Exploiting State Constraints in Heuristic State-Space Planning

Ioannis Refanidis and Ioannis Vlahavas
Aristotle University, Dept. of Informatics
54006, Thessaloniki, GREECE
E-mails: {yrefanid, vlahavas}@csd.auth.gr

Abstract

In the last years, some very promising domain independent heuristic state-space planners for STRIPS worlds, like ASP/HSP, HSPr and GRT, have been presented. These planners achieve remarkable performance in some domains, like the blocks world, the logistics and the gripper, but they are not effective in other domains, like the grid and the mystery. In this paper we propose the use of state constraints in heuristic state space planning. We claim that one of the causes for the pre-mentioned failures is the absence of domain specific knowledge about properties that characterize every valid and complete state. We propose the inclusion of state constraints in the domain definition and we present how they can be exploited by heuristic planners in order to decompose a problem into sub-problems that are easily solvable. We give performance results that exhibit significant speedup in the problem solving process. Finally, we give a notion of how problem decomposition can accelerate other planners, like GRAPHPLAN and BLACKBOX.

Introduction

In the last years, part of the planning community turned towards heuristic state-space planning, developing new domain independent algorithms, which achieve significant performance. The first planner was ASP (Bonet, Loerincs, and Geffner 1997), followed by HSP, HSPr (Boner and Geffner 1998, 1999) and GRT (Refanidis and Vlahavas 1999a). These domain independent heuristic planners, which adopted the STRIPS notation (Fikes and Nilsson 1971), search for solutions in the space of the states. For their guidance they use variations of a relatively simple idea, estimating the number of steps needed to achieve a fact from a given state, in order to estimate distances between states. The above planners exploit these estimates in a best-first or in a hill-climbing way.

HSP took part in the AIPS-98 planning competition, solving more problems than the other contestants did, but it needed more time on average. HSPr and GRT improved the way, in which the distances are computed, accelerating drastically the problem solving process. GRT took the original idea one step further, taking into account the interactions that occur while trying to achieve different facts simultaneously; thus, it produced better estimates and solved even more problems.

Although HSPr and GRT performed quite satisfactory in domains like the blocks world (Kautz and Selman 1996), the logistics (Veloso 1992) and the gripper (AIPS-98), they failed to handle effectively other domains like the grid and the mystery. A detailed presentation of the domains used in the AIPS-98 planning competition can be found at ftp.cs.yale.edu/pub/mcdermott/software/pddl.tar.gz.

In this paper we propose the use of domain dependent knowledge about axioms that apply to any valid and complete state of a domain. For example, in the logistics domain such axioms demand that each plane lies at exactly one airport or a package is either at a location, or within a truck or plane. Similar axioms can be formed for any other domain.

We name this kind of axioms XOR constraints, because they can be formalized as XOR relations between sets of ground facts, where exactly one of them can hold in each state. We believe that this kind of constraints should constitute an integral part of each domain's definition and specification languages (e.g. PDDL) should support them. However, XOR constraints can also be computed automatically, taking into account the definitions of the operators and the initial state (Gerevini and Schubert 1998, Refanidis and Vlahavas 1999b).

We enhanced GRT with the ability to handle state constraints. However any other heuristic planner could have been used instead. The enhanced GRT uses state constraints at the pre-processing phase, in order to analyze a planning problem in a sequence of sub-problems. Generally, these sub-problems are easily solved by heuristic planners; thus the total time needed to solve them is substantially shorter than the time needed to solve the original problem. The same algorithm could also be applied to the sub-problems, decomposing them in sub-sub problems and so on. Another possibility is to use a separate planner, e.g. GRAPHPLAN (Blum and Furst 1995) or SATPLAN (Kautz and Selman 1996), to solve the sub-problems.

In the rest of the paper, we briefly present the recent heuristic state-space planners and illustrate why they fail to handle certain domains, like the grid and the mystery. Then, we introduce state constraints and present how heuristic state-space planners can exploit them so as to decompose the initial problem into easily solvable sub-problems. Next, we present performance results obtained by the GRT planner, which has been enhanced with the ability to handle state constraints. Furthermore, we

present the effect of state constraints on other planners, like GRAPHPLAN and BLACKBOX (Kautz and Selman 1998). Finally, we conclude the paper and we pose future directions.

Heuristic State-Space Planners

In the last years, remarkable progress has been made in the area of heuristic domain independent planning for STRIPS worlds. The first such planner was ASP, followed by HSP and recently by HSPr and GRT.

