4SP: A four-stage incremental planning approach[1]

Eva Onaindia, Laura Sebastia and **Eliseo Marzal**[2]

Abstract. GraphPlan-like and SATPLAN-like planners have shown to outperform classical planners for most of the classical planning domains. However, these two propositional approaches do not exhibit good results for large-size problems due to the graph size they have to handle.

In this paper we show a new approach for planning as an attempt to combine the advantanges of a graph-based analysis and a partial-order planner. The fast responses obtained in the experimental results show that the application of this technique can report significant benefits in terms of a reduction in search space and that the average performance of this planner is much better: it takes a bit longer to solve some easy problems but it is capable to easily solve large problems.

1 INTRODUCTION

Recently, a study to compare the performance and limits of six planners reports that, to date, no one planner has demonstrated clearly superior performance [6]. The conclusion of the study is that the best planner varies accross problems. The planners were tested on the UCPOP suite problems and on a particular software testing domain from the authors [6]. No planner solved all of the problems. Some planners as STAN [4] were faster in general and others solved a few more problems the others did not. As for the software testing domain, UCPOP [1] clearly dominated and solved many more problems than the others.

Moreover, we tested STAN [4] and Blackbox [5] on large problems from the blocks world domain and noticed that none of them were able to solve problems involving more than fifteen blocks.

Our motivation is to develop a new planning approach which also offers a good performance for large size problems. In order to tackle this issue, we present a planner which integrates a graph-based preprocessing technique that incrementally exploits the problem knowledge and a partial-order planner (POP).

Our new planning approach, 4SP, executes a four-stage algorithm:

- First stage: its task is to build up a graph that will contain the set of all of the possible actions which can be executed at each time point.
- Second stage: the result of this phase is a more refined graph which only contains a subset of actions that must necessarily occur in a correct solution.
- Third stage: the purpose of this stage is to guarantee a partial consistency between the actions in the graph. The graph obtained from this phase will even comprise, in some particular cases, all of the actions of a correct solution.
- Fourth stage: this stage is aimed at adding the missing actions in the plan and finding an ordering relation for all the actions in the plan (total consistency). The result of this phase will be the final solution plan.

The experimental results show that a proper integration of a graph-based preprocessing technique and a POP reports significant benefits in terms of a reduction in search space and it is also capable to solve large size problems. The layout of the paper is as follows: in sections 2 through 5 we will focus on each planning stage, section 6 shows the obtained results and section 7 draws conclusions from this work and summarizes some directions for future work.

2 THE FIRST STAGE

The first phase of the algorithm creates a graph inspired in a Graphplan-like expansion. This graph, named *problem graph*, may partially or totally encode the problem. The problem graph is a directed, layered graph with two kinds of nodes (literals and actions) and two kinds of edges (precondition-edges and add-edges). The levels alternate action levels containing action nodes and literal levels containing literal nodes.

- An action-level A_j consists of all of the action instantiations which are applicable in the previous literal-level L_{j-1} and are different from any other action instatiation in the graph. That is, A_j is composed of all of the action instantiations a_{jk} which satisfy these two requiments:
  - all preconditions of a_{jk} are present in the previous literal-level L_{j-1} and
  - a_{jk} does not occur in any previous action level
- A literal level L_j is a set of propositions implictly representing the different world states reachable after executing actions in A_j. More specifically, let $A_j = \{a_{j1}, a_{j2}, \ldots, a_{jn}\}$ be the set of action instantiations that can be executed at action level A_j; the set of literals in L_j is defined as $L_{j-1} \cup \mathsf{AddEff}(a)$[3] $\forall a \in A_j$, that is literals in the previous level L_{j-1} plus the add effects of each action in A_j.

The first level in the problem graph is the literal-level L_0 and it is formed by all literals in the initial situation. A_1 consists of all of the action instantiations which are applicable in L_0. L_1 is the set of literals in L_0 plus the add effects of each action in A_1 and so forth. The problem graph creation terminates when a literal level containing all of the literals from the goal situation is reached in the graph or when no more new actions can be applied. Notice that the delete

[1] This work has been partially funded by the Spanish Government CICYT-FEDER project 1FD7-0887

[2] Dept. Sistemas Informaticos y Computacion, Technical University of Valencia, 46071 Valencia, Spain, email: {onaindia, lstarin, emarzal}@dsic.upv.es

[3] AddEff, DelEff and Pre stand for the add effects, delete effects and preconditions of an action respectively.

effects of actions are ignored during the problem graph creation and therefore interactions between actions are not taken into account at this stage. This makes the first stage be a very fast polynomial time process.

It must also be noticed that a problem graph is neither a state-space graph nor a Planning Graph [2]. There are two main differences with respect to a Planning Graph:

a) levels in the problem graph do not stand for time steps but for instantiation steps which can entail more than one execution step. An action level A_j denotes that every action in A_j will be executed at a time step $t \geq j$ and at least one action from A_{j-1} must be executed firstly.
b) since delete effects of actions are ignored in the problem graph we do not have to deal with mutual exclusion relations among nodes at this stage. The next phase will be responsible for identifying the relation between two actions in the same level: mutually exclusive (they interfere with each other), complementary actions (one of the actions adds an effect the other needs) or independent actions (there is no explicit order between them).

