*parc*Plan: a Planning Architecture with Parallel Actions, Resources and Constraints

Jonathan Lever and Barry Richards
IC-Parc,
Imperial College,
London,
SW7 2AZ
UK
{jml,ebr}@doc.ic.ac.uk

Appears in proceedings of 8th ISMIS, North Carolina, October 1994, Springer Verlag.

Abstract

We describe the generic planning architecture *parc*PLAN, which uses constraint-solving to perform both temporal and non-temporal reasoning. The architecture allows considerable temporal sophistication in the specification of actions, integrity constraints and planning problems, and produces plans with a high degree of concurrency in action execution. A feature of *parc*PLAN is the capability to represent and minimise costs associated with plans. *parc*PLAN has been implemented in the Constraint Logic Programming language ECL^{i}PS^e, and performs well on large-scale planning and resource allocation problems.

1 Introduction

This paper describes the generic planning architecture *parc*PLAN. This architecture is a development of IQ-PLAN [6], and continues to offer the temporal expressiveness of IQ-PLAN while including mechanisms to represent and minimise costs associated with plans, and adding constraint-solving to perform inference.

In planning, actions can be treated as instantaneous—as is typically the case in *state-space planning* (e.g. STRIPS [4])—or can be considered to have duration as in interval-based planning [1]. The *parc*PLAN architecture is interval-based, and is also *non-linear* ([9],[7],[10]) in that the order of action-execution in a plan is independent of the order in which actions are introduced during search.

In plans produced by *parc*PLAN, actions can occur concurrently in a variety of overlapping configurations. The conditions of an action need not be specified as holding *prior* to the start of execution of the action—a more exact specification that involves the precise point during execution of the action at which the condition is required can be given. Similar comments apply to the effects of actions. This allows concurrency of action-execution to be maximised.

An aspect of planning strategies of particular significance for non-linear planning is *least-commitment*. A strategy exemplifies least-commitment if, when faced with a number of alternative choices, it carries forward all possibilities rather than making a (backtrackable) commitment. There are many levels at which choices arise in planning and thus many levels at which the principle may be applied, the most far-reaching being least-commitment on action ordering ([7], [10]). The benefit of least-commitment is in a reduction of choice points, but this must be balanced against the cost of consistency-checking. The strategy used in *parc*PLAN preserves least-commitment on action ordering through part of the planning process, but outputs plans in which actions are totally ordered.

Constraint-solving techniques are relevant to least-commitment as they typically operate by eliminating inconsistent choices without committing to particular values. Constraint-solving has been used to implement least-commitment on the selection of objects used in actions [8], other aspects of resource reasoning [13], and interval durations ([10], [12], [3]). Chapman's TWEAK system [2] applies constraint-solving to temporal and non-temporal aspects of planning, and Yang has described an application of finite domain constraint solving to reason about conflict resolution in planning, taking the TWEAK system as a basis [14]. A planning system that uses temporal constraint propagation in a sophisticated and flexible way was introduced by Allen and Koomen in [1], but the overhead imposed by the temporal reasoner proved to be a bottleneck [5].

In *parc*PLAN, constraint-solving supports least-commitment in many aspects of planning, including resource allocation and action ordering. Temporal reasoning is also achieved through constraint-solving. Time points are represented by finite-domain variables, allowing *parc*PLAN to utilise finite-domain constraint propagation techniques. This commitment to a discrete rather than continuous model of time means that time points cannot be placed arbitrarily close as there is a smallest indivisible time unit between non-simultaneous time points, and that there is a maximum time period that a plan can occupy. However, these factors do not preclude the design of a useful and theoretically sound planner. The fact that the planner operates on a strong underlying temporal representation supports the interval-based, non-linear planning strategy we employ and provides for considerable expressiveness in problem representation.

The *parc*PLAN architecture is *generic*: to apply it to some problem domain it is necessary to provide specifications of actions and integrity constraints. Integrity constraints arise from the problem domain and can be classified as *temporal* or *non-temporal.* Temporal integrity constraints place bounds on the durations of properties and actions, or prohibit certain temporal configurations of properties and/or actions. Non-temporal integrity constraints express other aspects of the problem domain—such as quantitative constraints on resources.

A feature of *parc*PLAN is the capability to represent and minimise costs associated with plans. Such facilities are essential for a planning architecture to have industrial potential, but have been neglected in the planning literature. In *parc*PLAN it is possible, for example, to associate a cost with the performance of a particular action by a particular agent, or to associate a cost with the total quantity of some resource required for the execution of a plan. The architecture includes a number of different cost minimisation strategies that can be utilised within the planning algorithm. The effect of different options can easily be explored by setting global parameters of the architecture, allowing the algorithm to be tuned to particular applications. As a consequence of these features - as well as facilities available in the architecture with respect to temporal reasoning - *parc*PLAN can be seen as a generic shell for solving planning, scheduling and resource management problems involving cost minimisation. In this paper we describe application to blocks-world planning including resources, and to a resource allocation problem provided by British Airways.

