

Planning with Respect to an Existing Schedule of Events

Andrew Coles, Maria Fox, Derek Long and Amanda Smith

Department of Computer and Information Sciences,
University of Strathclyde, Glasgow, G1 1XH, UK
email: `firstname.lastname@cis.strath.ac.uk`

Abstract

Decomposition has proved an effective strategy in planning, with one decomposition-based planner, SGPLAN, exhibiting strong performance in the last two IPCs. By decomposing planning problems into several loosely coupled subproblems, themselves planning problems, a planner can be used to solve each of the subproblems individually. The subplans can then be combined to form a solution to the original problem. When planning for subproblems, it is necessary to account for the interactions between the actions used to solve the current subproblem and the actions chosen to solve other subproblems. The approach taken in SGPLAN is inspiring, but some aspects of the decomposition process are not fully described in the literature. In particular, how subplans are merged to form a complete plan is not discussed anywhere in detail. This paper presents an approach to planning at this subproblem level, detailing how the choices made whilst solving one subproblem can be influenced by the conflicts with other subproblems, and introduces a novel technique employing *wait events* that can be included in subproblem solution plans to allow the context of the existing schedule to be directly considered.

1 Introduction

Problem decomposition has proved to be a successful approach to solving large planning problems. SGPLAN (Chen, Wah, & Hsu 2006) is a planner based on decomposition that has achieved excellent results in both of the last two International Planning Competitions (Hoffmann 2005; Gerevini *et al.* 2006). A key problem in decomposition is how to combine the solutions to subproblems. The difficulties that must be addressed include how to ensure that solutions to separate subproblems influence the solutions to other subproblems, how to interleave actions from separate subproblem solutions and, where appropriate, how to explore positive interactions to remove redundant activity.

SGPLAN clearly solves these problems, but the existing literature explaining how it achieves the efficient integration of subplans with the global schedule is very brief. In this paper we present a technique for planning to solve subproblems given an existing schedule. We started this work with the intention of rationally reconstructing this part of the decomposition-based planning approach of SGPLAN but we

have been forced to address some very challenging issues in the process, and we believe (and provide some empirical evidence to support the claim) that our approach to subplan integration offers advantages over that employed by SGPLAN.

We begin by setting the context and presenting the relevant aspects of SGPLAN. In particular we focus on the problem of how to solve a subproblem in the context of a global schedule, and how to integrate a subplan into a schedule of solutions to other subproblems. We raise a number of issues that are essential to resolve in achieving this task efficiently and that are unanswered (or only briefly answered) by the published literature on SGPLAN. We then describe what we believe to be a novel approach to addressing these issues. Because of the brevity of the literature it is apparent that the authors of SGPLAN did not consider subplan merging to be an issue of major interest. We therefore hypothesise that our approach will produce better quality plans and finish the paper by presenting results that a new planner based on our approach, TSGP (Transparent Sub-Goal Planner — the name is deliberately intended to reflect the debt owed to SGPLAN), indeed produces plans with shorter makespan than those of SGPLAN and is better able to exploit the concurrency available in many domains.

2 Background and Context

This section describes the broad framework in which the approaches that follow are implemented. For clarity, the scope of discussion in this paper is limited to STRIPS planning problems, with no numerical or temporal constraints.

Our approach is currently based on the decomposition strategy taken in SGPLAN 4, where problems are decomposed on the basis on their top-level goals, although the techniques we describe are applicable with other decomposition schemes. Consider a planning problem $\langle A, I, G \rangle$, where A is the set of actions, I is the set of literals in the initial state and G is the set of top-level goals — literals which must hold in a goal state. When decomposing around top level goals, a planning problem $\langle A, I, g_i \rangle$ is created for each literal g_i in the top-level goal set G . These subproblems can be solved using a conventional planner as a subsolver, and the solutions combined to form a *global schedule*. A global schedule is a partial-order plan (Nguyen & Kambhampati 2001), represented as a directed graph in which the nodes are the actions to solve each subproblem and the edges

represent ordering constraints between them. In the general case one cannot assume that the subproblems can be solved in isolation. Therefore, the subsolver must be modified to account for flaws introduced into the global schedule by action choices within a subproblem solution. SGPLAN uses a modified FF (Hoffmann & Nebel 2001) as a subsolver.

2.1 The plan–update–penalty cycle

FF is deterministic: in its standard configuration it will produce identical plans for a given subproblem each time it is invoked. Invoking FF to solve each subproblem and naïvely combining the solutions will lead to a global schedule containing flaws. In this context, the flaws represent negative interactions between pairs of actions. A flaw arises between a pair of actions A, A' if A occurs before A' , and if A deletes a precondition of A' not re-established by an intermediary action. To avoid FF simply returning the same solution to a subproblem each time it is invoked, leading to the same flaws being generated in interaction with solutions to other subproblems, it is necessary to penalise the states searched by FF to dissuade the selection of actions which introduce flaws into the global schedule. The approach taken by SGPLAN, and in this work, is based around penalising the flaws between a developing subplan and the global schedule, using a penalty that is modified on successive attempts to solve the same subproblem.

