interactions. Indeed, we found that the normal open conditions heuristic h_{oc} is better than our relaxed heuristic on these problems. It may also be possible that the least commitment strategies employed by the POP algorithms become a burden in serial domains, since eventually all actions need to be ordered with respect to each other. One silverlining in this matter is that most of the domains where POP algorithms are supposed to offer advantages are likely to be parallel domains from the planner's perspective–either because the actions will have durations (making the serial/parallel distinction moot) or because we want solution output by the planner to offer some degree of scheduling flexibility.

Plan Quality: We also evaluated the quality of plans generated by REPOP, since plan quality is seen as an important issue favoring POP algorithms. To quantify the quality of plans generated, we consider three metrics: (i) the cumulative cost of the actions included in the plan (ii) the minimum time needed for executing the plan and (iii) the scheduling (execution) flexibility of the plan.

For actions with uniform cost, the action cost is equal to the number of actions in the plan. Table 1 shows that REPOP produces plans with lower action cost compared to both Graphplan and AltAlt in all but one problem (*rocket-ext-b*).

We measure the minimum execution time in terms of the *makespan* of the plan, which is loosely defined as the minimum number of time steps needed to execute the plan (taking the possibility of concurrent execution into consideration). Makespan for the plans produced by Graphplan is just the number of steps in the plan, while the makespan for plans produced by AltAlt (and other state space planners) is equal to the number of actions in the plan. For a partially ordered plan P generated by REPOP, the makespan is simply the length of the longest path between a_0 and a_∞. Specifically, $makespan(P) = \max_{a \in P} est(a)$, where $est(a)$ is the earliest start time step for the (instantaneous) action a. To compute est, we can start by initializing est to 0 for all $a \in P$. Next, we repeatedly update them until fixpoint using the following rule: For all $(a_i \prec a_j) \in \mathcal{O}$, $est(a_j) := \max\{est(a_j), 1 + est(a_i)\}$. Table 1 shows that the solution plans generated by REPOP are highly parallel, since the makespans of these plans are significantly smaller than the total number of actions. Graphplan's solutions have smaller makespans in several problems, but at the expense of having substantially larger number of actions.



(a) A parallel plan generated by Graphplan

(b) A partially ordered plan

Figure 1: Example illustrating the execution flexibility of partially ordered plans over (Graphplan's) parallel plans.

Finally, we measure the execution flexibility of a plan in terms of the number of actions in the plan that do not have any precedence relations among them. The higher this measure, the higher the number of orders in which a plan can be executed ("scheduled"). Figure 1 illustrates a parallel plan P_1 and a partially ordered plan P_2, which are generated by Graphplan and REPOP, respectively. Both plans have 4 actions and a makespan value of 2, but P_2 is noticeably more flexible than

Problem	UCPOP (time)	REPOP			Graphplan			AltAlt	
		Time	#A/ #S	#flex	Time	#A/ #S	#flex	Time	#A
gripper-8	–	1.01	21/ 15	.57	66.82	23/ 15	.69	.43	21
gripper-10	–	2.72	27/ 19	.59	47min	29/ 19	.71	1.15	27
gripper-12	–	6.46	33/ 23	.61	–	–	–	1.78	33
gripper-20	–	81.86	59/ 39	.68	–	–	–	15.42	59
rocket-ext-a	–	8.36	35/ 16	2.46	75.12	40/ 7	7.15	1.02	36
rocket-ext-b	–	8.17	34/ 15	7.29	77.48	30/ 7	4.80	1.29	34
logistics.a	–	3.16	52/ 13	20.54	306.12	80/ 11	6.58	1.59	64
logistics.b	–	2.31	42/ 13	20.0	262.64	79/ 13	5.34	1.18	53
logistics.c	–	22.54	50/ 15	16.92	–	–	–	4.52	70
logistics.d	–	91.53	69/ 33	22.84	–	–	–	20.62	85
bw-large-a(9)	45.78	(5.23) –	(8/ 5) –	(2.75) –	14.67	11/4	2.0	4.12	9
bw-large-b(11)	–	(18.86) –	(11/ 8) –	(3.28) –	122.56	18/ 5	2.67	14.14	11
bw-large-c(15)	–	(137.84) –	(17/ 10) –	(5.06) –	–	–	–	116.34	19
travel1	149.74	(4.32) –	(9/9) –	(0.0) –	0.32	9/ 9	0.0	0.53	9
simple-grid1	56.40	(0.0) –	(6/ 6) –	(0.0) –	0.42	6/ 6	0.0	1.48	6
simple-grid2	–	(2.43) –	(10/ 10) –	(0.0) –	0.95	10/ 10	0.0	1.58	10
simple-grid3	–	–	–	–	3.96	16/ 16	0.0	15.12	16

Table 1: "Time" shows *total* running times in cpu seconds, and includes the time for any required preprocessing. Dashed entries denote problems for which no solution is found in 3 hours or 250MB. Parenthesized entries (for blocks world, travel and grid domains) indicate the performance of REPOP when using h_{oc} heuristic. #A and #S are the action cost and time cost respectively of the solution plans. "flex" is the execution flexibility measure of the plan (see below).

