Scaleability in planning*

Vassilis Liatsos and Barry Richards

IC-Parc, William Penney Lab
Imperial College, London SW7 2AZ, UK
{vl,ebr}@icparc.ic.ac.uk

Abstract. We explore the performance of three planners, viz. *parc*PLAN, IPP and `Blackbox`, on a variant of the standard blocks-world problem. The variant problem has a restricted number of table positions, and the number of arms can vary (from 1 upwards). This type of problem is typical of many real world planning problems, where resources form a significant component. The empirical studies reveal that least commitment planning, as implemented in *parc*PLAN, is far more effective than the strategies in IPP and `Blackbox`. But the studies also reveal a serious limitation on the scaleability of *parc*PLAN's algorithm.

1 Introduction

In this paper we study the performance of *parc*PLAN [10] on a variant of the standard blocks-world problem, and for perspective compare it to IPP [9] and `Blackbox` [8]. In the standard problem the table is assumed to be indefinitely large; in the variant there is a fixed number of table positions. This restriction constitutes a resource limitation that significantly increases the difficulty of the problem. There is a further variation in regard to resources. In the standard problem there is just one arm available to move blocks; in the variant there may be several arms, the number varying from instance to instance. Problems of the variant type exhibit a critical dependency among three factors: the number of resources available, the degree of overlap among actions, and the length (duration) of individual actions. This dependency is typical of a large class of real-world planning problems, which is the reason for focusing on the variant blocks-world problems.

Any "viable" generic planner must be able to handle problems of this type successfully. The question is whether *parc*PLAN implements a scaleable generic strategy for these problems. This is the issue we investigate below. Here the comparative perspective provided by IPP and `Blackbox` is useful.

The *parc*PLAN system is a temporal planner. Actions, properties and goals are all indexed to intervals, which are represented in terms of their end-points. The search strategy is constraint-based and least commitment. This applies not only to temporal reasoning but to all forms of reasoning. The planning architecture is essentially "failure driven"; that is, it introduces an action into the

* This work was supported in part by a grant from the Engineering and Physical Sciences Research Council, Grant No. GR/L71919

plan only if it fails to find a feasible plan for the "current" set of actions. The solutions generated by *parc*PLAN use the minimal number of actions necessary to achieve all the goals.

There is a view that planning strategies like that in *parc*PLAN are "doomed" to be inefficient. Whilst this may well be true, we show that *parc*PLAN is better-suited than IPP and `Blackbox` in solving variant blocks-world problems. It is more efficient and scales much better.

This is not to imply, however, that *parc*PLAN is an effective generic planner. It too has significant limitations. The aim of the paper is twofold: first, to demonstrate that least commitment planning, as implemented in *parc*PLAN, has a great deal more "mileage" in it than has been recognised; second, to identify one of the critical limitations on the effectiveness of *parc*PLAN's algorithm. This limitation must be overcome if the system is to evolve into a "viable" generic planner. We think that this will require changing the search algorithm.

The paper is structured as follows. Section 2 presents the *parc*PLAN approach to representing planning problems and generating solutions. Section 3 reports the results of an empirical study of single-resource planning and Section 4 the results of a similar study of multiple-resource planning. Finally, Section 5 summarises the conclusions.

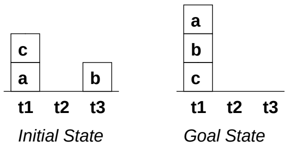
2 *parc*PLAN and the planning domain

The *parc*PLAN system falls broadly within the planning paradigm of Allen & Koomen [1]. That is, it is an interval-based temporal planner designed to support parallel action execution. But it differs from [1] in several respects. First, it treats intervals in terms of their end-points, and allows both indefinite and numerical reasoning. Second, it integrates temporal and resource reasoning, thereby supporting the requirements of reasoning about action overlap. This is realised in a module called *parc*TREC, which is described in [5]. Third, it implements a constraint-based strategy for introducing actions into a plan. This is essential to the process of generating optimal plans, which itself is another differentiating aspect of the *parc*PLAN architecture.

