SUBGOAL ORDERING AND GRANULARITY CONTROL FOR INCREMENTAL PLANNING*

CHIH-WEI HSU

Department of Computer Science
and the Coordinated Science Laboratory
University of Illinois, Urbana-Champaign
Urbana, Illinois 61801
United States of America

chsu@manip.crhc.uiuc.edu

YIXIN CHEN

Department of Computer Science and Engineering
Washington University
Saint Louis, Missouri 63130
United States of America

chen@cse.wustl.edu

BENJAMIN W. WAH

Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois, Urbana-Champaign
Urbana, Illinois 61801
United States of America
wah@uiuc.edu
http://manip.crhc.uiuc.edu

Received (19 March 2006)
Revised (18 May 2006)
Accepted (1 July 2006)

In this paper, we study strategies in incremental planning for ordering and grouping subproblems partitioned by the subgoals of a planning problem. To generate a rich set of partial orders for ordering subproblems, we propose an algorithm based on a relaxed plan that ignores the delete lists. The new algorithm considers both the initial and the goal states and can effectively order subgoals in such a way that greatly reduces the number of invalidations during incremental planning. We have also considered trade-offs between the granularity of the subgoal sets and the complexity of solving the overall planning problem. We propose an efficient strategy for dynamically adjusting the grain size in partitioning in order to minimize the total complexity. We further evaluate a redundant-ordering scheme that uses two different subgoal orders to improve the solution quality, without greatly sacrificing run-time efficiency. Experimental results on using Metric-FF, YAHSP, and LPG-TD-speed as the embedded planners in incremental planning show

*Research supported by National Science Foundation Grant IIS 03-12084.

that our strategies are general for improving the time and quality of these planners across various benchmarks. Finally, we compare the performance of the three planners, the incremental versions using these planners as embedded planners, and SGPlan$_{4.1}$.

Keywords: Basic planner, incremental planning, partitioning, subgoal ordering, subgoal grouping.

1. Introduction

In this paper, we study new strategies in incremental planning for solving planning problems represented in STRIPS. *Incremental planning*[13] solves a planning problem in a multi-step fashion by achieving in each step (or stage) all the facts (subgoal facts of the final goal or some other facts) considered in this and the previous steps. As is illustrated in Figure 1, the planner tries to satisfy all the goal facts up to the current step in each step of the process. The framework studied is for the STRIPS domains but can be extended to domains with durative actions.

In a STRIPS domain, a planning problem $\mathcal{P} = (\mathcal{F}, \mathcal{O}, \mathcal{I}, \mathcal{G})$ is a tuple with four components, where $\mathcal{F}$ is a finite set of all the facts, and $\mathcal{O}$ is a finite set of all the actions. An action $o \in \mathcal{O}$ has three attributes: $pre(o)$, a set of facts that defines the preconditions of o; $add(o)$, a set of facts that defines the add-effects of o; and $del(o)$ a set of facts that defines the delete-effects of o. *State* $S = \{f_1, \cdots, f_{n_S}\}$ is a subset of facts in $\mathcal{F}$ that are true; $\mathcal{I}$ is the set of facts in the initial state; and $\mathcal{G}$ is a set of subgoal facts to be made true in the goal state.

The resulting state of applying a sequence of actions to S is defined recursively as follows:

$$Result(S, (o_1, \cdots, o_n)) = Result(Result(S, (o_1, \cdots, o_{n-1})), o_n),$$

where $Result(S, o)$ is $(S \cup add(o)) \backslash del(o)$ if $pre(o) \in S$ and S otherwise.

The planning task of $\mathcal{P}$ is to find a sequence of actions $(o_1, \cdots, o_n)$ that transforms the state from $\mathcal{I}$ to a goal state S_g where all the facts in $\mathcal{G}$ are true:

$$\mathcal{G} \in S_g = Result(\mathcal{I}, (o_1, \cdots, o_n)).$$

Incremental planning entails the decomposition of $\mathcal{G}$ into N disjoint subgoal sets, $\mathcal{G}_1, \cdots, \mathcal{G}_N$, and the solution of the sequence of N subproblems $\mathcal{P}_1, \cdots, \mathcal{P}_N$, where

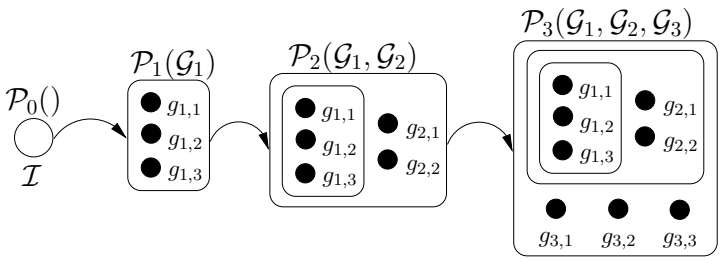


Fig. 1. An illustration of incremental planning that decomposes $\mathcal{P}$ with subgoal sets $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$ into subproblems $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$. Each dot is a subgoal fact in $\mathcal{G}$.

$\mathcal{G} = \bigcup_{i=1}^{n} \mathcal{G}_i$. Here, $\mathcal{P}_i$ aims to generate a plan $(o_{i,1}, \cdots, o_{i,n_i})$ from State S_{i-1} of achieving $\mathcal{G}_1, \ldots, \mathcal{G}_{i-1}$ to State S_i of achieving $\mathcal{G}_1, \ldots, \mathcal{G}_i$.

Incremental planning has been studied for many years in AI. It has been applied in planning under uncertainties in which a planner reacts to new uncertain events by planning incrementally,[5] and in dynamic domains in which a planner outputs valid prefixes of a final plan before it finishes planning.[12] Recently, it has been used to decompose large planning tasks into subproblems in such a way that a subset of them can be solved more efficiently than the original problem.[13] It is different from subgoal partitioning in SGPlan$_{4.1}$[3] in which each subproblem representing a subgoal is solved individually, and inconsistent global constraints across the subgoals are resolved at the end. Since global constraints only exist as biases in each subproblem and may not be satisfied after solving the subproblems, the subproblems will have to be re-evaluated. In contrast, incremental planning will not incur violated global constraints because all previous subgoals have to be satisfied in solving a subproblem. However, backtracking to a different order of subgoal evaluations may be needed when a feasible plan to a subproblem cannot be found.

