Matching Problem Features with Task Selection for Better Performance in HTN Planning

Reiko Tsuneto, James Hendler and **Dana Nau**
Department of Computer Science
and Institute for Systems Research
University of Maryland
College Park, MD 20742 USA
{reiko, hendler, nau}@cs.umd.edu

Leliane Nunes de Barros
Laboratory of Integrated System
University of São Paulo
Av. Luciano Gualberto 158 trav. 3
05508-900 São Paulo Brazil
leliane@lsi.usp.br

Abstract

During the planning process, a planner may often have many different options for what kind of plan refinement to perform next (for example, what task or goal to work on next, what operator or method to use to achieve the task or goal, or how to resolve a conflict or enforce some constraint in the plan). The planner's efficiency depends greatly on how well it chooses among these options.

In this paper, we present and compare two types of strategies that an HTN planner may use to select which task to decompose next. Both strategies facilitate efficient planning by making it easier for the planner to identify plans that can be pruned from the search space—but since the strategies accomplish this in two different ways, each works better on different kinds of problems. We present experimental results showing how characteristics of the planning domain can be used to predict which strategy will work best, so that these domain characteristics can be used to select strategies across application domains when building practical planning systems.

Introduction

One characteristic of partial-order planners—regardless of whether they use hierarchical task network (HTN) decomposition or STRIPS-style operators—is that they search a space in which the nodes are partially developed plans. The planner refines the plans into more and more specific plans, until either a completely developed solution is found or every plan is found incapable of solving the problem. During this process, a planner may often have many different options for what kind of refinement to perform next.

A planner that uses STRIPS-style operators may need to choose which unachieved goal to work on next, which operator to use to achieve a goal, or which technique to use to solve a conflict. An HTN planner usually has an even larger array of options: it may need to choose which unachieved task to work on next, which method to use to accomplish the task, or which constraints to impose on the plan. The planner's efficiency depends greatly on its plan refinement strategy, which is the way it goes about choosing among this options. In particular, a large amount of the cost of generating plans can be caused by making the wrong refinement choices and having to backtrack over them later (Tsuneto *et al.* 1996; Tsuneto, Hendler, & Nau 1997).

Many comparative analyses have been done for STRIPS-style planning (Minton, Bresina, & Drummond 1994; Kambhampati, Knoblock, & Yang 1995; Barret & Weld 1994) aimed at identifying what kinds of domain characteristics make different planning strategies perform best. For practical purposes, the results of such performance analyses can help a knowledge engineer to select a problem-solving method adequate to a given application. Such practical considerations are particularly important for HTN planning since it has been used in a number of practical planning systems (Aarup *et al.* 1994; Wilkins & Desimone 1994; Agosta 1995; Smith, Nau, & Throop 1996). Many heuristics have been developed for domain independent HTN planning systems, most notably O-Plan (Currie & Tate 1991) and SIPE-2 (Wilkins 1990) including some heuristics which are evaluated in this paper. However, little analysis has been done to explain the relationships between each heuristic's performance and domain characteristics, and to evaluate the performances based on them.

In this paper, we present two types of task selection strategies that facilitate efficient HTN planning by doing task selection in a way that makes it easier for the planner to identify and prune infeasible plans. One strategy, which we call the Left-to-Right selection strategy, makes use of the explicit step ordering information in order to select the tasks that come earlier in the plan. The second strategy, which we call the ExCon strategy, pre-processes the domain description to automatically extract a kind of constraint we call *external conditions*. During the planning, ExCon's first preference is for the tasks that can establish or threaten the current external condition.

Moreover, we present empirical results to confirm the following two hypotheses about how various features of planning domains relate to the applicability of the above two strategies:

- in problems where there are many ordering constraints among the goal tasks and their subtasks, the

Left-to-Right selection strategy performs better than the ExCon strategy (when Left-to-Right is not used as a tie-breaking strategy);

- the ExCon strategy performs better than the Left-to-Right strategy in problems where the goal tasks (and their subtasks) are highly interleaved.

Section 2 describes the two task selection strategies. Section 3 presents the results of our experiments, which were carried out using UMCP, a domain-independent HTN planning system (Erol 1995) created at the University of Maryland. Section presents conclusion and future work.

Task Selection Strategies

In this section, we describe the LtoR and ExCon task selection strategies. In these descriptions (and throughout the paper), our notation for domain and problem descriptions is that used in the UMCP planner.

The Left-to-Right strategy

In STRIPS-style planning, there are three ways to plan actions: plan forward, plan backward and a combination of both. Constructing plans starting from the initial state gives the planner the advantage of having more information about the world state the planner is dealing with and thus makes it easier to solve interactions between actions. Planning backward from the goals has the advantage of producing lower branching factors because there are usually fewer actions applicable to satisfy a goal than a state. Although both the approaches of forward planning and bi-directional planning have been used successfully by some planners (Fink & Veloso 1994; Blum & Furst 1997), the backward planning approach has been the most popular. For HTN planning, the backward planning approach does not have an obvious advantage since the planner constructs plans by decomposing tasks into subtasks by applying decomposition methods. On the other hand, the forward planning approach has an advantage similar to STRIPS-style planning. First, like STRIPS-style planning, the initial state is a complete description of the state. Since only actions can affect the world state, inserting actions starting from the initial state can provide more state information that is useful in reasoning about later actions. Furthermore, an HTN domain can contain explicit step orderings between subtasks, which make it easier for the planner to select earlier tasks.