2.1 Representation

In *parc*PLAN properties and actions are uniformly indexed to intervals. This is realised in expressions of the form *Q@[start,end)*, which says that the property (action) Q holds (occurs) over the time interval *[start,end)*. The interval *[start,end)* is closed on the left and open on the right; that is, the left point belongs to the interval and the right is the first point not belonging to the interval. The reasons for this treatment need not be elaborated here. Below we illustrate how a planning problem is formulated in the variant blocks world.

The basic problem is depicted in part (i) of Fig. 1. Note that there are three table positions in the problem ($t1, t2, t3$), and that the blocks are originally configured as shown in the "initial state". The task is to move the blocks into



(i) Representation of original plan state

Facts:

on(a,t1)@[0,T1), on(b,t3)@[0,T2),
on(c,a)@[0,T3), clear(b)@[0,T4),
clear(c)@[0,T5), clear(t2)@[0,T6)

Goals:

on(a,b)@[T7,20), on(b,c)@[T8,20),
on(c,t1)@[T9,20)

Temporal Constraints:

$$0 < T1, \ldots, 0 < T6, T7 < 20, \ldots, T9 < 20$$

(ii) Specification of original plan state

Fig. 1. A blocks world example

the positions shown in the "goal state", using the arm(s) available and only the three table positions. The "initial state" and the "goal state" together reflect the original state of the planner.

One can see immediately that this problem is more difficult than the standard blocks-world formulation where the table is indefinitely large. Any standard problem can be solved in a maximum of $2n - 2$ moves using one arm, where n is the number of blocks. Thus, a 3-block problem requires no more than 4 moves. The variant problem here requires 5 moves, if there is just one arm.

The informal characterisation of the original plan state in part (i) of Fig. 1 is specified more formally in part (ii). The specification has three components, viz. Facts, Goals and Temporal Constraints. The set of facts, together with some of the temporal constraints, reflect the situation in the "initial state", while the set of goals, together with other temporal constraints, reflect the situation in the "goal state". Note that there is no reference to the "initial state" in the facts, and no reference to the "goal state" in the goals. Here everything is indexed to intervals. Each of the facts begins to hold from time point 0, and is assumed to persist up to a certain unspecified time point, indicated by a variable. [1] Each of the goals must be established before time point 20, again indicated by a variable, and persists up to time 20. The constraints among the time points require that in the original plan state the facts are either earlier or co-temporal with the goals.

In general, the temporal constraints, which always relate two time points, are handled in the global constraint network Temporal Constraints. The variables here have the status of domain variables, with their domains consisting initially

[1] The Prolog convention is used to indicate variables: their names start with an uppercase letter.

Action:	*move(Block,From,To,Arm)@[Start,End)*
Conditions:	
1.	*on(Block,From)@[T1,Start),*
2.	*clear(To)@[T2,End),*
3.	*clear(Block)@[T3,T4)*
Effects:	
4.	*on(Block,To)@[End,T5),*
5.	*clear(From)@[Start,T6)*
Constraints:	
6.	*designates(Block,block),*
7.	*designates(Arm,arm),*
8.	*designates(From,block_table), designates(To,block_table),*
9.	$Block \neq From$, $Block \neq To$,
10.	$T3 < Start$, $End < T4$

Fig. 2. Specification of the *move* action operator

of the integers 1 to 20. These domains will be pruned as the plan evolves from the original plan state and new temporal constraints are added.

The action operator *move* is a 4-place relation, specified in terms of Conditions, Effects, and Constraints. The relations among the conditions, effects and actions are less constrained than those in a state-based planner. Since actions occur over intervals, conditions need not always be true before the action begins, and effects may become true before it ends. This is the case for the *move* action, which is defined in Fig. 2.