Although some intractable planning problems can be solved efficiently by incremental planning, the approach does not always work well in a naive mode that randomly orders the subgoals. To improve the effectiveness of the technique, we study in this paper two general approaches.

a) *Subgoal ordering.* The ordering of subgoals may have great impact on both run time and solution quality. In the ideal case, solving $\mathcal{P}_i$ would only require finding actions from S_{i-1} to achieve $\mathcal{G}_i$, without invalidating those facts found previously for $\mathcal{G}_1, \ldots, \mathcal{G}_{i-1}$. For example, it would be best to order $\mathcal{G}_1$ and $\mathcal{G}_2$ in Figure 1 in such a way that, when solving $\mathcal{P}_2(\mathcal{G}_1, \mathcal{G}_2)$, the subgoal facts found by solving $\mathcal{P}_1(\mathcal{G}_1)$ do not have to be invalidated. In that case, the extra actions and search time for re-achieving $\mathcal{P}_1(\mathcal{G}_1)$ will be saved.

In practice, since the ideal order is unknown until the problem is actually solved, heuristic approaches for ordering subgoals have been developed. A well-known approach for ordering subgoals is reasonable ordering.[13] Although it tries to avoid some unnecessary invalidations of previous subgoals, it does not work well on many of the 4^{th} International Planning Competition[6] (IPC4) benchmarks because it deduces the partial orders of subgoals by considering the goal state alone. Without taking the initial state into consideration, it can only generate partial orders that are invariant to any initial state.

b) *Subgoal grouping.* Another aspect that may impact run time is the grouping of subgoals in incremental planning. Current planning approaches fall into two extremes. Many planners simply group all subgoals into a single problem and resolve them simultaneously. In contrast, traditional incremental planning schemes add only one subgoal from $\mathcal{G}$ in each step. Although both are natural choices, they do not always lead to the the best run time and plan quality.

Based on subgoal ordering and granularity control, we present in this paper a

general incremental planning framework for enhancing the performance of existing STRIPS planners. In the next section, we propose a new ordering algorithm based on a relaxed plan built from the initial state to the goal state. The new ordering relations consider both the initial and the goal states and can effectively order subgoals in such a way that greatly reduces the number of invalidations during incremental planning. In Section 3, we study a general strategy that groups subgoals together in each step. Based on trade-offs between the size of subgoal groups and the complexity of each subproblem, we show an optimal group size that minimizes the time for solving the overall problem. We propose an efficient strategy for dynamically adjusting the size of subgoal groups in order to operate in this optimal region. In Section 4, we present the results of an incremental version using the above techniques. To improve plan quality without sacrificing run time, we introduce a strategy that evaluates two subgoal orders and that selects the best plan. Our extensive experimental results show that incremental planning is a general approach that can improve both the run time and plan quality of target planners.

2. Subgoal Ordering

The order of resolving subgoals can have significant effects on the performance of incremental planning. An ideal order is one in which each subgoal does not invalidate any previous subgoals already achieved. That is, if fact f in $\mathcal{P}_i$ has been achieved (*i.e.*, $f \in \mathcal{G}_i$), then f should stay true in all subsequent states.

If each subgoal does not invalidate any previous subgoal during planning, then incremental planning effectively decomposes a planning task into a sequence of subproblems, each resolving a small number of additional subgoals. On the other hand, if many subgoals are invalidated after being made true, then incremental planning becomes less useful, because the previous efforts to achieve certain subgoals will be wasted when their subgoals are invalidated.

2.1. *Previous Reasonable-Ordering Algorithm*

The goal of the algorithm is to detect partial orders between some pairs of subgoals g_i and g_j and to determine if a plan must invalidate g_i before reaching g_j. If any plan that reaches g_j must invalidate g_i first, then g_i should be ordered *after* g_j because it will be invalidated anyway if it is resolved before g_j. A heuristic procedure in FF[13] to incompletely detect some of the reasonable orders consists of two steps:

a) For each subgoal fact g, it generates a FALSE set $F(g)$ that includes some facts to be invalidated before reaching g. This is found by enumerating all actions that support g as an add effect and by finding the common delete effects of all supporting actions.

b) For each pair g_i and g_j, it checks all supporting actions for g_j. If any of these actions either deletes g_i or requires some facts in $F(g_i)$ as preconditions, then g_j is ordered before g_i.

```
1.  procedure Relaxed-Plan-Ordering (P = (F, O, I, G))
2.      construct planning graph G from I to G without computing mutual exclusions;
3.      extract a relaxed plan from G;
4.      for each pair of subgoals g_i and g_j
5.          PREC ← true;
6.          for each action o in P that has g_j as an add effect do
7.              if (g_i ⊈ del(o)) and (pre(o) ∩ F(g_i) == ∅) then
8.                  PREC ← false;
9.              end_if
10.         end_for
11.         if (PREC or g_j is reached before g_i in the relaxed plan) then
12.             order g_j before g_i;
13.         else
14.             order g_i before g_j;
15.         end_if
16.     end_for
17. end_procedure
```

Fig. 2. The relaxed-plan ordering algorithm that uses a planning problem $\mathcal{P}$ with m subgoals $\mathcal{G} = (g_1, \cdots, g_m)$ as input and that outputs an ordered sequence of subgoals.

A deficiency of reasonable ordering is that it only analyzes the interactions of those subgoal facts in $\mathcal{G}$ but does not consider the initial state $\mathcal{I}$. Therefore, it can generate invariant ordering relations that only hold true for *any* initial state, which are rare in practice. For instance, our tests on all the IPC4 domains show that reasonable ordering can find some partial orders in 42 out of the 50 instances in the Airport domain, 42 out of the 100 instances in the Pipesworld domain, and none in the other five domains (Satellite, Promela, UMTS, Settlers, and PSR).

2.2. *Proposed Relaxed-Plan Ordering*

Figure 2 shows our proposed relaxed-plan ordering algorithm that takes both $\mathcal{I}$ and $\mathcal{G}$ into consideration in generating partial orders. It consists of three steps:

a) Line 2: Build a planning graph[2] G from initial state $\mathcal{I}$ until a fixed point is reached. Starting from a level with all facts in $\mathcal{I}$, G alternates between a level of all possibly achievable actions and a level of all possibly achievable facts. It stops at a certain fixed-point level where no new actions or facts can be added. In contrast to planning graphs in Graphplan, we do not compute mutual exclusions in this step.

b) Line 3: Based on G, extract a relaxed plan from $\mathcal{I}$ to $\mathcal{G}$ by ignoring delete effects. This is the standard backward chaining in FF's relaxed-plan heuristic.[11]

c) Lines 4-16: Determine a proper order between each pair of subgoals g_i and g_j by examining all actions in G that makes g_j true. If any of these actions either deletes g_i or needs a fact in the FALSE set $F(g_i)$ as a precondition, then g_j is ordered before g_i; otherwise, g_i and g_j are in the same order as they appear in the relaxed plan. If a cycle occurs in the order, then the subgoals involved must be at the same level of the relaxed planning graph. In that case, we will use the original sequence specified in the problem file to order these subgoals.

