Opportunistic Branched Plans to Maximise Utility in the Presence of Resource Uncertainty

Amanda Coles¹

Abstract. In many applications, especially autonomous exploration, there is a trade-off between operational safety, forcing conservatism about resource usage; and maximising utility, requiring high resource utilisation. In this paper we consider a method of generating plans that maintain this conservatism whilst allowing exploitation of situations where resource usage is better than pessimistically estimated. We consider planning problems with soft goals, each with a violation cost. The challenge is to maximise utility (minimise the violation cost paid) whilst maintaining confidence that the plan will execute within the specified limits. We first show how forward search planning can be extended to generate such plans. Then we extend this to build branched plans: tree structures labelled with conditions on executing branches. Lower cost branches can be followed if their conditions are met. We demonstrate that the use of such plans can dramatically increase utility whilst still obeying strict safety constraints.

1 INTRODUCTION

Opportunities for communication with remote autonomous agents are often scarce, whether in space, underwater, or disaster-recovery environments. The ideal of on-board planning is currently difficult to achieve due to two primary factors: the reluctance of controllers to trust fully autonomous behaviour and the computational constraints of remote agents. It is therefore necessary to provide agents with plans for long periods, whenever communication is possible.

In such situations conservatism is ubiquitous: the desire for continued safe operation of autonomous agents restricts the amount of exploration that can be performed. To give an example, it is estimated that the Mars exploration rover Sojourner spent 50% of its time on the surface idle as a result of either having completed all planned activities, or due to plan failure [10]. Space agencies often generate plans on the ground, primarily by hand or with supporting software, using highly conservative estimates of energy consumption [16].

In this work, we consider the problem of creating plans that are cost-effective, whilst adhering to the strict safety constraints required. We consider over-subscription problems, where each goal has an associated cost, incurred if it is not reached. Such goals may arise, for instance, from the many competing science activities a Martian rover could perform. We first extend a forward-chaining over-subscription planning approach to support uncertainty in the numeric effects of actions. The resulting planner is capable of optimising quality in terms of the goal costs, whilst ensuring the plan completes with the requisite degree of confidence.

Using this planner, with a high confidence level, one can find a solution that will succeed under a wide range of outcomes. This is both a strength, and a weakness: the plan is statistically likely to succeed, but is also pessimistic. At execution time, we have additional knowledge – we know the resource usage of past actions – and although we must be pessimistic about the future, we may reach a point where a lower-cost goal state is reachable with acceptable confidence.

As on-board replanning is often not possible, we propose a technique for augmenting a plan with conditioned branches for optional use at execution time. We search for these branches by calling the planner several times, from the states along the plan reached by assuming uncertain numeric effects have their expected (mean) outcome. As the original high-confidence plan is pessimistic, it is likely that resource usage will be closer to the mean than to the values the planner is permitting, and hence, at execution time, such branches will often be used. Our approach gives the advantage of maintaining control over operations (only a finite space of plans could be executed), whilst allowing better costs through exploiting opportunities that arise during execution. This is related to the idea of creating policies, but differs in that we do not have to generate complete policies for all eventualities.

To evaluate our approach, we compare to a single pessimistic plan; a simulation of what could be achievable by on-board replanning; and make an indicative comparison to a policy-based approach. Our results show improved utilities with respect to a single plan, and indicate scalability with respect to policy based approaches.

2 BACKGROUND

Here we define formally the problem we are solving and compare existing approaches in the literature to solving related problems.

2.1 Problem Definition

A planning problem is a tuple \( (F, v, I, G, A, C, \theta) \) where:

- \( F \) is a set of propositional facts; \( v \) is a vector of numeric variables;
- \( I \) is the initial state: a subset of \( F \) and assignments to (some) variables in \( v \);
- A condition is a first-order logic formula over facts in \( F \) and Linear Normal Form (LNF) constraints on \( v \), each written:

\[
(w,v \ op \ l)
\]

...where \( op \in \{>, \geq\}; l \in \mathbb{R}; \) and \( w \) is a vector of real values.
- \( G \) describes the goals: a set of conditions. Each \( g \in G \) has an associated cost \( c(g) \in \mathbb{R}^+ \) if \( g \) is not true at the end of the plan.
- \( A \) is a set of actions, each \( a \in A \), with:
  - \( \text{Pre}(a) \): a (pre)condition on its execution;
  - \( \text{Eff}^-(a), \text{Eff}^+(a) \): propositions deleted (added) by \( a \);
  - \( \text{Eff}^{\text{num}}(a) \): a set of numeric variable updates that occur upon applying \( a \). Each is of the form \( (v \ op \ D(v, \text{params})) \) where