In HTN planning, the Left-to-Right (LtoR) task selection strategy will decompose a non-primitive task only when there are no other non-primitive tasks ordered to come before it. As a tie-breaking rule to handle the case where more than one non-primitive task has only primitive tasks ordered before it, we use the *Fewest Predecessors* (FP) heuristic, which selects non-primitive tasks which have the least number of tasks ordered before them. This heuristic has the advantage that it does not have to check if the preceding tasks are primitive or not because a non-primitive task A has fewer tasks ordered before it than a non-primitive task B if A precedes B. So using the FP heuristic automatically implements LtoR and each computation takes polynomial time with respect to the number of tasks in the partial plan.

While LtoR has a similar advantage to forward planning in STRIPS-style planning, the LtoR strategy we present does not necessarily "plan forward" unless the goal tasks and their subtasks are totally ordered. For example, LtoR may select to decompose a non-primitive task A before it selects a non-primitive task B, yet the subtasks of A may be ordered after the subtasks of B as a result of satisfying some state constraints.

The ExCon strategy

The ExCon strategy makes use of a type of condition called an *external condition*. Here, we give a brief description of what external conditions are and how the planner can use them to select tasks. The formal definition of external conditions and the complete algorithm of the ExCon strategy will be presented in another paper (Tsuneto, Hendler, & Nau 1998).

External conditions can be described intuitively as follows. Suppose that to accomplish some task in a plan P, we decide to use some method M. Furthermore, suppose that there is some condition c that must be satisfied in order for M to be successful, but that there is no way to decompose M into a sub-plan that achieves c. In this case, the plan P cannot be successful unless the condition c is somehow achieved elsewhere in P. Thus, we say that the condition c is external to the method M.

For example, suppose you want to eat breakfast. Your typical breakfast is either pancakes made from pancake mix or cereal. We can encode this situation as an HTN planning problem in which there is a task called Eat-Breakfast-Task that has two decomposition methods: one to eat pancakes (as shown in Figure 1) and one to eat cereal. As shown in Figure 1, the pancake method decomposes Eat-Breakfast-Task into three subtasks: Prepare-Table, Cook-Pancake-Task, and Eat. Let us assume that the methods for these tasks involve (1) putting the syrup, fork and knife on the table, (2) cooking the pancakes, and (3) serving and eating the pancakes, respectively.

Consider which conditions of the pancake method are external conditions. The method has four state constraints: the constraints (initially (*Egg* ?egg)) and (initially (*Milk* ?milk)) are static state constraints and not considered for external conditions; the constraint (between (*Have* pancake-mix) n0 n1) is an external conditions because the task of preparing the table (i.e. n0: (Prepare-Table)) does not make (*Have* pancake-mix) true and the other two tasks are ordered after n0 where the condition must be established; the constraint (before (*Hot* ?pc) n2) is not an external condition because the condition "the pancake ?pc is hot" can be caused by task n1. Thus, the constraint (between (*Have* pancake-mix) n0 n1) is the only external condition is this method.

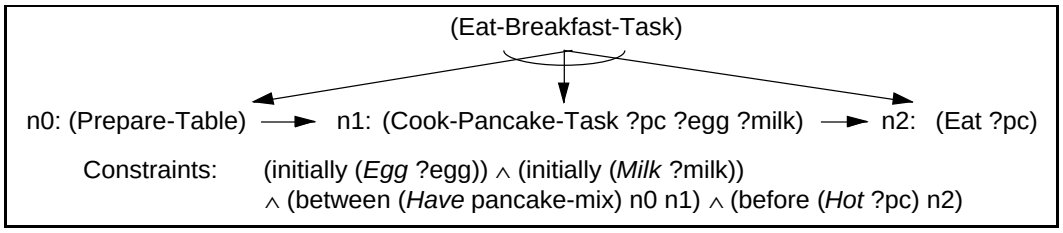


Figure 1: The pancake method for (Eat-Breakfast-Task). The downward-pointing arrows represent task decomposition, and the right-pointing arrows represent precedence. The tasks in the decomposition are label as n0, n1, and n2 so that they can be referred to in the constraint formula.

