Analyzing External Conditions to Improve the Efficiency of HTN Planning

Reiko Tsuneto, James Hendler and Dana Nau

Department of Computer Science
and Institute for Systems Research
University of Maryland, College Park, MD 20705
{reiko, hendler, nau}@cs.umd.edu

Abstract

One difficulty with existing theoretical work on HTN planning is that it does not address some of the planning constructs that are commonly used in HTN planners for practical applications. Although such constructs can make it difficult to ensure the soundness and completeness of HTN planning, they are important because they can greatly improve the efficiency of planning in practice. In this paper, we describe a way to achieve some of the advantages of such constructs while preserving soundness and completeness, through the use of what we will call *external conditions*. We describe how to detect some kinds of external conditions automatically by preprocessing the planner's knowledge base, and how to use this knowledge to improve the efficiency of the planner's refinement strategy. We present experimental results showing that by making use of external conditions as described here, an HTN planner can be significantly more efficient and scale better to large problems.

Introduction

Problem Description

Applied work in AI planning has typically favored approaches based on hierarchical decomposition rather than causal chaining. In particular, most successful planners for practical applications have used *hierarchical task network* (HTN) planning (Sacerdoti 1977; Tate 1977; Currie and Tate 1991; Wilkins 1990), an AI planning methodology that creates plans by *task decomposition*. This is a process in which the planning system decomposes tasks into smaller and smaller subtasks, until primitive tasks are found that can be performed directly. HTN planning systems have knowledge bases containing *methods* (also called *schemas* by some researchers). Each method includes (1) a prescription for how to decompose some task into a set of subtasks, (2) various restrictions that must be satisfied in order for the method to be applicable, and (3) various constraints on the subtasks and the relationships among them. Given a task to accomplish, the planner chooses an applicable method, instantiates it to decompose the task into subtasks, and then chooses and instantiates other methods to decompose the subtasks even further. If the constraints on the subtasks or the interactions among them prevent the plan from being feasible, the planner will backtrack and try other methods.

Although HTN-style planners have been the ones most used in practical applications (Aarup *et al.* 1994; Wilkins and Desimone 1994; Agosta 1995; Smith *et al.* 1996; Estlin *et al.*[1] 1997), most theoretical work to date is in the area of partial-order refinement planners such as UCPOP (Penberthy and Weld 1992) and more recently in graph- and circuit-based planners (Blum and Furst 1997; Kautz and Selman 1992).

One exception has been theoretical work on HTN planning at the University of Maryland. This work has shown HTN planning to be strictly more expressive than planning with STRIPS-style operators (Erol *et al.* 1994b), has established the soundness and completeness of HTN planning algorithms (Erol *et al.* 1994a), and has explored the complexity of HTN planning problems (Erol *et al.* 1996) and the efficiency of various search strategies (Tsuneto *et al.* 1996; Tsuneto *et al*. 1997). The code for the UMCP domain-independent planner (which we used for the experiments reported in this paper) is available at <http://www.cs.umd.edu/projects/plus/umcp>.

Unfortunately, the above work did not incorporate several of the planning constructs used in previous HTN planning systems, such as unsupervised conditions and high-level effects (Tate 1977). Such constructs are often used in practical applications of HTN planning because they can be very useful for improving the efficiency of planning—but the UMCP planner does not incorporate them because they can make it difficult to guarantee the planner's soundness and completeness.

As an example, consider high-level effects. Some HTN-style planners allow the user to state in the planner's knowledge base that various non-primitive tasks will

[1] Their DPLAN and MVP planners are not completely HTN planners, but they employ a combination of HTN planning and operator-based planning.

achieve various effects. Such planners can use these high-level effects to establish applicability conditions, and can prune the partial plan if a high-level effect threatens an applicability condition. This can interfere with the planner's soundness. For example, if a non-primitive task *t* has a high-level effect *o*, but one of the sub-plans that *t* can be decomposed into does not have the effect *o*, then this may cause the planner to produce an inconsistent plan. Also, even when there is a high-level effect to establish a condition, there may be a threat to that establishment that can only be found by looking at primitive tasks. This problem can be overcome, but only by imposing restrictions on how high-level effects are used (Bacchus and Yang 1991, Young *et al.* 1994).

Despite these problems, the ability to recognize important effects early is critical to the good performance of a planner. By examining those actions most likely to effect others, pruning of the search space can occur and much backtracking can be avoided. Ways are needed to provide the sort of information needed for this pruning without sacrificing soundness or completeness.

