Subgoal Ordering and Granularity Control for Incremental Planning*

Chih-Wei Hsu and *Benjamin W. Wah*
Dept. of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
{chsu, wah}@manip.crhc.uiuc.edu

Yixin Chen
Dept. of Computer Science and Engineering
Washington University
St. Louis, MO 63130, USA
{chen@cse.wustl.edu}

Abstract

In this paper, we study strategies in incremental planning for ordering and grouping subproblems partitioned by the subgoals of a planning problem when each subproblem is solved by a basic planner. To generate a rich set of partial orders for ordering subproblems, we propose a new ordering algorithm based on a relaxed plan built from the initial state to the goal state. 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 show an optimal region of grain size that minimizes the total complexity of incremental planning. We propose an efficient strategy to dynamically adjust the grain size in partitioning in order to operate in this optimal region. We further evaluate a redundant-execution scheme that uses two different subgoal orders in order to improve the quality of the plans generated without greatly sacrificing run-time efficiency. Experimental results on using three basic planners (Metric-FF, YAHSP, and LPG-TD-speed) show that our strategies are general for improving the time and quality of each of these planners across various benchmarks.

1 Introduction

In this paper, we study new strategies in incremental planning for solving planning problems. *Incremental planning* [8] 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.

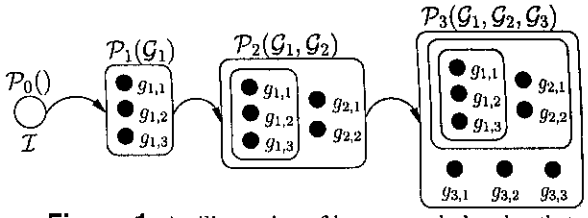


Figure 1. An illustration of incremental planning that decomposes $\mathcal{P}$ with subgoal sets $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$ into subproblems $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$. Each dot is a subgoal fact in $\mathcal{G}$.

The framework studied in this paper 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. *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 planning task of $\mathcal{P}$ is to find a sequence of actions $(o_1, \cdots, o_n)$ that transform the state from $\mathcal{I}$ to a goal state where all the facts in $\mathcal{G}$ are true.

Incremental planning entails the decomposition of $\mathcal{G}$ into N disjoint subgoal sets, $\mathcal{G}_1, \cdots, \mathcal{G}_N$, where $\mathcal{G} = \bigcup_{i=1}^{n} \mathcal{G}_i$, and the solution of the sequence of N subproblems $\mathcal{P}_1, \cdots, \mathcal{P}_N$. 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. It has been used in dynamic

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

IEEE International Conf. on Tools for Artificial Intelligence, 2005.

1082-3409/05 $20.00 © 2005 IEEE

domains in which a planner outputs valid prefixes of a final plan before it finishes planning. Recently, it has been used to decompose large planning tasks into smaller subproblems in such a way that a subset of them can be solved more efficiently than the original problem [8]. It is different from subgoal partitioning in SGPlan_g [3] we have developed. In SGPlan_g, each subproblem that represents 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 is not 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 make the technique effective, we study in this paper two important issues.

a) *Subgoal ordering.* The ordering of subgoals may have great impact on both the search efficiency and the 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 facts found previously for $\mathcal{G}_1, \ldots, \mathcal{G}_{i-1}$. For example, it would be desirable 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. Thus, we do not need extra actions and search time to re-achieve $\mathcal{P}_1(\mathcal{G}_1)$.

A well-known approach for ordering subgoals in incremental planning is reasonable ordering [8]. Although it tries to avoid some unnecessary invalidations of previous subgoals, it does not work well on many IPC4 [4] benchmarks because it deduces partial orders among subgoals by considering only the goal state. Without taking the initial state into consideration, it can generate partial orders that are invariant to any initial state.

In this paper, 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.

b) *Subgoal grouping.* Another aspect that may impact search efficiency 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 minimum time or the best quality in planning.

In this paper, we study a general strategy that groups multiple subgoals together in each step. Based on tradeoffs between the size of subgoal groups and the complexity of each subproblem, we show that there is an optimal group size that minimizes the time for solving the overall problem. We propose an efficient strategy to dynamically adjust the size of subgoal groups in order to operate in this optimal region. To improve plan quality without sacrificing planning time, we introduce a strategy that evaluates two subgoal orders together.