External conditions are somewhat similar to unsupervised conditions used in O-Plan2, although there are several differences in their definitions.[1] First, unlike external conditions, unsupervised conditions may also specify a condition that can be established by a subtask in the decomposition method. Second, the types of conditions are explicitly specified in O-Plan2 to give the domain writer more power to control the search. So unsupervised conditions are explicitly specified while external conditions can be automatically extracted. Moreover, conditions not specified as unsupervised conditions may be regarded as external conditions. For example, only_use_if conditions in O-Plan2 are conditions that are used to filter out inapplicable decomposition methods. If the conditions are for non-static state conditions (i.e. the conditions that may change as results of other actions), then they are considered external conditions by our definition. Similarly, some only_use_for_query conditions may be considered external conditions. For a summary and comparisons of condition types used in O-Plan2, Nonlin and SIPE-2, see (Tate, Drabble, & Dalton 1994).

After some method M is instantiated, every external condition of M becomes one of the applicability conditions in the partial plan: the conditions that must be established if the plan is to be successful. The ExCon strategy makes the planner work on the tasks that may establish or threaten an applicability condition of the partial plan first. Task selection using external conditions requires three steps: (1) Pre-process the planner's knowledge base to automatically extract the external conditions for each decomposition method; (2) During task decomposition, put the external conditions of the decomposition method onto the stack; (3) When selecting a task to decompose, choose a task which may potentially establish or threaten the condition on top of the stack.

The ExCon strategy needs a heuristic for breaking ties. We have tried using two different heuristics for this purpose. One heuristic is the LtoR heuristic described earlier. The other is the "fewest alternatives first" (FAF) heuristic, which selects the task that has the smallest number of applicable methods. In the rest of this paper, we use the names ExCon-FAF and ExCon-LtoR to refer to ExCon with the FAF and LtoR tie-breaking strategies, respectively.

[1] Also, unsupervised conditions in Nonlin (Tate 1977) and external-condition goals in SIPE-2 (Wilkins 1990) are defined in a similar manner to unsupervised conditions in O-Plan2.

Experiments

Since both the LtoR and ExCon strategies merely specify the order in which a planner will prefer to decompose tasks, they have no effect on the planner's soundness and completeness. However, they do affect the planner's efficiency; and we hypothesize that they do so in the following ways:

- The FAF heuristic, which selects the task that has the fewest decomposition methods, has been shown to work well in a large variety of planning problems (Joslin & Pollack 1994; 1996; Tsuneto *et al.* 1996). However, one major deficiency of the FAF heuristic is that it ignores pruning. By choosing a plan element with a larger number of refinement options, the planner may be able to do more pruning later on.[2] Thus, we hypothesize that the ExCon-FAF strategy should outperform the FAF strategy, especially on problems where the goal tasks are highly interleaved.
- In planning domains in which there are many constraints on the ordering of the subtasks, the LtoR heuristic should be able to outperform the FAF heuristic by expanding the tasks in an order that facilitates the pruning of infeasible plans. Similarly, the ExCon-LtoR strategy should be able to outperform the ExCon-FAF strategy in such problem domains.

To test these hypotheses, we implemented four task selection strategies, FAF, LtoR,[3] ExCon-FAF, and

[2] In fact, O-Plan's refinement choice mechanism (Currie & Tate 1991) has been using a combination of the FAF heuristic called Branch-1 with a Branch-N heuristic that estimates how many possibilities lie down the refinement path if it is taken. This Branch-1/Branch-N heuristic itself is based on the Hayes-Roths' OPM planner (Hayes-Roth *et al.* 1979).

[3] In our implementation of FAF and LtoR, we used the FP heuristic as the tie-breaking rule for FAF and the FAF

Algorithm refine(*PartialPlan*)

1. *(Pruning)* If the constraint formula of *PartialPlan* is *False* then prune this plan by returning empty set.
2. *(Task decomposition)* Otherwise, if the constraint formula is *True* and there are non-primitive tasks in *PartialPlan*, then decompose a task and return the resulting partial plans.
3. *(Solution check)* If the constraint formula is *True* and there are no non-primitive tasks in *PartialPlan*, then satisfy auxiliary constraints and return the resulting plans as solutions.
4. *(Constraint enforcement)* If the constraint formula is neither *True* nor *False*, then satisfy constraints, simplify constraint formula, and propagate auxiliary constraints. Return the resulting partial plans.

Figure 2: The default refinement strategy in the UMCP planner.

ExCon-LtoR, using the UMCP planning system, a domain-independent HTN planning system (Erol 1995), and compared their performance in a number of different planning domains, including the Random Travel Planning domain, the Translog domain, the Blocks World domain and the Flat Tire domain. For our experiments, we ran UMCP 1.0 on Sun ULTRA workstations using Allegro Common Lisp 4.3. We incorporated each task selection strategy into UMCP's default commitment strategy, which is described below. The code for the UMCP planner is available at http://www.cs.umd.edu/projects/plus/umcp/manual/. The domains used in the experiments are available at http://www.cs.umd.edu/projects/plus/umcp/domains/.

The default refinement strategies in UMCP For our experiments, refinement choices other than task selection are made using UMCP's default refinement strategy. UMCP's default refinement strategy repeatedly (1) decomposes a non-primitive task, (2) enforces each of the newly inserted constraints (3) evaluates and simplifies the constraint formula, and (4) propagates the previously postponed constraints. Figure 2 shows the algorithm that UMCP uses to decide what refinement to do next. It takes a partial plan as an input and returns a set of partial plans as the result of the refinement performed.