Approach

In this paper, we describe one way to address the problem discussed above, through the use of what we call *external conditions*. We also show that if an HTN planner recognizes and makes use of external conditions in an appropriate fashion, this can greatly improve the efficiency of HTN planning, while preserving soundness and completeness.

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*. Then the plan *P* may still be successful, but only if some other portion of *P* achieves *C*. In this case, we say that the condition *C* is *external* to the method *M*.

External conditions do not really have a good analog in STRIPS-style planning, but they are somewhat analogous to the unsupervised conditions used in the Nonlin planner (Tate 1977) and external-condition goals used in the SIPE-2 planner (Wilkins 1990). The main difference is that instead of being specified explicitly as unsupervised conditions or external-condition goals are, external conditions occur as a result of the structure of the planner's knowledge base, and can be detected by examining the knowledge base. We also believe that external conditions may be a good way to capture many of the benefits found in some planners via the use of high-level effects.

Background

How Constraints Are Handled in HTN Planning

Just as a STRIPS-style planning operator specifies preconditions that must be satisfied before the operator can be executed, an HTN method *M* may include specifications of several different kinds of conditions that must be satisfied in order for *M* to be used successfully. We describe the most important ones here, using the notation that is used for them in the UMCP HTN planner.

- A *predicate task*, which is sometimes called a GOAL node (Tate 1977) or an achievement task, is analogous to a precondition in a STRIPS-style planning operator: it is a condition that must be true at the time that the method *M* begins executing.
- An *initial state constraint* specifies a condition that must be true in the initial state in order for *M* to be used. For example, if *M* contains the constraint (initially (~*Q* ?x)), this constraint can be satisfied only if it is possible to bind the variable ?x to some value C such that (*Q* C) is false in the initial state. Normally, one would use an initial state constraint only to refer to a condition that will never change throughout the plan.
- A *'before' state constraint* such as (before (*P* ?x) n) specifies a condition that must be true just before some subtask n of the method *M*. This state constraint can be *established* by a task T if T has the effect (*P* ?y), T precedes the task n in the plan, and the variables ?x and ?y are instantiated to the same value. Similarly, (before (*P* ?x) n) can be *threatened* by a task T if T has the effect (~*P* ?z), T precedes the task n, no establisher of the constraint occurs after T and before n, and the variables ?x and ?z are instantiated to the same value.
- A *'between' state constraint* specifies a condition that needs to be true during a specific time interval. For example, the 'between' state constraint (between (*P* ?x) n1 n2) is satisfied if the condition (*P* ?x) is true from the end of the task n1 to the beginning of the task n2. A 'between' state constraint can be established or threatened in a manner similar to a 'before' constraint.

Task Selection and FAF

One characteristic of partial-order planners—regardless of whether they use 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 (promotion, demotion, or variable separation) to resolve a goal 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 constraint (from among a number of different possibilities) 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 these options.

Of the various plan refinement strategies that have been explored by AI planning researchers, one of the best that has been found so far is a strategy that we call the "fewest alternatives first" (FAF) strategy, which chooses the refinement that has the fewest immediate child nodes in the search. The first use of FAF by planning researchers seems to have occurred relatively recently (Currie and Tate 1991), but a similar heuristic has been known in the constraint satisfaction literature for more than 20 years (Bitner and Reingold 1975; Purdom 1983). In recent experimental studies comparing various versions of FAF with other well known plan refinement strategies (Joslin and Pollack 1994; Tsuneto *et al.* 1996), FAF outperformed the other refinement strategies on many planning problems and planning domains. Furthermore, theoretical analyses of FAF (Tsuneto *et al.* 1997) have shown that FAF can produce exponential savings in the size of the planner's search space. Since FAF is a general search heuristic,. however, it does not always do as well as other strategies that use domain-specific planning knowledge.

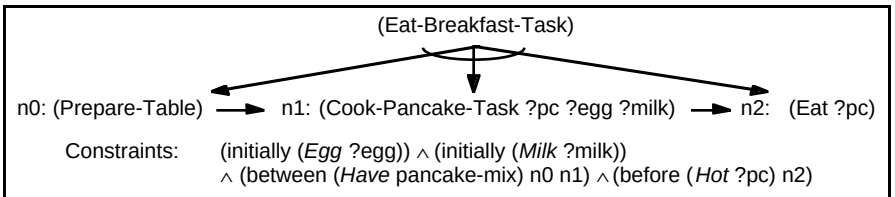

Figure 2: The pancake method for (Eat-Breakfast-Task). The downward-pointing arrows represent task decomposition, and the right-pointing arrows represent precedence. The tasks in the decomposition are labeled as n0, n1, and n2 so that they can be referred to in the constraint formula.[3]

When there are two or more goal tasks in a problem, the tasks can often be interleaved; i.e., an effect of one goal task can be used to satisfy the state constraints of another goal task. By interleaving goal tasks, the planner can create plans that have fewer redundant actions—and in some cases, interleaving may be the only way to generate a plan at all.