While the Conditions (goals) and Effects (facts) are the familiar ones, the temporal constraints, given in line 10, allow for unfamiliar relationships. For example, the condition *clear(To)@[T2,End)* need not be true before the action starts. There is no constraint that $T2 < Start$ and hence the condition must only become true sometime before *End*. Similarly, the effect *clear(From)@[Start,T6)* becomes true before the action ends. In fact, it becomes true when the action starts and remains true up to *T6*, which may or may not be before the action ends; *T6* has no specific relation to *End*.

The non-temporal constraints are the usual ones. For example, the constraint in line 6, together with the Conditions and Effects, asserts that the object which is moved has to be a block and not a table position. The constraints in line 8 state that the source and destination of the block can be either a block or a table position. The two constraints in line 9, which are non-codesignation constraints, state that a block cannot be moved from itself, nor it can be moved on top of itself. Note, however, that there is no constraint $From \neq To$, which would prevent moving a block from X to X. When there is only one arm and hence actions must occur in a linear sequence, lifting up a block and putting it down in the same place is not productive. However, when there are multiple arms allowing actions to overlap, this type of action can be effective in obtaining an optimal solution.

There is a final component specifying the problem domain; this takes the form of rules, which state that certain relations between properties and actions cannot occur in a valid plan. In the case of properties, for example, two domain rules hold: (1) an object (whether a block or table position) cannot be clear

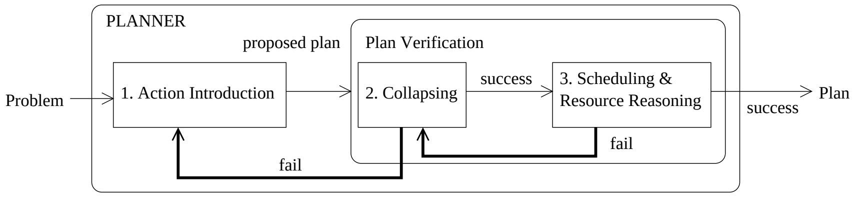


Fig. 3. *parc*PLAN's plan cycle

and have a block on top of it at the same time; (2) an object cannot have two different blocks on it at the same time. How these and other rules are captured need not detain us here.

2.2 Planning Algorithm

The planning algorithm in *parc*PLAN involves three basic steps: (1) action introduction, (2) collapsing and (3) scheduling and resource reasoning.[2] Action introduction and collapsing are two ways in which a goal can be achieved. Introducing an action to achieve a goal is realised by unifying the goal with one of the effects of the action operator and constraining the interval in the goal to be within the interval of that effect. Achieving a goal through collapsing is realised by unifying the goal with one of the facts in Facts and constraining the goal interval to be within the fact interval. If the resulting network of temporal constraints is consistent, collapsing is successful.[3] Figure 3 shows the planning cycle in *parc*PLAN, including the three basic steps and the failure points. These are explained in detail below.

Step 1: Action Introduction In the action introduction step *parc*PLAN looks for a goal which *does not* have a potential collapser. A fact *P@[S2,E2)* is a *potential collapser* for a goal *Q@[S1,E1)* if P and Q can codesignate, and $S1 \geq S2$ and $E2 \geq E1$ are consistent with the existing temporal constraints.[4] If there is a goal without a potential collapser, *parc*PLAN introduces an action into the plan by unifying an effect of the action operator with that goal, that is, by posting the appropriate codesignation and temporal constraints. It adds all the other effects of the action to the Facts, and all the conditions of the action to the Goals; and it removes the achieved goal from the Goals. Finally, it adds all the remaining constraints arising from the unification and the domain rules. The constraints are held in two separate networks, one for the codesignation constraints and the other for the temporal constraints.[5] *parc*PLAN continues to introduce actions in the plan until all the goals have at least one potential collapser; only then does the planner enter the collapsing step.

---

[2] The term "collapsing" originates from Allen and Koomen [1].
[3] A path consistency algorithm is used to detect inconsistency in the temporal network.
[4] Obviously, a goal can have more than one potential collapser.
[5] Note that all these constraints are resolved using least commitment.