To illustrate the process of the problem graph creation we will take the *hanoi* problem for three disks (big -B-, medium -M- and small -S-) and three pegs (P1, P2 and P3) as an example (Table 1). The first step is to build the literal level L_0 which comprises all of the literals in the initial situation (literals are numbered as they appear in the graph). Following, the action level A_1 is created by finding all possible applications of the operator Move ?disk ?place1 ?place2 over the literals in L_0, where ?place1 and ?place2 may indicate a disk or a peg. Once the first action level is created, the next literal-level, L_1, will include the set of literals in L_0 plus the new effects added by the two actions in A_1. In Table 1 literals 2 and 3 are in bold to indicate they also belong to the goal state. The column next to A_1 shows the preconditions required by each action and the add effects generated by the action (P stands for preconditions and E stands for add effects).

L_0		A_1		L_1	
B on P1	1	Move S M P2	P={3,4,5}, E={7,8}	B on P1	1
M on B	2	Move S M P3	P={3,4,6}, E={8,9}	**M on B**	2
S on M	3			**S on M**	3
clear S	4			clear S	4
clear P2	5			clear P2	5
clear P3	6			clear P3	6
				S on P2	7
				clear M	8
				S on P3	9

Table 1. Hanoi problem graph (1)

Next step is to generate the actions in A_2 by applying the operator Move ?disk ?place1 ?place2 over the literals in L_1. Only the instantiations which have not been inserted in the graph yet (that is, instantiations different from Move S M P2 and Move S M P3) are considered. In order to do this, we only take into account those instantiations which involve at least one of the new literals inserted at L_1 (7, 8 or 9), as the rest of instantiations already appear at the previous action-level A_1. The second level of actions and literals (A_2 and L_2) are shown in Table 2. The new literals are those numbered from 10 to 12.

A_2		L_2	
Move S P2 M	P={4,7,8}, E={3,5}	B on P1	1
Move M B P2	P={2,5,8}, E={10,11}	**M on B**	**2**
Move M B P3	P={2,6,8}, E={10,12}	**S on M**	**3**
Move S P3 M	P={4,8,9}, E={3,6}	clear S	4
Move S P2 P3	P={4,6,7}, E={5,9}	clear P2	5
Move S P3 P2	P={4,5,9}, E={6,7}	clear P3	6
		S on P2	7
		clear M	8
		S on P3	9
		clear B	10
		M on P2	11
		M on P3	12

Table 2. Hanoi problem graph (2)

Step 3 follows the same rules explained above to create A_3 and L_3. All of the literals in the goal situation finally appear at L_3 (Table 3) and consequently the process of the problem graph creation finishes.

A_3		L_3	
Move B P1 P2	P={1,5,10}, E={13,14}	B on P1	1
Move B P1 P3	P={1,6,10}, E={14,16}	**M on B**	**2**
Move M P2 B	P={8,10,11}, E={2,5}	**S on M**	**3**
Move M P3 B	P={8,10,12}, E={2,6}	clear S	4
Move M P2 P3	P={6,8,11}, E={5,12}	clear P2	5
Move M P3 P2	P={5,8,12}, E={6,11}	clear P3	6
Move S P2 B	P={4,7,10}, E={5,15}	S P2 B	7
Move S P3 B	P={4,9,10}, E={6,15}	clear M	8
Move S M B	P={3,4,10}, E={8,15}	S on P3	9
		clear B	10
		M on P2	11
		M on P3	12
		B on P2	13
		clear P1	14
		S on B	15
		B on P3	**16**

Table 3. Hanoi problem graph (3)

The problem graph may include all of the actions of a solution plan. For all of the tested domains (see section Experimental Results), except the *hanoi* problem, we obtained a problem graph which includes all necessary actions in a valid solution plan. However, this cannot be guaranteed because, as it was said above, the problem graph creation finishes at a level where all the literals from the goal situation are present, even though additional actions could be applied at this final level.

The advantage of the problem graph is that its size is much smaller than the Planning Graph and the cost of creating this graph is hardly appreciable even when dealing with large size problems.

3 THE SECOND STAGE

The goal of this phase is to extract the information concerning the planning problem from the problem graph. At this stage, a new graph, named *basic graph*, is obtained. The basic graph is created by selecting from the problem graph only those actions which must necessarily appear in a valid solution.

The basic graph is a directed, layered graph with only action nodes. The number of levels in the basic graph is the number of action levels in the problem graph plus two additional levels, an initial and a final action level. The former contains one action a_0 which has effects and no preconditions; the final level includes one action a_n with preconditions and no effects.

The process to create the basic graph starts with preconditions of a_n (goal literals). The objective is to find a set of actions in the problem graph having these goals as add effects. The found actions are inserted in the basic graph and their preconditions form a new set of subgoals which in turn are solved by following the same process. Once there are some actions in the basic graph, the new preconditions can be achieved with actions from the problem graph or basic graph. At the end of this phase the basic graph will be a subset of the actions in the problem graph which belong to a correct solution.