While interleaving tasks is essential in HTN planning, it is often hard for the planner to know which tasks can be interleaved successfully. The planner does not know exactly what effects a non-primitive task has until the task is decomposed into primitive tasks; and the decomposition process can introduce more state constraints to the plan that may not already be established. The failure to interleave tasks successfully often leads to a large backtracking cost.

Although FAF does generally better than other strategies, its performance is still ragged for many multiple-goal problems. If the planner is trying to establish more than one constraint, often it will do so by trying to interleave tasks; and if it tries to interleave several tasks at once, then the number of possible ways to interleave them can be combinatorially large. Thus, if the interleaving doesn't work, the planner can waste large amounts of time exploring a large search space. In order to avoid such difficulties, the planner needs to know which state constraints should be established early by interleaving tasks, and the task selection should use such knowledge to arrange the search.

External Conditions

An external condition of a method is a state constraint that cannot be established by any task that may result from the method. Thus, if the method is to be used successfully in a plan, the plan must establish this state constraint by something *external* to the method (such as the initial state or some other task in the planning problem).

More formally, let $M = \langle T, \phi \rangle$ be a method, where T is the set of subtasks created by the method and ϕ is the set of constraints for the method. Then a condition c is an *external condition* of M if:

1. c is a state constraint but not an initial state constraint;
2. c must be necessarily true to satisfy ϕ; and
3. no primitive descendants[2] of tasks in T can establish c.

As an example, suppose you want to eat a breakfast of 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 2) and one to eat cereal. As shown in Figure 2, 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.

The pancake method has four state constraints. The following analysis shows that the only external condition for this method is (between (*Have* pancake-mix) n0 n1):

- (initially (*Egg* ?egg)) and (initially (*Milk* ?milk)) are not external conditions, because they are initial state constraints.
- (between (*Have* pancake-mix) n0 n1) is an external condition, because preparing the table does not cause a pancake mix to be there and no subtasks can occur before n0.

---

[2] A descendent of a task t is a task that appears as a result of recursively decomposing t.

[3] To distinguish between task names and predicate names (both here and throughout this paper), we put the latter in italics.

- (before (*Hot* ?pc) n2) is not an external condition, because the condition "the pancake ?pc is hot" is caused by the task n1.

The ExCon Strategy

After some method *M* is instantiated as part of some partial plan, every external condition of *M* becomes one of the *applicability conditions* in the partial plan; i.e., it is a condition that must be established *somewhere* in any completion of the partial plan if the completion is to be successful for the current problem. If the applicability condition cannot be satisfied, the planner may not be able to tell this until long after it has instantiated the method *M*, in which case the planner will incur large backtracking costs. In many situations, the planner has to backtrack over more than one applicability condition, which multiplies the backtracking costs.

Consider the breakfast example again. Suppose we have a planning problem with two goal tasks, Shopping-Task and Eat-Breakfast-Task (see Figure 3). Suppose that the initial state does not contain (Have pancake-mix), but that depending on what is on sale at the store, one possible outcome of Shopping-Task is to buy a pancake mix. Given the goal, a task-selection strategy such as FAF may choose Eat-Breakfast to work on first. This will generate two partial plans, one using the pancake method (Plan1 in the figure) and the other using the cereal method. The planner has to choose a non-primitive task such as Cook-Pancake-Task or Shopping-Task in Plan1 to work on next.

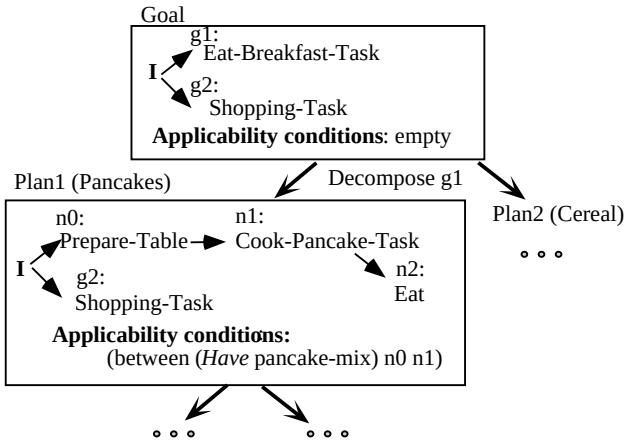

Figure 3: A part of the search tree for the breakfast problem. 'I' represents the initial state.