The main idea underlying these planners is the computation of estimates for the number of actions needed to achieve certain facts from a given state. Suppose S is a state defined as a collection of ground facts, $S=\{p_1, p_2, ..., p_n\}$, a a ground action with $P(a)$, $D(a)$ and $A(a)$ being its preconditions, delete and add lists respectively, and q another domain ground fact. Then, the distance between q and S is estimated recursively as:

$$g(q,S)=\begin{cases} 0, \text{ if } q \in S \\ i+1, \text{ if for some action } a,\ q \in A(a) \text{ and } \sum_{p \in P(a)} g(p,S) = i \\ \infty, \text{ if } q \text{ is unreachable from } S \end{cases}$$

Function $g(q,S)$ does not define a unique value, since for each ground fact q there may be more than one ground actions a that achieve it, while the same may also hold for the preconditions of these actions. Thus, the computed values depend strongly on the action-selection strategy. Moreover, it is possible that $g(q,S)$ assigns a finite distance to a fact that cannot be achieved.

For another state T, we can define its distance from S as $g(T,S) \overset{def}{=} \sum_{q \in T} g(q,S)$.

ASP and HSP used the above function to compute estimates between the intermediate states and the goal. The main problem was that the above computations, which are time consuming, were performed for each intermediate state, taking up to 85% of the processing time (Bonet and Geffner 1999). This problem was tackled both by HSPr and GRT.

HSPr computes the distances between all the facts of the domain and the initial state, once at a pre-processing phase, and then it searches backwards from the goals. Similarly, GRT computes the distances between all the facts of the domain and the goals once at a pre-processing phase and then it searches forward from the initial state. As a result, both HSPr and GRT are significantly faster than their ancestors.

Another inefficiency of the above heuristic is that it considers all the facts of the domain as strictly independent; thus, the cost of achieving them together is equal to the sum of the costs of achieving each one of them individually. This problem was tackled by GRT, which does not use the formulas presented above. GRT annotates each achieved ground fact not only with its distance from the goals, but also with a set of other ground facts that may be co-achieved. These facts are called *related facts* and are taken into account when computing distances. With this modification GRT achieves better estimates and generally solves more problems that the other heuristic planners (Refanidis and Vlahavas 1999a).

The Problem

In (Bonet and Geffner 1999) and (Refanidis and Vlahavas 1999), performance results for HSPr and GRT respectively are presented, from which it is evident that these planners are quite competitive in comparison with other planners, like BLACKBOX, STAN (Long and Fox 1998) and IPP (Koehler et al. 1997), at least in the blocks world, the logistics, the rocket (Blum and Furst 1995) and the gripper domains. However, both papers state that these planners cannot handle other domains, like the grid and the mystery. In this section we illustrate why the above planners cannot handle such domains. We will use the GRT planner as a reference, but the comments also apply to ASP/HSP(r).

The main disadvantage of GRT planner is that in every state it always selects to apply the action leading to the adjacent state with the least distance from the goals. If the various facts of a domain were independent, this strategy would be optimal. However, this is not the case and quite often the result is a local optimal state (the problem appears in GRT, although it does not consider the facts as being totally independent). Therefore, the planner should temporarily select actions that lead to states with greater distances from the goals, before selecting actions that lead to the goals. This is better illustrated in Figure 1.

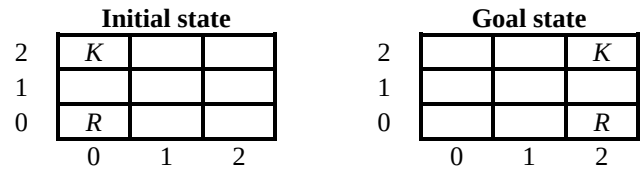

Figure 1. A 3x3 Grid problem.

The problem is from the grid domain, where *K* is a key and R is a robot. The valid actions are *get*, *leave* and *move*, with their obvious meanings. The initial and the goal states are the following (we will use a Prolog-style representation):

Initial state = [*at*('*R*',*n0_0*), *at*('*K*',*n0_2*)]
Goal state = [*at*('*R*',*n2_0*), *at*('*K*',*n2_2*)]

Fact	Distance from Goals	Related Facts
at('*R*',*n2_0*)	0	[ ]
at('*K*',*n2_2*)	0	[ ]
at('*R*',*n1_0*)	1	[ ]
at('*R*',*n0_0*)	2	[ ]
at('*R*',*n0_1*)	3	[ ]
at('*R*',*n2_1*)	1	[ ]
at('*R*',*n2_2*)	2	[ ]
in('*R*','*K*')	3	[*at*('*R*',*n2_2*)]
at('*R*',*n1_2*)	3	[ ]
at('*K*',*n1_2*)	7	[*at*('*R*',*n1_2*)]
at('*R*',*n0_2*)	4	[ ]
at('*P*',*n0_2*)	8	[*at*('*R*',*n0_2*)]

Table 1. The Greedy Regression Table for the 3x3 Grid problem.

