Parallel resolution of decomposed planning problems[⋆]

L. Sebastia, E. Onaindia, and E. Marzal

Dpto. Sistemas Informaticos y Computacion
Universidad Politecnica de Valencia
e-mail: {lstarin, onaindia, emarzal}@dsic.upv.es

Abstract. Planning is known to be a difficult task. One of the approaches used to reduce this difficulty is problem decomposition under divide-and-conquer methodology. This paper introduces a new decomposition technique for planning problems in STRIPS domains which is based on the idea of landmark, that is, literals that must be true in any solution plan. These landmarks are ordered and grouped in different sequential sets named intermediate goals (IG), being each IG a subgoal to solve. Our final objective is to build an independent planning sub-problem for each IG, using IG_i as goal state and an approximation to the state that would be reached on solving sub-problem $i-1$ as initial state. Once obtained all the sub-problems, a multiprocessor system will be able to solve them concurrently.

1 Introduction

Basically, AI planning problems can be reduced to search problems in a plan or state space, which can be very large. Most of the planning approaches are aimed at reducing this search space through new techniques such as graph analysis, local search, heuristic search, etc. Another mechanism is problem decomposition, that is, decomposing the planning problem into smaller sub-problems, solving them and combining the obtained solutions. Although problem decomposition has not been widely used in AI planning, it brings two advantages: (1) the obtained sub-problems may be easier to solve than the original problem and (2) a concurrent resolution (in a multiprocessor system) of these sub-problems may report important time savings.

In a traditional decomposition planning approach [9] based on a divide-and-conquer methodology, the goal set $\mathcal{G}$ of a planning problem is partitioned in disjoint subsets $\mathcal{G}_i$, such that $\mathcal{G} = \cup\mathcal{G}_i$. A plan is concurrently computed for each $\mathcal{G}_i$, and afterwards these plans are merged in order to obtain the final solution plan for the original problem. Combining these sub-plans may bring about conflicts between each other that are hard to solve.

We present a new decomposition technique for STRIPS domains to avoid these merging conflicts. This technique transforms the original problem into a set of ordered intermediate goals (IG) $\{IG_1, IG_2, \ldots, IG_n\}$, which represent common partial states that must be reached along the plan execution. A plan is computed for each sub-problem $(\mathcal{I}, IG_1), (SR_1, IG_2), \ldots, (SR_{n-1}, IG_n), (SR_n, \mathcal{G})$, where SR_i is the state reached

⋆ This work has been partially funded by the Spanish government CICYT projects TIC2002-04146-C05-04, TIC2001-4936 and DPI2001-2094-C03-03 and by the UPV project 20020681.

when solving IG_i. The process of gathering together the obtained sub-plans only entails a concatenation operation thus avoiding the conflicts that usually arise during the merging process.

One of the main drawbacks of this technique is that sub-problems have to be solved sequentially, as it is necessary to have the resulting state SR_i from solving IG_i in order to solve IG_{i+1}. To tackle with this difficulty, we have developed a process to approximate the state SR_i without having to solve the corresponding sub-problem. This approximate state to SR_i is called intermediate state (IS). Then, solving the whole problem $(\mathcal{I}, \mathcal{G})$ implies solving the list $(\mathcal{I}, IG_1)$, (IS_1, IG_2), $\ldots$, (IS_{n-1}, IG_n), $(IS_n, \mathcal{G})$, where IS_i denotes the computed intermediate state that acts as initial state for the corresponding sub-problem. These sub-problems are completely independent and, therefore, they can be solved in a multiprocessor (or distributed) system. Figure 1 summarizes this technique.