Step 2: Collapsing The task at this point is to collapse all the remaining goals while maintaining the consistency of the codesignation constraints and temporal constraints. Collapsing is handled as a constraint satisfaction problem where each goal is associated with a finite domain variable whose domain represents the potential collapsers for that goal. Choosing a value for such a variable collapses the associated goal with a particular potential collapser. To detect failures as early as possible the collapsing phase follows a strategy based on the *first-fail principle*; the most constrained goals are collapsed first, i.e. the ones with the least number of potential collapsers.[6]

Step 3: Scheduling & Resource Reasoning If all the goals are successfully collapsed, *parc*PLAN enters the third step of the planning cycle. At this point the plan consists of the set of actions which have been introduced so far. Typically, these are partially specified (due to choices remaining in codesignation constraints) and partially ordered (due to choices remaining on the order of time points). The task now is to determine whether the actions can be made fully specific and scheduled subject to the available resources. This is essentially a disjunctive scheduling problem complicated by three interdependent factors, viz. the number of available resources, the degree of action overlap, and the duration of the actions. To solve this problem (the *resource feasibility problem*) efficiently requires special techniques for integrating temporal and resource reasoning. One such technique is presented in El-Kholy & Richards [4], where Boolean meta-variables are used to support the necessary integration. The technique here adopts the same search model as [4]; however the search strategy is based on the probe-backtracking technique [13, 14]. Search terminates when *parc*PLAN succeeds in fully specifying the actions and scheduling them subject to the available resources. At this point, a plan has been found.

Failure Points Clearly, there are two points where *parc*PLAN can fail. It can fail in the third step, because it cannot fully specify the actions consistently or cannot schedule them with the resources available. It can also fail in the second step, because it cannot successfully collapse the remaining goals. These failures represent points in the search where control needs to be returned to the previous step.

If failure is encountered in scheduling & resource reasoning, control is returned to the collapsing step, and *parc*PLAN searches for an alternative collapse. If it finds a successful collapse, control passes back to the third step of scheduling & resource reasoning; otherwise control is returned to the action-introduction step. To reduce the amount of "backtracking" between the last two steps, a global constraint is used to monitor the resource usage during collapsing. This constraint monitors the lower and upper bounds for resource requirements at each start point (see [4]). If at any point the lower bound exceeds the available resources, a failure is signalled. The idea here is that collapsing should not waste time looking at plans which exceed the resources available.

[6] Goals with only one potential collapser are collapsed immediately prior to any search.

Failure in the second step, i.e., failure to find a successful collapse, means that *at least one* of the goals needs to be achieved by introducing a new action. The problem is to find which goal that is; here search confronts a major challenge. This is not special to *parc*PLAN but is shared in different ways by all planners.[7]

Since it is not clear which goal needs to be select for action introduction, *parc*PLAN follows a least commitment strategy. The strategy adds an action to the plan but delays selecting a specific goal until the collapsing step has again been reached. This is implemented by introducing an action which can be used for *any* of the remaining goals. We call this a *generalised* action.[8] The limiting case is the wholly indefinite action $move(X, Y, Z, A)$, where X is a block, Y, Z are either a block or a table position, and A is an arm. Once a generalised action has been introduced, control passes back to the collapsing step. Note that only one such action is required. If collapsing succeeds, one of the goals will be collapsed with an effect of the generalised action. At this point control passes to the third step. If, however, *parc*PLAN fails to find a successful collapse, it returns to the first step and introduces another generalised action into the plan. Clearly, this strategy introduces only those actions which are absolutely necessary to achieve the goals. This guarantees a plan with the minimal number of actions.

3 Experimental Study: Single Arm

In this section we focus on problems where there is just one arm available to move blocks. Here the third step in the planning cycle, viz. scheduling & resource reasoning, is simplified. Since there is only one arm, all the actions must be disjoint, i.e. the plan must be linear. As a result, when actions are introduced into the plan, a constraint is imposed to ensure that the actions are disjoint. This is sufficient to guarantee that if a successful collapse is found, the actions introduced so far can be scheduled to generate a plan.