At the pre-processing phase, GRT estimates the distances between all the ground facts and the goal state. This information is stored in a table, named "Greedy Regression Table", since its data are obtained by greedy applying (inverted) actions to the goals. Table 1 shows this table for the 3x3 Grid problem. The related facts are also shown.

According to Table 1, the distance between the initial state and the goals is 10. There are two applicable actions to the initial state, moving *R* to *n1_0* and moving *R* to *n0_1*. The resulting state after moving *R* to *n1_0* has a distance from the goals equal to 9, while the resulting state after moving *R* to *n0_1* has a distance from the goals equal to 11. So the planner decides to move *R* to *n1_0* and subsequently to *n2_0*. However, it is obvious that the optimal sequence of actions would initially move the robot to *n0_1*, next to *n0_2*, get the key etc.

The planner does not initially select the optimal action, because it leads to a state with a greater distance from the goals. In order to decide to move the robot towards the key, the planner should exhaust all the other valid plans and then try to move the robot to worse states (this requires that the planner remembers past states and does not visit them again). In difficult problems the number of states that the planner has to visit before following the right direction is extremely large. That is the reason why HSPr and GRT fail to handle problems from the grid and mystery domains.

This problem also occurs in the logistics domain. However, the ability of the trucks and the planes to move instantaneously to any valid location or airport respectively, decreases the difficulty of the problem significantly.

An ideal planner should detect that, in order to move the key from *n0_2* to *n2_2*, it is necessary that the robot gets the key, so it is needed to achieve initially the fact *at*('*R*',*n0_2*), shifting the achievement of *at*('*R*',*n2_0*) in a later time. But the planner does not know that the facts *at*('*R*',*n0_0*), *at*('*R*',*n2_0*) and *at*('*R*',*n0_2*) are related in some way, because the domain definition does not contain this information. Therefore, it is necessary to provide the planner with information about relations that exist between the facts of the domain.

State Constraints

Defining State Constraints

The state constraints we selected to use are relations between ground facts, where exactly one of them can hold in each valid and complete state. We call these constraints XOR constraints, because they can be considered as XOR relations between facts, which are true in a given state if and only if exactly one of the facts participating in the relation holds (in (Gerevini and Schubert 1998) these constraints are called *single valuedness constraints* or *sv constraints*, but sv constraints concern instantiations of the same predicate, while our XOR constraints can be relations between ground facts of different predicates). Such relations can be defined at almost every domain. For example, in the logistics domain we can define the following XOR constraints:

xor([*at*(*Truck*, _)]) :- *truck*(*Truck*).
xor([*at*(*Plane*, _)]) :- *plane*(*Plane*).
xor([*at*(*Package*, _), *in*(*Package*, _)]):-
package(*Package*).

By writing "_" we mean that this variable could be instantiated to every valid value, according to the predicate definition and the instantiations of the other variables. The above definitions mean that for each set of values of the named variables that appear in a XOR relation, for all the possible values of the no-name variables (those noted by "_"), according to the domain definition, exactly one ground fact can hold in each valid and complete state. Note that the above state constraints are general definitions that can be grounded in several ways, according to the different ways in which the named variables can be instantiated.

For example, the first XOR constraint means that for each *Truck* and for all the possible locations in the domain, exactly one ground fact of the form *at*(*Fact*,*Location*) can hold in a valid and complete state. This state constraint can be instantiated for each different truck of the domain. The meaning of the second XOR constraint is similar. Finally, the third XOR constraint can be interpreted as follows: for each *Package*, for all the locations and for all the *Trucks* and *Planes*, exactly one fact of the form *at*(*Package*,*Location*) or *in*(*Package*,*Truck/Plane*) can hold in each valid and compete state.

In some cases, it is possible to have XOR constraints that incorporate AND relations. For example, if in the logistics domain the predicate *out*(*Package*) is defined, which means that a package is loaded neither in a *Truck*

nor in a *Plane*, then the relevant constraint would be written as:

xor([*and*([*at*(*Package*, _), *out*(*Package*)]),
in(*Package*,_)]):-*package*(*Package*).

However, AND relations can be avoided by combining different predicates in one. In the above example, one could omit predicate out, considering that the presence of the fact *at*(*Package*,*Location*) in a state implies that the *Package* is not loaded.

Exploiting State Constraints

We will illustrate the use of the state constraints with the example of Figure 2. The domain is the same with that of Figure 1, but the problem is a little more complicated, having two keys (P1 and P2) and two robots (R1 and R2).