Based on the subgoal ordering and the granularity-control strategies developed, we present a general incremental planning framework that can be integrated with existing STRIPS planners to enhance their performance. Our extensive experimental results on benchmarks show that incremental planning is a general approach that can improve the performance of different planners across multiple domains without sacrificing solution quality.

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 achieved already. 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 [8] 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 supporting g as an add effect and by finding the common delete effects of all supporting actions.

1. **procedure Relaxed-Plan-Ordering** ($\mathcal{P} = (\mathcal{F}, \mathcal{O}, \mathcal{I}, \mathcal{G})$)
2. construct planning graph G from $\mathcal{I}$ to $\mathcal{G}$ without computing mutual exclusions;
3. extract a relaxed plan from G;
4. **for** each pair of subgoals g_i and g_j
5. **set** PREC $\longleftarrow$ **true**;
6. **for** each action o in P that has g_j as an add effect
7. **if** ($g_i \nsubseteq del(o)$) **and** ($pre(o) \cap F(g_i) == \varnothing$)
8. **then** PREC $\longleftarrow$ **false**;
9. **end_for**
10. **if** PREC == **true**
11. **then** order g_j before g_i;
12. **else if** g_j is reached before g_i in the relaxed plan
13. **then** order g_j before g_i;
14. **else** order g_i before g_j;
15. **end_for**
16. **end_procedure**

Figure 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.

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

A deficiency of reasonable ordering is that it only analyzes interactions of the subgoal facts in $\mathcal{G}$ but does not consider the initial state $\mathcal{I}$. Therefore, it can generate invariant ordering relations that 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 50 instances in the Airport domain, 42 out of 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 G [2] from initial state $\mathcal{I}$ until a fixed point is reached. Starting from a fact 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 will level off at a certain fixed-point level where no more 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 [6].

c) Lines 4-15: 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 those subgoals.

An important property of 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 ordering relations.

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

2.3 A Comparison of The Ordering Schemes

To compare the three schemes for ordering subgoals: original ordering provided by the planning model, reasonable ordering, and 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:

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

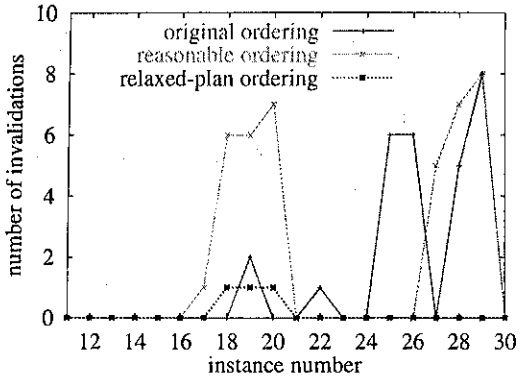
where N_i^{inv} is the number of times g_i is invalidated. To compute N_i^{inv}, suppose g_i first appears in the k^{th} subproblem ($g_i \in \mathcal{G}_k$). We have:

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

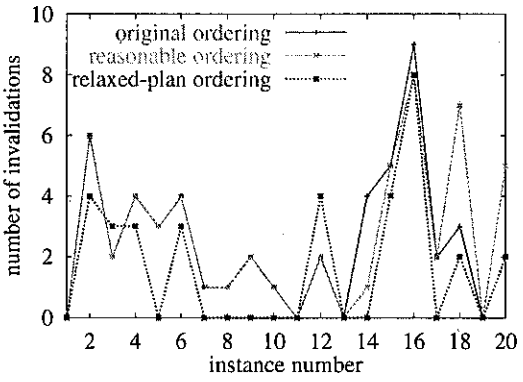
where $Inv(g_i, j)$ is 1 if g_i is invalidated in the subplan for $\mathcal{P}_j$ and is 0 otherwise.

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 few invalidations under the original or random ordering.

Figure 3 compares N^{inv} on the various instances of the Airport and Pipesworld domains with respect to the three ordering schemes when Metric-FF [5] is used as the embedded planner. Figure 3a show that relaxed-plan



a) Airport domain.



b) Pipesworld domain.

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

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. The results in Figure 3b show that relaxed-plan ordering is better than original ordering and reasonable ordering in all the 20 instances of the Pipesworld domain except instance 3 and 12.

Note that the difference in the number of invalidations in Figure 3 among the three schemes may be seemingly small. However, 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.