¹ King’s College London, UK.amanda.coales@kcl.ac.uk
Executing (rather than finding) plans with choice points [6, 12].

takes the opposite view to our work, generating optimistic schedules in scheduling has considered building branches ‘just in case’ [7]: this
uous numeric variables and uncertain numeric effects. Related work very different: probabilistic propositional effects, rather than contin-
certainty in the problem. The problems being considered though are
ing cost in the presence of soft goals, and considering how a plan
these techniques, addressing the additional challenges of minimis-
ingulation when operations staff wish to maintain tight control and confi-
able trajectories, is less scrutable than one with fewer options, a limi-

3 OVER-SUBSCRIPTION PLANNING UNDER RESOURCE UNCERTAINTY

Over-subscription planning problems are characterised by a surfeit of goals, and a means of determining which combinations of goals are preferable. Each goal \( g \) is assigned a cost \( c(g) \), and the metric cost of a plan is the sum of the costs of the goals which it does not reach.

One plan is then preferable to another if its metric cost is lower. When planning with resource uncertainty, we have the additional consideration that some plans are more or less likely to complete. There is an inherent trade-off: a good high-confidence plan will be more conservative and hence have higher cost than a good less-confident plan.

In this section, we explore the issues arising where over-subscription and uncertainty meet. First, we detail how we adapt a forward-chaining search approach for over-subscription planning, to consider the uncertainty in effects on numeric variables and to ensure the plan succeeds with the desired confidence. Second, we discuss a compromise between a single, linear solution, and a full-policy solution to this class of problems, extending a conservative initial plan with branches for use at execution time if conditions are suitable.

3.1 Adapting Forward-Chaining Search

In order to effectively use a forward-chaining approach for the class of problems featured in this work, two important considerations are how to manage uncertainty during search, and which heuristic to use.

For the first, we turn to the planner RTU [2] and its Bayesian Network approach, described earlier in Section 2.1. For a given plan, the Bayesian network captures the distribution of variables’ values in each of the states along the plan trajectory, given the effects of the actions. At each state during search, we can query the network to ensure the plan will succeed acceptably often: as noted in Section 2.1, with \( P \geq \theta \), each state \( S \) must satisfy the conditions \( C \), and if an action \( a \) is applied in \( S \), \( S \) must satisfy any preconditions of \( a \).

This part of the approach does not change fundamentally with the shift to over-subscription planning. Rather, what is more involved is the heuristic guidance needed. As in the case where all goals are hard, we need some sort of estimate of ‘actions to go’ until all goals are met. Further, as some goals might not be reachable from a given state, we would like to identify this too: if we have already have an incumbent solution with some cost, but carry on searching, we can prune states based on knowledge of unreachable soft-goals, i.e. reachable cost. To serve both of these purposes, we take as our basis the non-LP heuristic used in LPRP [5]: a variant of the Metric Relaxed Planning Graph (RPG) heuristic [13], extended to handle PDDL3 preferences. As the ‘soft goals’ in this work are a subset of
2. Action layer
3. 6. A relaxed plan is extracted, containing actions to meet each of the heuristic (computed at each state) as follows:

It is important to note that at point 5 here, graph expansion only stops when each goal has been met, or has been proven to be unreachable even under the relaxed semantics. Thus, if a goal does not appear, it cannot be satisfied in any state reached from $S$. This is a rich source of heuristic knowledge about the cost of reachable states: if the metric comprises a weighted sum of binary variables denoting whether each goal is achieved, an admissible estimate of the cost of reachable states is the sum of cost of the goals not met during graph expansion. Then, as discussed above, if search is bounded by the cost of an incumbent solution, any state with admissible cost in excess of the metric comprises a weighted sum of binary variables denoting whether each goal is achieved, an admissible estimate of the cost of reachable states:

3.2 Opportunistic Branching

This forward-chaining search approach finds a sequential solution plan to a planning problem which, statistically, will respect each constraint, given the uncertain nature of execution. When planning with a high degree of confidence, for instance, $\theta = 0.999$, the resulting plan is necessarily quite conservative. It will still occasionally fail (with $P < 0.001$) but on average, the plan will not come close to violating its constraints and may therefore compromise cost.