If sometime later the planner finds that the Shopping-Task does not actually buy a pancake mix, then the entire search branch derived from Plan1 fails. The cost of such backtracking can be significantly large if the planner worked on the task Cook-Pancake-Task down to the detailed level and then finds the failure while working on the Shopping-Task.

The idea of the ExCon strategy is to shift the attention of the planner to concentrate on establishing applicability conditions in the partial plan. This requires the following:

1. When it loads its knowledge base, the planner must pre-compute external conditions for every decomposition method in the domain and store the information.
2. The data structure of a partial plan keeps the stack of applicability conditions. Initially, the partial plan has no applicability conditions. During a method instantiation, the external conditions of the method are pushed onto the top of this stack. The condition on top is the current priority to the planner.
3. When selecting a task to decompose, the priorities are given to (1) tasks which can possibly establish the current top condition, or (2) tasks which can possibly threaten the current top condition, based on the presence of a primitive establisher.

The algorithm for the third step (selecting a task to decompose) is shown in Figure 4. For selecting tasks in Steps 1, 4, and 5, the algorithm uses whatever task-selection strategy the user wishes (we use FAF for this purpose in the experiments described in the next section). We now describe the details of the algorithm.

Algorithm select-task-ExCon(*PartialPlan*)

1. If the applicability condition stack of *PartialPlan* is empty, then select a task from *PartialPlan* and return the result.
2. Else, set *c* to the first element of the applicability condition stack in *PartialPlan*.
3. If *c* is true in *PartialPlan*, then remove *c* from the applicability condition stack and go back to Step 1.
4. If there is no primitive task that establishes *c*, then compute possible establishers for *c*. Select a task among them and return the result.
5. Else, compute possible threats for *c*. Select a task among them and return the result.
6. If there are no possible establishers or possible threats, remove *c* from the applicability condition stack of *PartialPlan* and go back to Step 1.

Figure 4: The task selection algorithm for ExCon.

In Step 1, if there are no applicability conditions to achieve, then the planner selects and returns a task. When there are applicability conditions, Step 2 picks the one on top of the stack. If the current condition is already established without threats in the partial plan, then Step 3 removes the condition from the stack and goes back to select something else. Otherwise, Step 4 computes the non-primitive tasks in the plan that can possibly establish the condition, provided the condition is not established by any primitive task currently in the plan. If it is established by a primitive task, then possible threats are computed and one is selected in Step 5. Otherwise, there are only primitive tasks that might affect this condition, so Step 6 will remove it from the stack and go back to select another one.

Note that since ExCon's task-selection strategy merely specifies the order in which a planner will prefer to expand tasks, it has no effect on the planner's soundness and completeness: a planner that is sound and complete without

it will also be sound and complete with it.

Experiments

We implemented FAF and ExCon in UMCP version 1.0, using Allegro Common Lisp version 4.3 on SUN workstations. We ran two sets of experiments: one on a small artificial domain and one on the Translog domain. We incorporated each task selection strategy into UMCP's default commitment strategy, and used depth-first search for all the experiments. Although we do not show CPU-time results for the experiments described below, the CPU times in our experiments were basically proportional to the node counts.

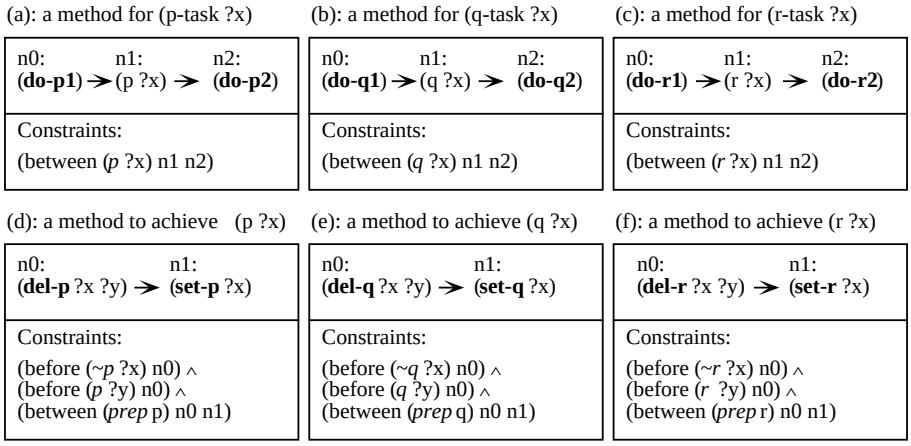

Figure 5: The decomposition methods for the test domain. Each non-primitive task has exactly one method specified. The tasks shown in boldface are primitive tasks.