For single-arm problems the burden of generating a plan is carried largely by the first two steps in the cycle. Our objective below is to examine this burden experimentally. We study a number of different problem sets, each containing 100 different instances. We name each set (benchmark) by indicating its size, i.e. the number of blocks, the number of table positions and the number of arms. For example, the set having 10 blocks, 4 table positions and 1 arm is denoted 10x4x1.

We explore each of the three planners, *parc*PLAN, IPP and `Blackbox`, on each problem set. We impose a uniform time-out of 600 seconds for each instance.[9] For this study we use `Blackbox` 3.4, and the PDDL-version of IPP used in the AIPS planning competition. These systems have a performance advantage over

[7] In IPP and `Blackbox`, for example, the same problem occurs when a plan is not found using n time points; extending the plan to $n+1$ time points generates a much larger search space.

[8] Joslin and Pollack use a similar approach in [7].

[9] The experiments were run on a Pentium II 300 MHz using Red Hat Linux 2.0.32. Benchmarks are available from [11]

Table 1. *parc*PLAN: single arm problems

Benchmark	Solved	CPU time	Actions
4x4x1	100	0.20	0.08
6x6x1	100	0.70	0.02
8x8x1	100	1.53	0.08
10x10x1	100	4.65	0.25
12x12x1	100	8.49	0.45
14x14x1	100	14.10	0.64
16x16x1	100	26.27	0.93
18x18x1	100	36.52	0.96
20x20x1	100	64.82	1.32

*parc*PLAN since they are compiled for the particular architecture; *parc*PLAN runs on ECL^{i}PS^e 4.1 [3].

As we mentioned above, *parc*PLAN will always generate a plan with the minimal number of actions. For the single-arm problems, the actions in the plan must be disjoint. On the assumption that their duration is some uniform constant length, *parc*PLAN yields plans with the shortest overall duration.

Note that finding such plans is not in general easy. Gupta & Nau [6] show that optimal planning in the standard blocks world is NP-complete.

3.1 Easier problems

We first look at limiting case benchmarks, i.e. problem sets where the number of table positions is the same as the number of blocks. These are similar to, though not exactly like, the standard blocks-world problems. The aim here is to study the performance of the three planners as the size of the problem increases.

Table 1 shows the performance of *parc*PLAN measured on two dimensions, viz. the number of problems solved and the average CPU time (measured in seconds) to solve them. The problems range from 4 blocks 4 table positions, to 20 blocks 20 table positions. Recall that there is a cut-off of 600 seconds. *parc*PLAN solves all 100 instances of all the problems well within the cut-off. On the largest problem set it takes an average of about 65 seconds to solve each instance. It is interesting to note the effect of problem size on the average CPU time and the average number of generalised actions introduced to solve a particular instance. The rate at which CPU time rises is roughly similar to the rate at which the number of generalised actions is introduced.

To run IPP and `Blackbox` on the same benchmarks, we translate the problems into PDDL format [12]. We explore two alternative specifications. The Optimal version uses two actions `pickup` and `putdown`; this generates plans which are guaranteed to have the shortest plan length. The Feasible version uses one action `move`; here plans may contain superfluous actions and hence may not be the shortest.

Table 2 shows the performance of both Optimal and Feasible versions of IPP and `Blackbox` on the benchmarks above. Neither version of the two planners scales on these problems although the Feasible versions consistently solve more

Table 2. IPP and `Blackbox`: single arm problems

	IPP				Blackbox			
	Optimal		*Feasible*		*Optimal*		*Feasible*	
Benchmark	Solved	CPU time	Solved	CPU time	Solved	CPU time	Solved	CPU time
4x4x1	100	0.15	100	0.14	100	0.19	100	0.17
6x6x1	100	1.09	100	1.52	100	15.06	100	14.22
8x8x1	94	56.15	94	35.79	22	246.43	61	29.46
10x10x1	8	97.18	44	112.19	0	–	6	515.66
12x12x1	0	–	17	224.65	n/a	n/a	n/a	n/a

Table 3. *parc*PLAN: single arm, limited table problems

Benchmarks	Solved	CPU time	Actions
10x4x1	72	81.28	1.38
20x8x1	71	68.80	1.06
30x12x1	63	267.78	1.71

problems than the Optimal ones. The results clearly show that the performance of the Feasible version of IPP (which is the best of these) does not compare well to *parc*PLAN. The performance of IPP falls sharply, first between 6x6 and 8x8, and then between 8x8 and 10x10. Problem size here is a real challenge for both IPP and `Blackbox`.