If the constraint formula of the partial plan is *False*, then UMCP prunes the partial plan by returning an empty set in Step 1. When the constraint formula is *True*, UMCP decomposes a non-primitive task t in the partial plan at Step 2. Decomposing t involves, for each decomposition method M of t, (1) replacing t with the subtasks in M, and (2) replacing the current constraint formula C in the partial plan with the conjunction of C and the constraint formula in M. If t is a predicate task, t is also phantomized by creating a plan with the task t replaced with a do-nothing task and the constraint formula specifying the predicate is accomplished at the beginning of the do-nothing task. Step 3 checks if the partial plan is a solution plan or not. If there are no non-primitive tasks in the partial plan and the constraint formula is *True*, then UMCP satisfies the remaining auxiliary constraints by instantiating variables and ordering steps, and return the solution plans. If the constraint formula is neither *True* nor *False*, then UMCP enforces the constraints in the partial plan at Step 4. Enforcing constraints involves adding step ordering to the tasks, binding variables to a value, according to the constraint types. If it requires further task decomposition to fully enforce some constraint, the constraint will be put into the auxiliary constraint lists to be enforced later. If a constraint is not enforceable, then an empty plan is returned.

Plan selection Our experiments used best-first search for the Block-World problems and depth-first search for all the other problems. For best-first search, each time the planner wants to select a partial plan for refinement, it selects the one with the lowest value for the following quantity:

number of non-primitive tasks
+ number of tasks (both primitive and non-primitive)
+ number of ordering and variable constraints that are postponed.

This selection heuristic is based on the heuristic presented by Gerevini and Schubert (Gerevini & Schubert 1996) and seems to perform well on many problems.

Random Travel Planning Domain

In our description of the LtoR strategy earlier, we pointed out that the explicit step orderings given in the task descriptions make it easier for the planner to select the tasks in a left-to-right manner. To see how much this can help the performance of the planner, we created a small domain called Random Travel Planning where there is only one level of hierarchy (i.e. no non-primitive task is decomposed into another non-primitive task). In this domain, there are three types of goal tasks, Sightsee, Travel and Eat. Their decomposition methods are shown in Figure 3. The task Sightsee is to go sightseeing ourselves (Method 1 in the figure) or to join a tour bus (Method 2), depending on if we are tired or not. Going sightseeing makes us tired. Taking a flight, or eating a food makes us recover from tiredness. In other

heuristic as the tie-breaking rule for LtoR.