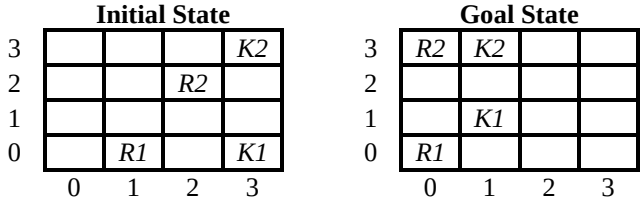

Figure 2. A 4x4 Grid problem

The initial and the goal states of this example are defined as follows:

Initial state = { *at*('*R1*',*n1_0*), *at*('*R2*',*n2_2*), *at*('*K1*',*n3_0*), *at*('*K2*',*n3_3*) }
Goal state = { *at*('*R1*',*n0_0*), *at*('*R2*',*n0_3*), *at*('*K1*',*n1_1*), *at*('*K2*',*n1_3*) }

The XOR constraints we will use are the following:

xor([*at*(*Robot*, _)]) :- *robot*(*Robot*)
xor([*at*(*Key*, _), *hold*(_, *Key*)]):-*key*(*Key*)

The above definitions have four ground instantiations, one for each Robot and one for each Key. Hereafter we will use the notation XOR_{OBJ} to refer to the instantiated XOR constraint concerning object OBJ.

The first information the planner can extract concerns pairs of facts, one from the initial state and one from the goals that belong to the same ground XOR relation (we suppose that the goals constitute a complete state - we treat the case in which this hypothesis does not hold later). The pairs identified for the current problem are the following:

XOR_{R1}: *at*('*R1*',*n1_0*) - *at*('*R1*',*n0_0*)
XOR_{R2}: *at*('*R2*',*n2_2*) - *at*('*R2*',*n0_3*)
XOR_{K1}: *at*('*K1*',*n3_0*) - *at*('*K1*',*n1_1*)
XOR_{K2}: *at*('*K2*',*n3_3*) - *at*('*K2*',*n1_3*)

At the pre-processing phase, GRT planner computes estimates for the distances of all the domain's facts from the goals, together with the lists of related facts, but it does not keep information about the actions that achieved the various facts.

However, in order to exploit the state constraints, this information should be retained. By keeping the ground actions that achieve the various facts, the table structure used by GRT planner is transformed into a directed acyclic graph structure. We call this structure GRG, the acronym of the words "Greedy Regression Graph".

The nodes of this graph represent the facts of the domain. Each node is labeled with a non-negative integer value, which is an estimate for the number of backward steps needed to achieve its fact starting from the goals. Moreover, it is labeled with the corresponding list of related facts. Finally, it is labeled with the name of the ground action, which first achieved the node's fact. This action characterizes all the arcs that point to the node. Figure 3 shows a part of the GRG structure for the 4x4 Grid problem (the Figure does not show the related facts).

Having constructed the GRG structure, for every ground XOR relation we can derive sequences of actions that transform the initial state fact to the corresponding goal state fact. For each such sequence we are interested only in the actions that change facts of the corresponding XOR relation; we are not interested in actions that provide auxiliary preconditions.

For the four ground XOR constraints, the sequences of actions that transform the initial facts to the goal facts are the following (note that, while Figure 3 refers to the 'inverted' actions, here the original ones are used instead):

XOR_{R1}: *at*('*R1*',*n1_0*) - *at*('*R1*',*n0_0*)
move('*R1*', *n1_0*, *n0_0*)

XOR_{R2}: *at*('*R2*',*n2_2*) - *at*('*R2*',*n0_3*)
move('*R2*', *n2_2*, *n2_3*), *move*('*R2*', *n2_3*,*n1_3*),
move('*R2*', *n1_3*, *n0_3*)

XOR_{K1}: *at*('*K1*',*n3_0*) - *at*('*K1*',*n1_1*)
get('*R1*', '*K1*', *n3_0*), *leave*('*R1*', '*K1*', *n1_1*)

XOR_{K1}: *at*('*K2*',*n3_3*) - *at*('*K2*',*n1_3*)
get('*R2*', '*K2*', *n3_3*), *leave*('*R2*', '*K2*', *n1_3*)

Having identified the above sequences of actions, we check their preconditions, searching for facts that are included in a XOR relation, other than the sequence where they are appearing. In our example, the actions of sequences XOR_{R1} and XOR_{R2} do not have any precondition that is part of another XOR relation. However the actions of sequences XOR_{K1} and XOR_{K2} do have such preconditions. To be more specific, in the XOR_{K1} sequence, actions *get*('*R1*','*K1*',*n3_0*) and *leave*('*R1*','*K1*',*n1_1*) have *at*('*R1*',*n3_0*) and *at*('*R1*',*n1_1*) as preconditions respectively, which are members of the XOR_{R1} relation. Similarly, in the XOR_{K2} sequence, actions *get*('*R2*','*K2*',*n3_3*) and *leave*('*R2*','*K2*',*n1_3*) have *at*('*R2*',*n3_3*) and *at*('*R2*',*n1_3*) as preconditions respectively, which are members of the XOR_{R2} relation.