In order to find the correct action for each literal (subgoal), 4SP applies the following property:

Property 1 (literal consistency) *A literal p required by an action a_k ($p \in \mathsf{Pre}(a_k)$) is said to be consistent if these two requirements hold:*

- *there is a sequence of actions $a_i \rightarrow a_{i+1} \ldots a_{k-1} \rightarrow a_k$ such that $p \in \mathsf{AddEff}(a_i)$ and $p \notin \mathsf{DelEff}(a_j)\ \forall j \in [i+1, k-1]$.*
- *for each action a_i such that $p \in \mathsf{DelEff}(a_i)$ there is a sequence $a_i \rightarrow a_{i+1} \ldots a_{k-1} \rightarrow a_k$ with an action a_j, $j \in [i+1, k-1]$, such that $p \in \mathsf{AddEff}(a_j)$.*

In order to check the literal consistency it is necessary to propagate effects of an action a_i each time a causal link $a_i \rightarrow a_j$ is asserted. The propagated effects of an action a_j are computed by means of the following procedure:

1. $\mathsf{PDelEff}(a_0) = \mathsf{DelEff}(a_0)$
   $\mathsf{PAddEff}(a_0) = \mathsf{AddEff}(a_0)$
2. Let $p_0, p_1, \ldots, p_n$ be the paths in the graph that have a_j as destination node. Let $A = \{a_{0,j-1}, a_{1,j-1}, \ldots, a_{n,j-1}\}$ be the set of predecessor actions of a_j, each corresponding to a path.
   (a) $\mathsf{PAddEff}(a_j) = \{x \in \mathsf{PAddEff}(a_i) : a_i \in A / (\exists a_k \in A \wedge x \in \mathsf{PDelEff}(a_k)) \rightarrow a_k < a_i\}$ [4]

   $\mathsf{PDelEff}(a_j) = \{x \in \mathsf{PDelEff}(a_i) : a_i \in A / (\exists a_k \in A \wedge x \in \mathsf{PAddEff}(a_k)) \rightarrow a_k < a_i\}$

   (b) $\mathsf{PAddEff}(a_j) = \mathsf{PAddEff}(a_j) - \mathsf{DelEff}(a_j)) \cup \mathsf{AddEff}(a_j)$
   $\mathsf{PDelEff}(a_j) = \mathsf{PDelEff}(a_i) - \mathsf{AddEff}(a_j)) \cup \mathsf{DelEff}(a_j)$

[4] $a_k < a_i$ denotes action a_k is executed before action a_i

3.1 Algorithm for the basic graph

The aim of this second phase is to obtain a basic graph where the property of literal consistency is satisfied for each action precondition. In order to check whether a literal is consistent or not the delete effects of the producer action of a causal link must be propagated according to the procedure stated in section 3. Figure 1 shows some cases of inconsistent literals. In Figure 1.a, action a_1, which is selected to satisfy the precondition y of a_2, provokes a conflict as it deletes the precondition x of a_3. Figure 1.b shows a similar example where the action a_1 which is introduced in the basic graph to satify a precondition also deletes a literal of the same needer action.

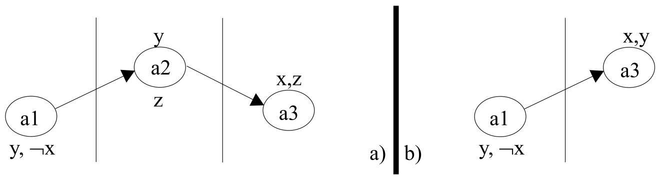

Figure 1. Some examples of inconsistent literals

The key point of the algorithm for the basic graph is to select the proper actions to satisfy the preconditions of the actions inserted in the basic graph. In order to select an action a_i from the problem graph to solve a precondition p of an action $a_j \in A_j$ in the basic graph, the algorithm proceeds as follows:

1. **Find a set of actions for** p. Find all actions at any level $A_i \leq A_j$ having p as an add effect.
   - If the actions found in the problem graph belong to different action levels we say p is an OPEN literal. In this case a set R containing all the different levels the literal belongs to is created. When a literal is OPEN it is not possible to make a decision about which action to choose and the literal remains as OPEN until more information is available.
   - If all actions found in the problem graph belong to the same action level we say p is a KNOWN precondition. In this case it is not possible yet to choose the proper action to satisfy p but at least the level of the producer action is known. All of the actions which produce p are gathered in a cluster and the common preconditions, add and delete effects of the cluster are identified. From this point, the cluster is treated as a single action (with its preconditions, add effects and delete effects) until one of the actions in the cluster is selected.
   - If there exists only one action to satisfy p then the action is clearly identified and it is inserted in the basic graph. In this case p is a CLOSED literal.

   From a set of literals to be solved, 4SP selects first CLOSED literals, then KNOWN literals and finally OPEN literals.
2. **Study the delete effects of the selected actions**. If p is a KNOWN or CLOSED literal then one of the actions in the cluster, or the selected action, must necessarily be used to solve the precondition p.

4SP analyzes the consequences of propagating the common delete effects of the actions in the cluster or the delete effects of the selected action respectively. The aim of the propagation is to find out whether any other literal in the basic graph becomes inconsistent after adding the causal link $a_i \rightarrow a_j$ for p ($a_i \in A_i$ will be the selected action if p is a CLOSED literal or one of the actions in the cluster if p is a KNOWN literal). Let's suppose that a precondition q of an action a_k in the basic graph ($a_k \in A_k, A_k \geq A_i$) becomes inconsistent after propagating the delete effects of a_i:

(a) If q is a CLOSED or KNOWN literal, an ordering between a_i and the producer action of q for a_k is added.