2.4 Dynamic Reordering

Although relaxed-plan ordering can detect more partial orders than reasonable ordering, the search may still get stuck in incremental planning. 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 state S_1 has been reached in solving $\mathcal{P}_1(\mathcal{G}_1)$ in Figure 1 and that $\mathcal{G}_2$ is ordered before $\mathcal{G}_3$. It is possible that the embedded planner fails 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 try 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 at point D initially and is closer to C than 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 getting a search stuck at some difficult subgoals with a large heuristic distance from the current state. For example, in the previous airplane example, if the airplane is at A and there is a path $A \rightarrow B \rightarrow C$, the relaxed-plan heuristic will indicate that the subgoal of visiting B has a shorter distance from A than that of visiting C. Therefore, using dynamic re-ordering, the airplane will try to get to B first before reaching C.

3 Granularity in Incremental Planning

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

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 by incremental planning. 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 be only 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 ($|G_i|$, number of subgoals in one stage) 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 a grain size of 2 and 12 subproblems.

We have designed experimentally a strategy to dynamically determine the granularity in incremental planning. Given a planning problem $\mathcal{P}$ with subgoal set $\mathcal{G}$, we first partition the subgoals into ten subsets, where each subset has $\frac{|\mathcal{G}|}{10}$ 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 the algorithm above, we have used multiplicative instead of additive increases in determining a proper grain size. Multiplicative increases allow 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 find a good grain size. For the same reason, the improvement due to dynamic decreases of the 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 three basic planners: Metric-FF [5], YAHSP 1.1 (*http://www.cril.univ-artois.fr/~vidal/Yahsp*), and LPG-TD-speed 1.0 (*http://zeus.ing.unibs.it/lpg/registerlpg-td.html*). We have chosen YAHSP and LPG-TD-speed because they are top planners in IPC4 and their binary codes are available for integration as basic 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. Because 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.

Table 1. Trade-offs between grain size $|G_i|$ and the total time T_{total} to solve the Satellite-15 instance with 24 subgoals using incremental planning. Also shown are the number of subproblems N and the average time to solve 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

We have also compared our planner with SGPlan_g (*http://manip.crhc.uiuc.edu/programs/SGPlan*) [3]. The 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 a basic planner because it already performs subgoal partitioning and employs Metric-FF as a basic planner. As is discussed earlier, SGPlan_g employs a different partition-and-resolve approach.

Table 2 summarizes the results of the various combinations of basic solvers, grain sizes, and subgoal ordering schemes on the IPC4 AIRPORT-34 instance [4]. This instance involves the planning of eight planes to take off while three are going to their 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 the threshold of four.

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

Table 3 compares the performance of our incremental implementation with that of the corresponding ba-

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

	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 CPU 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	Argument of strategy with one subgoal in each subproblem
d	Argument of strategy with dynamically adjusted grain size

sic planner, using a quality measure of the number of actions and a CPU time limit of 30 minutes. It also compares the performance of the incremental version of Metric-FF with that of SGPlan$_g$. To evaluate our proposed framework, we tested all IPC4 STRIPS domains. We also conducted experiments on two IPC3 domains (Depots and Freecell) [7] and one IPC2 domain (Logistics) [1]. The latter three domains are considered difficult for incremental planning. In Freecell, there is a strong inter-dependency or interference among subgoals, whereas instances in Logistics have a number of positive goal interactions. Last, Depots is indeed a combination of Logistics and the well-known Blocksworld.

The results show that incremental planning can generally solve more instances and use less times than the original planners. We have also found that Metric-FF-inc performs better in terms of planning time than SGPlan$_g$ on a majority of the instances. SGPlan$_g$ is better in several domains because it not only partitions a problem by its subgoals but also performs landmark analysis and symmetry-group detection [3] to further decompose a problem into smaller subproblems.

Table 3. Quality-time comparisons of the incremental version of three planners and the non-incremental version. The table also compares the performance of the incremental version of Metric-FF and SGPlan$_g$ 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
Comparisons Between Metric-FF-inc and 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
Comparisons Between Yahsp-inc and 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
Comparisons Between LPG-TD-speed-inc and 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
Comparisons Between Metric-FF-inc and SGPlan									
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	0.10	0.10	0.35	0.35	0.00	0.05	0.00	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 incremental method (INC) (smaller values are better for both time and quality)
(t_n, q_n) = (time, quality) by 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

Although incremental planning can be ten times faster than the original planner, it is likely to lead to longer plans. These quality degradations exist in a num-

ber of domains, such as Pipesworld, Satellite, Freecell, Depots and Logistics, and are illustrated in in Figures 4. The figure plots the trade-offs between time and quality, where the performance measures have been normalized with respect to those of the non-incremental version.