P_1, since P_1 implies ordering constraints such as $a_1 \prec a_4$ and $a_2 \prec a_3$, but P_2 does not. To capture this flexibility, we define, for each action a, $flex(a)$ as the number of actions in the plan that *do not* have any (direct or indirect) ordering constraint with a. $flex(P)$ is defined as the average value of $flex$ over all the actions in the plan. It is easy to see that for a serial plan P, $\forall_{a \in P} flex(a) = 0$, and consequently $flex(P) = 0$. In our example in Figure 1, $flex(a) = 1$ for all a in P_1, and $flex(a) = 2$ for all a in P_2. Thus, $flex(P_1) = 1$ and $flex(P_2) = 2$. It is easy to see that P_2 can be executed in more ways than P_1. Table 1 reports the $flex()$ value for the solution plans. As can be seen, plans generated by REPOP have substantially larger average values of $flex$ than Graphplan in blocks world and logistics, and similar values in gripper. Graphplan produces a more flexible plan in only one problem in the rocket domain.

Problem	UCPOP	+CE	+HP	+HP+CE
gripper-8	*	6557/ 3881	*	1299/ 698
gripper-10	*	11407/ 6642	*	2215/ 1175
gripper-12	*	17628/ 10147	*	3380/ 1776
gripper-20	*	*	*	11097/ 5675
rocket-ext-a	*	*	30110/ 17768	7638/ 4261
rocket-ext-b	*	*	85316/ 51540	28282/ 16324
logistics.a	*	*	411/ 191	847/ 436
logistics.b	*	*	920/ 436	542/ 271
logistics.c	*	*	4939/ 2468	7424/ 4796
logistics.d	*	*	*	16572/ 10512

Table 2: Ablation studies to evaluate the individual effectiveness of the new techniques: heuristic for ranking partial plans (HP) and consistency enforcement (CE). Each entry shows the number of partial plans generated and expanded. Note that REPOP is essentially UCPOP with HP and CE. (*) means no solution found after generating 100,000 nodes.

Before ending the discussion on plan quality, we should mention that it is possible to use post-processing techniques to improve the quality of plans produced by state-space and CSP-based planners. However, such post-processing, in addition to being NP-hard in general [1], does not provide a satisfactory solution for online integration of the planner with other modules such as schedulers and executors [6; 25].

Ablation Studies: We now evaluate the individual effectiveness of each of the acceleration techniques, viz., heuristic functions for ranking partial plans (HP), and consistency enforcement (CE).Table 2 shows the number of partial plans generated and expanded in the search when each of these techniques is added into the original UCPOP. We restrict our focus to the parallel domains where REPOP seems to offer significant advantages.

In the logistics and rocket domains, the use of h_{relax} heuristic accounts for the largest fraction of the improvement from UCPOP. Interestingly, h_{relax} fails to help scale up UCPOP even on very small problems in the gripper domain. We found that the search spends most of the time exploring inconsistent partial plans for failing to realize that a left or right gripper can carry at most one ball. This problem is alleviated by consistency enforcement (CE) techniques through detection and resolution of implicit conflicts (e.g. the conflict between $carry(ball1, left)$ and $carry(ball2, left)$). As a result, REPOP can comfortably solve large gripper problems, such as *gripper-20*.

Among the consistency enforcement techniques, both reachability analysis and disjunctive constraint representation appear to complement each other. For instance, in problem *logistics.d*, if only reachability analysis is used with the heuristic h_{relax}, a solution can be found after generating 255K nodes. When disjunctive representation is also used, the number of generated nodes is reduced by more than 15 times to 16K.

6 Related Work

Several previous research efforts have been aimed at accelerating partial order planners (c.f. [11; 12; 13; 16; 23; 24; 6; 4]). While none of these techniques approach the current level of performance offered by REPOP, many important ideas separately introduced in these previous efforts are either related to or are complementary to our techniques. IxTeT [6] uses distance based heuristic estimates to select among the possible resolutions of a given open condition flaw (although no evaluation of the technique is provided). It is interesting to note that IxTeT's use of distance based heuristics precedes their

independent re-discovery in the context of state-search planners by McDermott [20] and Bonet and Geffner [3]. In [4], Bylander describes the use of a relaxation heuristic based on linear planning for POP; it however seems not to be very effective. The idea of postponing the resolution of unsafe links by posting disjunctive constraints has been pursued by Smith and Peot in [23] as well as by Kambhampati and Yang in [16]. Our work shows that the effectiveness of this idea is enhanced significantly by generalizing the notion of conflicts to include indirect conflicts. The notion of action-proposition mutexes defined in Smith and Weld's work on temporal graphplan [26] is related to our notion of indirect conflicts introduced in Section 4. Finally, there is a significant amount of work on flaw selection strategies (e.g., the order in which open condition flaws are selected to be resolved) [11] that may be fruitfully combined with REPOP. The techniques for recognizing and suspending recursion ("looping") during search may also make a useful addition to REPOP [24].