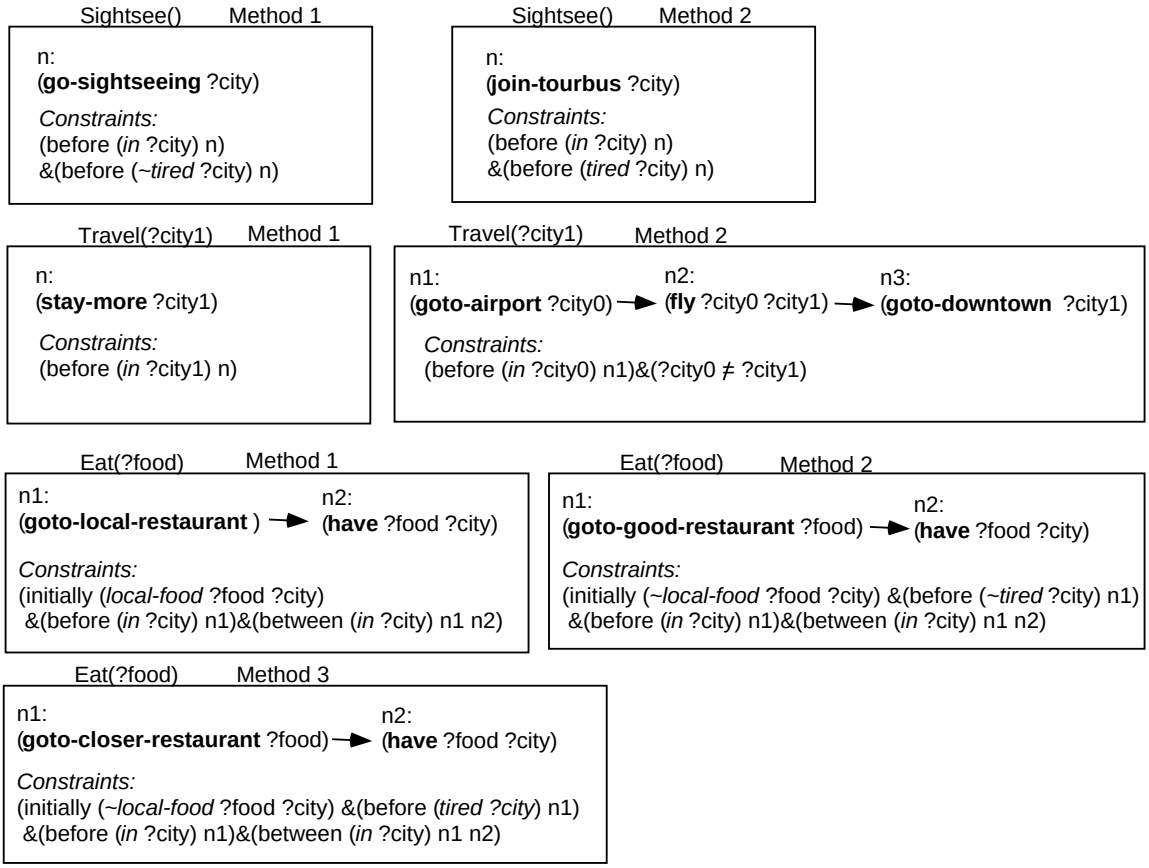


Figure 3: The decomposition methods for the Random Travel Planning domain. The tasks shown in boldface are primitive tasks.

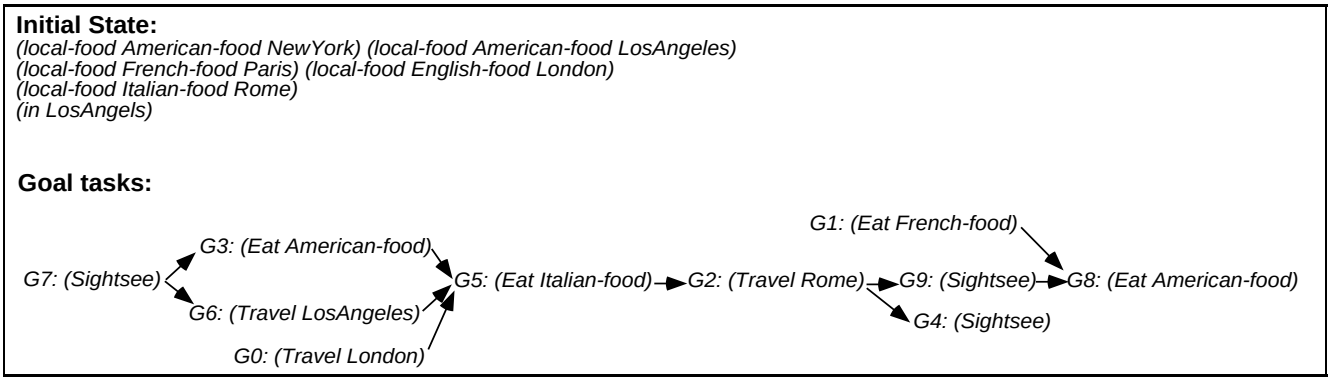


Figure 4: A sample problem of $\omega = 15$ (the actual number of unordered pairs of task is 14) in the Random Travel Planning domain.

words, the primitive task (go-sightseeing ?city) has an effect (*tired* ?city) and the primitive tasks (fly ?city0 ?city1) and (have ?food ?city0) have an effect (∼*tired* ?city0). The task (fly ?city0 ?city1) also has effects (∼*in* ?city0) and (*in* ?city1). Other primitive tasks have no effects. The task Travel is to move to another city, if it's different from the current location. The task Eat is to go to a restaurant for a type of food we want and eat there. If the type of food we want is local to the location, such as Italian food if the current location is Rome, then going to a local restaurant suffices (Method 1). If not, we go to a good restaurant if we are not tired (Method 2), or we go to a closer restaurant if we are (Method 3). In these methods, every 'before' and 'between' state constraint is an external condition.

A problem in this domain consists of 10 goal tasks, randomly generated. A goal task is either (Sightsee), (Travel ?city), or (Eat ?food). where ?city is a value randomly chosen from {LosAngeles, NewYork, London, Paris, Rome}, and ?food is a value randomly chosen from {American-food, English-food, French-food, Italian-food}. Since the subtasks in each method are totally ordered, how the problems are ordered depends on the step orderings between the goal tasks. If the goal tasks are totally ordered, then every partial plan generated from the goal also has the tasks totally ordered. The step orderings between the goal tasks are randomly generated based on the parameter ω, which defines the maximum number of pairs of goal tasks that can be left unordered. Lower values of ω indicate that there are more ordering constraints among the goal tasks. The initial state consists of the food-city pairs for each city such as (*local-food* Italian-food Rome) and the current location, i.e. (*in* ?city), which is randomly assigned. A sample problem of $\omega = 15$ is shown in Figure 4

We created 20 problems each for $\omega = 5$, 10, 15, 20 or 25 and solved them using FAF, LtoR, ExCon-FAF and ExCon-LtoR strategies. The results are shown in the Table 1and Figure 5. For low ω values, LtoR does better than FAF because LtoR can use the step orderings to correctly choose the earliest tasks. Many applicability conditions considered by ExCon-LtoR can be easily established at the time the conditions are inserted into the plan by using LtoR selection. Also, there are fewer non-primitive tasks that may affect the establishment of the current applicability condition, so the performance of ExCon-LtoR is similar to that of LtoR.