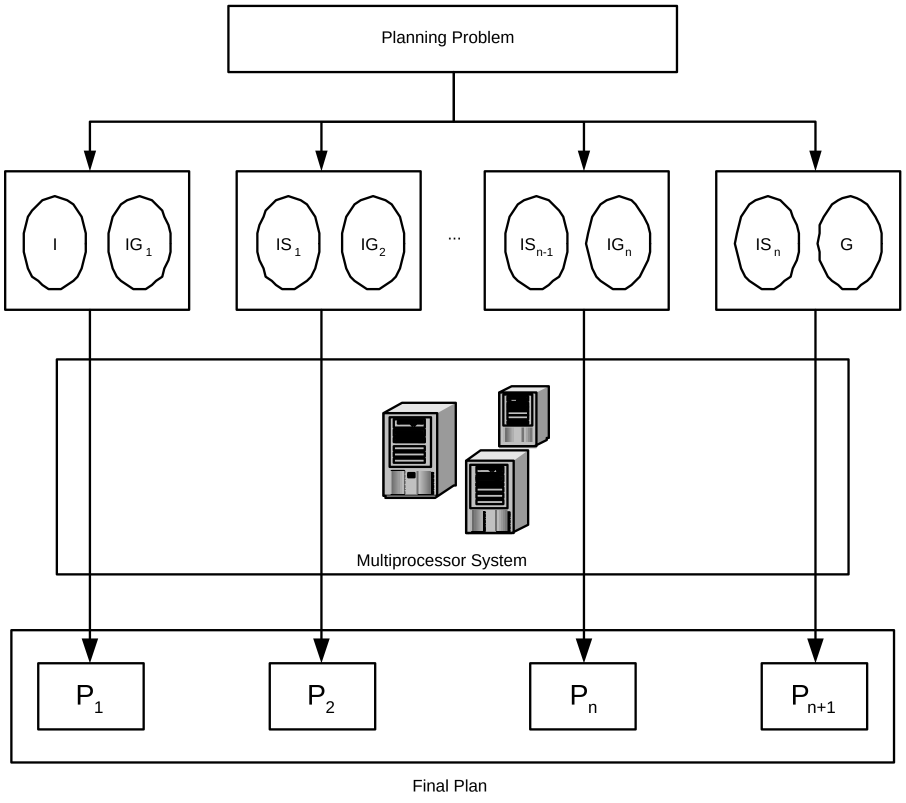

Fig. 1. Schema of our decomposition technique.

The main objective of this technique is to decompose a planning problem into a set of independent sub-problems, so that these sub-problems can be solved concurrently. This objective involves:

– the construction of the intermediate goals (IGs), which is explained in the following section;
– the computation of the intermediate states (ISs), introduced in section 3 and
– the resolution of the obtained sub-problems using different planners; the experimental results are shown in section 4.

Finally, we conclude by summarizing the decomposition technique introduced in this paper and pointing out further work to be done.

2 Decomposing the planning problem

The first step of our decomposition technique consists of computing a set of intermediate goals (IGs). In order to build them, we use the concept of landmarks graph (LG). A fact l is a **landmark** iff l is true at some point in all solution plans [8]. Landmarks are ordered according to three ordering relations [7]. First, we say there is a **necessary** order between l and l' if l must be true in the immediate preceeding state, that is, l is a prerequisite for achieving l'. A **reasonable** or an **obedient** order between l and l' states that it is reasonable to achieve l first, because otherwise l' would be achieved twice in the plan. The main difference between these two types of orders is that a reasonable order is based on necessary orders whereas an obedient order is based on both necessary and reasonable orders. A **LG** is a graph (N, E) where N is the set of the extracted landmarks and E represents the necessary, reasonable and obedient orders among landmarks. It is important to remark that both processes (landmarks extraction and orders computation) are incomplete, thus the information in the LG can also be incomplete.