An alternative to finding a linear solution plan, addressing this limitation, is to find a policy: state–action pairs that, beginning with the initial state, dictate which action to execute. In the presence of continuous variables, some sort of approximation is necessary, with each policy state representing a number of reachable states. Otherwise, in theory, when applying an effect whose outcome is governed by some distribution, an infinitely large number of states is reached, identical modulo different values of the variable altered. A linear plan is a coarse approximation, where all the states reachable after an action are collapsed into single policy state, associated with which is the next step of the plan. Such a representation is compact, but as discussed, has its limitations. More sophisticated approaches (e.g. [15]) use discretization approaches, where applying an action in a state will lead to one of a finite number of policy states. Such policies have better cost performance than a linear plan, but are considerable in size, with scalability being the main limitation of such approaches.

As a compromise measure between these, we build a partial policy. The spine of the policy is a linear plan that, with $P \geq \theta$, will execute successfully. Attached to this are branches for opportunities which, if execution-time conditions permit, can be followed to reach a lower-cost goal state. The structure of such plans can be represented naturally as tree $(V,E)$. Each $v \in V$ is an action from $A$, with $v^0$ (the root of the tree) corresponding to the first plan step. Each $(i,j) \in E$ is labelled with one or more condition–cost pairs, $(f_k,c_k)$, where:

- After applying the action $v^i$, if the state $S_i$ reached satisfies one of these conditions $f_k$, execution may continue with step $v^j$;
- If there are several $(i,j) \in E$ with at least one condition satisfied, a single $v^j$ is chosen. We select (arbitrarily) one of:

$$\arg\min\{c_k | (f_k,c_k) \in \text{labels}(i,j) \land S_i \models f_k\}$$

Each $f_k$ is derived by computing, using the Bayesian network, the weakest preconditions of a plan with cost $c_k$ rooted at $v^j$. It specifies the constraints on the continuous state variables required to ensure that, statistically, if the $j$ branch is chosen, it will execute successfully with $P \geq \theta$. As a simple example, consider a branch with a single resource-using action, with an effect $\nu = N[-10,3]$, i.e. decreasing $\nu$ by a normally distributed amount (mean 10, standard deviation 3). If $\theta = 0.99$ and there is a condition $c \in C$ that states $(\nu \geq 0)$, this must be true with $P \geq 0.99$ after the effect has occurred. Thus, the weakest precondition of this branch is calculated as the smallest value of $\nu$ for which this holds: approximately, $\nu \geq 15.35$. Note that this slightly changes the interpretation of $\theta$ for branched plans: whilst the linear plan generated from each node completes with confidence $\theta$, the overall confidence in completion of the branched plan may become less than $\theta$. We return to this point in our evaluation.