The planning cycle used in SGPLAN is described in the literature as follows (Chen, Wah, & Hsu 2006). Penalties are calculated for each state considered by the FF subsolver, to penalise actions choices that introduce flaws between the subproblem solution and the global schedule. The penalty applied to a state is based on a weighted sum of the costs of flaws between the subproblem solution leading to the state, the relaxed plan FF generates leading from the state to the goal and the global schedule. The weights applied are determined using Lagrange multipliers, with specific multipliers, $\lambda_{p,q}$, being maintained to penalise conflicts between subproblems p and q . The application of this multiplier to the cost of flaws associated with a state causes FF to select different paths through the search space depending on conflicts between actions selected and the global schedule.

FF, modified in this manner, can then be used within a plan–update–penalty cycle as follows:

1. Solve each subproblem in isolation using FF
2. Combine the solutions to form a global schedule
3. Update Lagrange multipliers according to flaws in the global schedule (terminating if there are no flaws)
4. Solve each subproblem in turn with modified FF, adding its actions to the global schedule when a subsolution is found and return to step 3

The Lagrange multipliers are dynamically updated at step 3 on the basis of the flaws present in the global schedule. When a solution has been found for all subproblems, each Lagrange multiplier $\lambda_{p,q}$ is incremented for each flaw present between subproblems p and q . Consequently, the modified FF will, when solving a subproblem, avoid making action choices that introduce flaws between that subproblem

and others with which it has already had negative interactions. Search terminates if, at any point, the number of flaws in the global schedule is zero.

3 Open Issues in Subplan Merging

In SGPLAN, the modified version of FF plans with respect to an existing schedule, with the penalties influencing its decisions. Planning with respect to an existing schedule requires more than planning to solve a single problem. The solutions to subproblems can interact both positively, by sharing actions, and negatively, where some actions delete preconditions required by other actions. The planning process for solving each subproblem must be informed of constraints posed by the solution to other subproblems, without being unduly complicated and losing the benefits of the decomposition approach. The literature on SGPLAN is vague about exactly how to plan for a subproblem given an existing global schedule, and how to merge the generated subproblem solution into this schedule. These are the key issues we address in this paper.

3.1 Identifying an initial state for subplan merging

Each invocation of the subsolver requires as input a complete planning problem: an initial state, a goal state, and the actions provided to solve the problem. The latter two are trivial in the case of the decomposition described: the goal state is the single goal of this subproblem and the actions are those of the original planning problem. When solving the first subproblem the initial state is simply that of the original problem. The first subplan then forms the first global schedule, but it is then unclear where to begin planning for subsequent subproblems. One possibility is always to plan from the initial state of the original problem, which can result in the new plan deleting preconditions required by actions already in the global schedule and thus introducing flaws.

Alternatively, it may be beneficial to start from a later state arising after executing some of the actions in the global schedule. Waiting for the execution of some actions might help by allowing their effects to be used in solving the current subproblem. Conversely, it may be impossible to solve the current subproblem after the execution of some of the actions in the global schedule. There are, therefore, several possible initial states. Indeed, assuming actions in the global schedule are totally ordered, there is a different possible initial state from which to begin solving the current subproblem corresponding to each of the states reached after the application of each action in turn. Our understanding of the published description (Chen, Wah, & Hsu 2006) is that SGPLAN considers planning from each of these initial states in turn, until one is found that improves the global schedule according to a certain evaluation criterion.

3.2 Combining subproblem solutions

When a subplan has been found, it must be merged into the global schedule. In the general case, one can assume that there will be interactions between the actions in the sub-

problem solution and the global schedule, and thus the new global schedule formed will be flawed.

The insertion of subproblem solution plans into the global schedule is further complicated by the independent generation of subplans. If two plans require the same action for completion then the subsolver, in solving the two problems independently, will select the action twice: once for each subproblem. It may well be possible (and desirable) to share one instance of this action between the subproblems. However, it is not straightforward to decide when common actions between subplans should be shared and when repeated instances of actions are needed.

In their paper (Chen, Wah, & Hsu 2006) the authors of SGPLAN compare their solution to this problem with plan merging (Yang 1997) and plan reuse. However, it is left unclear how actions that follow the proposed initial state might be interleaved with the actions selected to solve the subproblem being considered.