The performances of FAF and ExCon-FAF are also similar for the low ω value problems. Since FAF uses the LtoR heuristic for tie-breaking, FAF picks up the tasks relatively from left-to-right, although it skips Eat in preference to Sightsee or Travel. So, similarly to ExCon-LtoR, only a few possible non-primitive tasks exist that may affect the establishment of the current applicability condition. Since the performance of the ExCon strategy greatly depends on its tie-breaking strategy for low ω value problems (i.e. problems where goal tasks are not interleaved much), ExCon-LtoR does better than ExCon-FAF.

For the high ω value problems, there are not very many ordering constraints among the goal tasks, so there can be many more interactions among goal tasks. In these problems, the performance of LtoR is worse than any other strategy because LtoR does not have enough step ordering information to correctly work in a left-to-right manner and thus not be able to find constraints that can never be established early. FAF performs better than LtoR, but not as well as ExCon-FAF or ExCon-LtoR. The performances of ExCon-LtoR and ExCon-FAF are similar because for the problem with highly interleaved goal tasks, ExCon selects tasks and does not have to use its tie-breaking strategy (i.e. LtoR or FAF).

Translog Domain

Translog (Andrews *et al.* 1995) is a transport logistics domain where the methods of transportation are specified based on the locations, the types of packages, and availabilities of the necessary equipment. It is a considerably larger domain than many used in planning. It is specified with 17 compound tasks, 42 primitive actions, and 29 predicates. We have randomly chosen 10 problems from the one-package problems created by Kettler (Kettler 1995) for this experiment. Table 2 lists the results. For one-package (i.e. one goal task) problems, there are many ordering constraints among the subtasks. So the relative performances of the task selection strategies are similar to the low ω values problems in the Random Travel Planning domain.

Blocks World and Fixit Problems

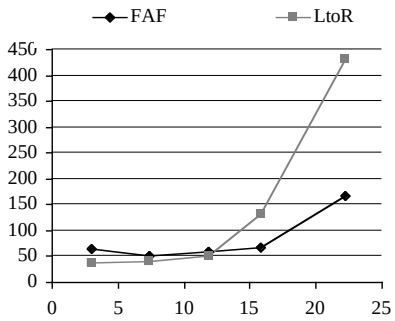
For the next set of experiments, the problems Sussman's anomaly, tower invert3 and tower invert 4 in the Blocks World domain and the fixit problem in Russel's Flat Tire domain were tested. For these two domains, there are not very many ordering constraints among tasks as one-package Translog problems. The results are shown in Table 3.

The non-primitive tasks in the Flat Tire domain have either one or two decomposition methods. So, the FAF heuristic is not very decisive. The LtoR strategy does not work quite as well because there are not very many ordering constraints among the tasks in the domain. ExCon-FAF creates more partial plans than FAF does. ExCon may not do well if search branches fail for reasons such as the failure of variable bindings, or state constraints except external conditions.

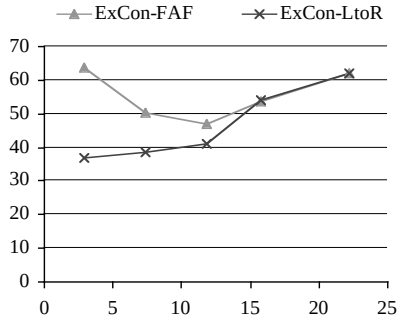
For Blocks World problems, the performances of FAF and LtoR are the same. There are no explicit step orderings specified between non-primitive subtasks, so LtoR always uses the tie-breaker FAF to select a task. However, there are only two non-primitive tasks on and clear in this domain. The FAF heuristic prefers the task on to clear, but does not decide on which one of the on tasks or clear tasks, so the task selection returns the one that happens to be found in the partial plan first. Thus the performances of FAF and LtoR depend on what order the tasks are specified in the problem. For

ω	Actual	FAF		LtoR		ExCon-FAF		ExCon-LtoR	
		Plans	Time	Plans	Time	Plans	Time	Plans	Time
5	2.95	63.55	0.32	36.95	0.16	63.60	0.36	36.90	0.17
10	7.35	51.25	0.27	41.00	0.21	50.45	0.29	38.40	0.19
15	11.85	58.55	0.32	50.80	0.26	46.90	0.28	41.05	0.22
20	15.8	66.65	0.47	131.85	1.16	53.75	0.33	54.15	0.34
25	22.2	168.10	2.57	431.55	5.54	62.15	0.44	62.00	0.41

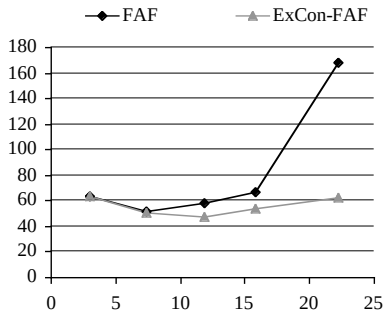
Table 1: The results of the Random Travel Planning problems. The actual average number of unordered pairs is shown next to ω values. "Plans" is the average number of partial plans created, and "Time" is the average non-garbage collection CPU time in seconds.