An important property of our relaxed-plan ordering is that it is strictly stronger than reasonable ordering in the sense that all partial orders detected by reasonable ordering are also detected by relaxed-plan ordering but not vice versa. This is true because in reasonable ordering, g_j is ordered before g_i only when *all actions* supporting g_j invalidate g_i, whereas in relaxed-plan ordering, g_j is ordered before g_i when *all actions in planning graph* G supporting g_j invalidate g_i. In other words, relaxed-plan ordering can detect more partial orders than reasonable ordering because it uses a subset of all supporting actions for g_j in G when deriving the relations.

Note that our algorithm can use the relaxed plans already constructed in each search step in the planners studied in this paper. It does not have any additional memory requirements and incurs little overhead for computing the partial orders.

2.3. *A Comparison of the Ordering Schemes*

To compare the three schemes for ordering subgoals: the original order provided by the planning model, reasonable ordering, and our proposed relaxed-plan ordering, we measure their quality using N^{inv}, the total number of invalidations in incremental planning. Given m subgoals $g_1, \cdots, g_m$, N^{inv} is defined as:

$$N^{inv} = N_1^{inv} + N_2^{inv} + \cdots + N_m^{inv},$$

where N_i^{inv} is the number of times g_i is invalidated. To compute N_i^{inv}, suppose g_i first appears in the k^{th} subproblem ($g_i \in \mathcal{G}_k$). We have:

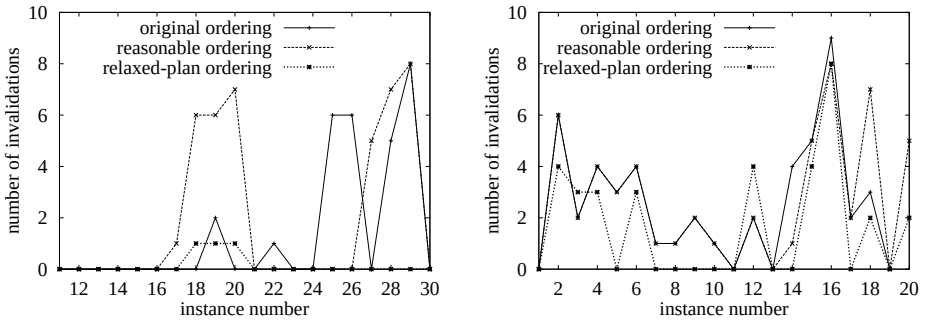
$$N_i^{inv} = \sum_{j=k+1}^{N} Inv(g_i, j),$$

where $Inv(g_i, j)$ is 1 if g_i is invalidated in the subplan for $\mathcal{P}_j$ and is 0 otherwise.

We have evaluated the three ordering schemes for two of the IPC4 domains: Airport and Pipesworld. These are domains where a lot of invalidations can occur under random ordering. The other five IPC4 domains are relatively easy in the sense that there are a few invalidations under the original or random ordering.

Figure 3 compares N^{inv} for the various instances of the Airport and Pipesworld domains with respect to the three ordering schemes in which Metric-FF[8] is used as the embedded planner. Figure 3a shows that relaxed-plan ordering is consistently better (has smaller N^{inv}) than reasonable ordering in 20 instances of the Airport domain (the values in the first ten instances are too small to be compared) and is better than original ordering in all but Instances 18 and 20. Figure 3b shows that relaxed-plan ordering is better than original ordering and reasonable ordering in all the 20 instances of the Pipesworld domain except Instances 3 and 12.

Although the differences in the number of invalidations among the three schemes in Figure 3 may be seemingly small, the search overhead actually grows exponentially with respect to the number of invalidations. This happens because once an earlier subgoal is invalidated, the final state of the invalidated subgoal is also invalidated, and all subsequent subgoals based on that state must be reevaluated.



a) Airport domain. b) Pipesworld domain.

Fig. 3. A comparison of original ordering, reasonable ordering, and relaxed-plan ordering in terms of N^{inv} on two IPC4 domains when using Metric-FF as the embedded planner.

2.4. *Dynamic Reordering*

Although relaxed-plan ordering can detect more partial orders than reasonable ordering, it may still get stuck and requires backtracking. This happens when the embedded planner takes too long to find a feasible plan from the final state in one stage to the next stage with a new group of subgoals. For instance, assume that S_1 has been reached in solving $\mathcal{P}_1(\mathcal{G}_1)$ in Figure 1 and that $\mathcal{G}_2$ is ordered before $\mathcal{G}_3$. The embedded planner may fail to find a feasible plan in the time allowed for solving $\mathcal{P}_2(\mathcal{G}_1, \mathcal{G}_2)$ but may find a feasible plan easily when solving $\mathcal{P}_2(\mathcal{G}_1, \mathcal{G}_3)$.

An effective way to alleviate the above problem is to re-order the subgoals and to achieve those easy-to-reach subgoals first. For example, if an airplane at A needs to visit B and C and there is a path $A \rightarrow B \rightarrow C$, then it is more efficient to get to B before reaching C than to get to C before reaching B. This ordering may not be detected in the initial relaxed-plan ordering because the airplane may be initially at D and is closer to C than to B. Therefore, relaxed-plan ordering will order C before B. But as planning progresses, the current state changes and the airplane may be moved to A, where B is closer than C. At this point, re-ordering B before C will allow the airplane to move to B first.

To this end, we propose to dynamically re-order unsatisfied subgoals, based on a heuristic estimate of the distance from the current state to each subgoal. We set a threshold of planning time for each subproblem and invoke dynamic re-ordering when the threshold is exceeded. At that time, we use FF's relaxed-plan heuristic to estimate the distance from the current state to each of the unsatisfied subgoals, and order the subgoals in an ascending order of their estimated distances.

In practice, the above strategy can effectively avoid a search getting stuck at some difficult subgoals with a large heuristic distance from the current state. For example, if an airplane is at A and there is a path $A \rightarrow B \rightarrow C$, then the relaxed-plan heuristic will indicate a shorter distance to visit B from A. Therefore, using dynamic re-ordering, the airplane will try to get to B first before reaching C.

Table 1. Trade-offs between grain size $|G_i|$ and the total time T_{total} for solving the Satellite-15 instance with 24 subgoals using incremental planning. Also shown are the number of subproblems N and the average time for solving a subproblem T_s. The optimal grain size is to have 2 subgoals in each subproblem and 12 subproblems.