For example, Figure 2 shows a problem defined over the depots domain [3]. In this domain, crates must be stacked onto pallets located in a particular distributor or depots by using the hoist in each location. Crates can be transported from one location to another by using trucks. In this problem, we can find these three landmarks (among others): $\{$clear(pallet2), lifting(hoist2 crate0), on(crate0 pallet2)$\}$. That is, these literals must be achieved in any solution plan. The necessary orders existing between these landmarks define a reasonable order:

$$\left.\begin{array}{l}\mathsf{clear(pallet2)} \leq_n \mathsf{on(crate0\ pallet2)}\\ \mathsf{lifting(hoist2\ crate0)} \leq_n \mathsf{on(crate0\ pallet2)}\end{array}\right\}\ \mathsf{clear(pallet2)} \leq_r \mathsf{lifting(hoist2\ crate0)}$$

This reasonable order indicates that it is better to clear pallet2 (literal clear(pallet2)) before hoist2 is grabbing crate0 to stack it on pallet2 (literal lifting(hoist2 crate0)): if we achieve first lifting(hoist2 crate0), we have to destroy it because we need hoist2 (which is the only hoist in location distributor1) to lift crate1 in order to clear pallet2.

An **intermediate goal (IG)** is a set of landmarks that must be true at the same point in a solution plan.

In order to ensure all the literals in an IG belong to the same state when executing a *good* plan, we define the following properties that an IG must accomplish:

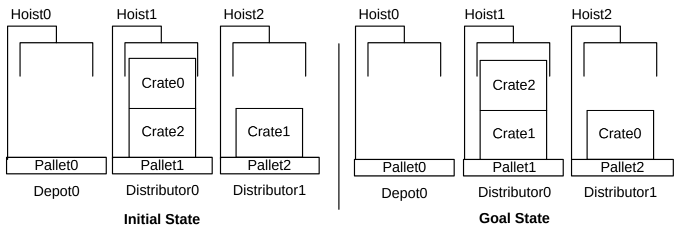

Fig. 2. Example on the depots domain.

– **Property 1:** An IG represents a goal state so all the literals in an IG must be consistent between each other.
– **Property 2:** A literal l belongs to an IG if and only if all its prerequisites have been visited before l (that is, they have been included in a previous IG).
– **Property 3:** A literal l should not belong to an IG if there exists an unvisited literal l' which has a reasonable or obedient order with l.
– **Property 4:** A literal l having a successor literal l' should be propagated to the following IGs until l' is included in an IG. This is to prevent l from being achieved twice as l might be deleted by an inconsistent literal.
– **Property 5:** Once a top level goal g has been included in an IG, g should belong to the following IGs for the same reason as Property 4.

The IGs are computed by going through the LG and grouping together those landmarks that fulfil the properties given above. The process of building IG_i works as follows:

1. First, an approximation to IG_i is computed with all the landmarks l that have a predecessor literal l' in IG_{i-1}. That is, $IG_i = \{l \in LG / \exists l' \in IG_{i-1} : l' \leq l\}$.
2. Then, we refine this first approximation in three stages:
   (a) Delay landmarks l that have a predecessor literal l' such that l' has not been included in a previous IG (properties 2 and 3).
   (b) Add to IG_i those literals in IG_{i-1} that accomplish properties 4 and 5.
   (c) Check inconsistencies between the literals in IG_i (property 1).

Figure 3 shows the first three IGs obtained for the problem in Figure 2. IG_1 indicates that crate0 and crate1 should be lifted at the same time. Then, IG_2 states that we must clear pallet1 before hoist1 grabs crate1 to stack it on pallet1, fact that is contained in IG_3, joint with lifting(hoist2 crate1). Once we have hoist1 holding crate1 and hoist2 holding crate0, these crates must be stacked onto pallet1 and pallet2, respectively. The corresponding literals are in IG_4 and IG_5 contains the remaining literal in the goal state (on(crate2 crate1)).