The ordering of subgoals is crucial in incremental planning for generating shorter plans. 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 our 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 ordering to be five times the time spent for the first ordering plus one minute. The time of incremental planning reported is then the total time to find both orders and within the 30-minute limit. The reported solution is the one with the shorter length.

Table 4 summarizes the results of our redundant-ordering scheme. Figure 5 also shows the new quality-time distributions on the Airport, Satellite, and Logistics domains. The results show improved quality with little increase in time, as compared to the original implementation without redundant ordering.

Table 4. Quality-time comparisons of the incremental version of three planners with redundant ordering and the non-incremental version. The table also compares the performance of Metric-FF-inc-redundant and SGPlan_g 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
Comparisons between Metric-FF-inc-redundant and 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	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
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
Comparisons between Yahsp-inc-redundant and 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
Comp. between LPG-TD-SP-inc-redundant and LPG-TD-SP									
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
Comparisons between Metric-FF-inc-redundant and SGPlan									
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	0.05	0.15	0.00	0.00	0.80	0.00	0.00	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

5 Conclusions

In this paper, we have developed strategies to order and group subgoals in incremental planning when each partitioned subproblem is solved by a 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 efficient strategy that adjusts dynamically the grain size of partitioning in order to operate with the optimal grain size that minimizes the complexity of incremental planning. We have also shown improved solution quality by evaluating two alternative subgoal orders during planning.

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, C.-W. Hsu, and B. W. Wah. SGPlan: Subgoal partitioning and resolution in planning. In *Proc. Fourth Int'l Planning Competition*. Int'l Conf. on Automated Planning and Scheduling, June 5 2004.

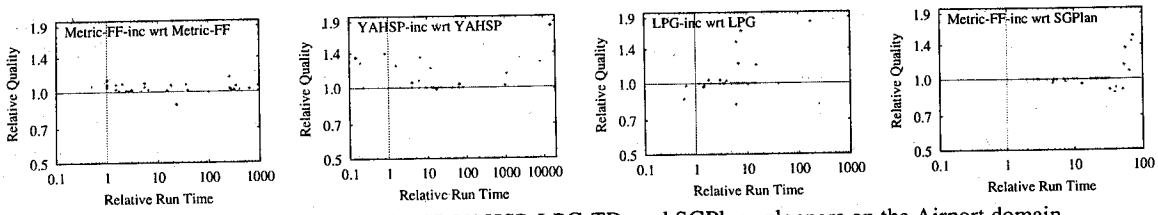
[4] S. Edelkamp and J. Hoffmann. Classical part, 4th international planning competition. http://ls5-www.cs.uni-dortmund.de/~edelkamp/ipc-4/, 2004.

[5] J. Hoffmann. The metric-ff planning system: Translating ignoring delete lists to numeric state variables. *J. of Artificial Intelligence Research*, 20:291–341, 2003.

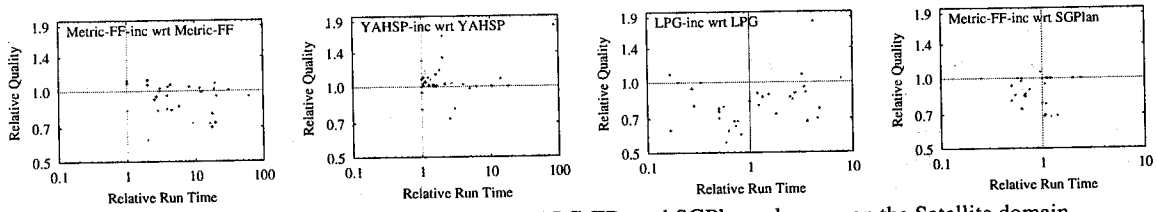
[6] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *J. of Artificial Intelligence Research*, 14:253–302, 2001.

[7] Int'l Planning Competition. AIPS 2002 competition domains. http://www.dur.ac.uk/d.p.long/IPC/domains.html, 2002.