N	1	2	3	4	6	**12**	24
$\|G_i\|$	24	12	8	6	4	**2**	1
T_s	6.6	2.2	1.4	1.1	0.03	**0.002**	0.002
T_{total}	6.6	4.3	4.2	4.3	0.2	**0.02**	0.06

3. Granularity in Incremental Planning

After determining the ordering of subgoals, we need to partition them into N disjoint sets: $\mathcal{G}_1, \cdots, \mathcal{G}_N$, where $\mathcal{G} = \bigcup_{i=1}^{n} \mathcal{G}_i$. These sets will be used in incremental planning in which $\mathcal{P}_i, i = 1, \ldots, N$, will generate a plan to achieve $\mathcal{G}_1, \ldots, \mathcal{G}_i$.

In one extreme, the smallest grain size is one in which each subgoal set has one subgoal fact; that is, $G_i = \{g_i\}$. In this case, there will be a large number of iterations in incremental planning, although each subproblem is trivial. In the other extreme, all subgoals are grouped into one subgoal set; that is $G_1 = \{g_1, \ldots, g_m\}$. In this case, the benefit of incremental planning is not exploited.

The efficiency of incremental planning depends on the granularity of the subgoal sets chosen. There is a trade-off between this granularity and the complexity of solving the overall problem. One can estimate the total planning time by the sum of the planning times for solving each subgoal individually. If the grain size is too small, then each subproblem will be easy to solve, but there will be many subproblems to be evaluated and the complexity of incremental planning will be high. In contrast, if the grain size is large, then there will only be a few subproblems to be evaluated, but each subproblem will be very expensive to solve. It is clear that there is an optimal grain size that minimizes the total time of incremental planning.

Table 1 illustrates the trade-offs between grain size ($|G_i|$) and the total planning time in incremental planning (T_{total}) for solving the Satellite-15 instance in IPC4. Note that T_{total} is not equal to the product of the number of subproblems (N) and the average time for solving one subproblem (T_s). The reason is that T_s is only the time for solving each partition of subgoals, without considering any previously achieved subgoals. The results show that neither the smallest nor the largest grain size leads to the optimal total time. In this example, the best total time is obtained by 12 subproblems, each with a grain size of 2.

We have experimentally designed a strategy for determining the granularity in incremental planning. Given a planning problem $\mathcal{P}$ with subgoal set $\mathcal{G}$, we first partition the subgoals into ten subsets, each with $\frac{|\mathcal{G}|}{10}$ subgoals. For example, in the Satellite-15 instance illustrated, the initial grain size is $\frac{24}{10} = 2.4$ subgoals. We then test if the number of states evaluated in solving the first subproblem is less than a threshold (4 in our experiments), and double the grain size (number of subgoals in each subset) if the number of states evaluated is less than the threshold. We perform

this iterative doubling until the threshold is exceeded.

In our algorithm, we have used multiplicative instead of additive increases in determining a proper grain size. The former allows us to estimate a coarse grain size quickly. This is preferable because the number of subgoals is relatively small in most domains, and the effectiveness of granularity control depends on how fast one can find a good grain size. Likewise, improvements due to dynamic decreases in grain size will be small when the number of subgoals is not sufficiently large.

4. Experimental Results

In this section, we describe our results on comparing the various ordering and granularity-control strategies using, respectively, Metric-FF,[8] YAHSP 1.1,[14] or LPG-TD-speed 1.0[7] as the embedded planner in incremental planning. We have chosen YAHSP and LPG-TD-speed because they are top planners in IPC4 and their binary codes are available for integration as embedded planners in incremental planning. (YAHSP won the second prize in the Suboptimal Propositional Track, whereas LPG-TD won the second prize in the Suboptimal Temporal Metric Track.) We did not use Downward, another leading IPC4 planner, because it is not available for testing. As YAHSP and LPG-TD-speed are only available in binary form, they will incur additional overheads in parsing instance files each time they are called. Since this overhead should be incurred once initially, we discount the overhead of parsing instance files multiple times in our experiments.

We have also compared our planner with $\text{SGPlan}_{4.1}$.[3] This planner won the first prize in the Suboptimal Temporal Metric Track and the second prize in the Suboptimal Propositional Track in IPC4. It cannot be used as an embedded planner because it already performs subgoal partitioning and employs Metric-FF as its basic planner. $\text{SGPlan}_{4.1}$ employs a different partition-and-resolve approach that solves each subproblem individually and that resolves inconsistent global constraints across the subproblems at the end.

Each planner is called with two arguments: *domain_file* and *instance_file*, where *instance_file* contains the initial and the goal states. We first pre-process the inputs by Metric-FF's parser and get the internal representation of the initial and the goal states. All the subproblems generated from a problem use the same *domain_file* but use different *instance_file*s with unique initial and goal states.

In the first iteration of planning, the single subproblem to be solved uses the original initial state as its initial state and a goal state with one subgoal. The final state of solving the first subproblem is obtained by matching the sequence of actions found with the internal representation generated by Metric-FF's parser. Using this state as the initial state, the second subproblem tries to achieve a goal state with one more subgoal. The process is repeated until all the subgoals have been achieved.

Table 2 summarizes the results of the various combinations of planners, grain sizes, and subgoal ordering schemes for the IPC4 AIRPORT-34 instance.[6] This instance involves the planning of eight planes to take off while three are going to their

Table 2. An evaluation of the various combinations of solvers, grain sizes, and subgoal ordering schemes on AIRPORT-34.

	Strategy	Time	States	Actions	N^{inv}
Metric-FF	no partitioning	–	–	–	–
	reasonable-order	–	–	–	–
	optimal-order(1)	9.18	540	427	0
	random-order(1)	–	–	–	–
	original-order(1)	–	–	–	–
	proposed(1)	9.23	540	427	0
YAHSP Planner	no partitioning	–	–	–	–
	reasonable-order	–	–	–	–
	optimal-order(1)	0.23	41	443	0
	optimal-order(d)	0.22	30	443	0
	random-order(1)	–	–	–	–
	random-order(d)	–	–	–	–
	original-order(1)	–	–	–	–
	original-order(d)	–	–	–	–
	proposed(1)	0.22	41	443	0
	proposed(d)	0.20	30	443	0
LPG-TD-speed	no partitioning	58.49	n/a	427	0
	reasonable-order	292.06	n/a	493	3
	optimal-order(1)	4.56	n/a	427	0
	random-order(1)	–	–	–	–
	original-order(1)	–	–	–	–
	proposed(1)	4.68	n/a	427	0
SG	partition+resolve	96.75	n/a	427	0