Implementation of ExCon in UMCP

Automatically extracting external conditions. Computing precisely which state constraints are external conditions is not a trivial matter since it requires the planner to know the exact variable bindings that can occur during the planning. To see which tasks affect which constraints, UMCP uses a *possible-effects table* to store information about which non-primitive tasks are capable of causing various kinds of effects. Since the exact effect of each non-primitive task depends on which decomposition methods are used and how the variables are bound, the table only specifies which non-primitive task can possibly affect each predicate. The table is a table of pairs $\langle p, t \rangle$ where p is a positive or a negative predicate symbol and t is a non-primitive task. A pair $\langle p, t \rangle$ is in the table if one of the possible decompositions of the task t contains a primitive task s such that one of s's effects is the literal (*p arguments*).

The possible effects table can be constructed by a planner by preprocessing the domain. During the planning, the planner can look up the table to see which non-primitive task in the current partial plan can possibly establish or threaten certain constraints in order to prune partial plans that have no way of satisfying necessary state constraints.

To extract the external conditions of each method in the domain, we modified UMCP to look at each non-initial state constraint and find all the subtasks that are not explicitly ordered after the point where the condition needs to be true. If none of those subtasks can possibly establish the condition, the state constraint is marked as an external condition. This method will not necessarily extract all external conditions. We are currently trying to find a better way to extract them.

Computing possible establishers and possible threats. As shown in Figure 4, Steps 4 and 5 of ExCon's task-selection compute the possible establishers and the possible threats of the applicability condition. In our implementation, the planner uses the possible effects table to compute them. First the planner finds all non-primitive tasks in the partial plan that are not ordered after the point where the condition needs to be true. It then looks up the possible effects table to see if any of them can possibly establish or threaten the condition, and returns the result. Although this method returns (as possible establishers or possible threats) some tasks that can never establish nor threaten the condition, it finds every possible establisher and possible threat to the condition.

Experiments on an artificial domain

We hypothesized that in comparison with FAF, ExCon should perform best in problems that contain lots of interleaved tasks, and also in problems where there are many possible ways to interleave the same tasks. To test these hypotheses, we experimented with ExCon on the test domain described below.

The test domain contains methods for accomplishing compound tasks called p-task, q-task, and r-task. As shown in Figure 5, these methods decompose the compound tasks into other tasks. Most of the other tasks are primitive tasks, but a few of them (p, q, and r) are predicate tasks. Most of the primitive tasks (do-p1, do-p2, do-q1, do-q2, do-r1 and do-r2) have no preconditions and effects. Each predicate task has two methods that are capable of achieving it: one of the methods shown in Figure 5(d)–5(f), and a Do-Nothing method.[4]

[4] If a predicate task has already been established, it is said to be *phantomized* because the planner will not need or want to establish it yet again. There are several well known ways to handle phantomization. UMCP handles it by inserting a Do-Nothing method (basically a no-op) into the plan.

The primitive task (del-p ?x ?y) has the effects (~*p* ?y) and (*prep* p). The task (set-p ?x) has the effects (*p* ?x) and (~*prep* p). The predicate task (p W) for some value W can be achieved in two ways: by phantomizing it if the literal (*p* W) is true at the beginning of the task (p W); or by doing (del-p W Z) followed by (set-p W) if at the beginning of the task (p W), the literal (*p* W) is false and the literal (*p* Z) is true for some value Z. The tasks del-q and del-r are defined similarly to the task del-p, and the tasks set-q and set-r are defined similarly to the task set-p. An initial state for this domain consists of three ground atoms (*p* w_p) (*q* w_q) and (*r* w_r), where w_p, w_q and w_r are constant values randomly chosen from the set {C1, C2, C3, C4, C5, C6}.

In this test domain, the amount of interleaving can be altered by varying the arguments of the goal tasks: a problem is highly interleaved if the arguments of most p-task goal tasks are the same, and it is less interleaved if the arguments of most p-task goal tasks are different.