(b) If q is an OPEN literal, the new set of action levels for q is computed as $R = \{A_h\}/h \in [i+1, k]$, and so only actions in an action level of R are now considered as potential producers for q. In short, the range of action levels for a precondition q is restricted by discarding all levels lower and equal than the action level which deletes q. In both cases of Figure 1 literal x of action $a3$ becomes inconsistent due to the propagation of the delete effects of action $a1$. In figure 1-a), x could be achieved with actions in the same level as $a3$ or from any lower level, whereas in figure 1-b) x could only be achieved with actions from its own level.

3. **Select or limit the number of actions for p.**

(a) When p is an OPEN literal the producer action for p is not known, not even the action level for that action. As it was explained above, the number of action levels (R) of a literal can be restricted as long as new information is inserted in the basic graph. In this way, it may eventually happen that $|R| = 1$ and thus the literal becomes KNOWN or CLOSED. Otherwise, when all CLOSED and KNOWN literals have been studied (at this point one iteration of the algorithm is completed), the upper action level of all OPEN literals is removed. Actions from lower levels are preferred because the lower the level of the action is the less actions will have to be introduced to achieve its preconditions. The behaviour of the algorithm always follows a "principle of minimality" which is also applied at other points. This will be explained later on.

(b) If p is a KNOWN literal, and therefore there is a potential set of actions to achieve p, the algorithm discards those actions for which property 1 does not hold. That is, $\forall a_i/p \in \mathsf{AddEff}(a_i)$, if the causal link $a_i \rightarrow a_j$ violates property 1 for any other literal in the basic graph then a_i is removed from the set of actions. Sometimes the application of this property is not sufficient to discriminate among a set of actions and consequently additional criteria are to be applied.

The algorithm selects firstly actions in the basic graph than in the problem graph when achieving a precondition of an action. The reason is that it is preferable to use actions already existing in the basic graph than adding a new action (given two literals p and q to be satisfied, if an action $a1$ in the problem graph achieves p and an action $a2$ in the basic graph achieves both p and q then $a2$ will be selected). On the other hand, when there are several potential producer actions for a precondition, and all of them satisfy property 1, the algorithm selects the one which has less preconditions unresolved. The application of these two criteria indicate the process for creating the basic graph is oriented towards obtaining the minimal set of actions. In case none of the them allows to discriminate among the actions, any action will be valid.

3.2 Properties of the basic graph

After the second stage two different results can be obtained:

- **No basic graph exists**. This happens when it is not possible to find a sequence of actions to achieve a particular literal, and it is usually due to the lack of an operator to achieve such a literal. Let's take the example in Figure 2 where the action a_i deletes the precondition p of action a_k. In case there is no operator for achieving p, the only way to satisfy the literal is with the effects of the initial situation and therefore the precondition p of the action a_k will never be a consistent literal. Notice that the action a_k appears in the problem graph (and consequently in the basic graph) because at the literal-level L_j both p and q are present and q is a new literal achieved in the previous action-level A_j. If no basic graph is obtained the problem is unsolvable.

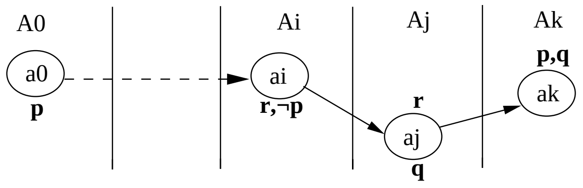

Figure 2. No basic graph

- **A basic graph is obtained**. Although the basic graph is created, this does not guarantee the problem is solvable. That is, the fulfilment of property 1 is not sufficient to discover unsolvable problems. However, if a solution exists for a problem, all of the actions in the basic graph will be part of such a solution.
  Basically, the basic graph comprises optimal sequences of actions to achieve each subgoal literal independently. Only a few interactions among actions are considered at this stage, as those due to the introduction of causal links. For this reason, in most cases it is not possible to establish a set of consistent ordering constraints among all of the actions in the basic graph.

The issues of completeness, soundness and termination on unsolvable problems are tackled in section 4.

3.3 An application example

In order to show the process for creating a basic graph we will take the example of the *hanoi* problem. The starting point is the problem graph shown in Tables 1, 2 and 3.

The algorithm begins with the literals in the final situation: 2, 3 and 16. Literals 2 and 3 are OPEN because literal 2 can be achieved with the initial action a_0, as it is one of the initial effects, and actions at level A_3; literal 3 is also produced by action a_0 and two actions at level A_2. The algorithm selects literal 16 (CLOSED) and action Move B P1 P3 is inserted in the basic graph at $A3$. The preconditions of the new actions are 1, 6 and 10. Literal 1 is CLOSED (level A_0), literal 6 is OPEN (levels A_0, A_2 and A_3) and literal 10 is KNOWN (two actions at level A_2). Since literal 1 is a CLOSED literal (as it only appears at A_0), a new causal link between a_0 and Move B P1 P3 is added in the basic graph (Figure 3).

Next step is to study literal 10. There are two choices at A_2 for literal 10 (Move M B P3 and Move M B P2). Both actions have a

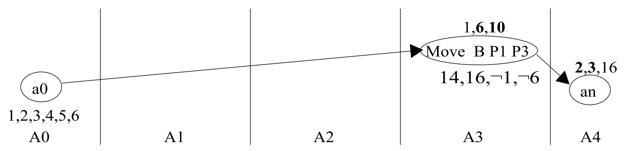

Figure 3. Basic graph 1 for the hanoi problem

common delete effect, literal 2, which makes precondition 2 of action a_n be an inconsistent literal (after applying the effects propagation).