Keys

–	Strategy failed to solve instance in one hour of run time on an AMD MP2000 system running Linux AS3
States	Number of states evaluated (not provided in LPG)
Actions	Length of the solution plan
N^{inv}	Total number of subgoal invalidations
1	Strategy with one subgoal in each subproblem
d	Strategy with dynamically adjusted grain size

parking positions. We have evaluated the three original planners without partitioning and with default parameters, and our incremental planner under reasonable ordering, original ordering, random ordering, proposed relaxed-plan ordering, and a hypothetical optimal ordering. The optimal order serves as an (impractical) upper bound on performance and was determined by examining the order in which subgoals were satisfied after the solution plan has been generated. Note that dynamic granularity cannot be applied in LPG-TD-speed because the number of states evaluated is not available. Also, dynamic granularity is not applicable in Metric-FF for this instance because the number of states evaluated in solving the subproblem in the first stage is larger than our threshold of four.

The results show that AIRPORT-34 cannot be solved by Metric-FF and YAHSP but can be solved by LPG-TD-speed and SGPlan$_{4.1}$. They also show that incremental planning can solve the instance much faster than the original planners.

Table 3 compares the performance of our incremental version with that of the

Table 3. Quality-time comparisons of the incremental version of three planners and the non-incremental version. We also compare the performance of the incremental version of Metric-FF and $\text{SGPlan}_{4.1}$ that uses Metric-FF as its basic solver.

	Domain	F_i	F_q	F_t	F_w	F_{wt}	F_{wq}	F_1	F_2	F_u
Metric-FF-inc vs. Metric-FF	airport	0.66	0.02	0.04	0.00	0.00	0.00	0.28	0.00	0.00
	pipesworld	0.20	0.02	0.24	0.10	0.00	0.12	0.14	0.04	0.14
	pw-tankage	0.16	0.12	0.04	0.00	0.00	0.00	0.44	0.02	0.20
	optical	0.71	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.29
	philosophers	0.41	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.59
	psr-small	0.56	0.08	0.14	0.00	0.00	0.02	0.04	0.12	0.04
	satellite	0.39	0.00	0.42	0.00	0.00	0.06	0.06	0.00	0.08
	freecell	0.30	0.15	0.35	0.20	0.00	0.00	0.00	0.00	0.00
	depots	0.20	0.00	0.70	0.00	0.00	0.00	0.10	0.00	0.00
	logistics	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
Yahsp-inc vs. Yahsp	airport	0.60	0.06	0.04	0.00	0.00	0.00	0.26	0.00	0.02
	pipesworld	0.10	0.46	0.04	0.32	0.00	0.04	0.00	0.00	0.00
	pw-tankage	0.18	0.18	0.28	0.22	0.00	0.00	0.08	0.02	0.04
	optical	0.93	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.07
	philosophers	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
	psr-small	0.38	0.00	0.22	0.02	0.00	0.24	0.00	0.00	0.00
	satellite	0.83	0.00	0.08	0.00	0.00	0.06	0.00	0.03	0.00
	freecell	0.00	0.60	0.00	0.15	0.00	0.00	0.05	0.20	0.00
	depots	0.30	0.30	0.15	0.00	0.00	0.00	0.05	0.15	0.05
	logistics	0.02	0.00	0.89	0.00	0.00	0.10	0.00	0.00	0.00
LPG-TD-speed-inc vs. LPG-TD-speed	airport	0.70	0.00	0.12	0.08	0.00	0.00	0.10	0.00	0.00
	pipesworld	0.24	0.10	0.14	0.34	0.00	0.02	0.10	0.02	0.04
	pw-tankage	0.16	0.04	0.10	0.20	0.00	0.02	0.22	0.06	0.20
	optical	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	philosophers	0.31	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.69
	psr-small	0.30	0.14	0.12	0.24	0.00	0.00	0.02	0.02	0.00
	satellite	0.06	0.03	0.36	0.36	0.00	0.03	0.00	0.08	0.03
	freecell	0.05	0.00	0.20	0.00	0.00	0.00	0.00	0.70	0.05
	depots	0.15	0.05	0.35	0.40	0.00	0.05	0.00	0.00	0.00
	logistics	0.00	0.00	0.77	0.21	0.00	0.02	0.00	0.00	0.00
Metric-FF-inc vs. $\text{SGPlan}_{4.1}$	airport	0.72	0.00	0.16	0.00	0.00	0.00	0.12	0.00	0.00
	pipesworld	0.40	0.24	0.02	0.08	0.00	0.08	0.00	0.18	0.00
	pw-tankage	0.22	0.24	0.00	0.06	0.00	0.02	0.22	0.10	0.12
	optical	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
	philosophers	0.00	0.00	0.00	0.00	0.41	0.00	0.00	0.59	0.00
	psr-small	0.58	0.00	0.14	0.00	0.00	0.02	0.02	0.12	0.04
	satellite	0.31	0.03	0.17	0.25	0.00	0.03	0.00	0.06	0.08
	freecell	0.25	0.00	0.55	0.05	0.00	0.05	0.10	0.00	0.00
	depots	0.15	0.05	0.65	0.05	0.00	0.10	0.00	0.00	0.00
	logistics	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00

Keys: (t_i, q_i) = (time, quality) by the inc. method (INC) (smaller values are better)
(t_n, q_n) = (time, quality) by the non-inc. method (NON-INC)
F_i: Fraction that $t_i \leq t_n$ and $q_i \leq q_n$; F_q: Fraction that $t_i > t_n$ and $q_i < q_n$
F_t: Fraction that $t_i < t_n$ and $q_i > q_n$; F_w: Fraction that $t_i > t_n$ and $q_i > q_n$
F_{wt}: Fraction that $t_i > t_n$ and $q_i = q_n$; F_{wq}: Fraction that $t_i = t_n$ and $q_i > q_n$
F_1: Fraction solved by INC but not by NON-INC
F_2: Fraction solved by NON-INC but not by INC
F_u: Fraction unsolved by both INC and NON-INC

original planner, using a quality measure on the number of actions and a run-time limit of 30 minutes on an AMD MP2000 system with Linux AS3. It also compares the performance of the incremental version of Metric-FF with that of $\text{SGPlan}_{4.1}$.

To evaluate our proposed framework, we conducted experiments on all IPC4 STRIPS domains, the Depots and the Freecell domains in IPC3,[4] and the Logistics domain in IPC2.[1] The latter three domains are considered difficult for incremental planning: Freecell has a strong inter-dependency or interference among its subgoals;

Logistics has a number of positive goal interactions in its instances; and Depots is a combination of Logistics and the well-known Blocksworld.