3.3 Planning with penalties

Conventional forward state-space search planning terminates when the planner has reached a state in which the heuristic value is equal to zero. In the case of planning for an entire problem with no reference to other problems this is sufficient as a termination criterion. In the case where penalties are added to the heuristic values of states, it is rare that a state will have a heuristic value of zero as the action choices to reach that state will almost certainly have given rise to conflicts between the plan to reach that state and the global schedule. It may be necessary to commit to action choices that introduce conflicts when solving a subproblem if it can only be solved in one, or a few, ways. The conflicts would then need to be resolved later in the cycle by modifying the action choices in another subproblem solution where there is greater choice in how to solve the subproblem.

In the case where no non-zero states are present, it is not clear how to terminate search. Of course, it is still possible to identify local goal states, because they have a zero non-penalised heuristic value. It may, however, be beneficial to continue searching beyond this point to attempt to reach a state with an overall heuristic value of zero (including penalties), or at least one with a lower penalty value than the best state found so far.

4 Planning with an Existing Schedule

Having described issues arising when applying decomposition to planning, we now introduce a paradigm for planning with respect to an existing schedule: a vital component of such planning systems. In general, one cannot assume that subplans found entirely in isolation can be interleaved *post hoc* to find a global solution plan. Thus, the subsolver must plan within the context of the existing subplans and attempt to plan in such a way as to minimise conflicts whilst still solving the subproblem itself.

We now propose an approach to planning to achieve a subproblem in the context of an existing schedule. We set out to imitate SGPLAN but, because of the questions surrounding how SGPLAN performs this essential step, we have devel-

oped what we believe to be an alternative approach. Our approach resolves the open issues we have identified above.

4.1 Planning at the subproblem level

Our paradigm for planning at the subproblem level is to follow a forward-chaining approach within a neighbourhood that is different from the one explored by the modified FF in SGPLAN. In conventional forward-chaining planning the nodes in the search space are propositional world states and the arcs between nodes correspond to the application of actions. An arc $S \rightarrow S'$ exists if there is an action A applicable in S and the effects of A on S lead to S' . Whilst a goal state has not been found, a node is selected, heuristically, to expand in the search landscape, generating *neighbours* — states that can be reached from it.

To reflect the context in which a subproblem is being solved, we make two modifications to the basic FF-style forward-chaining approach outlined. First, as in SGPLAN, the heuristic values of states are modified to reflect conflicts between the developing subplan and the existing solutions to other subproblems, which together form the global schedule. Second, the neighbourhood is extended to contain actions that might be reused from the existing schedule and some *ordering constraints* that might be added to try to avoid conflicts. These approaches provide indirect and direct interaction, respectively, between the developing subplan and the global schedule. The calculation of penalties is described later in this paper and the remainder of this section focuses on the extended neighbourhood.

We extend the neighbourhood of a state by the use of three sorts of events:

1. *New action* events, as in traditional planning with FF — an action selected from those applicable in the current state.
2. *Action reuse* events — an action selected from those applicable in the current state, which is already present in the a solution to another subproblem. This results in subproblem solutions sharing actions.
3. *Wait* events, where the solution to the current subproblem pauses in anticipation of the execution of some action, A , from a solution to another subproblem, irrespective of whether it is applicable in the current state. A is called the *target* of the wait. Unlike reusing an action, waiting for an action does not allow for its effects to be relied upon in solving the current subproblem, as the preconditions were not satisfied in the current state.

The first of these corresponds to the action choices made in conventional forward-chaining planning. The latter two imply constraints between the actions in the current subplan and the global schedule. The ordering constraints for reuse events are simply a side-effect of sharing a single action between two subproblems. The ordering constraints implied by wait events correspond to *promotion* and *demotion* in partial-order planning: events prior to the wait itself are promoted to occur before the action that is the target of the wait, and events after the wait are demoted to occur after it. By adding this capability into the subsolver, delays can be

inserted between actions in subplans in order to avoid conflicts, and such delays may be necessary to find a feasible global schedule in some planning problems. It is this opportunity to resolve conflicts with waits that distinguishes their exploitation from action reuse.

As a special case, the neighbourhood for the *initial* state is extended to contain states corresponding to the reuse of action *sequences* from the global schedule. We establish an arbitrary, fixed total-ordering on the actions in the schedule and then attempt to execute the actions in turn starting from the initial state. If an action is applicable (i.e. its preconditions are satisfied) a new state is reached, and added to the neighbourhood of the initial state in the subproblem. If it is not applicable (as a result of flaws in the global schedule), we ignore it. Thus, new successors are only created for applicable action sequences. The process terminates when all actions in the global schedule have been considered. The successors added to the initial state correspond to reusing action *sequences*, taken from the global schedule. The neighbourhood in other (non-initial) states differ in that only *single* actions are considered for reuse, reducing the branching factor. By considering reusing action sequences in the initial state we avoid having to invoke the subsolver from each possible initial state, as is the case on SGPLAN, and hence the increased branching factor is worthwhile.