At least one of these two actions must be chosen to solve literal 10; precondition 2 of a_n is an OPEN literal so the number of action levels for literal 2 is restricted to $\{A_3\}$. A cluster with both actions for literal 10 is created (Figure 4). We discard Move M B P3 because this action gives rise to a literal inconsistency as it deletes literal 6 which is a precondition for the action Move B P1 P3, whereas Move M B P2 does not cause any conflict.

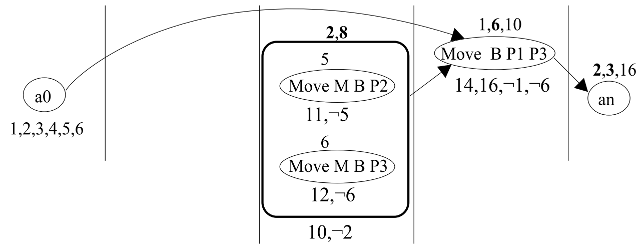

Figure 4. Basic graph 2 for the hanoi problem

After some more steps, the situation is as shown in Figure 5. At this point, we have no criteria to choose between the actions in the clusters.

1. At least one of the two actions in the cluster at A_1 must be chosen to generate literal 8. Both actions produce a literal inconsistency (Move S M P2 deletes precondition 5 of action Move M B P2 and action Move S M P3 deletes precondition 6 of action Move B P1 P3) and we cannot discriminate between them by any other criteria.
   (a) If we tried to solve the conflict generated by Move S M P2 then literal 5 would have to be achieved for action Move M B P2; there are two actions that have literal 5 as an add effect (Move S P2 M in the basic graph and Move S P2 P3 in the problem graph) but both generate a literal inconsistency (the former deletes literal 8 which is a precondition of Move M B P2 and the latter deletes literal 6 which is a precondition of Move B P1 P3).
   (b) If we tried to solve the conflict generated by Move S M P3 then literal 6 would have to be achieved for action Move B P1 P3; there are two actions having literal 6 as an add effect (Move S P3 M in the basic graph and Move S P3 P2 in the problem graph); as the former is in the basic graph and it does not cause any conflict we conclude that the correct choice to solve literal 8 is by selecting the action Move S M P3.
2. Once the action in the cluster at $A1$ is known we proceed with the cluster at $A2$. Since we have selected Move S M P3 at A_1, it is easy to see that Move S P3 M is the correct action at A_2 because all of its preconditions are already solved whereas Move S P2 M would need an additional action to achieve its precondition 7.
3. The same criteria can be applied to the cluster at A_3, thus resulting in the selection of Move M P2 B as all of its preconditions are already solved with actions in the basic graph.

At this point the upper action levels for literals 5 and 6 are removed. Literal 5 is then solved with action a_0 and literal 6 with the action selected at $A2$, Move S P3 M.

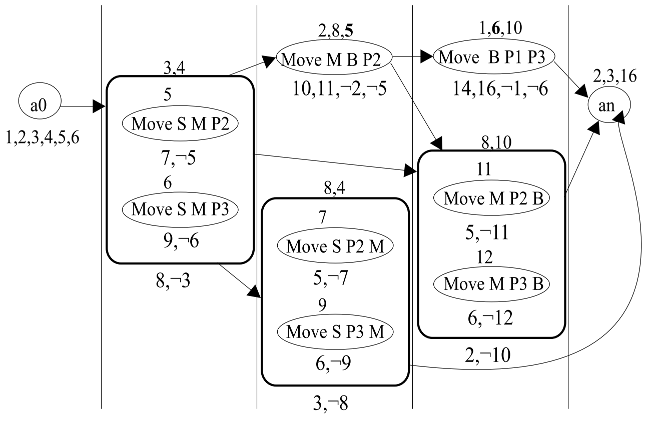

Figure 5. Basic graph 3 for the hanoi problem

The resulting basic graph (Figure 6) comprises five out of the seven necessary actions to solve the problem.

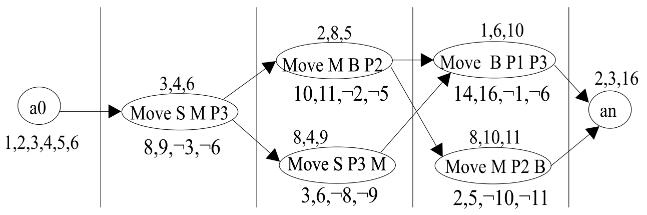

Figure 6. Final basic graph for the hanoi problem.

4 THE THIRD STAGE

Property 1 guarantess that, if a solution exists for the problem, all of the actions in the basic graph will belong to a correct solution. However, having obtained a basic graph is not a sufficient condition to ensure the problem is solvable. On the other hand, even though the problem is solvable, it may be impossible to set an order among all of the actions in the basic graph. In short, the set of actions in the basic graph constitute a first approximation towards a final plan but still some further refinements can be done on the graph.

The third stage is aimed at solving these issues and obtaining a more refined graph. The behaviour of the third stage is guided by the following property:

Definition 1 (partial consistency) *A basic graph is said to be partially consistent if it is possible to set a total-order relation between each pair of actions in the same action level of the basic graph.*

The application of this property will enable:

1. to discover unsolvable problems,
2. to get a more refined graph, closer to a final solution plan,
3. to provide a support towards obtaining an optimal solution (the shortest solution, i.e. the one with the less number of actions).