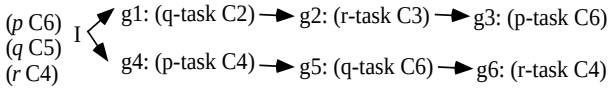

Figure 6: A sample problem with 2 goals, 3 predicates and 50% overlap. I represents the initial state which consists of (*p* C6), (*q* C4) and (*r* C4).

We generated test problems as follows. Goals were random sequences of one (p-task only), two (p-task and q-task) or three (p-task, q-task and r-task) different tasks that need to be done. How many different tasks were in the goals decided the number of predicates in the problems. A problem consisted of two or three goals, with no ordering constraints between them. We randomly assigned arguments to the tasks based on an "overlap rate" of 10%, 50% or 90%. For example, if the overlap rate were 100%, all the arguments of the p-task tasks would be identical to the *p* value in the initial state. If the overlap rate were 0%, the arguments for p-task's and the *p* value in the initial state would all be unique. If the overlap rate were 30%, there would be a 30% probability that the argument of a p-task is used in another p-task or the atom *p* in the initial state. We varied the overlap rate to create problems with various degrees of interleaving. We also varied the number of predicates appearing in the problems to change the chances that the planner tries to interleave multiple predicate tasks. A sample problem is shown in Figure 6.

We created and tested 100 problems of 1, 2 or 3 predicates used, 10%, 50% or 90% overlap, and 2 or 3 goals, totaling 1800 problems tested. We counted the number of search nodes, i.e. partial plans, created during the planning and computed the average for each type of problem. The results are shown in Table 1.

The ExCon strategy first extracts the external conditions of each of the methods. The methods in Figure 5(a)–(c) have no external conditions while the methods in Figure 5(d)–(f) have two external conditions each (the first two 'before' state constraints). Also, the Do-Nothing method for each predicate task has one external condition. (ex. the constraint (before (*p* ?x) n) is the external condition for the Do-Nothing method of the task (p ?x).)

Table 1: The average numbers of search nodes created over the 100 randomly generated problems in the artificial test domain.

	FAF	ExCon	FAF	ExCon	FAF	ExCon
2 goals	1 predicate		2 predicates		3 predicates	
90%	10.5	10.5	22.5	21.9	35.8	32.6
50%	12.3	12.3	30.2	26.2	56.2	38.3
10%	14.2	14.2	36.1	29.7	68.7	43.8
3 goals	1 predicate		2 predicates		3 predicates	
90%	16.5	16.2	40.2	38.8	79.1	52.5
50%	34.5	34.8	176	64.4	1414	105
10%	53.6	53.5	473	101	3302	141

Table 1 shows our experimental results. At 90% overlap, most predicate tasks can be phantomized and the amount of backtracking is minimal. Thus, the performances of the two strategies are almost the same. As the overlap rate decreases to 50% and 10%, less and less predicate tasks can be phantomized. The planner has a harder time trying to order tasks consistently in such a way that phantomization will work between the goals. For example, consider the problem shown in Figure 6. If the planner orders the task g3 before the task g4, the subtask (p C6) of g3 can be phantomized by the initial state. On the other hand, if the planner orders the task g6 before the task g2, then the subtask (r C4) of g6 can be phantomized. However, the planner cannot phantomize both (p C6) and (r C4) because of the ordering constraints.

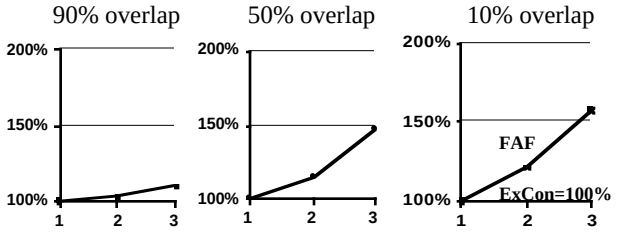

Figure 7: The relative performance of FAF and ExCon on 2-goal problems. The x-axis gives the number of predicates in the goals, and the y-axis gives the ratio (# search nodes by FAF) / (# search nodes by ExCon).

Figures 7 and 8 are graphs of the data in Table 1, that show how the relative performance of FAF and ExCon depends on number of predicates in the goal. Note that as the number of predicates increases, FAF's performance degrades much more quickly than ExCon's. This is because FAF creates many more search nodes in order to phantomize predicate tasks. Since ExCon works on one predicate at a time until it is established or fails, the planner does not have to backtrack over multiple predicates as it does with FAF.