2 The *parc*PLAN Architecture

2.1 Representation

Properties and actions holding or occurring over time are represented by terms indexed to time intervals. We use the notation *holds(p,start,end)* to denote that the property or action p holds or occurs over the maximal time interval *[start,end).* The start and end points of time intervals are either integers or finite-domain variables. During planning, the planner works with a database and a set of goals. Both are represented as sets of terms of the form *holds(p,start,end).*

2.2 Action Specification

Actions are specified by the following:

1. a list of *conditions*
2. a list of *effects*
3. a list of *resource variables*, with associated resource types
4. a list of *constraints*
5. a *cost term*

The conditions and effects of an action are properties or actions indexed to time intervals. The conditions are required for the action to be executed, while the effects are those properties and actions which the action establishes. In particular, a term representing the specified action itself appears among the effects. The resource variables are finite-domain variables that represent resources required in the execution of the action. These variables are associated with a type, according to the type of resource they represent. The constraints are asserted when the action is introduced. The cost term is used in planning to derive the cost of performing the action. The form of the cost term is unspecified by the architecture, and its evaluation is handled by application-specific predicates.

The following specification of the *move* action of the blocks world shows the expressiveness available in *parc*PLAN. The details are discussed below:

conditions:	*holds(on(Block,From),T1,Start),* *holds(clear(To),T2,End),* *holds(clear(Block),T3,T4)*
effects:	*holds(move(Block,From,To,Arm),Start,End),* *holds(on(Block,To),End,T5),* *holds(clear(From),T6,T7)*
resource variables:	*Arm-arm*
constraints:	*designates([Block,From,To],block),* $Block \neq From, Block \neq To$, $T6 = Start + 2, T2 \leq End - 2$, $T3 < Start, End < T4, Start + 5 \leq End$
cost term:	*cost(move(Block,From,To,Arm),Start,End)*

Here, the temporal representation offers considerable expressiveness by allowing the point from which the destination block must be clear and that from which the source block becomes clear to be different to the endpoints of the interval *[Start,End)* over which the action occurs. In the above specification the source block becomes clear only after two time units have elapsed from the beginning of the action—perhaps only after the moving block has been lifted sufficiently away from it—through the temporal constraint $T6 = Start + 2$. Similarly, the constraint $T2 \leq End - 2$ requires the destination block to be clear for two time units before the moving block arrives. This kind of flexibility allows maximum interleaving of *move* actions to be achieved by the planner. The final constraint, $Start + 5 \leq End$, forces the action to take at least 5 time units.

2.3 Resource Handling and Costs

For particular applications, resources are specified by a type and quantity. In some cases it is useful to define a notion of equivalence of resources of some particular type, as this can improve the efficiency of some strategies open to the planner by avoiding isomorphic solutions of identical cost.

There are four generic mechanisms that allow costs to be associated with plans. The first is a global notion of cost arising from the total number of resources used in the plan. For particular applications, the cost of using a resource of a each type must be specified. *parc*PLAN sets up delayed calls that monitor the usage of the resource as planning proceeds, and these calls update the value of the cost expression as necessary.

The second notion of cost is local to individual actions, and typically represents the cost of performing a particular action by a particular agent. This is handled by means of the cost term included in the action specification, as described above. The cost term should contain those parameters of the action relevant to the cost, and the evaluation of the cost must be carried out by an application-specific definition. As certain parameters of the action may not have been fixed at the point in planning that the action is introduced, it is often essential to use delay mechanisms in this procedure so that the cost will be re-evaluated as the parameters are filled in.

The third notion of cost is neither local to individual actions nor global to the plan, but is associated with the number of changes in a sequence of values. The fourth notion allows specification of preferred values for variables, and can for example be used to specify a preference that a particular agent carry out a particular action. In both cases, delay mechanisms are used to monitor assignments and update the cost term when necessary.