Algorithm 1 outlines our branch-planning approach. Initially, we

<table>
<thead>
<tr>
<th>Algorithm 1: Branch Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data:</strong> $S$, an initial state; $U$, a cost bound on search</td>
</tr>
<tr>
<td>1 $\Pi \leftarrow \text{plan}(S,U)$;</td>
</tr>
<tr>
<td>2 $(V,E) \leftarrow$ tree for II, with vertex $i$ denoting step $a_i$;</td>
</tr>
<tr>
<td>3 $U' \leftarrow$ cost of $\Pi$;</td>
</tr>
<tr>
<td>4 $S' \leftarrow S$;</td>
</tr>
<tr>
<td>5 for each $a_i \in \Pi$ in order do</td>
</tr>
<tr>
<td>6 $S'' \leftarrow \text{apply } a_i \text{ to } S'$;</td>
</tr>
<tr>
<td>7 $S'' \leftarrow S'' \text{ setting each variable to its mean value}$;</td>
</tr>
<tr>
<td>8 $(V',E') \leftarrow \text{BranchPlan}(S'',U')$;</td>
</tr>
<tr>
<td>9 if $V'$ is non-empty then</td>
</tr>
<tr>
<td>10 $j \leftarrow \text{root of } V'$;</td>
</tr>
<tr>
<td>11 $i' \leftarrow i + 1$;</td>
</tr>
<tr>
<td>12 while $i'$ and $j$ have at most one outgoing edge</td>
</tr>
<tr>
<td>13 and are labelled with the same action do</td>
</tr>
<tr>
<td>14 increment $i'$ and $j$ by 1;</td>
</tr>
<tr>
<td>15 add subtree of $(V',E')$ rooted at $j$ to $(V,E)$;</td>
</tr>
<tr>
<td>16 for each $a_i \in \Pi$ in order do</td>
</tr>
<tr>
<td>17 for each $(i,j) \in E$ do label with $(\text{condition},\text{cost})$ of plans from $j$;</td>
</tr>
<tr>
<td>18 for each $(f,c)$ on edge $(a_i,a_{i+1})$ do if $c = U'$ then $f = \emptyset$;</td>
</tr>
<tr>
<td>19 return A plan tree, $(V,E)$</td>
</tr>
</tbody>
</table>
call \textit{BranchPlan}(I, \infty), and hence at line 1, the planner is invoked. From lines 5 to 15, the steps of the plan found are considered in turn. Line 7 is key to our branch-generation approach: each variable is set to its 50th percentile, i.e. assuming resource change so far has been nominal, rather than extreme\(^2\). This forms the basis of the initial state for a recursive call to \textit{BranchPlan}. If this returns a non-empty plan tree, then due to the cost bound imposed, it necessarily reaches a goal state with better cost than that of \(I\), and the tree is merged in. As the plan tree may begin with a plan segment identical to that following \(i\), line 12 skips over the first steps of the new plan tree whilst it remains the same as the plan that would be executed anyway. Then, any remaining portion of the tree, rooted at \(j\), is added at the point where the plan diverges. Having built the tree, the final loop (lines 16–18) label the edges out of each step in the plan with condition–cost pairs: one such pair \((f, c)\) for each tree traversal (i.e. plan tail) reachable from \(j\), labelled with its weakest preconditions \(f\), and the cost reached \(c\). The exception are edges along \(I\): at line 18, \(f\) for these is cleared, i.e. the default execution behaviour if no acceptable other edge’s condition is met is to continue with \(I\).

An example of the output from this algorithm is shown in Figure 1 (two non-branching sections are omitted for space reasons). The initial plan found, cost 116, is the right-most path through the tree. The octagonal vertices denote points from which the recursive call to \textit{BranchPlan} found a better plan: from the first, with cost 76.5; from the second, cost 0. In both cases, the solutions overlapped with the existing plan, so the branches are added where they first diverge. After \textit{com	extunderscore soil	extunderscore data} in \(w\), \textit{sample	extunderscore soil	extunderscore store} \(w\), we see an example of multiple edge labels: the right-most path is labelled with two condition–cost pairs. Which path is taken depends on the value of energy at execution time: from \([219.2, 354.7]\) the left path; otherwise, the right path.

Note that we assume it is reasonable to consider the plans in a sequential order. In the presence of multiple agents, this is not necessarily reasonable as, conceivably, permitting non-related steps in the solution returned by the planner may lead to a more effective branch plan. For instance, if branching after \(a\) but before \(b\) leads to a cost-effective branch, but not vice-versa, this will not be found if the linear plan happens to order the steps \(b, a\). An effective solution to this is still an open question. One option is to fix the division of goals between agents \textit{a priori} and plan for them individually. (This would be necessary, in any case, if the agents are unable to communicate or are not orchestrated, once execution has begun.)

\section{Evaluation}

To evaluate our approach we investigate its performance on three problems. The first, a Martian rover problem, is derived from the over-subscription variant of the Rovers domain used in the Fifth International Planning Competition \cite{11}. We keep the soft goals from the domain, and one rover, but rather than adding ‘traversal cost’ to the metric we made each navigate action consume a normally distributed amount of energy (mean given by its traversal cost), and then added to \(C\) (always \((\geq \text{energy } 0))\). Our second domain is based on the activities of an Autonomous Underwater Vehicle (AUV), with soft-goals to have performed inspection, cleaning, and maintenance tasks. Navigating and performing tasks takes a normally distributed amount of time; the challenge is to minimise total soft-goal cost, whilst meeting a deadline to the desired confidence. Our third domain concerns planning for a power substation \cite{3}, minimising the number of control actions (stepping/switching components) to keep the voltage within ±5% of 132kV, given a demand forecast. Our model is an extension of this where, now, at each point, demand is offset by a normally distributed amount from the forecast, and the planning of control actions must account for this fluctuation (with confidence \(\theta\)).