Let a_1 and a_2 be two actions of a same level action, $\mathsf{Pre}(a_1) = \{x_1, y\}$, $\mathsf{Eff}(a_1) = \{z_1, \neg y\}$, $\mathsf{Pre}(a_2) = \{x_2, y\}$, $\mathsf{Eff}(a_2) = \{z_2, \neg y\}$. Clearly, it is not possible to set a correct ordering constraint between both actions. There are two different justifications for this situation:

- If z_1 and z_2 (or just one of them) were OPEN literals at some time during the second stage and no action deleted these literals during the process, the algorithm will have assumed the lowest level as their producer literal level. In some cases this is not the correct option and the consequence is that the subset of actions in the basic graph do not represent an optimal solution. This type of conflict is named *effect conflict* and it is usually due to a lack of information during the second stage. The algorithm will choose then a different producer level for z_1, z_2 or both at this stage. In order to solve an effect conflict the planner carries out the following operations:
  a) eliminate the action that achieves the literal in the current solution
  b) take the next upper level as the producer level for the literal
  c) resolve the process as usual by selecting an action in the new level
- If the only possible way to satisfy z_1 and z_2 is by means of actions a_1 and a_2 respectively, then we say this is a *precondition conflict*. The name comes from the fact that the literals involved in the conflict are the preconditions of the actions (in the example, a_1 needs literal y and deletes y and likewise for a_2). In this case, the literal in conflict has to be achieved again by a new action (from the problem graph) or an existing action (from the basic graph). Notice that this is the same operation the planner carries out when solving an action precondition. The only additional checking is to discover the correct ordering for the new action a_3 ($a_1 \rightarrow a_3 \rightarrow a_2$ or $a_2 \rightarrow a_3 \rightarrow a_1$).

The third stage is mainly devoted to solve ordering conflicts among the actions at the same level in the basic graph. In order to do this, the algorithm performs operations like introducing new actions to solve conflicts or replacing one action by another one for achieving a particular literal.

The partial consistency property allows for detecting unsolvable problems. If an effect conflict or precondition conflict cannot be resolved by any means then the problem is unsolvable. Notice that all of the actions, either in the basic graph or problem graph, are considered when solving a conflict. Therefore, an irresoluble conflict leads to an unsolvable problem.

The issue of optimality is related to these two points:

- As it was said above, the algorithm always applies criteria so as to generate the minimal set of actions: selecting firstly actions in the basic graph over those in the problem graph and preferring actions which have the less number of preconditions unresolved.
- An effect conflict implies there is an alternative solution for achieving a literal. The algorithm tends to use actions at the lowest levels for those OPEN literals which are never deleted by the propagation of effects. This behaviour may yield a non-optimal solution because a non-correct action level may be chosen for an action. The third stage is aimed at solving this problem.

4.1 Example: Monkey test 1

Let's apply property 1 to the basic graph obtained for the *hanoi* problem. It is possible to set a total-order relation between the two actions at $A3$ by ordering $\mathsf{Move\ B\ P1\ P3}$ before $\mathsf{Move\ M\ P2\ B}$ (the latter deletes the precondition 10 of the former action). And for the two actions at $A2$, the consistent order is to put $\mathsf{Move\ M\ B\ P2}$ before $\mathsf{Move\ S\ P3\ M}$. Consequently, the basic graph from the *hanoi* problem satisfies property 1 and no further operations are needed for this problem at this stage. Obviously, there are two missing actions in this partial solution but they will be discovered by the POP at the fourth stage.

A_1		L_1		A_2	
GoTo P1 P2	P={1,2} E={6}	M1 onfloor	1	GoTo P2 P1	P={1,6} E={2}
GoTo P1 P3	P={1,2} E={7}	M1 at P1	2	GoTo P2 P3	P={1,6} E={7}
GoTo P1 P4	P={1,2} E={8}	Box at P2	3	GoTo P2 P4	P={1,6} E={8}
		Ban at P3	4	GoTo P3 P1	P={1,7} E={2}
		Knf at P4	5	GoTo P3 P2	P={1,7} E={6}
		M1 at P2	6	GoTo P3 P4	P={1,7} E={8}
		M1 at P3	7	GoTo P4 P1	P={1,8} E={2}
		M1 at P4	8	GoTo P4 P2	P={1,8} E={6}
				GoTo P4 P3	P={1,8} E={7}
				Climb P2	P={3,6} E={9}
				PBox P2 P1	P={1,3,6} E={2,10}
				PBox P2 P3	P={1,3,6} E={7,11}
				PBox P2 P4	P={1,3,6} E={8,12}
				GetKnf P4	P={5,8} E={13}

Table 4. Partial problem graph for the *monkey* problem

Num	Literal	Num	Literal
1	M1 onfloor	9	M1 onbox P2
2	M1 at P1	10	B1 at P1
3	Box1 at P2	11	B1 at P3
4	Bananas at P3	12	B1 at P4
5	Knife at P	13	M1 hasknife
6	M1 at P2	14	M1 onbox P4
7	M1 at P3	15	M1 onbox P3
8	M1 at P4	16	M1 onbox P4
		17	**M1 hasbananas**

Table 5. Literals in the problem graph from the monkey test1 problem

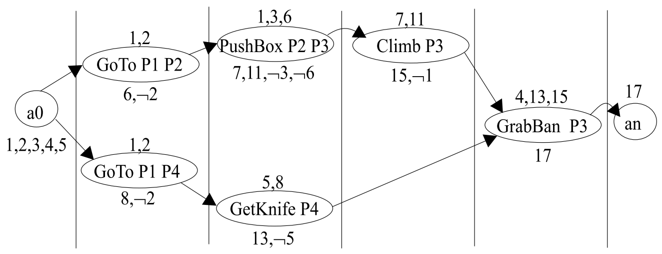

Figure 7. Basic graph for the monkey test 1 problem after stage 2.