During the IGs construction, it is possible to reach a dead-end caused, among other reasons, by the existence of cycles in the LG. The way reasonable and obedient orders are extracted can produce cycles in the LG, which indicate that at least one literal in the cycle must be achieved minimum twice. This complicates the process of computing

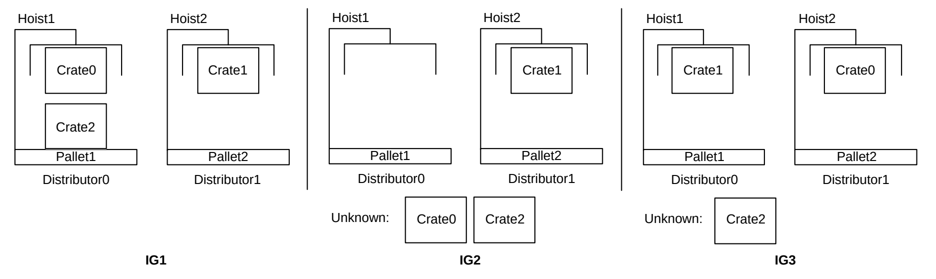

Fig. 3. IGs obtained for the example on the depots domain.

the IGs because if a literal l is in a cycle, it will always be delayed by step 2a in the algorithm above, so l would never be included in an IG. To avoid this deadlock, we relax property 3, that is, we select a literal from the cycle and ignore whether its predecessor literals have been visited or not.

3 Solving sub-problems concurrently

Once we have decomposed a planning problem into a set of ordered IGs, we build the corresponding sub-problems. Sub-problem i will have IG_i as goal state and the initial state should be the state reached (SR_{i-1}) when achieving IG_{i-1}. However, this state SR_{i-1} is unknown until the literals in IG_{i-1} are reached. Consequently, we cannot solve sub-problem i until sub-problem $i-1$ has been completely solved, that is, we cannot solve sub-problems in a multiprocessor system. The objective of this section is to tackle with this problem. Therefore, we will compute an approximation to each SR_i (that we call intermediate state -IS-) beforehand, so that we will be able to solve sub-problem i at the same time sub-problem $i-1$ and the rest of sub-problems are being solved. This computation is based on the concept of *property space*, which is explained in the following subsection. Subsection 3.2 introduces the algorithm to build the intermediate states. When sub-problems are solved concurrently there may be a difference between the inferred IS and the actually reached state. This difference causes that the corresponding sub-plans cannot be concatenated, so a repairing process (explained in subsection 3.3) is needed.

3.1 Property Spaces

Our technique to approximate the initial state of each sub-problem is based on the notion of *property space* offered by TIM [2]. A *property* is formed by combining a predicate and an index to one of its argument positions. This property is said to belong to values which can occupy the corresponding position in some valid literal in the domain. A *transition rule* specifies which properties an object gains or losses as a result of the application of an operator. Figure 4 shows the transition rules for the type hoist in the depots domain. In this example, an object of type hoist will always have property [at1], whereas it gains property [available1] and losses property [lifting1] when operators drop or load are applied. Other information extracted by TIM when performing the domain analysis is the concept of property space. A *property space* (PS) is a set of property sets that a particular type of objects in the domain can have in each state. That is, any object of that particular type must have a set of these properties in each state. The PSs are computed from the transition rules. For example, the PS for the type hoist is {[at1]}, {[lifting1], [available1]}, indicating a hoist will always be at one location (at1) and it will be lifting a crate (lifting1) or available (available1). For type crate, TIM finds the following PS: {[clear1, on1, at1], [on1, on2, at1], [lifting2], [in1]}. This means that a crate will only have one of these property sets (in square brackets).

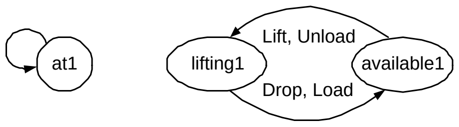

Fig. 4. Transition rules of type hoist.

3.2 Algorithm to compute the intermediate states

Given a sub-problem i (formed by IS_{i-1} as initial state and IG_i as goal state), the objective of this section is to build an approximation to the state SR_i we will reach after solving sub-problem i. This state IS_i will be the initial state for sub-problem $i+1$. The first approach is to consider IS_i as IG_i plus the literals in IS_{i-1} that are consistent with the literals in IG_i. However, this is not sufficient as there may be many more literals to be considered. For example, the state reached after achieving IG_3 in Figure 3 contains the following literals: {clear(pallet1), clear(pallet2), lifting(hoist1 crate1), lifting(hoist2 crate0), ...} and IG_4 is {on(crate1 pallet1), on(crate0 pallet2)}. If this IG is solved by any planner, the resulting state would also contain literals (available hoist1) and (available hoist2), but since these literals do not appear in IG_4, they cannot be included in IS_4.