The results show that incremental planning can generally solve more instances using less run times than the original planners. Metric-FF-inc has been found to require less run times than $\text{SGPlan}_{4.1}$ for a majority of the instances. $\text{SGPlan}_{4.1}$, however, is better in several domains because it not only partitions a problem by its subgoals but also carries out landmark analysis and symmetry-group detection[3] to further decompose a problem into smaller subproblems.

Although incremental planning is in general faster than the original planner ($F_i + F_t + F_{wq} > F_q + F_w + F_{wt}$ for most domains), it is likely to lead to longer plans. These quality degradations ($F_q + F_w + F_{wq} > F_i + F_t + F_{wt}$) exist in a number of domains, such as Pipesworld, Satellite, Freecell, Depots and Logistics.

The ordering of subgoals is crucial for generating shorter plans in incremental planning. In the original version of our incremental planners, we have implemented one subgoal order. As there are significant improvements on run time, we have implemented a second version that evaluates two alternative subgoal orders in order to search for better plans. Our planner first tries the proposed relaxed-plan ordering and then the original ordering specified in the problem definition. Since we would like to restrict the total time spent, we set a time limit for the second alternative to be five times the time spent for the first alternative plus one minute. The time of incremental planning reported is then the total time for both alternatives and within the 30-minute limit. The reported solution is the one with the shorter plan.

Table 4 summarizes the results of our redundant-ordering scheme. The results show improved quality as compared to that of the original implementation without redundant ordering. For example, in the Logistics domain, the original incremental version generates solutions of worse quality for all the instances ($F_t = 1$), whereas Metric-FF-inc with redundant ordering can improve both the quality and the run time for a majority of the instances ($F_i = 0.92$).

Figure 4 depicts the quality-time trade-offs of each benchmark between $\text{SGPlan}_{4.1}$ and three versions of the target planner: the original planner, its incremental version, and the incremental version with redundant ordering. For each benchmark, we first select the best embedded planner to be used in incremental planning according to the number of instances solved (and using run time to break ties) by the original planner. We then plot the quality and time normalized with respect to those of $\text{SGPlan}_{4.1}$ for the best planner selected, the corresponding incremental version, and the incremental version with redundant ordering. The reason to select only one planner for comparison is that, if this planner is better than the others in terms of run time, then its incremental version is also the best among the incremental versions.

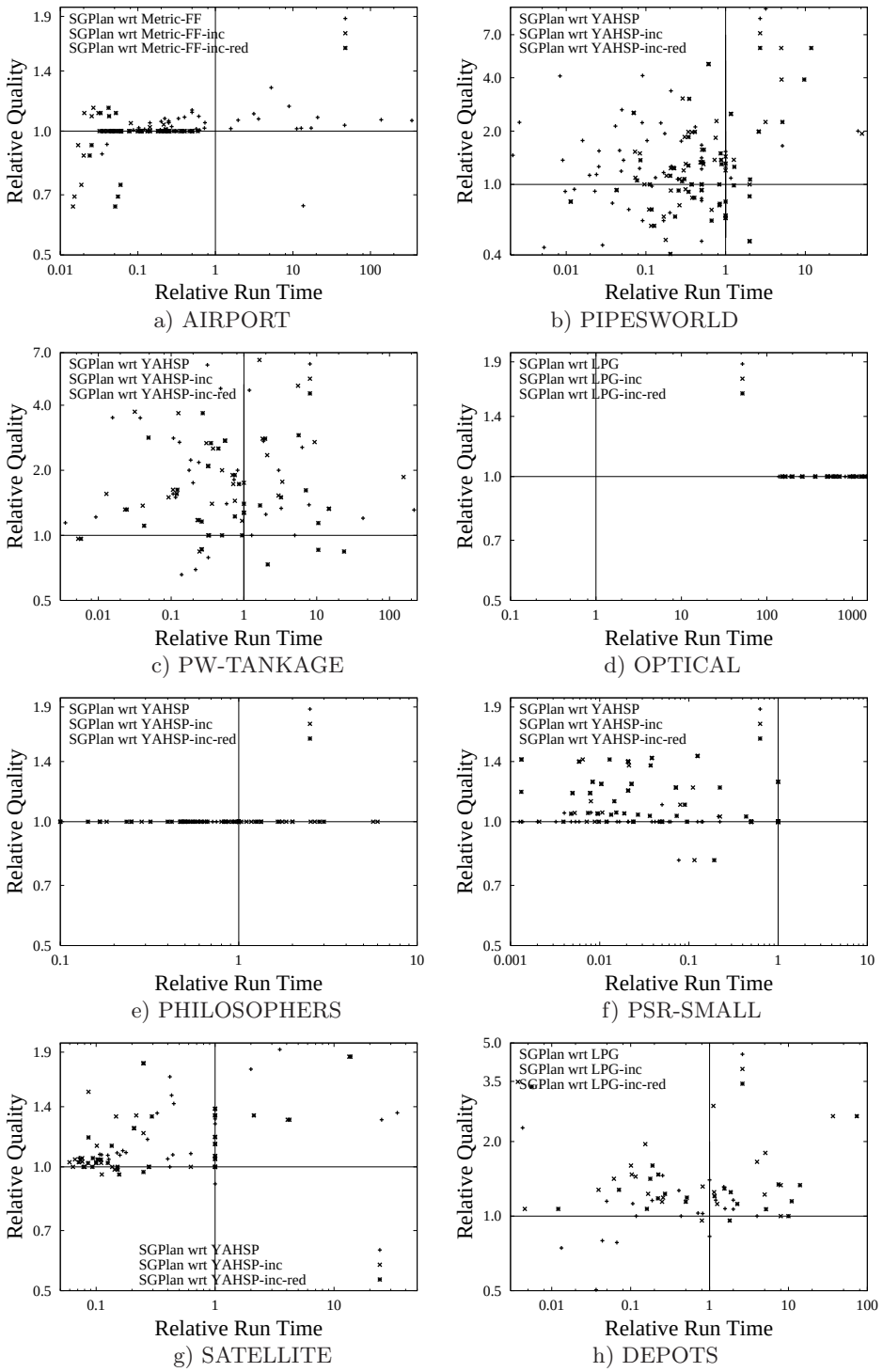
In the AIRPORT domain (Figure 4a), the two incremental versions of Metric-FF use much less run times than $\text{SGPlan}_{4.1}$, while the non-incremental version generally incurs more run times than $\text{SGPlan}_{4.1}$. In this domain, relaxed-plan ordering can reduce the number of airplanes in the airport quickly because the "take-off" subgoals