4.2 Inserting a subplan into the global schedule

Having obtained a candidate subplan (an ordered list of new-action, reuse, and wait events) the solution needs to be merged into the global schedule. Representing the global schedule as a directed graph (nodes correspond to actions from each subproblem and edges correspond to ordering constraints between actions), we merge the subplan into the global schedule in two phases:

1. Nodes are added to the global schedule for each of the new action events in the subplan.
2. Ordering constraints are added between adjacent events within the subplan — in the case of reuse or wait events, these will cause ordering constraints between nodes added for this subplan and nodes in the original schedule.

These operations are very similar to the partial-plan modification steps of a classical partial-order planner (Penberthy & Weld 1992), or the local neighbourhood search operations available to LPG (Gerevini & Serina 2002).

Within this graph, a topological order traversal is then performed to construct a totally ordered plan. If the topological order traversal does not visit all nodes, then a cycle is present and the schedule is not valid. In this case, the subplan to be merged into the global schedule is deemed to be invalid. To preserve stability, if two concurrent actions *A* and *B* in a schedule were traversed in the order *A-B* before the new subplan was merged, they will be traversed in the same order after the merge.

5 Computing Penalties

Once a totally ordered plan has been constructed, it is possible to calculate the penalties between actions in the plan.

Algorithm 1: Conflict Calculation for a Candidate Plan

Data: *C* - a candidate plan, $\lambda_{i,j}$ for all pairs of subproblems
Result: Penalty cost for *C*

```

1 penalty  $\leftarrow$  0;
2 activedeletes  $\leftarrow$  [];
3 foreach action A in C do
4   | conflicts  $\leftarrow$   $\emptyset$ ;
5   | foreach precondition P of A do
6   |   | add activedeletes[P] to conflicts;
7   | end
8   | foreach node N in conflicts do
9   |   | increment penalty according to equation 1
10  |   | between N and schedule node for A;
11  | end
12  | foreach delete effect F of A do
13  |   | add node corresponding to A to
14  |   | activedeletes[F];
15  | end
16  | foreach add effect F of A do
17  |   | activedeletes[F]  $\leftarrow$   $\emptyset$ ;
18  | end
19 end
20 return penalty

```

Penalties arise when the attempted execution of the plan reaches a state where the next action cannot be applied as it makes use of a precondition invalidated by an effect of earlier action, giving rise to a conflict between two actions. In our implementation, as in SGPLAN, for each pair of subproblems (*p*, *q*) a Lagrange multiplier $\lambda_{p,q}$ is maintained; and a conflict between two actions, one from subproblem *p* and one from *q*, has cost $\lambda_{p,q}$. In our implementation, where action reuse allows a single action to be shared between several subproblems, the cost of a conflict between a pair of actions from subproblems $P_{1\dots m}$ and $Q_{1\dots n}$ is:

$$\sum_{i \in P} \sum_{\substack{i \neq j \\ j \in Q}} \lambda_{i,j} \quad (1)$$

The flaws are calculated using the algorithm outlined in Algorithm 1. By iterating through the plan and maintaining a set of the literals that are no longer available for use as a precondition, the conflicts between each action and prior actions can be established. With this information, the equation given in Equation 1 can be used for each active flaw to calculate the contribution it makes to the overall penalty based on the appropriate Lagrange multipliers.

In order to identify potential conflicts between actions and a final goal, a dummy action is added to the end of each subplan with the goal as a precondition, allowing Algorithm 1 to operate unchanged.

5.1 Using penalties in scheduling

Within the schedule, in addition to the constraints described in Section 4.2, we consider adding further ordering constraints if they reduce the overall sum of penalties. For

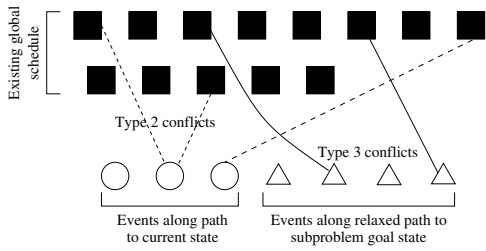


Figure 1: Computing Penalties for a Candidate Schedule including Relaxed Plan Events

each action, A , it is easy to find the bounds in the ordering that limit the placement of A . Within these bounds we use a greedy approach, iterating over the actions in the global schedule after which A could be ordered, calculating the penalties arising between A and the global schedule. The earliest ordering constraint resulting in the lowest penalty score is then added. This process is repeated for each of the actions in the subproblem solution. The result is a schedule which is non-optimal in terms of penalties but for which an attempt has been made to minimise them. For this reason we refer to this approach as a *min-flaw strategy*. Because this strategy is not guaranteed to find a good position for A we can choose to replace it with a simple ordering strategy in which A is placed in the earliest possible position in the schedule, regardless of the penalty score.