In order to illustrate the behaviour of the third stage (in particular, the effect conflict) we will take the *monkey and bananas* problem as an example. The literals in the problem graph are shown in Table 5, two action levels from the problem graph (levels $A1$ and $A2$) are shown in Table 4 and the basic graph is represented in Figure 7.

The first aspect to point out is that the set of actions in the basic graph does not yield an optimal solution. Actions Goto P1 P2 and Goto P1 P4 would force the introduction of an additional action like Goto P4 P1 or Goto P3 P1. When applying property 1 we notice there is a conflict at $A1$ as both movement actions require and delete literal 2.

During the process of creating the basic graph, literals 6 and 8 were OPEN literals as they are both generated by actions $A1$ and $A2$ (Table 4). Since no action at $A1$ would delete a precondition 6 or 8 of actions at $A2$ in the basic graph, the algorithm selected actions from $A1$ as the producer actions for literals 6 and 8 (the lowest level), and the consequence is that the comprised solution in the basic graph is non-optimal.

At the third stage, the algorithm proceeds to solve the effect conflict between the two actions at $A1$. The planner attempts to take $A2$ as the new producer level for literal 8 and applies the procedure for selecting an action. Two of the choices at $A2$ provoke again an ordering conflict (Goto P2 P4 and PushBox P2 P4 require and delete literal 6 and so conflict with action PushBox P2 P3 which also needs and removes literal 6); another choice would be Goto P3 P4 which does not satisfy the requirement of minimality because its precondition 7 is unresolved.

Subsequently, the planner checks what happens when attempting to find another way of solving literal 6 (leaving literal 8 at $A1$). There are two possibilities, one does not accomplish the requirement of minimality (Goto P3 P2) and the other choice does not involve any conflict (Goto P4 P2). Then the planner chooses Goto P4 P2 to replace the action producing literal 6 at $A1$. The final and optimal solution is shown in Figure 8.

5 THE FOURTH STAGE

As we explained above, the goal of the fourth stage is to obtain the final plan. In principle, the third stage in 4SP could be omitted and to execute directly the fourth stage after the second one. However, the task of finding the correct ordering constraints in the plan is accomplished in two steps (firstly ensuring a partial consistency in the graph and then finding an ordering for all of the actions in the graph) because important benefits can be gained:

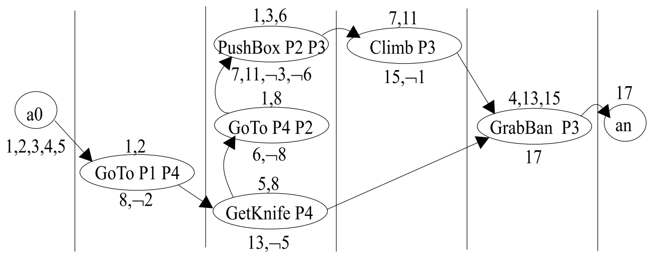

Figure 8. Final Basic graph for the monkey test 1 problem (after stage 3).

- First and foremost to delay the use of a POP until it is strictly necessary. POP are very good at solving threats among actions, which is the final step of our planning algorithm (finding a final ordering relation among all actions in the plan).
- By finding a total-order relation between each pair of actions in the same action level it would be possible to obtain the final solution plan without having to execute the fourth stage.
- The third stage is also used as a way to verify the basic graph entails a solvable problem and an optimal solution. This point is very important since the POP would be unable to discover the solution is working with is non-optimal or there is no solution for the problem.

Our partial-order planner [8] is based on the UCPOP planner [7] and therefore completeness in guaranteed when starting from an empty initial plan. The POP is given the plan obtained at the third stage as an initial input plan. When the POP input is an empty plan, a complete search space is generated and all choices to solve an OPEN precondition or a conflict are considered in the resolution process. However, when the input is not an empty plan, completeness is not guaranteed because this non-empty plan is just the result of one branching line of the search space which would have been generated by a complete search method. A way to recover completeness in the POP is by means of the White Knight concept [3].

Hard interactions among actions in different sequences of different levels are not taken into account when building the basic graph. Therefore, it might be impossible to establish an ordering relation for the set of all actions in the basic graph. This means that, when a precondition p of an action a_j is deleted by one action a_i, $A_i < A_j$, which belongs to another sequence of actions, p must be restored by a new action (application of the white knight technique). This situation gives rise to two different types of basic graphs. Let $\mathcal{A}$ be the set of actions in a basic graph and $\mathcal{S}$ be the set of actions that constitute a solution plan for a given problem:

- *complete basic graphs*, when $\mathcal{A} = \mathcal{S}$. In this case, a total order relation can be established among all actions in $\mathcal{A}$.
- *incomplete basic graphs*, when $\mathcal{A} \subset \mathcal{S}$. In this case, there are inconsistencies between actions of different sequences. New actions would have to be added to solve these interactions.