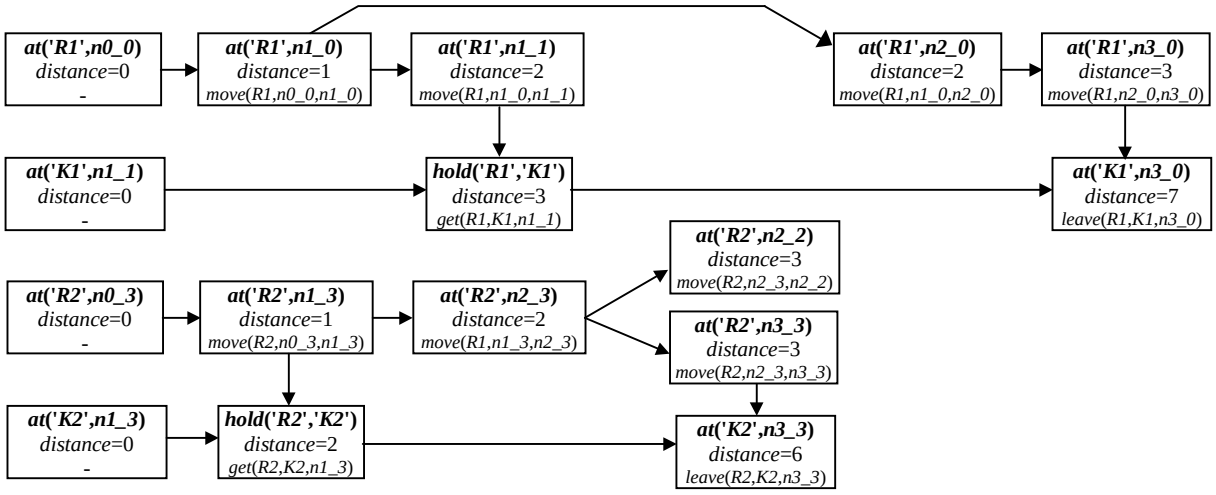

Figure 3. Part of the Greedy Regression Graph for the 4x4 Grid problem.

In case where there is no XOR constrained fact appearing in a sequence of actions of another XOR relation, the original problem can be easily solved by any heuristic planner. But if there are XOR constrained facts appearing as preconditions in actions of other XOR sequences, the original problem has to be analyzed into sub-problems. To do this, for each ground XOR relation we identify all the facts that are members of this relation and that are:

(I) members either of the initial state, or of the goals, or
(II) preconditions of a ground action that appears in another XOR sequence, or
(III) add effects of an action of their own XOR sequence, which action has a fact of another XOR relation as precondition.

Having identified the above facts, we construct a graph, conjoining them with arcs that denote ordering constraints, using the following rules:

(I) For each XOR relation, all the facts are ordered after the initial fact and before the goal fact (if any).
(II) The facts that are add effects of actions in their own XOR sequence are ordered according to the positions of their actions in the XOR sequence.
(III) The facts that are preconditions in any action are placed in the same order with the facts being add effects in the same action (these facts belong to different XOR relations).
(IV) The facts of a XOR relation that are preconditions in different actions of another XOR sequence are ordered according to the ordering of their actions.
(V) For each XOR relation, the facts that are preconditions in actions of another XOR sequence are ordered before the facts that are add effects in their own XOR sequence.

We call the resulted graph as the *ordering graph* of the problem, since it denotes the order in which its facts have to be achieved. Figure 4 shows the *ordering graph* for the 4x4 Grid problem. Lines with arcs denote ordering constraints, while lines without arcs denote that the two facts are ordered together.

It is not obligatory that for each XOR relation there exist a fact within the goals. If for some relation there is not such one, then the first two of the above five steps are unnecessary for it. The case in which a XOR relation has no fact in the initial state is more difficult, because this indicates lack of knowledge about the initial state, demanding contingent planning to solve the problem for all possible different initial states.

From the ordering graph it is possible to construct intermediate states (possibly incomplete) that have to be achieved. This task is a graph-solving problem, which, in some domains, may be difficult. However, we follow a simple approach that usually works well.

Starting from the initial state, in each intermediate state we try to include a fact from each XOR relation, with the following properties:

(I) it has not been included in a previous intermediate state,
(II) it is not ordered after another fact of the same XOR relation that has not already been included in a previous intermediate state, and finally
(III) it is not ordered together with a fact of another XOR relation that cannot be included in the current intermediate state.

In case where for a XOR relation there are more than one facts with the above properties, we select one of them randomly. Finally, in case where for a XOR relation there is no fact with the above properties, we do not select any fact, leaving the intermediate state incomplete.