Once this first approximation of IS_i is computed, IS_i is refined by using the notions of PS and transition rule. The objective is to complete the intermediate state by including the missing properties for each object in the problem, that is, to complete IS_i with the literals that are side-effects of the literals in IG_i. Firstly, we build a set of possible PS ($possiblePS(o)$) for an object o in two stages:

1. a set named $possiblePS(o)$ is formed with the PSs of the object o resulting from the applicable transition rules over the properties that o has in the state IS_{i-1}. This way we take into account the actual transitions that an object may suffer when applying an operator.
2. If no PSs are obtained in the previous step, then we initialize $possiblePS(o)$ with all the PSs for object o.

For example, if we consider the object crate2 in IG_2, the resulting $possiblePS$(crate2) after applying the corresponding transition rules will be: $\{$[at1, clear1, on1], [lifting2], [at1, on2, on1]$\}$. This means crate2 can have the same properties as it has in IS_1 ([at1, clear1, on1]) or it can be lifted by hoist1 ([lifting2]) or it can have another crate on top ([at1, on2, on1]). In other words, the only property set that it is not available for crate2 in IS_2 is to have crate2 in a truck ([in1]).

After calculating the set $possiblePS(o)$, we identify the properties that o owns ($addedProps(o)$) through the literals in IG_i and the properties o does not own ($delProps(o)$) by extracting the literals in IS_{i-1} that are inconsistent with literals in IG_i. The final set of properties for o is obtained by removing $delProps(o)$ from $possiblePS(o)$ and also those property sets that do not contain all the properties in $addedProps(o)$. In the example above, $addedProps$(crate2) is the empty set and $delProps$(crate2) is $\{$on1$\}$. Therefore, we will discard [at1, clear1, on1] and [at1, on2, on1] from $possiblePS(o)$ because these property sets contain the property $\{$on1$\}$.

Once we have the final set $possiblePS(o)$, we instantiate the properties in each PS using the literals in IS_{i-1} and IG_i and we select the PS with the minimum number of non-instantiated literals. The remaining non-instantiated arguments in a literal are instantiated taking into account the actions that add the instantiated literals. Therefore, non-instantiated literals are matched against the corresponding effects of these actions. In our example, the remaining PS [lifting2] is matched against the literal lifting(? crate2), so we do not know which hoist will be lifting the crate. If we study the actions that give rise to literal clear(pallet2) from IS_1, we can conclude that the correct instantiation is lifting(hoist1 crate2). Finally, the literals of the selected PS are put in IS_i.

3.3 Postprocess for repairing plans

Once all the ISs have been computed, the sub-problems can be solved concurrently. That is, we build sub-problem i consisting of IS_{i-1} as initial state and IG_i as goal state. These sub-problems are solved by a planner and then, the obtained solutions to these sub-problems are concatenated to build the final solution plan. As these sub-problems are independent, they can be solved at the same time in a multiprocessor system.

The states reached during the concurrent resolution (RS) may differ from the computed ISs, which may cause the impossibility to concatenate all the sub-plans. This difference between the RS and the IS is repaired by calculating the necessary actions to reach IS from RS. As the experiments will show, this repairing process can be time consuming, but it has only been applied in the depots problems.

As for correctness, our decomposition technique is correct since it will always return a valid plan as a result of the application of this repairing process. However, this technique is incomplete mainly due to the fact that the LG is incomplete.