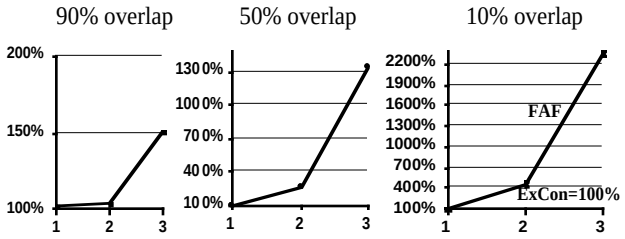

Figure 8: The relative performances of FAF and ExCon for 3-goal problems. The x-axis gives the number of predicates in the goals, and the y-axis gives the ratio (# search nodes by FAF) / (# search nodes by ExCon).

The graphs in Figure 9 show how the relative performance of FAF and ExCon depends on the overlap rate. For the problems with 1 predicate, the difference between FAF and ExCon is insignificant. For the problems with 2 or 3 predicates in the goals, FAF is clearly spending more time backtracking than ExCon.

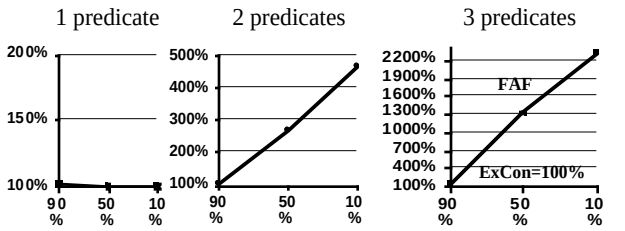

Figure 9: The relative performance of FAF and ExCon on 3-goal problems. The x-axis gives the overlap rate, and the y-axis gives the ratio (# search nodes by FAF) / (# search nodes by ExCon).

ExCon does slightly worse than FAF on average for the problems with 3 goals, 1 predicate, and 50% overlap. For other types of problems, ExCon may perform worse than FAF on certain problems but on average, ExCon performs better. ExCon may not do well if search branches fail for reasons such as the failure of variable bindings, or state constraints except external conditions. We need to study commitment strategies as a whole to improve such situations. in the future

Experiments with the Translog Domain

We now describe our experiments with FAF and ExCon in the Translog domain (Andrews, *et al.* 1995). Translog is a transportation logistics domain specifying various delivery methods for various types of packages. For example, it specifies granular packages such as grain must be delivered by hopper trucks. If the packages are the same type and the itineraries are the same, then two delivery tasks can be interleaved by using the same truck and carrying the packages at the same time. Even if the itineraries are different, if the destination of a package A is the initial location of the other package B, then the task of moving the truck to the initial location of B can be interleaved by carrying and delivering package A first.

We tested (a) one-package delivery problems, (b) two package delivery problems where the packages are of the same package type and have the same initial location and destination, (c) two package delivery problems where the packages are of the same package type and the destination of one of the packages is the same as the initial location of the other package, (d) two package delivery problems where the packages are of the same type but none of the initial locations or the destinations are the same, and (e) two package delivery problems where the packages are of the different types. We randomly created 50 problems, 10 problems for each type of problems, with various package types (regular, bulky, liquid, livestock or granular) and locations (among 15 locations). The results are shown in Table 2.

Table 2: Average number of nodes generated by FAF and ExCon on Translog problems. For two-package delivery problems, "Package types" shows whether the packages have the same type and so can be carried by the same delivery truck.

Problem	Package types	FAF	ExCon	FAF/ExCon
1pack (a)	—	67.9	67.9	1.00
2pack (b)	same	932.6	1331.4	0.70
2pack (c)	same	799.9	721.7	1.11
2pack (d)	same	1415.9	731.9	1.93
2pack (e)	different	1366.3	518.2	2.64

For one-package delivery problems, there is almost no interleaving in the problem, so the performances of FAF and ExCon are similar. For two-package delivery problems, the performances of the two strategies depends on how much tasks can be successfully interleaved between two goal tasks. For the problems of type (b), the task of moving trucks to the necessary locations can be completely interleaved between the two goals. So FAF can perform well on these problems. On our ten test problems, FAF produced fewer search nodes on average than ExCon although ExCon outperformed FAF on 7 out of 10 problems. For the problems of type (c), the task of moving a truck to the initial location of the second package can be interleaved by delivering the first package first. ExCon does this more efficiently than FAF. For the problems of type (d), although the two packages are of the same type, and same trucks can be used, no interleaving can actually work, since the locations are all different. Both ExCon and FAF make attempts to interleave tasks, but ExCon realizes the failure faster than FAF does. The similar occurrence takes place for the problems of type (e) where the two package types are different.