6 Planning with Penalties

Armed with a general approach to determining the penalties when planning for subproblems with respect to an existing schedule we now consider the planning itself, building a new planner, TSGP. At the subproblem level, we adopt an approach based on FF, using the Relaxed Planning Graph (RPG) heuristic to guide an enforced hill-climbing (EHC) search, restricted first to ‘helpful’ actions and then, where EHC fails, resorting to best-first search considering all actions, helpful or otherwise.

We now describe how the RPG heuristic is extended to account for interactions between a candidate subproblem solution and the global schedule, in terms of an updated heuristic value and a selection of nodes for which to consider waiting. We also explain how Lagrange multipliers associated with conflicts in the global schedule are updated, and what modifications we have made to the FF planning strategy.

6.1 Computing heuristic values for search nodes

In FF, heuristic information about each state in the search landscape is determined by finding a relaxed plan to the goal from that state; this relaxed plan is found by building a relaxed planning graph and extracting a plan from it¹. The relaxed plan is used to provide two sources of heuristic information. First, the length of the relaxed plan is taken as a heuristic value (a goal distance estimate). Second, the actions at the first time step of the relaxed plan are said to be *helpful* actions. In FF’s hill-climbing search, the neighbourhood of states is restricted to contain only helpful actions.

¹A non-optimal relaxed plan can be found in polynomial time.

In TSGP all the actions in the relaxed plan are used as a source of heuristic guidance and play an important part in search control. At any point in the forward-chaining search landscape we have the events *chosen* to lead from the initial state to the current state (in terms of actions, reused actions, and waits) and the relaxed plan to the goal. By translating the relaxed plan into a series of sequential new action events, and appending these events to the list of those chosen to reach the current state, we can build a candidate solution — albeit one with flaws, since the relaxed plan is probably not executable under non-relaxed semantics. Nonetheless, this flawed candidate solution can be merged into the global schedule using the techniques described earlier, and the penalties between the relaxed plan portion of the flawed candidate solution and the rest of the schedule can be used as a source of heuristic guidance.

Figure 1 illustrates the case where a flawed candidate solution has been merged with the global schedule. In the figure, square nodes represent existing actions in the global schedule; circular nodes represent the action events chosen to reach the current state in the search landscape; and triangular nodes represent the new action events added from the relaxed plan to build the flawed candidate solution. Using Algorithm 1, described in Section 5, we can find conflicts between the actions in the combined schedule. When using flaw detection and penalisation for heuristic guidance, the flaws can be partitioned into three sets:

1. conflicts arising between actions in the global schedule not involving the current subproblem;
2. conflicts between the actions chosen to reach the current state, and the actions in the global schedule (lines between square and circular nodes in Figure 1);
3. conflicts between the actions taken from the relaxed plan, and the actions in the global schedule (lines between square and triangular nodes).

We consider two ways of assigning costs to these flaws. In both cases, the first conflict category carries cost 0, and the last category carries a cost determined by Equation 1.

Our first approach assigns cost 0 to the flaws in the other category, in an attempt to address the zero-state problem discussed in Section 3.3. The effect of this is that the cost associated with a state is determined entirely by its relaxed plan, ignoring conflicts between the global schedule and the path to that state. Since goal states have empty relaxed plans, they have a penalty and heuristic cost of 0, allowing search to terminate. The trade-off, however, is that arbitrarily expensive actions can be chosen to reach goal states without carrying cost themselves.

Our second approach uses Equation 1 to assign costs to the second conflict category; in this case, a goal state (with RPG heuristic value 0) will only have a combined heuristic value of 0 if it can be scheduled without introducing flaws. Thus, the zero-state problem remains, but more penalty information is incorporated into the heuristic. In an effort to improve search control, we deduce which of the category two conflicts cause *fixed penalties*. If conflicts arise between the global schedule and the subproblem solution, when evaluating a state S , the only way to eliminate the conflict is to

apply more actions to S . The first point at which this can be done corresponds to point immediately after the actions chosen to reach S (i.e. immediately after the circle nodes), and as such any penalties arising prior to this point are fixed. Applying actions to S to reach successor states cannot eliminate these penalties, as the new actions would necessarily be scheduled after where the penalty occurred. Thus, the sum of the costs of the fixed penalties serves as a lower bound on the heuristic value of successors. This lower bound can be then be used to prune states during search: any state S' whose fixed penalty costs equal or exceed the heuristic value of the best state seen so far can be pruned, as neither it nor its successors can improve upon this best state.

6.2 Helpful waits

In Section 4.1, we discussed the addition of wait events to the neighbourhood of the forward-chaining planner. Using wait events, the subsolver in TSGP is capable of making some scheduling decisions: if a wait event is applied when searching for a subplan, then all events occurring after the wait in the subplan are necessarily scheduled after the event which has been waited for. When evaluating a state within the search landscape the relaxed plan to the goal is appended to the path to the state in question and used to form a (flawed) candidate schedule. Conflicts between the relaxed plan and the candidate schedule are identified, and a penalty cost added to the heuristic value of the state for each conflict. Conflicts are found between the relaxed plan and actions from other subproblem solutions when either an action, A_S , in another subplan scheduled after the current state, deletes a precondition of an action, A_R , in the relaxed plan *or* an action, B_R , in the relaxed plan deletes a precondition of a later action, B_S , taken from another subplan.