7 Conclusion and Future Work

The successes in scaling up classical planning using CSP and state space search approaches have generally been (mis)interpreted as a side-swipe on the scalability of partial order planning. Consequently, in the last five years, work on POP paradigm has dwindled down, despite its known flexibility advantages. In this paper we challenged this trend by demonstrating that the very techniques that are responsible for the effectiveness of state search and CSP approaches can also be exploited to improve the efficiency of partial order planners dramatically. By applying the ideas of distance based heuristics, disjunctive representations for planning constraints and reachability analysis, we have achieved an impressive performance for a partial order planner, called REPOP, across a number of "parallel" planning domains. Our empirical studies show that not only does REPOP convincingly outperform Graphplan in parallel domains, the plans generated by REPOP have more execution flexibility. This is very interesting for two reasons. First of all, most of the real-world planning domains tend to have loose ordering among actions. Secondly, the ability for generating loosely ordered plans is very important in hybrid methods that involve on-line integration of planning with scheduling.

There are several avenues for extending this work. To begin with, our partial plan selection heuristics do not take negative interactions into account. This may be one reason for the unsatisfactory performance of REPOP in serial domains. One way to account for the negative interactions, that we are considering currently, involves using the partial state information provided by the pre- and post-cutsets of actions. Our work on AltAlt [22] suggests that the cost of achieving these partial states can be quantified in terms of the level in the planning graph at which the propositions comprising these states are present without any mutex relations. Another idea we are pursuing is to use n-ary state invariants (such as those detected in [5]) to detect and resolve more indirect conflicts in the plan. Finally, a more ambitious extension that we are pursuing involves considering more general versions of POP algorithms–including those that handle partially instantiated actions, as well as actions with conditional effects and durations.

References

[1] C. Backstrom. Computational aspects of reordering plans. *JAIR*. Vol. 9. pp. 99-137.

[2] A. Blum and M.L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*. 90(1-2). 1997.

[3] B. Bonet and H. Geffner. Planning as heuristic search: New results. In *Proc. ECP-99*, 1999.

[4] T. Bylander. A Linear programming heuristic for optimal planning. In *Proc. AAAI-97*, 1997.

[5] M. Fox and D. Long. Automatic inference of state invariants in TIM. *JAIR*. Vol. 9. 1998.

[6] M. Ghallab and H. Laruelle. Representation and control in IxTeT. In *Proc. AIPS-94*, 1994.

[7] A. Gerevini and L. Schubert. Inferring state constraints for domain-independent planning. In *Proc. AAAI-98*, 1998.

[8] P. Haslum and H. Geffner. Admissible Heuristics for Optimal Planning. In *Proc. AIPS-2000*, 2000.

[9] J. Hoffman and B. Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. Submitted, 2000.

[10] A. Johnson, P. Morris, N. Muscettola and K. Rajan. Planning in Interplanetary Space: Theory and Practice. In *Proc. AIPS-2000*.

[11] D. Joslin and M. Pollack. Least-cost flaw repair: A plan refinement strategy for partial-order planning. In *Proc. AAAI-94*.

[12] D. Joslin, M. Pollack. Passive and active decision postponement in plan generation. Proc. 3rd European Conf. on Planning. 1995.

[13] A. Gerevini and L. Schubert. Accelerating partial-order planners: Some techniques for effective search control and pruning. *JAIR*, 5:95-137, 1996.

[14] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proc. AAAI-96*.

[15] S. Kambhampati, C. Knoblock and Q. Yang. Planning as Refinement Search: A unified framework for evaluating design tradeoffs in partial-order planning. In *Artificial Intelligence*, 1995.

[16] S. Kambhampati and X. Yang. On the role of Disjunctive representations and Constraint Propagation in Refinement Planning In *Proc. KR-96*.

[17] S. Kambhampati. Planning Graph as (dynamic) CSP: Exploiting EBL, DDB and other CSP Techniques in Graphplan. *JAIR*. Vol. 12. pp. 1-34. 2000.

[18] D. Long and M. Fox. Efficient implementation of the plan graph in STAN. *JAIR*, 10(1-2) 1999.

[19] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proc. AAAI-91*.

[20] D. McDermott. Using regression graphs to control search in planning. *Artificial Intelligence*, 109(1-2):111–160, 1999.

[21] X. Nguyen and S. Kambhampati. Extracting effective and admissible state space heuristics from the planning graph. In *Proc. AAAI-2000*.

[22] X. Nguyen, S. Kambhampati and R. Nigenda. Planning Graph as the Basis for deriving Heuristics for Plan Synthesis by State Space and CSP Search. To appear in *Artificial Intelligence*.

[23] M. Peot and D. Smith. Threat-removal strategies for partial-order planning. In *Proc. AAAI-93*.

[24] D. Smith and M. Peot. Suspending Recursion Causal-link Planning. In *Proc. AIPS-96*.

[25] D. Smith, J. Frank and A. Jonsson. Bridging the gap between planning and scheduling. In *Knowledge Engineering Review*, 15(1):47-83. 2000.

[26] D. Smith and D. Weld. Temporal planning with mutual exclusion reasoning. In *Proc. IJCAI-99*, 1999.

[27] D. Weld. An introduction to least commitment planning. *AI magazine*, 1994.