Conclusions

In this paper, we have shown how an HTN planner can avoid many situations that cause complexity in multiple-goal and interleaved problems, by identifying and handling

external conditions. We have presented ExCon, a task selection strategy that makes use of these conditions. ExCon causes the planner to explore tasks that are likely to affect the applicability conditions of other tasks first, significantly reducing backtracking. In our empirical studies, ExCon consistently outperformed FAF on complex problems, doing increasingly well on the problems where the task interactions occurred recursively and where multiple goals were involved.

Since ExCon enables the planner to establish conditions in the plan at less detailed levels, it produces some of the same improvements in planning efficiency that one might try to get using planning constructs such as unsupervised preconditions or high-level effects. In addition, it has the following advantages:

- External conditions do not have to be specified explicitly by the user, but instead are found automatically by the planning system when it pre-compiles its knowledge base. This will make it much easier for users to maintain the knowledge base.
- ExCon is a task-selection strategy, not a search-space pruning heuristic: it simply specifies the order in which a planner will prefer to expand tasks. Thus, it has no effect on the planner's soundness and completeness: a planner that is sound and complete without it will also be sound and complete with it.

Acknowledgments

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. In Fox, M. and Zweben, M., editors, *Intelligent Scheduling*, 451–469. Calif: Morgan Kaufmann,.

Agosta, J. M. 1995. Formulation and Implementation of an Equipment Configuration Problem with the SIPE-2 Generative Planner. In *Proc. 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, CS-TR-3487, University of Maryland.

Bacchus, F. and Yang, Q. 1991. The downward refinement property. In *Proc. IJCAI-91*.

Bitner, J. and Reingold, E. 1975. Backtrack Programming Techniques. *CACM* 18(11).

Blum, A. L. and Furst, M. L. 1997. Fast Planning Through 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; Hendler, J.; and Nau, D. 1994a. UMCP: A Sound and Complete Procedure for Hierarchical Task-network Planning. In *Proc. Second Conference on AI Planning Systems (AIPS-94)*.

Erol. K.; Hendler, J.; and Nau, D. 1994b. HTN Planning: Complexity and Expressivity. In *Proc. AAAI-94.*

Estlin, T. A.; Chien, S. A.; and Wang, X. 1997. An Argument for Hybrid HTN/Operator-Based Approach to Planning. In *Proc. Fourth European Conference on Planning (ECP-97)*, 184–196.

Joslin, D. and Pollack, M. E. 1994. Least-Cost Flaw Repair: A Plan Refinement Strategy for Partial-Order Planning. In *Proc. AAAI-94*, 1004–1009.

Kautz, H. and Selman, B. 1992. Planning as Satisfiability. In *Proc. Tenth European Conference on Artificial Intelligence*, 359–363.

Penberthy, J. S. and Weld, D. 1992. UCPOP: A Sound, Complete, Partial Order Planner for ADL. In *Proc. KR-92.*

Pollack, M. E.; Joslin, D.; and Paolucci, M. 1997. Flaw Selection Strategies for Partial-Order Planning. *Journal of Artificial Intelligence Research* 6, 223-262.

Purdom, P. W. 1983. Search Rearrangement Backtracking and Polynomial Average Time. *Artificial Intelligence* 21: 117–133.

Sacerdoti, E. 1977. *A Structure for Plans and Behavior.* American Elsevier Publishing Company.

Smith, S. J.; Nau, D.; and Throop, T. 1996. Total-Order Multi-Agent Task-Network Planning for Contract Bridge. In *Proc. AAAI-96.*

Tate, A. 1977. Generating Project Networks. In *Proc. IJCAI-77*, 888–893.

Tsuneto, R.; Erol, K.; Hendler, J.; and Nau, D. 1996. Commitment Strategies in HTN Planning. In *Proc. AAAI-96*, 536-542.

Tsuneto, R.; Hendler, J.; and Nau, D. 1997. Space-Size Minimization in Refinement Planning. In *Proc. Fourth European Conference on Planning(ECP-97)*.

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

Wilkins, D. E. and Desimone, R. V. 1994. *Applying an AI planner to military operations planning.* In Fox, M. and Zweben, M., editors, Intelligent Scheduling, 685—709. Calif.: Morgan Kaufmann.

Young, R. M.; Pollack, M. E.; and Moore, J. D. 1994. Decomposition and Causality in Partial-Order Planning. In *Proc. Second Conference on AI Planning Systems.*