2.4 Integrity Constraints

Integrity constraints are application-specific and state certain relations between of pairs of properties and/or actions which cannot occur in a valid plan. They are often used to state that two properties cannot hold simultaneously. Integrity constraints for a specific application are defined by clauses for the predicate

integrity_constraint(p,startp,endp,q,startq,endq)

These clauses are used by the planner to ensure integrity as the plan is built up through the addition of properties and actions. Clauses defining integrity constraints are typically of the form

integrity_constraint(p,Startp,Endp,q,Startq,Endq) :-
    blocking_ask test $\rightarrow$ *action.*

Here, *test* is a constraint involving variables from *p*, *q*, *Startp*, *Endp*, *Startq* or *Endq*, whose truth or falsity is monitored by the generic utility *blocking_ask* supplied by the architecture. If *test* becomes true, *action* is executed, while if *test* becomes false, no action is taken. The action is often to assert a temporal constraint to make the intervals *[Startp,Endp)* and *[Startq,Endq)* disjoint.

2.5 The Planning Algorithm

The planning algorithm can be divided into three phases: action-introduction, collapsing, and labelling of resource and time point variables. The algorithm is broadly similar to that used by Allen and Koomen in [1], although *parc*PLAN goes beyond Allen and Koomen's planner in including metric temporal reasoning and facilities for resource handling.

Before describing the action-introduction phase we must define the notion of potential collapse for a goal with respect to a database:

> A goal *holds(q,startq,endq)* is said to have a *potential collapse* with respect to a database if the goal unifies with an element of the database.

In action-introduction, we select a goal from the current set of goals which has no potential collapse with respect to the current database. Given such a goal, we find an action which achieves the goal, that is, the goal unifies with one of the effects of the action. The action is then introduced by:

1. making the unification between goal and effect
2. adding the effects of the action to the current database

3. adding the conditions to the current goal set and removing the selected goal
4. asserting the constraints
5. evaluating the cost term and adding the result to the current cost

The action-introduction phase then continues with the updated values of database, goals and cost.

When all goals have a potential collapse, *parc*PLAN enters the collapsing phase. The task here is to simultaneously unify all goals with elements of the database. The difficulty is that many combinations of individual collapses are mutually exclusive as they require variables of terms in the database or goals to take incompatible values. In *parc*PLAN, the collapsing phase is handled as a constraint satisfaction problem. Each goal is associated with a finite-domain variable whose domain is a representation of the potential collapses for that goal. The constraints are that each goal unifies with some element of the database. Choosing a value for a variable chooses the particular element of the database with which the goal associated with the variable is unified. The architecture sets up delayed goals which monitor potential collapses during the choosing of values and implements a looking-ahead strategy over the constraints.

After successfully choosing values for the collapse variables, the database may still contain uninstantiated variables that represent resources or time points. The final phase of planning is to choose values for these remaining variables, whilst minimising the cost function. In fact, in certain variations on the basic planning algorithm that are available in the generic architecture, the choosing of values for collapse variables and of these other variables is interleaved. These various strategies are described in the next section.

If, however, no solution to the collapse exists, the planner must return to the action-introduction phase. At this point, a problem exists as to which goals to regard as unsatisfied and are therefore to be achieved through further actions [1]. This problem is not yet addressed in *parc*PLAN. In order to solve this problem, the failure of the collapse needs to be analysed and a subset of incompatible collapses extracted. The fact that the collapsing phase is handled as a constraint satisfaction problem in *parc*PLAN at least provides a start here, for the use of the first-fail principle in making the collapses leads to early detection of failure.

2.6 Cost Minimisation

The phases of collapsing and labelling resource and time point variables can be subject to a cost minimisation process. The basic algorithm used is branch-and-bound. The best labelling strategy to use is application-specific and *parc*PLAN provides various strategies at the generic level. The ability to compare their behaviour for specific applications by simply setting the value of a global parameter is very valuable. The optimisation strategies currently available are:

1. *length_of_plan*
2. *resource*
3. *eq_resource*
4. *resource_local_cost*
5. *eq_resource_local_cost*

The *length_of_plan* strategy minimises the plan length. The collapse variables are labelled first, then the time points of intervals over which actions occur, and then the remaining variables.

The *resource* strategy minimises the cost function built up during planning as described in Section 2.5 above. Resource variables are labelled first, followed by collapse variables and time point variables. When choosing the value of resource variables, the system prefers values that have already been used in the plan to unused values, helping to obtain a low bound early in the search.

The *eq_resource* strategy is a variation on the *resource* strategy. The values in the domain of the resource being labelled are divided into equivalence classes according to the application-specific definition of equivalence. The idea is that the values in a single equivalence class are interchangeable in a plan without disrupting the overall feasibility of the plan or changing its cost. This notion allows the construction of many isomorphic solutions of identical cost to be avoided in a way that is seen in its most basic form in the graph-colouring example given in [11]. In assigning a value to a resource variable, the values considered from each equivalence class are restricted to those values already used in the plan plus *only one unused value.*

In the *resource_local_cost* strategy, labelling is again initially on the resource variables. When labelling a resource variable, the available values are ranked in order of the increase in the lower bound of the cost function introduced by choosing the value for the variable at this stage of the labelling process. Values that introduce smaller increases in the cost function are preferred.