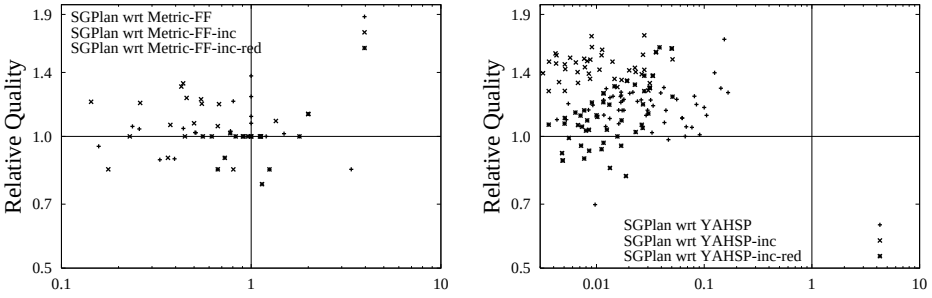
Table 4. Quality-time comparisons of the incremental version of three planners with redundant ordering and the non-incremental version. We also compare Metric-FF-inc-redundant and $\text{SGPlan}_{4.1}$. (See the explanations of keys in Table 3.)

	Domain	F_i	F_q	F_t	F_w	F_{wt}	F_{wq}	F_1	F_2	F_u
Metric-FF-inc-redundant vs. Metric-FF	airport	0.66	0.02	0.04	0.00	0.00	0.00	0.28	0.00	0.00
	pipesworld	0.28	0.06	0.20	0.10	0.00	0.04	0.14	0.04	0.14
	pw-tankage	0.16	0.16	0.02	0.00	0.00	0.00	0.44	0.02	0.20
	optical	0.71	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.29
	philosophers	0.41	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.59
	psr-small	0.56	0.08	0.14	0.00	0.00	0.02	0.04	0.12	0.04
	satellite	0.72	0.00	0.11	0.00	0.00	0.03	0.06	0.00	0.08
	freecell	0.40	0.45	0.00	0.15	0.00	0.00	0.00	0.00	0.00
	depots	0.86	0.05	0.00	0.00	0.00	0.00	0.09	0.00	0.00
	logistics	0.92	0.00	0.04	0.04	0.00	0.00	0.00	0.00	0.00
Yahsp-inc-redundant vs. Yahsp	airport	0.52	0.14	0.04	0.00	0.02	0.00	0.26	0.00	0.02
	pipesworld	0.08	0.50	0.02	0.30	0.06	0.04	0.00	0.00	0.00
	pw-tankage	0.24	0.28	0.06	0.26	0.02	0.00	0.08	0.02	0.04
	optical	0.93	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.07
	philosophers	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
	psr-small	0.28	0.00	0.20	0.02	0.24	0.26	0.00	0.00	0.00
	satellite	0.69	0.17	0.06	0.00	0.00	0.06	0.00	0.03	0.00
	freecell	0.00	0.65	0.00	0.10	0.00	0.00	0.05	0.20	0.00
	depots	0.23	0.27	0.14	0.00	0.09	0.00	0.05	0.14	0.09
	logistics	0.67	0.00	0.23	0.00	0.00	0.10	0.00	0.00	0.00
LPG-TD-SP-inc-red. vs. LPG-TD-Speed	airport	0.54	0.08	0.08	0.06	0.14	0.00	0.10	0.00	0.00
	pipesworld	0.18	0.20	0.06	0.36	0.04	0.00	0.10	0.02	0.04
	pw-tankage	0.10	0.12	0.04	0.22	0.04	0.00	0.22	0.06	0.20
	optical	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	philosophers	0.31	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.69
	psr-small	0.04	0.34	0.06	0.24	0.28	0.00	0.02	0.02	0.00
	satellite	0.19	0.06	0.06	0.53	0.06	0.00	0.00	0.08	0.03
	freecell	0.05	0.00	0.10	0.10	0.00	0.00	0.00	0.70	0.05
	depots	0.14	0.14	0.09	0.64	0.00	0.00	0.00	0.00	0.00
	logistics	0.62	0.15	0.00	0.21	0.02	0.00	0.00	0.00	0.00
Metric-FF-inc-redundant vs. $\text{SGPlan}_{4.1}$	airport	0.78	0.00	0.10	0.00	0.00	0.00	0.12	0.00	0.00
	pipesworld	0.30	0.40	0.00	0.12	0.00	0.00	0.00	0.18	0.00
	pw-tankage	0.22	0.26	0.00	0.02	0.06	0.00	0.22	0.10	0.12
	optical	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
	philosophers	0.00	0.00	0.00	0.00	0.41	0.00	0.00	0.59	0.00
	psr-small	0.58	0.00	0.14	0.00	0.08	0.02	0.02	0.12	0.04
	satellite	0.19	0.03	0.00	0.00	0.67	0.00	0.03	0.03	0.06
	freecell	0.50	0.10	0.05	0.00	0.20	0.05	0.10	0.00	0.00
	depots	0.36	0.09	0.00	0.00	0.55	0.00	0.00	0.00	0.00
	logistics	0.85	0.04	0.00	0.08	0.04	0.00	0.00	0.00	0.00

cannot be invalidated once they have been achieved. Although there are some dead-ends that cannot be recognized by the relaxed-plan heuristic,[10] they only occur in uncommon regions with traffic congestion[9] and do not occur elsewhere. In terms of quality, relaxed-plan ordering helps produce better solutions than the original orders, and incremental planning generates shorter plans than those of Metric-FF.