In the first case, a wait event would not be beneficial: waiting for the action A_S ensures only that the actions in the current plan added after this wait are necessarily scheduled after A_S . As wait events do not lead to states with different relaxed plans, the conflict between A_S and A_R would still be present. Hypothetically, one could add actions to support A_R in the subproblem solution, followed by the action A_R itself, and *then* a wait for A_S , thereby promoting A_R to occur before A_S . However, recalling that only conflicts between the relaxed plan and the global schedule are counted when evaluating a state, once the action A_R had been applied the conflict between A_R and A_S would be ignored.

In the second case, a wait event *could* be beneficial: by waiting for the action B_S the action B_R , which deletes one of the preconditions of B_S , is necessarily scheduled to occur after B_S in the global schedule. Thus, we define the notion of *helpful waits*: wait events for actions in the global schedule which conflict with actions in the relaxed plan due to a relaxed plan action deleting one or more of their preconditions. The restriction of the wait events in this manner to the helpful waits is similar to the restriction of action events to the helpful actions, pruning the list of wait events on the basis of the relaxed plan or, specifically, the apparent interactions between it and the global schedule. Helpful waits are found when evaluating states in the search landscape using a trivial modification to Algorithm 1 — when increment-

ing the penalties involving the preconditions of an action A (line 9), a wait event for the schedule node corresponding to A is added to the list of helpful waits if at least one member of *conflicts* corresponds to an action from the relaxed plan.

6.3 Updating multipliers

So far, we have discussed conflicts between actions from different subplans, and how Lagrange multipliers are used to weight the impact of these conflicts on the choices made in search. The cost of conflicts between two subplans is determined by multiplying the number of conflicts between them by the Lagrange multiplier stored for the corresponding pair of subproblems. That is, each conflict is penalised by the absolute value of the Lagrange multiplier.

Our approach to updating the subproblem–subproblem Lagrange multipliers in TSGP is the same as employed in SGPLAN. The overall process of finding a solution plan follows the plan–update–multipliers strategy described in Section 2.1. After the subsolver has been invoked for each subproblem (solving subproblems is a fixed, arbitrary order) the Lagrange multipliers are updated based on conflicts in the global schedule. For each conflict in the global schedule between solutions to subproblems p and q , the Lagrange multiplier $\lambda_{p,q}$ is incremented by 0.01. The initial value of the multipliers is 100.0, although in the first iteration of the plan–update–multipliers strategy conflicts between subproblem solutions and the global schedule are not considered.

As well as updating the Lagrange multipliers pertaining to subproblem costs, we also maintain a cost for each ground action in the planning problem. For each conflict in the global schedule between an action A and an action A' , the cost of A and A' are both increased by 0.01. Initially, the costs of all actions is assumed to be 1.0. Conceptually, the cost indicates how likely a given action is to be involved in a conflict with another and plays a useful role in influencing the construction of the relaxed plan, details for which are given below in Section 6.5.

6.4 Modifications to FF

When using enforced hill-climbing (EHC) to find solutions to planning problems, FF greedily selects the first state with a strictly better heuristic value than the current state. Actions are considered in a fixed arbitrary order. The first action considered for application from a given state will be always be used in preference to others if the state resulting from applying this action has a lower heuristic value than the parent state. The possibility of missing a successor with a better heuristic value is accepted in FF as a trade-off to increase the speed of plan generation. The disadvantages become more problematic when including penalties to alter the direction that the planner will take in order to reduce conflict with an existing schedule.

Taking the first strictly-better successor does not allow a great deal of sensitivity to the penalty values. Suppose a state S has two successor states, S_1 and S_2 with equal relaxed plan heuristic values but with differing penalty values. If S has a worse heuristic value than either of these two states, and the action leading to S_1 is ordered (arbitrarily) before that leading to S_2 , then S_1 will always be evaluated

before S_2 and chosen as the successor, regardless of their relative penalty values. This will remain the case until the penalty associated with S_1 eventually becomes so large that the state S_1 now has a higher heuristic value than the parent state. We use a steepest descent approach, rather than the conventional method of selecting the first achiever, to increase the sensitivity of the search to the penalty values.