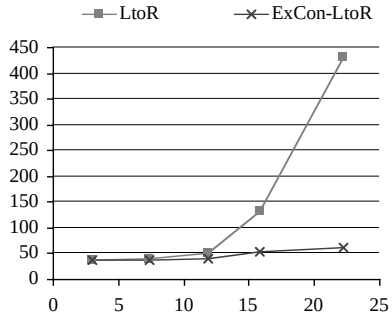
(a) FAF and LtoR



(b) ExCon-FAF and ExCon-LtoR



(c) FAF and ExCon-FAF



(d) LtoR and ExCon-LtoR

Figure 5: The results of the Random Travel Planning problems. The x-axis shows the average number of pairs of unordered goal tasks and the y-axis shows the average number of partial plans created.

Problems	FAF		LtoR		ExCon-FAF		ExCon-LtoR	
	Plans	Time	Plans	Time	Plans	Time	Plans	Time
1	218	5.51	217	5.45	219	5.32	217	4.81
2	87	1.04	87	1.02	87	0.91	87	0.91
3	303	9.97	302	9.05	309	11.59	302	8.13
4	442	16.59	243	6.56	258	7.38	243	6.18
5	77	0.79	76	0.76	77	0.81	76	0.79
6	80	1.04	78	1.03	80	1.04	78	1.04
7	77	0.79	76	0.79	77	0.77	76	0.78
8	80	1.07	78	1.03	80	1.05	78	1.09
9	80	1.03	78	1.03	80	1.03	78	1.02
10	287	7.79	331	8.47	529	10.50	331	8.45
average	173.1	4.56	156.5	3.52	179.6	4.04	156.5	3.32

Table 2: The results of the one-package Translog problems. Plans are number of partial plans created. Time is non-garbage collection CPU time in seconds.

Problems	FAF		LtoR		ExCon-FAF		ExCon-LtoR	
	Plans	Time	Plans	Time	Plans	Time	Plans	Time
sussman-anomaly	36	0.15	36	0.14	33	0.13	33	0.14
tower-invert3	45	0.22	45	0.21	49	0.28	49	0.28
tower-invert4	229	6.41	229	6.43	100	1.32	100	1.34
fixit	160	5.93	159	2.54	236	5.28	137	1.78

Table 3: The results on the Blocks World problems and fixit problem. Plans are number of partial plans created. Time is non-garbage collection CPU time in seconds.

Problems	FAF		LtoR		ExCon-FAF		ExCon-LtoR	
	Plans	Time	Plans	Time	Plans	Time	Plans	Time
sussman-anomaly	71	0.53	71	0.50	50	0.52	50	0.51
tower-invert3	80	0.76	80	0.72	36	0.22	36	0.23
tower-invert4	4983	221.72	4983	216.14	107	1.41	107	1.45

Table 4: The results on the Blocks World problems where the goal tasks are specified in the reversed order.

instance, the goals of the tower-invert4 problem used for the above experiments are ordered as (n1 on B C) (n2 on C D) (n3 on D A). Thus, the goal (n1 on B C) is decomposed first and then later the goal (n2 on C D) is decomposed before the decomposition of the goal task (n3 on D A). Table 4 lists the results of the same Blocks World problems, where the goals are specified in reverse order. The results show that the performances of FAF and LtoR are quite different from the results shown in Table 3, while the performances of the ExCon strategies are very similar regardless of the goal specification.

Conclusion

Many search techniques have been presented to increase the efficiency of planning. However, many of them lack clear explanations of how they are better than other techniques, what types of planning domains they work well with, and how much one technique may perform better than another in specific problems. The lack of study is especially significant in HTN planning, which many practical applications employ. Systematic studies of various search techniques are necessary. This includes identifying domain characteristics, and evaluating and comparing various techniques empirically and/or theoretically. Not only can the results of such performance analyses help further improve the search techniques, but they can also help a knowledge engineer select a problem-solving method adequate to a given application.

We have presented two types of strategies in HTN planning that select which non-primitive task to decompose. Each one is focused on different HTN problem domain characteristics in order to use domain information for pruning. The LtoR strategy selects a task that has no non-primitive tasks ordered before it and tries to develop a detailed plan starting from the initial state. It is easier for the planner to identify the unsatisfiable conditions associated with a task if the planner knows more about what primitive actions come before the task. Thus, the LtoR strategy performs well for problems where there are many ordering constraints between the goal tasks and their subtasks, because the ordering constraints help the planner decide which task comes earlier.

The ExCon strategy makes use of external conditions. The planner pre-processes its knowledge base to extract external conditions for each decomposition method. During task decomposition, the external conditions of the method used are placed onto the partial plan as its applicability conditions. When selecting a task to decompose, ExCon selects a task that may be used to satisfy an applicability condition of the partial plan. ExCon performs well on the problems where the goal tasks are highly interleaved.