In the two PIPESWORLD domains (Figures 4b and 4c), YAHSP and its incremental versions are much faster than $\text{SGPlan}_{4.1}$ for most of the instances, although


SGPlan wrt Metric-FF
SGPlan wrt Metric-FF-inc
SGPlan wrt Metric-FF-inc-red
Relative Quality
Relative Run Time
a) AIRPORT
SGPlan wrt YAHSP
SGPlan wrt YAHSP-inc
SGPlan wrt YAHSP-inc-red
Relative Quality
Relative Run Time
b) PIPESWORLD
SGPlan wrt YAHSP
SGPlan wrt YAHSP-inc
SGPlan wrt YAHSP-inc-red
Relative Quality
Relative Run Time
c) PW-TANKAGE
SGPlan wrt LPG
SGPlan wrt LPG-inc
SGPlan wrt LPG-inc-red
Relative Quality
Relative Run Time
d) OPTICAL
SGPlan wrt YAHSP
SGPlan wrt YAHSP-inc
SGPlan wrt YAHSP-inc-red
Relative Quality
Relative Run Time
e) PHILOSOPHERS
SGPlan wrt YAHSP
SGPlan wrt YAHSP-inc
SGPlan wrt YAHSP-inc-red
Relative Quality
Relative Run Time
f) PSR-SMALL
SGPlan wrt YAHSP
SGPlan wrt YAHSP-inc
SGPlan wrt YAHSP-inc-red
Relative Quality
Relative Run Time
g) SATELLITE
SGPlan wrt LPG
SGPlan wrt LPG-inc
SGPlan wrt LPG-inc-red
Relative Quality
Relative Run Time
h) DEPOTS



i) FREECELL j) LOGISTICS

Fig. 4. Quality-time distributions of the best target planner and its incremental versions for each instance normalized with respect to the corresponding metrics of $\text{SGPlan}_{4.1}$ for the same instance on all instances solvable by both planners. (Performance values larger than one are better for $\text{SGPlan}_{4.1}$.)

they both found much longer solution plans. Redundant ordering can help improve plan quality but is still not competitive with respect to $\text{SGPlan}_{4.1}$.

In the OPTICAL domain (Figure 4d), $\text{SGPlan}_{4.1}$ is significantly faster than the other planners because it does not backtrack in its state-space search but uses penalties to resolve global violations. In this domain, the "Ignoring-Delete-Lists" heuristic does not work well because there are local minima that need possibly an unbounded number of steps to escape from.[10] In this case, incremental planning still improves in terms of run time since it can reduce the amount of backtracking. The Philosophers subdomain (Figure 4e) requires a much smaller amount of backtracking than the OPTICAL subdomain because the number of steps to escape from a local minimum is bounded by a small constant.

In the PSR-SMALL domain (Figure 4f), the encoding that compiles away derived predicates makes all subgoals artificial and breaks the assumption that tasks can be decomposed by subgoals. As a result, both $\text{SGPlan}_{4.1}$ and incremental planning by subgoals found plans of similar quality as compared to those of YAHSP while using more times. Due to dynamic granularity control, incremental planning on YAHSP can further improve its speed.

In the Satellite domain (Figure 4g), $\text{SGPlan}_{4.1}$ generates better plans but is usually much slower than YAHSP and its incremental versions. In this domain, there are actions for changing the directions of satellites that are feasible from any states. By aggressively applying the "Ignoring-Delete-Lists" heuristics in its lookahead strategy, YAHSP uses as many of the applicable actions as possible from a relaxed plan and will be much faster in generating feasible solutions. Since YAHSP does not backtrack after finding a feasible plan, it will lead to longer solution plans.

In the Depots domain (Figure 4h), the incremental versions of LPG-TD-speed generate worse plans than LPG-TD-speed. These happen because the Depots domain is an extension of the Blocksworld domain and contains a number of reasonable

orders, but our incremental planner does not always follow these reasonable orders in preventing unnecessary invalidations.

In the Freecell domain (Figure 4i), Metric-FF-inc-red generates the best plans for over half of the instances. There are little differences in run times among the planners, and redundant ordering does not improve the run times.

Finally, in the Logistics domain (Figure 4j), incremental planning can greatly reduce the run times of all the instances. Not only can incremental planning with redundant orders generate better plans than YAHSP for a majority of the instances, it can also generate better plans than $\text{SGPlan}_{4.1}$ for some of the instances, although there is a clear gap in plan quality between $\text{SGPlan}_{4.1}$ and YAHSP.

In short, incremental planning can reduce search complexity without sacrificing plan quality, when the subgoal order and the grain size of the problem to be solved are properly chosen. Our study also shows that the selection of the basic planner has more impact on solution quality than incremental planning itself.

5. Conclusions

In this paper, we have developed strategies for ordering and for grouping subgoals in incremental planning, when each partitioned subproblem is solved by an embedded basic planner. Using a new ordering algorithm that considers both the initial and the goal states, we order subgoals in such a way that greatly reduces the number of invalidations during incremental planning. We have studied an effective strategy that dynamically adjusts the grain size of partitioning in order to operate with the best grain size and to minimize the complexity of incremental planning. We have also shown improved plan quality by evaluating two alternative subgoal orders during planning. Incremental planning, when integrated with existing planners, can generally solve more instances and uses less run times without sacrificing quality.

References

1. F. Bacchus. The AIPS'00 planning competition. *AI Magazine*, 22(3):47–56, 2001.
2. A. L. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.
3. Y. X. Chen, B. W. Wah, and C. W. Hsu. Temporal planning using subgoal partitioning and resolution in SGPlan. *J. of Artificial Intelligence Research*, (accepted to appear) Jan. 2006.
4. D. Long and M. Fox. 3rd International Planning Competition. http://planning.cis.strath.ac.uk/competition/, 2002.
5. E. Durfee and V. Lesser. Incremental Planning to Control a Blackboard-based Problem Solver. *Proc. of the Fifth National Conf. on Artificial Intelligence*, pages 58–64, August 1986.
6. S. Edelkamp and J. Hoffmann. Classical part, 4th International Planning Competition. http://ls5-www.cs.uni-dortmund.de/~edelkamp/ipc-4/, 2004.
7. A. Gerevini, A. Saetti, and I. Serina. An approach to temporal planning and scheduling in domains with predictable exogenous events. *J. of Artificial Intelligence Research*, 25:187–231, 2006.

8. J. Hoffmann. The Metric-FF planning system: Translating ignoring delete lists to numeric state variables. *J. of Artificial Intelligence Research*, 20:291–341, 2003.
9. J. Hoffmann and S. Edelkamp. The deterministic part of IPC-4: An overview. *J. of Artificial Intelligence Research*, 24:519–579, 2005.
10. J. Hoffmann and S. Edelkamp. Where Ignoring Delete Lists' works: Local search topology in planning benchmarks. *J. of Artificial Intelligence Research*, 24:685–758, 2005.
11. J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *J. of Artificial Intelligence Research*, 14:253–302, 2001.
12. P. Jónsson and C. Bäckström. Incremental planning. In *New directions in AI planning*, pages 79–90. IOS Press, 1996.
13. J. Koehler and J. Hoffmann. On reasonable and forced goal ordering and their use in an agenda-driven planning algorithm. *J. of Artificial Intelligence Research*, 12:339–386, 2000.
14. V. Vidal. A lookahead strategy for heuristic search planning. In *Proc. Int'l Conf. on Automated Planning and Scheduling*, 2004.