6.5 Modifications to relaxed plan extraction

The relaxed plan has two important functions in: its length is used to compute the heuristic value of each state and the actions occurring in the first time step are used to compute *helpful actions* to prune the search space. When building a relaxed plan graph for heuristic purposes, the first achiever for each literal is maintained. Conventional relaxed plan extraction from a relaxed planning graph regresses from the goal state, selecting the first achiever found for each required goal literal at each layer. The use of first-achievers introduces no-ops from the earliest point at which a literal is established to the point at which it is needed. No preference is given to one action over another: only the first achiever for each literal is considered. When conflict penalties between actions are introduced into planning, however, the achiever chosen may be important in determining the penalties attached to the relaxed plan from a given state.

By incorporating information about potential conflicts between this subplan and others into the relaxed planning graph, we can see that some achievers are preferable to others. To this end, we use a greedy strategy favouring what appears to be a low-cost achiever in terms of potential conflicts, rather than the first achiever. The aim is to produce what is likely to be a more useful relaxed plan, more indicative of a reasonable path to the goal from the state being evaluated. Since the actions in the first step of the relaxed plan are used to generate the helpful actions the generation of helpful actions will thereby be influenced to favour selecting lower penalty achievers.

The costs attached to actions in the relaxed planning graph are determined when the multipliers associated with conflicts are determined prior to a cycle of solving each subproblem. In this phase, as explained in Section 5, an action’s cost (initially 1) is incremented by 0.01 for each action in the global schedule with which it is in a conflict. The costs, in this manner, do not take into consideration other actions that may need to be added to the relaxed plan in order to achieve the preconditions for a given action; they simply indicate the likelihood of a single action itself leading to a conflict. In order to generate the relaxed plan whilst preferring low-cost actions, we note the lowest cost achiever for each literal whilst building the relaxed planning graph. Initially, this will be the first achiever found for the literal, as in the conventional method, but each time a new action that adds a given literal is found, the noted achiever is reviewed: if the new achieving action cost is strictly lower than this and it can achieve the literal at the same fact layer as before, then it is noted as the new lowest cost achiever for that literal. Relaxed plan extraction then proceeds as before, using lowest-cost achievers rather than earliest achievers.

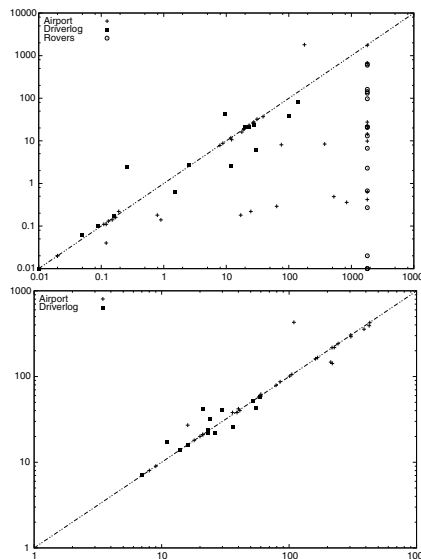


Figure 2: Comparison of planning times (above) and plan quality (below) with EHC and Steepest Descent strategies. Note that the cut-off time (1800 seconds) is used where plans are not produced.

7 Results

We performed two experiments, one based on ablation studies and one based on comparison with SGPLAN. The ablation studies explore the effects of using steepest descent search as an alternative to EHC, on both time to solve and plan quality, and the benefits of our min-flaw selection strategy. We used 60 problem instances from three domains and we present the results in scatterplot form (the diagonal line represents equal performance). The comparative experiment compares the plan quality obtained by TSGP and SGPLAN in the Airport, Rovers and DriverLog domains. The tests all use 30 minutes of CPU time and 1.5Gb of RAM on 3.4GHz Pentium 4s with 2Gb of physical memory.

Figure 2 shows that steepest descent search improved the time to solve problems but had no noticeable effect on plan quality. Figure 3 shows a similar result: min-flaw scheduling helped to shorten the time required to solve the problem but did not improve plan quality (indeed, plan quality deteriorates slightly with min-flaw).

Since we are proposing an implementation of a central part of the SGPLAN strategy, and arguing that our approach allows efficient scheduling of sub-plans with respect to the global solution, we believe it is important to demonstrate that our approach gives some advantages over that implemented in SGPLAN. TSGP has not been optimised for CPU time so we cannot expect to compete with SGPLAN in terms of speed. Indeed, although it can slightly outperform SGPLAN in terms of speed in the Airport domain, SGPLAN is almost always faster by approximately an order of magnitude. However, because of the way that we have addressed the scheduling issues described in this paper we expect to obtain greater concurrency in domains where non-trivial concurrency is available.