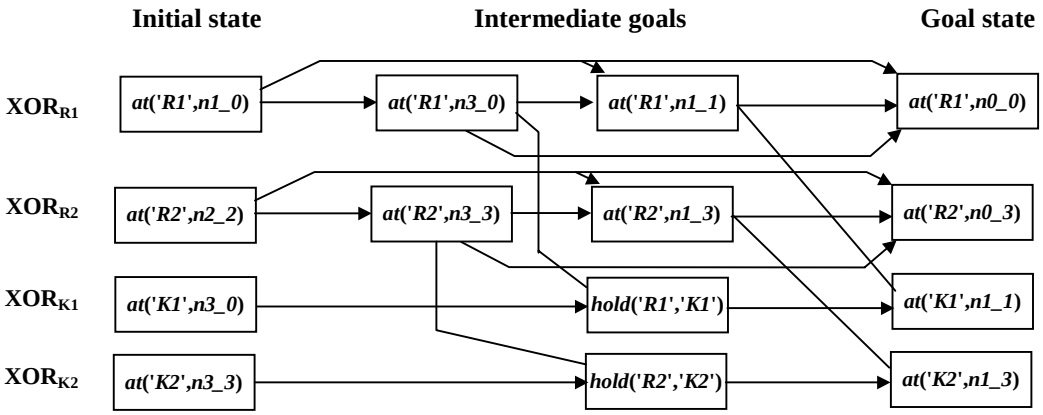

Figure 4. The ordering graph for the 4x4 Grid problem.

For the problem of Figure 4, the following three intermediate states have been extracted:

Intermediate state 1: { *at*('*R1*',*n3_0*), *at*('*R2*',*n3_3*), *in*('*K1*','*R1*'), *in*('*K2*','*R2*') }

Intermediate state 2: { *at*('*R1*',*n1_1*), *at*('*R2*',*n1_3*), *at*('*K1*',*n1_1*), *at*('*K2*',*n1_3*) }

Intermediate state 3: { *at*('*R1*',*n0_0*), *at*('*R2*',*n0_3*), *at*('*K1*',*n1_1*), *at*('*K2*',*n1_3*) }

where the last state is the goal state.

After constructing the three intermediate states, the planner has to solve three sub-problems, which are substantially easier than the original one; thus, the total time to solve them is shorter than the time needed to solve the original problem. It is also possible to apply XOR constraints also to the sub-problems, decomposing them in sub-sub-problems, which could result in better performance. A last possibility is to use a different planning algorithm to solve the sub-problems, like GRAPHPLAN or SATPLAN, instead of a heuristic state-space algorithm.

Performance Results

In order to have a notion for the effectiveness of the problem decomposition approach, we have enhanced the GRT planner with the ability to handle state constraints and to decompose problems into sub-problems. The implementation language was C++[1] and the test platform we used was a Pentium Celeron 300MHz, with 64-MB main memory. This platform is identical (except for the memory) with the one used at the AIPS-98 planning competition, so we obtained comparative results.

We forced the planner to solve problems from the logistics, the grid and the mystery domains. The XOR state constraints we have used for these domains are the following:

[1] The source code of the planner is available through contacting the authors. Also a Prolog version of the code is also available.

Logistics domain:
xor([at(Truck, _)]) :- truck(Truck).
xor([at(Plane,_)]) :- plane(Plane).
xor([at(Package,_), in(Package,_)]):- package(Package).

Grid domain:
xor([arm_empty, not_empty]).
xor([locked(Pos), open(Pos)]):- place(Pos).
xor([at_robot(_)]).
xor([at(Key,_), holding(Key)]):-key(Key).

Mystery domain:
xor([at(Truck, _)]) :- truck(Truck).
xor([at(Package,_), in(Package,_)]) :-package(Package).

For the convenience of the reader we have "translated" the original predicates' names of the mystery domain in more meaningful ones.

Table 2 presents comparative performance results of GRT in the logistics domain. The problems are those used at the AIPS-98 planning competition. The table presents measurements both for the original GRT planner and for the improved one. Short dashes for GRT mean that no solution has been found after 15 minutes.

As it is clear from this table, the use of state constraints and the problem decomposition process speeds up the problem solving process up to 30%. It should be noted here that the logistics domain is not the most suitable domain to demonstrate the utility of the state constraints. The unreal assumption underlying this domain, that each truck and plane can be moved in one step to any possible location, renders these problems easily solvable by any heuristic state-space planner. Therefore, although the sub-problems are trivially solved by GRT, the preprocessing phase that should be repeated for every sub-problem produces a significant overhead, which is comparable to the time needed to solve the original problems.