3.2 Harder Problems

We now consider harder benchmarks. Here we fix the number of table positions to be 40% of the number of blocks. These problems are much harder than their counterparts, where the number of table positions matches the number of blocks. Table 3 shows that *parc*PLAN finds these far from trivial.

It is interesting that the success rate for *parc*PLAN seems relatively stable. It fails on 28 of the 10x4 instances, 29 of the 20x8 cases, and 37 of the 30x12. While the problems vary enormously in size, the success rate does not. As for the variation in CPU time, this correlates roughly to the number of generalised actions introduced.

We tried IPP and `Blackbox` on these benchmarks but the results were poor. Neither planner solved any of the 20x8 and 30x12 problems. Clearly IPP and `Blackbox` are not well-suited to these problems. Although *parc*PLAN is much better, we begin to see the limitations on its scaleability. There is sharp fall in performance on the 30x12x1 problem set.

4 Experimental Study: Multiple Arms

Here we consider problems where there are multiple arms to move blocks. In this case the third step in the planning cycle, viz. scheduling & resource reasoning, has real work to do. After finding a successful collapse, *parc*PLAN must

Table 4. Multiple arms problems

	*parc*PLAN			IPP	
Benchmark	Solved	CPU time	Actions	Solved	CPU time
4x4x2	100	0.14	0.14	100	0.17
6x6x3	100	0.45	0.28	100	1.44
8x8x4	100	0.71	0.04	92	15.99
10x10x5	100	5.63	0.12	44	87.52
12x12x6	100	12.24	0.04	12	96.49
14x14x7	100	5.88	0.01	0	–
16x16x8	100	5.68	0.00	n/a	n/a
18x18x9	100	8.04	0.00	n/a	n/a
20x20x10	100	10.93	0.00	n/a	n/a

determine whether the actions generated so far can be scheduled subject to the arms available. This requires search, which may not succeed. In the event, the system must "backtrack" to seek another successful collapse, and if it finds one, it re-enters the scheduling & resource reasoning step. Clearly, this can lead to a considerable amount of search.

4.1 Easier Problems

We first look at the performance of *parc*PLAN and IPP on benchmarks corresponding to the easier problem sets in the previous section. The difference here is in the number of arms available; we fix this uniformly at half the number of blocks. The smallest benchmark is 4x4x2 (4 blocks, 4 table positions, 2 arms) and the largest is 20x20x10 (20 blocks, 20 table positions, 10 arms).

For these problems we could test only the Optimal version of IPP since the Feasible version does not reason about resources. We present the results for both *parc*PLAN and IPP in Table 4.[10]

The performance of IPP falls sharply on the 10x10x5 benchmarks; it can solve only 44 instances out of a 100 and takes an average of nearly 88 seconds to do so, which is an order of magnitude worse than *parc*PLAN. IPP's performance again falls sharply on the 12x12x6 problem set, where it solves only 12 instances. On the larger problems it cannot solve any instances within the 600 second cut-off.

In contrast, the performance of *parc*PLAN is impressive both on the number of instances solved and in the time required to solve them. It finds solutions for all instances of all benchmarks, with the worst average CPU time being just over 12 seconds on the 12x12x6 benchmark. Note that the relatively uniform performance profile reflects the relatively constant number of generalised actions introduced. Indeed such actions are very rarely used. For most of these problems there is no need to "backtrack" to the action-introduction step. This is why the strategy is so successful.

[10] We do not discuss `Blackbox` since its performance was poor relative to IPP.