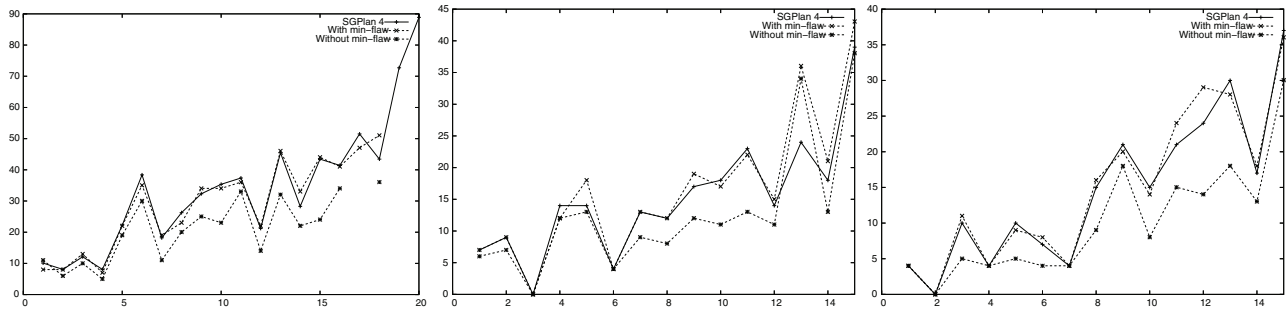


Figure 4: Example showing benefits of improved parallelism in plans compared with SGPLAN in Rovers domain (left), and DriverLog problems with 5 (middle) and 9 (right) trucks and drivers.

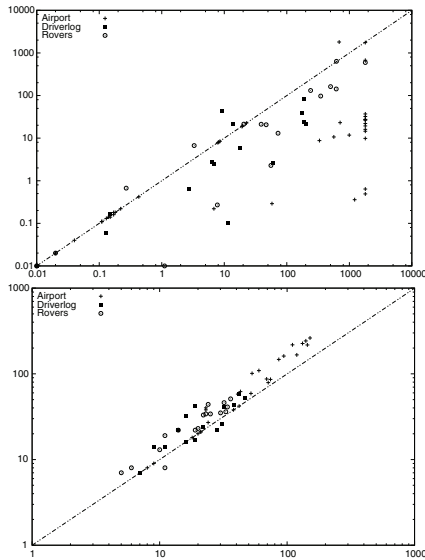


Figure 3: Comparison of planning times (above) and plan quality (below) with and without min-flaw strategy. Note that the cut-off time (1800 seconds) is used where plans are not produced.

Figure 4 (left) shows that TSGP achieves better plan quality than SGPLAN in the Rovers domain when min-flaw is disabled (and comparable quality when it is enabled). To further explore this phenomenon we ran tests using DriverLog problems split into two collections: one with 5 trucks and drivers and the other with 9 trucks and drivers. We would expect to see that the makespan of plans should tend to improve where more resources are available, particularly as problems get larger, so that the resources become significant. As can be seen, in Figure 4 (middle and right), TSGP does indeed show better makespan as the problem size increases. It can be seen that the improvements are most significant when min-flaw scheduling is not used. These results tend to confirm that the min-flaw strategy can improve time to plan, but at the cost of plan quality. This is a significant result, since it emphasises the trade-off that is present between attempting to find simple resolutions of conflicts between subplans and the overall quality of the final plan.

8 Conclusions

This paper describes an implementation of a core part of the decomposition and recombination strategy made prominent by SGPLAN. We focus on the problem of efficiently integrating a subplan with an existing global schedule. Although SGPLAN is now very well known, and widely used, this part of the decomposition-based planning process has only been briefly described yet requires ingenuity to do well. We have identified and addressed a number of issues left open in the published literature on SGPLAN and obtained a rational reconstruction of this fundamental part of the decomposition-based planning approach.

Acknowledgement

The authors are grateful to Benjamin Wah for many useful and detailed discussions on aspects of SGPLAN.

References

- Chen, Y.; Wah, B. W.; and Hsu, C. 2006. Temporal planning using subgoal partitioning and resolution in SGPlan. *Journal of Artificial Intelligence Research (JAIR)* 26:323–369.
- Gerevini, A., and Serina, I. 2002. LPG: A Planner based on Local Search for Planning Graphs. In *Proceedings of the 6th International Conference of Artificial Intelligence Planning and Scheduling (AIPS'02)*. Menlo Park, CA: AAAI Press.
- Gerevini, A.; Dimopoulos, Y.; Haslum, P.; and Saetti, A. 2006. Benchmark domains and problems of IPC-5. <http://zeus.ing.unibs.it/ipc-5/domains.html>.
- Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.
- Hoffmann, J. 2005. The deterministic part of IPC-4: An overview. *Journal of Artificial Intelligence Research* 24:519–579.
- Nguyen, X., and Kambhampati, S. 2001. Reviving partial order planning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, 459–466.
- Penberthy, J., and Weld, D. 1992. UCPOP: a sound, complete, partial-order planner for ADL. In *Proc. Int. Conf. On Principles of Knowledge Representation and Reasoning*, 103–114. Los Altos, CA: Kaufmann.
- Yang, Q. 1997. *Intelligent planning: a decomposition and abstraction based approach*. Springer-Verlag, London, UK.