Problem	Best AIPS-98		GRT without state constraints		GRT with state constraints	
	actions	time	actions	time	actions	time
prob01	26	767	30	280	30	240
prob02	32	4319	34	1320	34	980
prob03	-	-	60	5550	61	4200
prob04	-	-	69	19280	70	15100
prob05	24	2400	26	390	26	270
prob06	-	-	80	14390	81	10120
prob07	112	788914	37	1760	37	1310
prob08	-	-	48	16370	49	11420
prob09	-	-	98	50480	99	37530
prob10	-	-	117	23130	119	16120
prob11	30	6544	36	1540	36	1140
prob12	-	-	48	43060	50	33200
prob13	-	-	79	85580	80	69520
prob14	-	-	104	60200	105	39100
prob15	-	-	106	6750	108	5120
prob16	-	-	62	31580	61	24200
prob17	-	-	53	12190	53	8960
prob18	-	-	193	335050	193	271280
prob19	-	-	174	238980	172	165860
prob20	-	-	169	324120	170	240190
prob21	-	-	120	294230	121	223930
prob22	-	-	-	-	-	-
prob23	-	-	118	16860	117	11500
prob24	-	-	49	98540	49	68200
prob25	-	-	-	-	-	-
prob26	-	-	-	-	-	-
prob27	-	-	-	-	-	-
prob28	-	-	-	-	-	-
prob29	-	-	-	-	-	-
prob30	-	-	-	-	-	-

Table 2. Performance results for the logistics domain (time in msecs)

Problem	Best AIPS-98		GRT without state constraints		GRT with state constraints	
	actions	time	actions	time	actions	time
prob01	14	2505	-	-	16	1040
prob02	-	-	-	-	28	6630
prob03	-	-	-	-	65	21350
prob04	-	-	-	-	32	19920
prob05	-	-	-	-	153	118650

Table 3. Performance results for the grid domain (time in msecs)

Problem	GRT without state constraints		GRT with state constraints	
	actions	time	actions	time
prob01	5	30	5	75
prob02	11	12400	11	470
prob03	4	125	6	210
prob04	7	140	7	110
prob05	10	18100	10	360
prob06	-	-	18	2150
prob07	4	200	4	160
prob08	-	-	6	490
prob09	-	-	11	320
prob10	-	-	12	11370
prob11	7	180	7	150
prob12	-	-	7	130
prob13	-	-	17	22920
prob14	-	-	25	25400
prob15	-	-	11	2320

Table 4. Performance results for a simplified mystery domain (time in msecs)

The two columns entitled "Best AIPS-98" present the best plan found in this competition and the best solution time (which may be achieved by different planners).

Table 3 presents performance results in the grid domain. The five problems have also been taken from the AIPS-98 competition. In this domain the original GRT planner failed to solve any problem, while with the use of state constraints it solved all of them.

It should be noted that in the grid domain we had two levels of decomposition, i.e. after the decomposition of each original problem in sub-problems, the sub-problems were further decomposed in sub-sub-problems, which were finally solved. The planner always tries to decompose any problem into sub-problems. If this attempt fails, then it tries to solve the problem; otherwise it decomposes the problem into sub-problems and proceeds recursively with them.

Table 4 presents performance results in a simplified mystery domain. We removed from this domain the resources, i.e. the fuels of the cities and the capacities of the trucks, because neither the original heuristic planners nor the enhanced with state constraints new ones can handle effectively such resources. The issue of resources should be handled separately and is a future work challenge. The problems in Table 4 are the 15 first problems of the AIPS-98 competition, with the pre-mentioned simplification.

By removing resources from the mystery domain, this domain resembles the logistic domain, except that in this domain the trucks cannot be moved instantaneously to any location. As it is shown on Table 4, the original GRT did not solve many problems, while with the use of state constraints it solved all of them easily.

Finally, Table 5 presents how the problem decomposition process can affect positively other planners, i.e. GRAPHPLAN and BLACKBOX (we thank Blum, Furst, Kautz and Selman for making their code available[1]). We have used five 'traditional' problems, taken by the bibliography (Veloso 1992, Blum and Furst 1995). The measurements have been taken on a SUN Enterprise 3000 Unix server, running at 166 MHz and having 64-MB memory. In this case, we decomposed the problems separately and we forced the planners to solve both the original problems and the sub-problems. Table 5 shows how the problem decomposition can affect drastically the performance of the planners.