The difference between complete and incomplete basic graphs is specially important from the point of view of the fourth stage. In the case of complete basic graphs, the only remaining task is to sort the actions in the basic plan, whereas in the case of incomplete basic graphs, some actions will have to be added. In both cases, the remaining operations (ordering between actions and addition of new actions) are discovered by the existence of threats between the steps of the plan.

6 EXPERIMENTAL RESULTS

Due to a lack of time, the experiments shown in table 6 [5] correspond to a previous prototype of 4SP where the third stage is not implemented. Therefore, 4SP is not obtaining the optimal solution for problems such as monkey test 1, and it is not able to detect unsolvable problems.

Problems were taken from the UCPOP suite and Blackbox software distribution. All tests were run on a Sun Ultra 10 machine and results are given in seconds. We have run 4SP, BlackBox v3.6 [5] and STAN [4]. The results are classified into two groups: those for complete graphs and those for incomplete graphs (Table 6).

Problem	**Blackbox**	**STAN**	**Our method**	
Complete graphs	TT	TT	GT	TT
Sussman	0.02	0.028	0.005	0.006
Tw_rever4	0.03	0.03	0.011	0.021
Tw_rever5	0.06	0.03	0.023	0.04
Tower4	0.07	0.032	0.013	0.013
Tower5	0.21	0.07	0.024	0.025
Tower6	0.6	0.16	0.046	0.047
Tower9	111	10.53	0.225	0.225
T_largeA	0.82	0.53	0.079	0.08
T_largeB	4.34	2.63	0.289	0.3
T_largeC	—	82.143	1.792	1.8
T_largeD	—	—	4.508	4.52
Incomplete Graphs	TT	TT	GT	TT
Hanoi3d	0.11	0.039	0.019	0.176
Hanoi4d	1.41	0.061	0.035	1.81
Ferry	0.04	0.012	0.005	0.073
Monkeyt1	0.11	0.022	0.009	0.08
Monkeyt2	0.26	0.037	0.014	0.212

Table 6. Performance of Blackbox, STAN and our method on different problems

In most of the problems where 4SP was able to obtain a complete graph, the CPU time was reduced more than 50% compared to STAN and BlackBox. For example, in the blocks world domain, as the number of blocks increases, this difference is greater. This is specially remarkable in TowerLarge problems: 4SP was able to solve TowerLargeD problem that neither STAN nor BlackBox were able to.

For those problems with an incomplete graph, 4SP behaves slightly worse that STAN and BlackBox, although this difference is not as significant as in the case of complete graphs. As the average and standard desviation results show, 4SP behaviour is much more stable.

	Our method	**STAN**	**BlackBox**
Average	0.454	6.025	7.034
Standard Desviation	1.017	20.469	26.812

7 CONCLUSIONS AND FUTURE WORK

We have presented in this paper our four-stage planner 4SP. 4SP relies on the combination of an incremental preprocessing technique based on graph analysis and a POP. The basic graph is used to build a skeletal plan which is the POP's input. The most relevant aspect in 4SP is that the basic graph obtained with this graph-based technique already comprises the final solution plan for most of the tested domains.

Our objective was to develop a new planning approach by taking advantage of partial-order planning properties and reducing the inefficiency caused by the large search spaces generated by these planners. We have also shown that 4SP average outperforms other planning approaches as Graphplan or SATPLAN planners.

This is a first prototype of our planner 4SP. The obtained results confirm that the POP is still a bottleneck mainly for those problem which give rise to an incomplete graph. For this reason we suggest that the introduction of the third stage will significantly reduce the amount of work done by the fourth stage.

REFERENCES

[1] A. Barret, D. Christianson, M. Friedman, K. Golden, S. Penberthy, Y. Sun and D. Weld. *UCPOP v4.0 user's manual*, Technical Report TR 93-09-06d, Dept. of Computer Science and Engineering, University of Washington, Seattle, WA, (1996).

[2] A. Blum and M. Furst, *Fast planning through planning graph analysis*, Artificial Intelligence, **90(1-2)**, 281–300, (1997).

[3] D. Chapman, *Planning for Conjuntive goals*, Artificial Intelligence, **32-3**, 333–377, (1987).

[4] M. Fox and D. Long, *STAN and TIM public source code*, http://www.dur.ac.uk/CompSci/research/stanstuff/planpage.html, (1999).

[5] H. Kautz and B. Selman, *Blackbox Planner. Version 3.6*, http://http://www.research.att.com/ kautz/blackbox/, (1999).

[6] A. Howe, E. Dahlman, C. Hansen, M. Scheetz and A. von Mayrhauser, *Exploiting Competitive Planner Performance*, Proc. of the European Conference in Planning, 60–72, (1999).

[7] J.S. Penberthy and D.S. Weld, *UCPOP: A Sound, Complete, Partial Order Planner for ADL*, Proc. of the 1992 International Conference on Principles of Knowledge Representation and Reasoning, 103–114, (1992).

[8] L. Sebastia, E. Onaindia and E. Marzal, *Improving expressivity and efficiency in Partial-Order Causal Link Planners*, Proc. of the 18th Workshop of the UK Planning and Scheduling Special Interest Group, 124–136, (1999).

[5] GT stands for the time used in the graph creation and TT for the total time. We have used a blocksworld domain with 3 operators.