4 Experiments

In this section, we show a comparison between the results obtained for the planning problems from the two last planning competitions ([1], [3]), with and without our decomposition technique. In order to solve these problems, we have selected three plan-

Domain		FF	LPG	VHPOP	Domain		FF	LPG	VHPOP
Blocks	# probs	28.1%	63.33%	700%	**Elevator**	# probs	0%	0%	188.46%
114 probls.	Length	2.89%	-1.56%	0%	150 probls.	Length	-0.026%	-4.33%	-4.85%
	Time	$>$100%	-73.4%	-92%		Time	$>$100%	$>$100%	-95.6%
Freecell	# probs	-4%	70%	(0/10)	**Logistics**	# probs	0%	0%	8.45%
60 probls.	Length	2.44%	-10%	-	77 probls.	Length	-1.77%	-7.94%	-1.41%
	Time	9.57%	-18.47%	-		Time	-16.38%	$>$100%	-61.96%
Depots	# probs	17.65%	0%	800%	**Driverlog**	# probs	0%	-5%	50%
20 probls.	Length	-9.87%	-23.28%	30%	20 probls.	Length	3.43%	-7.99%	0%
	Time	-80%	-30%	-80%		Time	$>$100%	-12%	-53%
Satellite	# probs	-10%	0%	18.18%	**Zeno**	# probs	0%	0%	11.11%
20 probls.	Length	0.84%	1.31%	-2.39%	20 probls.	Length	-0.16%	-7.45%	0.8%
	Time	$>$100%	$>$100%	$>$100%		Time	$>$100%	19.23%	-63.82%

Table 1. Comparison between resolution with and without our decomposition technique.

ners that took part in the last planning competition: FF[5], LPG[4] and VHPOP[10]. Table 1 summarizes the obtained results; for each domain, it shows:

- the number of tested problems;
- the percentage of the increment in the number of solved problems (first row in each domain) when using our decomposition technique;
- the percentage of the increment in the plan length (second row) when using our decomposition technique;
- the percentage of the increment in the execution time (third row) when using our decomposition technique and using as many processors as obtained sub-problems.

Time to solve a complete problem is computed as the time for decomposing it plus the maximum between (1) the time used to solve all the sub-problems sequentially divided by the number of processors and (2) the time used to solve the largest sub-problem. Each sub-problem is solved in a processor, so problems decomposed into n sub-problems need n processors as maximum.

The most outstanding result is that the three planners are able to solve more problems when they are executed with our decomposition technique. This is specially remarkable in VHPOP, which is able to solve 380 problems (out of 481) using our decomposition technique and only 161 when it is executed with the original problem. Therefore, we can affirm that our technique decomposes the original problem into easier sub-problems. On the other hand, though solving a decomposed problem usually implies a lack of quality in the solutions (due to the problem is not being considered as a whole), our results show that the order in which the landmarks are included in the IGs allows to preserve the quality of the solutions and to obtain shorter plans in many cases (negative numbers in Table 1).

With respect to execution time, we can see that the use of our decomposition technique and the concurrent resolution of the obtained sub-problems can report time savings in many cases, which are more remarkable with more time-consuming planners.

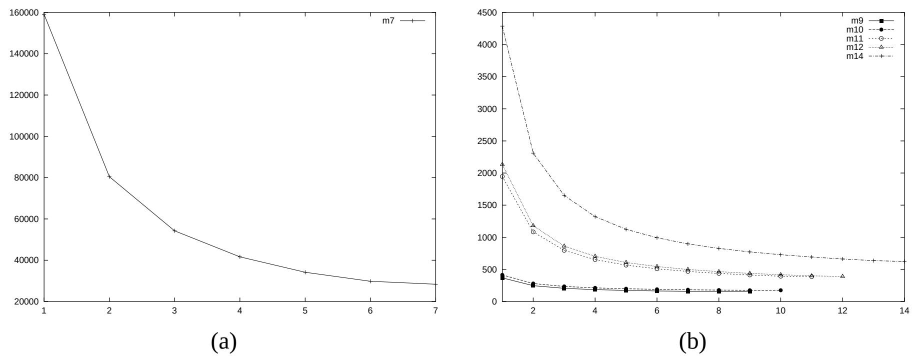

Fig. 5. Execution time of VHPOP in logistics problems when using up to 14 processors.