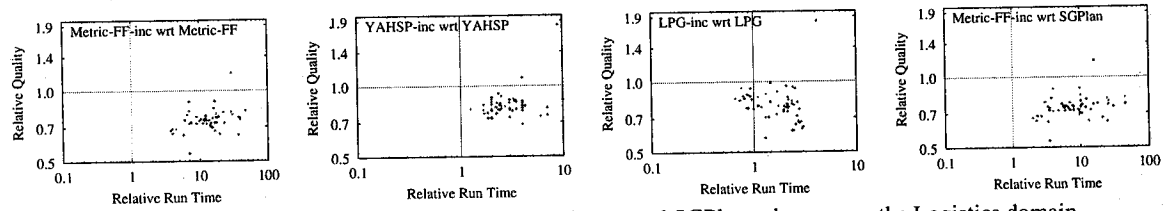
[8] J. Koehler and J. Hoffmann. On reasonable and forced goal ordering and their use in an agenda-driven planning algorithm. *J. of Artificial Intelligence Research*, 12:339–386, 2000.



a) Results comparing the Metric-FF, YAHSP, LPG-TD, and SGPlan$_g$ planners on the Airport domain.

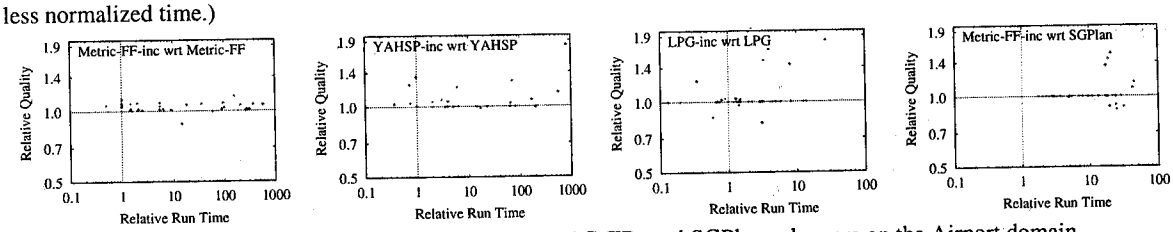


b) Results comparing the Metric-FF, YAHSP, LPG-TD, and SGPlan$_g$ planners on the Satellite domain.

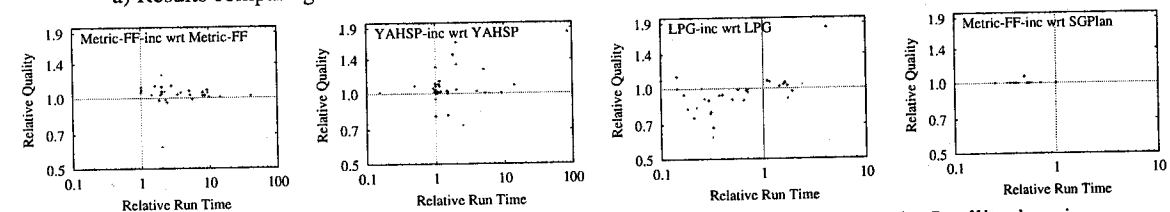


c) Results comparing the Metric-FF, YAHSP, LPG-TD, and SGPlan$_g$ planners on the Logistics domain.

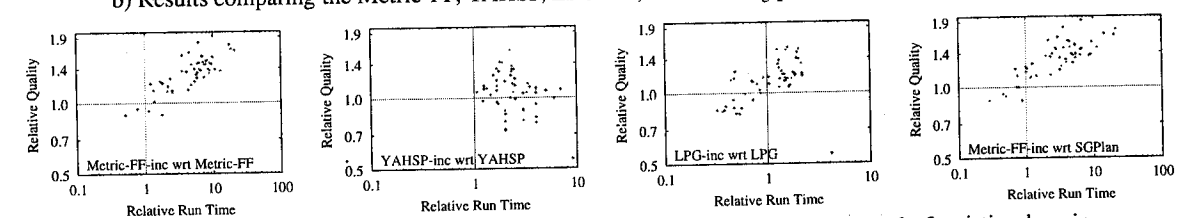
Figure 4. Quality-time distributions of the performance of the incremental version of a planner with respect to that of the non-incremental version on three domains. (The upper right quadrant in each plot represents better normalized quality and less normalized time.)



a) Results comparing the Metric-FF, YAHSP, LPG-TD, and SGPlan$_g$ planners on the Airport domain.



b) Results comparing the Metric-FF, YAHSP, LPG-TD, and SGPlan$_g$ planners on the Satellite domain.



c) Results comparing the Metric-FF, YAHSP, LPG-TD, and SGPlan$_g$ planners on the Logistics domain.

Figure 5. Quality-time distributions of the performance of the incremental version of a planner that evaluates two alternative subgoal orders with respect to that of the non-incremental version on three domains. (The upper right quadrant in each plot represents better normalized quality and less normalized time.)