Finally, the *eq_resource_local_cost* strategy combines the partitioning of resources into equivalence classes with the local cost strategy.

3 Performance

We first describe performance of *parc*PLAN for a formulation of the blocks-world problem including resources. There is a single action, which is to move a block from one position to another using a designated arm. Arms are treated as resources, the number of arms is limited, and one option is to minimise the number of arms needed to execute the plan[1]. Another option is to minimise the length of the plan. We also regard the table as limited to a number of positions on which blocks can be placed.

The following is a randomly generated problem with five blocks and five available arms:

Initial state:	*on(2, 1), on(1, 5), on(5, table1),on(4, 3), on(3, table2)*
Goal state:	*on(1, 2), on(2, 3), on(3, 4), on(4, 5), on(5, table3)*

Optimising on the number of arms required, we obtain the following solution::

move(2, 1, 3, arm2)	over interval *[1, 9]*
move(1, 5, 2, arm1)	over interval *[2, 10]*
move(5, table1, table3, arm3)	over interval *[3, 4]*
move(4, 3, 5, arm3)	over interval *[5, 6]*
move(3, table2, 4, arm3)	over interval *[7, 8]*

The action *move(Block,Source,Destination,Arm)* is read as "move *Block* from *Source* to *Destination* using *Arm*". This solution is found and proved to be optimal with respect to the number of arms required in 1.25 seconds user-time. This is for the $\mathrm{ECL}^i\mathrm{PS}^e$ implementation running on a Sparc 2.

Setting the optimisation criterion to length of plan, we obtain the following plan (which uses all five arms):

move(2, 1, 3, arm1)	over interval *[1, 7]*
move(1, 5, 2, arm2)	over interval *[2, 8]*
move(5, table1, table3, arm3)	over interval *[3, 4]*
move(4, 3, 5, arm4)	over interval *[4, 5]*
move(3, table2, 4, arm5)	over interval *[5, 6]*

This solution is found and proved to be optimal with respect to plan length in 0.78 seconds user-time.

For ten randomly generated problems involving ten blocks and five arms with optimisation set to minimise the number of arms required, the average user-time to find and prove the optimal

[1] *parc*PLAN minimises the number of arms required for a plan with a minimal number of actions: other plans requiring fewer arms but more actions may exist.

plan ranged between 6.17 seconds and 16.12 seconds, the average being 9 seconds. For problems of this scale in which no plan was found due to *parc*PLAN committing to a collapse after introducing insufficient actions, evaluation failed in an average of 2 seconds.

For ten randomly generated problems involving twenty blocks and twenty arms with optimisation set to minimise the length of the plan, the average user-time to find and prove the optimal plan ranged between 6.03 seconds and 10.5 seconds, the average being 7.3 seconds. For problems of this scale in which no plan was found, evaluation failed in an average of 4.4 seconds.

We have also applied *parc*PLAN to a resource allocation problem involving cost minimisation provided by British Airways. The problem has two aspects: to allocate aircraft to flights over the period of a day, and to perform re-allocation in the event of delays during the day. For start-of-the-day allocation, the input consists of:

1. a set of planes, with specification of initial position,time from which available for operation, and operational status of individual aircraft
2. a set of flights, specified by source and destination airports, departure time and arrival time
3. a set of tours - sequences of flights which it is preferred should be carried out by the same aircraft
4. a set of maintenance requests, specifying the aircraft required, and the location and time period of the maintenance

For re-allocation, the input is similar, but includes the current allocation of aircraft to flights, specification of a delay to the arrival-time of a flight, and the time from which re-allocations can be made. In both cases, the output required is an allocation of aircraft to flights that is feasible and which minimises a cost function. In the flight allocation problem, costs can arise in three ways:

1. from the number of aircraft used in the allocation
2. by allocation of an aircraft of special operational status to flights of certain length or to certain destinations
3. by breaking a tour through allocation of different aircraft to adjacent flights

These costs are each assigned a weight and are combined into a single cost function. In re-allocation, changing the aircraft previously allocated to a flight also incurs a cost.

A characteristic of the problem derives from the fact that Heathrow Airport, as the centre of British Airways' operation, is very much the hub of activities. The majority of aircraft start the day at Heathrow, almost all flights either depart from or arrive at Heathrow, and maintenance is carried out at Heathrow. While the allocation to flights from smaller airports is virtually deterministic, the allocation to flights leaving Heathrow, particularly later in the day when there have been many incoming flights, is strongly undetermined. As a consequence, there are many similar allocations that differ only in the choices made for one or two flights, and these allocations tend to have similar cost.