In the first two domains, when constructing linear plans, the Bayesian network solver can use analytic methods; in the third, it cannot, and Monte Carlo sampling is used, effectively simulating execution. In all three cases, when evaluating branch plans, Monte Carlo sampling is used. When sampling branched plans, the branch to follow is selected using the criteria in Section 3.2. Plan cost is recorded after each sample: we report the average cost achieved over 10000 samples, and the percentage of failures.

We consider two definitions of failure, the first ‘standard’ simply executes the plan with no safety checks and reports failure if a constraint in \(C\) is broken, this is useful for measuring properties of the plan. The second, ‘failsafe’, stops plan execution if some condition is met, and reports cost. In the transformer domain, where hard goals are present, we cannot simply stop so the failsafe is to a reactive closed-loop executive. Failsafe simulates a more realistic execution framework; the condition for invoking the failsafe measure is that in the plan tree there is no edge out of the most recently executed step with a satisfied condition\(^3\). This allows a fair comparison to a replanning approach, that executes an action of the plan then generates a new plan given the resulting state, and will thus exhibit this termination behaviour by default.

In theory an optimal plan will eventually be produced for every call of the planner; in practice, however, this only occurs in small problems as proving optimality requires exhaustion of the search space. We therefore impose time limits on both computation of the initial plan \((T_i)\) and each branch plan \((T_b)\). This introduces a trade-off between planning for longer to improve cost and ensuring termination. We use the results from Table 1 to make a number of comparisons between linear plans, branched plans and replanning.

\textbf{Branched vs Linear Plan Failure Rate}. Whilst the linear plan generated will meet the specified confidence threshold, the branched

\footnote{\textsuperscript{2} Other percentiles from the CDFs could be used.}

\footnote{\textsuperscript{3} Line 18 of Algorithm 1 is removed, so continuation of \(I\) (vs invoking the failsafe) is conditional on its success having likelihood \(\theta\).}
Table 1. Average Cost and % Failure Rates (FR) for the Initial Linear Plan (O); the Branched Plan (B); the Equivalent-Certainty Linear Plan (E); and Online Replanning (R). θ=0.95 for Transformer, 0.99 otherwise. Tests ran on a 3GHz CPU, all planner calls restricted to 3GB RAM.

Comparing the cost achieved with by a linear plan (Cost(O)) with $T_i=60$ and that of the branched plan (Cost(B)), in Table 1, we can see that on most problems the branched plan is able to achieve much lower cost plans upon execution; however, as discussed above this does come at the cost of a slightly increased failure rate. We therefore did further experiments: finding a linear plan, with $\theta=(100-\text{FR}(B))\%$ (N.B. often greater than its prior value), and allowing 900 seconds. The results are shown in Table 1 as Cost(E) and confirm that a linear plan cannot deliver equivalent performance.

We also ran the planner with $\theta=0.99$ and $T_i=900$, to confirm that the high cost of the linear plans found initially is not simply due to limited planning time. These are shown as ‘Cost(O) | 900’ in the table. There are relatively few problems (shown in bold) on which the cost of the original linear plan is improved by planning for 900s versus 60s (120s in Transformer), and on these the cost reduction is much lower than that made by using the branched plan. Note that all instances where the branched plan did not improve on a linear plan coincided with problems where no branches were generated (the nodes in each plan, Nodes(O) and Nodes(B) are equal). Our results confirm that branched plans reduce cost even if more time is spent generating the linear plan. If we reduce $T_i$ to just 10 seconds cost increases by a reasonable amount on 4, 7 and 9 problems in Rovers, AUV and Transformer respectively. This suggests that it is useful to invest more than 10 seconds in generating a good initial plan.

Additionally, we investigated what happens if the branch planning time ($T_b$) is halved. There was a marginal increase in the cost (<5%) of the branched plan in only four problems given the same $T_i$ (60 or 120s). Problems solved during branching are smaller than the original problem, as some useful activity in moving towards the goals has already been performed, so less time is required to generate plans.

We conclude that increasing $T_i$ not only has a bigger impact on overall solving time, but also has less impact on cost than increasing $T_b$.