Table 5. *parc*PLAN: 10x10 problems

	Solved Cases				*Failed Cases*	
Benchmark	Solved	CPU time	Actions	Collapses	Actions	Collapses
10x10x5	100	5.63	0.12	21.74	–	–
10x10x4	92	7.77	0.34	23.52	2.00	> 1800
10x10x3	93	10.77	0.97	16.23	2.57	> 1400
10x10x2	99	14.40	1.97	7.32	5.00	0

4.2 Harder Problems

There are problems, however, where *parc*PLAN must resort to introducing a larger number of generalised actions; these it finds more difficult to solve, and some instances it cannot solve at all. We will show that the difficulty lies in the collapsing step, although the difficulty is not the same in all cases.

Table 5 profiles the performance of *parc*PLAN on problem sets consisting of 10 blocks and 10 table positions, with the number of arms varying from 5 down to 2. The first line of the table repeats the results on the 10x10x5 benchmarks from Table 4. While the results in table 5 are encouraging, the performance profile is curious. Note that *parc*PLAN is at its worst on the 10x10x4 and 10x10x3 problem sets. It solves fewer instances from these sets than it does from either the 10x10x2 set or the 10x10x5 set. The reason behind this behaviour is that the resource-monitoring constraint in the collapsing phase is able to prune more options and detect failures earlier when the resources are more limited. The more constrained the problem, the more effective the resource-monitoring constraint is in guiding the search. One can see that the constraint has the desired effect by looking at the reduction in the number of collapses as resources are reduced.

For the failed case of the 10x10x2 problem set, note that *parc*PLAN introduces 5 generalised actions but never finds a successful collapse. In effect, it never proceeds to the third step of the planning cycle; it times out looking for a successful collapse. For the failed cases of the 10x10x4 and 10x10x3 problem sets *parc*PLAN finds an enormous number of collapsers but in each case it discovers there exists no schedule for the actions subject to the arms available. Here there are too many successful collapses, all of which lead to a dead-end.

In general, the reason for failure here lies in collapsing. Either *parc*PLAN cannot find a successful collapse or it finds too many collapses, all of which are dead-ends. These collapsing problems constitute a major challenge for *parc*PLAN, one which is shared in different ways by all generic planners.

We now make the problem sets even harder. We reduce the number of table positions to 4, keeping the number of blocks at 10; and again we vary the number of arms from 5 down to 2. Table 6 shows the results. Here *parc*PLAN's performance degrades uniformly with the decrease in the arms available. Again note the situation in the failed cases. In the 10x10x4 and 10x10x3 problem sets, *parc*PLAN finds a large number of successful collapses (over 650 in one case and 250 in the other); for each one it discovers that there is no viable schedule for the

Table 6. *parc*PLAN: 10x4 problems

		Solved Cases			Failed Cases	
Benchmark	Solved	CPU time	Actions	Collapses	Actions	Collapses
10x4x5	100	4.46	0.11	8.39	–	–
10x4x4	98	9.72	0.40	9.46	2.00	> 650
10x4x3	91	12.88	1.00	6.85	2.78	> 250
10x4x2	84	52.73	2.30	13.10	4.50	29.50

Table 7. IPP: 10x10 and 10x4 problems

10x10 benchmarks			10x4 benchmarks		
Benchmark	Solved	CPU time	Benchmark	Solved	CPU time
10x10x5	44	87.52	10x4x5	52	133.57
10x10x4	59	112.13	10x4x4	50	133.28
10x10x3	27	175.33	10x4x3	49	123.77
10x10x2	5	266.88	10x4x2	39	114.79

actions. In the 10x10x2 problem set it finds on average just under 30 successful collapses, and none lead to a viable schedule. Collapsing is the core issue.

Table 7 shows the performance of IPP on the same benchmarks.[11] The results indicate that the strategies of IPP and `Blackbox` are not well suited to solve problems where multiple resources are involved. This limitation has been noted by Srivastava and Kambhampati [15], who are exploring an extension to GraphPlan [2] with a view to handling such cases more efficiently.