We present timings for start-of-the-day allocation for a data set with sixteen aircraft, forty-one flights and four maintenance requests. Seven of the aircraft are of special operational status. This data set is typical of the problem size and was provided by BA. The timings given are seconds of user-time to find the optimal solution and to the conclusion of the branch-and-bound search proving optimality , using the current ECL^{i}PS^e implementation of the architecture running on a Sparc 2. A "-" in the table indicates that the computation was terminated after 30 minutes without result.

Heuristic	time to optimum	conclusion of proof
none	-	-
local cost	12	-
equivalent resources	44	541
equivalent resources + local cost	7.5	607

The equivalent resources strategy in combination with the local cost heuristic gives the fastest route to the optimal solution. However, the subsequent proof of optimality takes longer than under the equivalent resources strategy alone, as the more complex strategy does more work at each step in the labelling. We feel that there is scope to significantly reduce the time taken up by the proof of optimality by using one strategy to generate a good first solution and a second strategy in the subsequent search for a better solution.

The following table shows the effect of differentiation of resources on the equivalent resources strategy. Timings are given for allocation in the data set used above while varying the number of aircraft with special operational status. In fact, the maintenance requests were omitted from the data as they also effect the equivalence of aircraft. As the number of aircraft of special status increases, so does the number of equivalence classes that must be considered. Timings are user-time in seconds.

	Aircraft with special status					
	0	1	2	3	4	5
time to optimum	6.3	6.4	6.5	6.5	6.5	6.5
conclusion of proof	1.28	1.63	6.67	33.22	81.92	380.02

The local cost heuristic is particularly useful in the case of re-allocation, as introducing a change from the original schedule immediately increases the cost function, so that the values of the original schedule tend to be selected first. For the data set used above and a delay to a flight due to arrive at Heathrow at midday, the optimal repair is found and proved optimal in 3.7 seconds.

4 Conclusions

We have described the generic planning architecture *parc*PLAN, which uses constraint-solving to handle both temporal and non-temporal reasoning tasks. The architecture allows considerable temporal sophistication in the specification of actions, integrity constraints and planning problems, and naturally produces plans with a high degree of concurrency in action execution. *parc*PLAN includes generic mechanisms for cost handling and minimisation. The current CLP implementation of *parc*PLAN is able to solve large-scale planning and resource allocation problems with considerable efficiency.

Acknowledgements This research was carried out on ESPRIT project No. 5291, CHIC. We thank our colleagues on the CHIC project and at IC-Parc.

References

[1] J. Allen and J. Koomen. Planning using a temporal world model. In *Proceedings of the 2nd IJCAI*, Karlsruhe, 1983.

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

[3] K. Currie and A. Tate. O-plan: control in the open planning architecture. In *Proc. of the BCS Expert Systems 85 Conf.*, Warwick, 1985. Cambridge University Press.

[4] R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[5] J. C. Hogge. Tplan: a temporal interval-based planner with novel extensions. Technical Report UIUCDCS-R-87-1367, University of Illinois at Urbana-Champaign, September 1987.

[6] B. Richards, Y. Jiang, and H. Choi. On interval-based temporal planning - an IQ strategy. In Z. W. Ras and M. Zemankova, editors, *Proc. of the 6th International Symposium ISMIS 91*, pages 226–235, Charlotte, USA, 1991. Springer Verlag.

[7] E. D. Sacerdoti. *A structure for plans and behaviour.* American Elsevier, New York, 1977.

[8] M. Stefik. Planning with constraints (MOLGEN: Part 1). *Artificial Intelligence*, 16(2):111–139, 1981.

[9] G. J. Sussman. A computer model of skill acquisition. Technical Report AI-TR-297, MIT AI Laboratory Memo, Cambridge, MA, 1973.

[10] A. Tate. Generating project networks. In *Proc. of IJCAI-77*, Boston, MA, 1977.

[11] P. Van Hentenryck. *Constraint satisfaction in logic programming.* MIT Press, 1989.

[12] S. A. Vere. Splicing plans to achieve misordered goals. In *Proc. of IJCAI-85*, pages 1016–1021, 1985.

[13] D. E. Wilkins. Domain independent planning: representation and plan generation. *Artificial Intelligence*, 22:269–301, 1984.

[14] Q. Yang. A theory of conflict resolution in planning. *Artificial Intelligence*, 58:361–392, 1992.