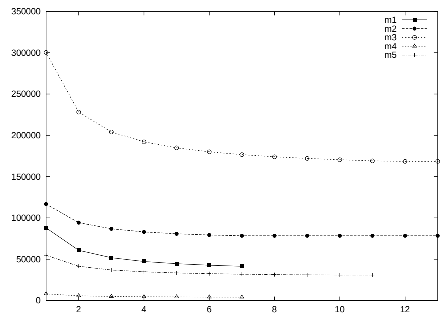

Fig. 6. Execution time of VHPOP in depots problems when using up to 13 processors.

Figure 5 shows the execution time (ms.) in average required by VHPOP to solve problems from the logistics domains when we use up to 14 processors. Problems are grouped according to the number of sub-problems in which they are decomposed. Figure 5(a) shows the problems decomposed into 7 sub-problems (so only 7 processors are needed) and Figure 5(b) shows the results for the problems decomposed into 9, 10, 11, 12 and 14 sub-problems. These figures show that we can obtain dramatic reductions in execution time when several sub-problems are solved concurrently.

Figure 6 shows the execution time (ms.) in average (grouped by instances of similar difficulty) required by VHPOP to solve problems from the depots domain, using up to 13 processors. In this case, we also get time reductions when using a concurrent resolution. However, these reductions are not as remarkable as in other domains, because the depots domain is the only one that requires a reparation process to obtain the final solution plan. This process can be time consuming but we still get important time savings, as Table 1 shows.

5 Conclusions and further work

In this paper we have introduced a new decomposition technique for planning problems in STRIPS domains. This technique is based on the idea of landmark. Landmarks are ordered and grouped into different ordered IGs, which can be solved either sequentially or concurrently. In both cases, the set of obtained sub-plans are concatenated to form the solution plan for the original problem. Due to the fact that each IG is dependent on the previous one, in order to solve sub-problems in a multiprocessor system, it is necessary to complete the initial state of each sub-problem, which is an inference of the state that would be reached after the resolution of the preceeding IG.

The experiments show, in first place, that the sub-problems obtained from this decomposition technique are easier to solve than the original problem since the three referenced planners can solve more problems. On the other hand, shorter plans are generated in many cases when planners are executed with our technique. Finally, the concurrent resolution of the obtained sub-problems can provide important time savings.

Our main efforts are now focused on reducing the computation time required by the repairing process (which is only needed in the depots domain) by (1) improving the repairing algorithm we are currently using and (2) improving the ISs generation so that we do not need a repairing process.

References

1. F. Bacchus. AIPS-2000 competition results. Technical report, University of Toronto, 2000. http://www.cs.toronto.edu/aips2000/.
2. M. Fox and D. Long. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, 9:367–421, 1998.
3. M. Fox and D. Long. Domains and results of the third international planning competition, 2002. http://www.dur.ac.uk/d.p.long/competition.html.
4. A. Gerevini and I. Serina. Lpg: a planner based on local search for planning graphs. In *AIPS'02*. AAAI Press, 2002.
5. J. Hoffmann. Extending ff to numerical state variables. In *ECAI'02*, pages 571–575. IOS Press, Amsterdam, 2002.
6. J. Hoffmann. Local search topology in planning benchmarks: A theoretical analysis. In *AIPS'02*, 2002.
7. J. Hoffmann, J. Porteous, and L. Sebastia. New theory about orders (technical report), 2002.
8. J. Porteous, L. Sebastia, and J. Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In *Recent Advances in AI Planning. ECP'01*. Springer Verlag, 2001.
9. Q. Yang. *Intelligent Planning. A Descomposition and Abstraction Based Approach*. Springer-Verlag. Berlin, Heidelberg, 1997.
10. H. Younes and R. Simmons. On the role of ground actions in refinement planning. In *Procs of the 6th Int. Conf. on AI Planning and Scheduling (AIPS'02)*. AAAI Press, 2002.