### Reasons for the Benefits of Branching

It is important to ascertain whether the cost decrease achieved by the branched planning is simply due to incidentally finding a better linear plan than

---

4 Data omitted from the table for space reasons.
the one forming the spine of the plan tree. To test this, for each plan tree, we enumerated all traversals from the root node, keeping only those that complete with $P \geq \theta$, i.e. are valid linear plans. In all cases, the quality of these plans was no better than that used as the spine of the plan tree, i.e. that found by the first call to the planner. As such, we can conclude that the benefit of the branching is not simply due to finding better linear plans due to increased total planning time.

**Scalability of Branched Planning vs MDPs** It is difficult to test this conclusively. Direct comparison to existing approaches is not possible as these either reason with uncertainty of a different nature, do not consider over-subscription planning, or are not available as runnable systems. However, we do note the results in [16] give indicative scalability for an MDP approach to solving similar problems. This planner was evaluated on Rovers problems only: the largest problems the planner scales to involve ‘1 rover, 11 locations, 20 paths and 6 goals’ and take ‘20,000 seconds’ to solve. Whilst we do not have exactly the same problems, nor the same hardware (they do not specify), an indicative comparison can be made to the size of the problems solved using our approach. The smallest problems we use are comparable to the largest solved by the MDP approach according to these parameters. As a point of comparison for time-taken, when we used $T_l=60s$ and $T_b=10s$, finding a branched plan for such problems takes just over 90 seconds. The largest problem had 19 locations, 60 paths and 19 goals; and a branched plan was generated in just 540 seconds. This is promising for the scalability of our approach, and is to be expected since we do not have the burden of computing complete, optimal, policies.

An additional consideration is the size of the resulting plan trees. In memory-limited situations it is not possible to store large policies, therefore the size of our branched plans, which are not complete policies, could be advantageous. We cannot compare to the MDP policies above as the raw data is not given. The ‘Nodes’ rows of Table 1 show that the number of nodes in the branched plan for each problem remains reasonable in relation to the size of a linear plan. Some of the branched plans generated for larger transformer instances are somewhat larger than others, because small swings in demand caused by uncertainty require changes in the number of control actions; here a small change can lead to a radically different plan. There is further scope for more sophisticated branch merging, to eliminate redundancy (reducing size); we leave this to future work.

**Branched Plans vs Online Replanning** Online re-planning can effectively build the best branches at execution time given knowledge of what has happened so far; it is therefore analogous to a branched plan with a branch for every necessary opportunity and should thus perform better. As we noted earlier, however, online replanning is often not possible, so we want to test how much of the benefit of online replanning we can get through building a branched plan, compared to the baseline of a linear plan. For this comparison we must use the failsafe execution semantics, as discussed earlier.

In the simulation, we generate a plan, limited to 60s of CPU time (120s in Transformer) and then simulate the execution of the first step. From the resulting state $S'$, we replan, limiting search to 10s (60s), as in our branched planning experiments. Then, we execute the first step of the plan from $S'$; and replan again; repeating until the planner returns an empty plan, or $C$ is violated (signalling failure) and report cost. As a point of efficiency, if the remainder of the plan from $S'$ is still acceptable (executes with confidence $\theta$), it serves as an incumbent solution for search, inducing a cost bound.

The average costs achieved by replanning are included in Table 1 as Cost(R), alongside costs FSCost(O), FSCost(B) of the closest analogue Failsafe approach, i.e. the same $T_l$ and $T_b$. Clearly, branched plans are at a disadvantage to replanning as the latter is equivalent to knowing the values for which to compute a branch at each point in the plan. However, it is pleasing to note that despite not knowing the actual execution-time state, the improvement between the original and branched plans is not considerably less than that between the original plan and replanning; meaning much of the benefit of replanning can be gained through the use of branched plans. Indeed taking the geometric mean across the ratios Cost(B)/Cost(O) and Cost(R)/Cost(O) shows that whilst branching reduces cost to 64% of Cost(O), replanning is not substantially better, reducing the cost to 58% of Cost(O).

We conclude with this excellent result: branched plans can achieve most of the benefit of online replanning, without the requirement for substantial on-board computational resources; and whilst also allowing prior scrutiny of the plan by operations personnel if desired.

**ACKNOWLEDGEMENTS**

The author would like to thank Sylvie Thiébaut, Derek Long and Andrew Coles for helpful discussions about this work. The work is funded by EPSRC Post-Doctoral Fellowship (EP/H029001/1).

**REFERENCES**