Our empirical studies show that LtoR consistently outperforms other strategies on problems with many ordering constraints. The results also show that ExCon does well on the problems where there are fewer ordering constraints and the goal tasks are highly interleaved. Moreover, ExCon-LtoR, the ExCon strategy combined with LtoR, does better than FAF or ExCon-FAF on problems with many ordering constraints because LtoR is used for tie-breaking. It does better than FAF or LtoR on problems with fewer ordering constraints because ExCon works well for interleaving tasks.

Since we have only run our experiments on a relatively small sample of problems the results presented in this paper are preliminary. We are currently working to fully analyze LtoR and ExCon both theoretically and empirically.

Acknowledgment

This research was supported in part by grants from ONR (N00014-J-91-1451), ARPA (N00014-94-1090, DABT-95-C0037, F30602-93-C-0039), ARL (DAAH049610297) and NSF (NSF EEC 94-02384, IRI-9306580).

References

Aarup, M.; Arentoft, M. M.; Parrod, Y.; Stader, J.; and Stokes, I. 1994. OPTIMUM-AIV: A knowledge-based planning and scheduling system for spacecraft AIV. *Intelligent Scheduling* 451–469.

Agosta, J. M. 1995. Formulation and implementation of an equipment configuration problem with the SIPE-2 generative planner. In *Proceedings of AAAI-95 Spring Symposium on Integrated Planning Applications*, 1–10.

Andrews, S.; Kettler, B.; Erol, K.; and Hendler, J. 1995. UM translog: A planning domain for the development and benchmarking of planning systems. Technical report, Dept. of Computer Science, University of Maryland, College Park, MD.

Barret, A., and Weld, D. S. 1994. Partial-order planning: Evaluating possible efficiency gains. *Artificial Intelligence 67(1)* 71–112.

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

Currie, K., and Tate, A. 1991. O-plan: the open planning architecture. *Artificial Intelligence 52* 49–86.

Erol, K. 1995. HTN planning: Formalization, analysis, and implementation. Ph.D. dissertation, Computer Science Dept., University of Maryland.

Fink, E., and Veloso, M. 1994. Prodigy planning algorithm. Technical report cmu-cs-94-123, Carnegie Mellon University, Pittsburgh, PA.

Gerevini, A., and Schubert, L. 1996. Accelerating partial-order planners: Some techniques for effective search control and pruning. *Journal of Artificial Intelligence Research 5* 95–137.

Hayes-Roth, B.; Hayes-Roth, F.; Rosenschein, S.; and Cammarata, S. 1979. Modeling planning as an incremental, opportunistic process. In *The Proceedings of Sixth International Joint Conference on Artificial Intelligence*, 375–383.

Joslin, D., and Pollack, M. E. 1994. Least-cost flaw repair: A plan refinement strategy for partial-order planning. In *The Proceedings of 12th National Conference on Artificial Intelligence*, 1004–1009.

Joslin, D., and Pollack, M. E. 1996. Is "Early Commitment" in plan generation ever a good idea? *Proceedings of AAAI-96* 1188–1193.

Kambhampati, S.; Knoblock, C. A.; and Yang, Q. 1995. Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence 76* 167–238.

Kettler, B. 1995. Case-based planning with high-performance parallel memory. Ph.D. dissertation, Computer Science Dept., University of Maryland.

Minton, S.; Bresina, J.; and Drummond, M. 1994. Total-order and partial-order planning: A comparative analysis. *JAIR 2* 227–262.

Smith, S. J.; Nau, D.; and Throop, T. 1996. Total-order multi-agent task-network planning for contract bridge. In *AAAI-96*, 108–113.

Tate, A.; Drabble, B.; and Dalton, J. 1994. The use of condition types to restrict search in an AI planner. In *The proceedings of Twelfth National Conference on Artificial Intelligence*, 1129–1134.

Tate, A. 1977. Generating project networks. In *The Proceedings of Fifth International Joint Conference on Artificial Intelligence*, 888–893.

Tsuneto, R.; Erol, K.; Hendler, J.; and Nau, D. 1996. Commitment strategies in hierarchical task-network planning. In *The Proceedings of the Thirteenth National Conference on Artificial Intelligence(AAAI-96)*, 536–542.

Tsuneto, R.; Hendler, J.; and Nau, D. 1997. Plan-refinement strategies and search-space size. In *The Proceedings of the Fourth European Conference on Planning(ECP-97)*, 416–428.

Tsuneto, R.; Hendler, J.; and Nau, D. 1998. Analyzing external conditions to improve the efficiency of htn planning. In *The Proceedings of the Fifteenth National Conference on Artificial Intelligence(AAAI-98)*.

Wilkins, D. E., and Desimone, R. V. 1994. Applying an AI planner to military operations planning. *Intelligent Scheduling* 685–709.

Wilkins, D. E. 1990. Can AI planners solve practical problems? *Computational Intelligence* 6(4):232–246.