[1] GRAPHPLAN is available at http://www.cs.cmu.edu/afs/cs.cmu.edu/usr/avrim/www/graphplan.html
BLACKBOX is available at http://www.research.att.com/~kautz/blackbox/index.html

Problem	GRAPHPLAN				BLACKBOX			
	without problem decomposition		with problem decomposition		without problem decomposition		with problem decomposition	
	actions	time	actions	time	actions	time	actions	time
logistics.a	-	-	62	600	69	67000	65	550
logistics.b	-	-	52	650	64	80000	59	580
logistics.c	-	-	60	700	77	101000	67	600
rocket.a	-	-	28	210	31	105000	29	180
rocket.b	-	-	29	240	28	104000	28	220

Table 5. Performance results for GRAPHPLAN and BLACKBOX planners, without and with problem decomposition (time in msecs)

Conclusions and Future work

In this paper we have presented how the use of state constraints can accelerate the problem solving process for state-space heuristic planners. By the term state constraints we refer to logical formulas that have to be true in every valid and complete state. In this work we have considered only XOR constraints, i.e. relations between sets of ground facts, where exactly one of them can hold in any valid and complete state.

We have shown that one reason why heuristic state-space planners fail to handle some domains is the absence of knowledge about axioms that hold in each valid and complete state. Then we have presented how these planners can exploit such knowledge, by decomposing the original problem into sub-problems that are easily solvable. The performance results exhibit significant speedup in the problem solving process. Furthermore, we have shown that the problem decomposition can accelerate other types of planners, like GRAPHPLAN and SATPLAN.

The main disadvantage of the problem decomposition approach is that the pre-processing phase, i.e. the computation of the distances between the domain's facts and the goals, has to be repeated for each sub-problem, which, in some cases, generates a considerable overhead in the total solution time. Thus, a future challenge is the acceleration of the pre-processing phase, using for example non fully-instantiated facts and actions, in combination with constraint satisfaction techniques.

Strictly concerning the problem decomposition process, future research includes the investigation of more efficient problem decomposition algorithms. Another point could be the computation of multiple distances for each ground fact at the pre-processing phase, i.e. the achievement of each ground fact in several ways, which would lead to multiple possible decompositions of each original problem in sub-problems.

Concerning the original heuristic state-space algorithms, one challenge is handling resources, i.e. capacities, fuels etc., in a universal manner. Another challenge is handing time. However, resources and time require number manipulation, which means that the algorithms have to be extended beyond the classical STRIPS planning paradigm.

Finally, we would like to investigate the exploitation of richer domain representations (apart from the state constraints) and how they can be combined with the existing planning algorithms.

References

Blum, A.L., and Furst M.L. 1995. Fast planning through planning graph analysis. In *Proc. of the 14th Intl. Joint Conf. on Artificial Intelligence (IJCAI-95)*, 636-642. Montreal, Canada: Intl. Joint Conference on Artificial Intelligence, Inc.

Bonet, B., Loerincs, G., and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *Proc. of 14th Intl. Conference of the American Association of Artificial Intelligence (AAAI-97)*, 714-719. Providence, Rhode Island: AAAI Press.

Bonet, B., and Geffner, H. 1998. HSP: Heuristic Search Planner. *Entry at the Artificial Intelligence Planning Systems (AIPS-98) Planning Competition.* Pittsburgh.

Bonet, B., and Geffner, H. 1999. Heuristic Planning: New Results. In *Proceedings of the 5th European Conference on Planning*, 359-371 (preprints). Durham, UK.

Fikes, R.E., and Nilsson, N.J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189-208.

Gerevini, A., and Schubert, L. 1998. Inferring State Constraints for Domain-Independent Planning. In *Proc. of the 15th Intl. Conference of the American Association of Artificial Intelligence (AAAI-98)*, 905-912. Madison, Wisconsin: AAAI Press.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proceedings of the 13th International Conference of the American Association of Artificial Intelligence (AAAI-96)*, 1194-1201. Portland, Oregon: AAAI Press.

Kautz, H., and Selman B. 1998. BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving. *Working notes of the Workshop on Planning as Combinatorial Search (AIPS-98).* Pittsburgh (1998).

Koehler, J., Nebel, B., Hoffmann, J., and Dimopoulos, Y. 1997. Extending Planning Graphs to an ADL Subset. In *Proceedings of the 4th European Conference on Planning (ECP-97)*, 273-285. Springer LNAI 1348.

Long, D., and Fox, M. 1998. Efficient Implementation of the Plan Graph in STAN. *Journal of Artificial Intelligence Research*, 10:87-115.

Refanidis, I., and Vlahavas, I. 1999a. GRT: A Domain Independent Heuristic for STRIPS Worlds based on Greedy Regression Tables. In *Proceedings of the 5th European Conference on Planning*, 346-358 (preprints). Durham, UK.

Refanidis, I., and Vlahavas, I. 1999b. On Determining and Completing Incomplete States in STRIPS Domains. In *Proc. of the IEEE Intl. Conference on Information, Intelligence and Systems*. 289-296, Washington, US.

Veloso, M. 1992. Learning by Analogical Reasoning in General Problem Solving. Ph.D. diss. Computer Science Dept., Carnegie Mellon Univ. (also available as techincal report: CMU-CS-92-174).