5 Conclusions

The results of our experiments suggest that least commitment planning has more "mileage" in it than some have recognised [7]. The implementation of the least commitment strategy in *parc*PLAN is clearly effective for variant blocks-world problems, particularly in comparison with strategies in IPP and `Blackbox`. These strategies may be better than *parc*PLAN for other problems, although that remains to be investigated.

The main conclusion, however, relates to *parc*PLAN itself. It has serious limitations associated with collapsing. How these might be resolved is not immediately clear. What is clear is that this is one of the critical challenges for *parc*PLAN. Since the challenge appears in different forms in other generic planners, it seems to be critical to the future of generic planning.

[11] `Blackbox` is ineffective for problems of this size and hence, we do not consider it further.

References

[1] James F. Allen and J. A. Koomen. Planning Using a Temporal World Model. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence, IJCAI-83*, pages 741–747, Karlsruhe, Germany, 1983.

[2] Avrim L. Blum and Merrick L. Furst. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence*, 90:281–300, 1997.

[3] ECRC. *ECL^iPS^e User Manual*. European Computer Industry Research Centre, Munich, 4.1 edition, 1998.

[4] A. El-Kholy and B. Richards. Temporal and Resource Reasoning in Planning: the *parc*PLAN approach. In *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96)*, pages 614–618, Budapest, Hungary, 1996. John Wiley and Sons, Ltd.

[5] Amin El-Kholy. *Resource Feasibility in Planning*. PhD thesis, Imperial College, University of London, 1996.

[6] Naresh Gupta and Dana S. Nau. On the Complexity of Blocks-World Planning. *Artificial Intelligence*, 56:223–254, 1992.

[7] D. Joslin and M. E. Pollack. Is "early commitment" in plan generation ever a good idea? In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 1188–1193, Portland, Oregon, 1996.

[8] H. Kautz and B. Selman. Blackbox: A New Approach to the Application of Theorem Proving to Problem Solving. *AIPS98 Workshop on Planning as Combinatorial Search*, pages 58–60, June 1998. Available from: `http://www.research.att.com/~kautz/blackbox/`.

[9] J. Koehler, B. Nebel, J Hoffmann, and Dimopoulos Y. Extending Planning Graphs to an adl Subset. In S. Steel and R. Alami, editors, *Fourth European Conference on Planning (ECP-97)*, number 1348 in Lectures Notes in Artificial Intelligence, pages 273–285, Toulouse, France, September 1997. Springer. Available from: `http://www.informatik.uni-freiburg.de/~koehler/ipp.html`.

[10] J. M. Lever and B. Richards. *parc*PLAN: a Planning Architecture with Parallel Actions, Resources and Constraints. In *Proceedings of the 9th International Symposium on Methodologies for Intelligent Systems*, pages 213–222, 1994.

[11] V. Liatsos and B. Richards. Planning benchmarks on a blocks world variant. Available from: `http://www.icparc.ic.ac.uk/parcPlan/ecp99/`.

[12] D. McDermott et al. PDDL – The Planning Domain Definition Language, version 1.2. Technical Report CVC TR-98-003, Yale Center for Computational Vision and Control, Yale University, October 1998. Available from: `http://www.cs.yale.edu/HTML/YALE/CS/HyPlans/mcdermott.html`.

[13] H. El Sakkout, T. Richards, and M. Wallace. Minimal Perturbation in Dynamic Scheduling. In H. Prade, editor, *13th European Conference on Artificial Intelligence. ECAI-98*. John Wiley & Sons, 1998.

[14] H. El Sakkout and M. Wallace. Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling. To appear in Constraints journal, 1999.

[15] B. Srivastava and S. Kambhampati. Scaling up Planning by teasing out Resource Scheduling. Technical Report ASU CSE TR 99-005, Department of Computer Science and Engineering, Arizona State University, Tempe, AZ 85287-5406, 1999. Available from: `http://rakaposhi.eas.asu.edu/yochan.html`.