Distance-based Goal-ordering heuristics for Graphplan

Subbarao Kambhampati* **& Romeo Sanchez Nigenda**
Department of Computer Science and Engineering
Arizona State University, Tempe AZ 85287-5406
Email: {rao,rsanchez}@asu.edu

Abstract

We will discuss the shortcomings of known variable and value ordering strategies for Graphplan's backward search phase, and propose a novel strategy that is based on a notion of the difficulty of achieving the corresponding subgoal. The difficulty of achievement is quantified in terms of the structure of the planning graph itself–specifically, the earliest level of the planning-graph at which that subgoal appears. We will present empirical results showing the surprising effectiveness of this simple heuristic on benchmark problems. We will end by contrasting the way distance-based heuristics are used in Graphplan and state-search planners like UNPOP, HSP and HSP-R.

1 Introduction

It has been known for sometime now that the backward search of Graphplan algorithm can be seen as solving a (dynamic) CSP problem [8; 17]. Given this relation, the order in which the backward search considers goals for expansion–the so-called "variable ordering heuristic", and the order in which the actions supporting those goals are considered – i.e., the "value ordering heuristic"–can have a very significant impact on Graphplan's performance [1; 5]. Despite this expectation, experiments with the traditional CSP variable ordering strategies such as "most constrained variable first" and "least constrained value first"–have been shown to be of at best marginal utility [7].

In this paper, we note that the straightforward adaptation of variable and value ordering strategies from CSP literature is not likely to be effective given the fact that planning graph is a dynamic CSP, where the assignment of values to variables at one level activate other variables in the next lower level. We then present a family of variable and value-ordering heuristics that are based on the difficulty of achieving truth (or falsity) value for a proposition, starting from the initial state. In all these heuristics, the degree of difficulty of achieving a single proposition is quantified by the index of the earliest *level* of the planning graph in which that proposition first appears. During backward search, we consider goals (variables) in the order of highest to lowest difficulty, and the actions that support the chosen goal variable are considered in the order of easiest to hardest to achieve preconditions. We will show that these heuristics based on level information have several important properties:

- First and most important, use of these heuristics improves the performance of Graphplan. The improvements are particularly impressive when backward search is applied to "solution-bearing" levels of the planning graph (as is to be expected of any effective variable and value ordering heuristic).
- The heuristics are surprisingly insensitive to the length of the planning graph. Specifically, Graphplan with our heuristics tends to return near-optimal quality plans in about the same time, even if we are searching in a planning graph that is considerably longer than the length of the minimal length solution. This makes it possible to start with a rough upper-bound guess on the length of the solution, expand the planning graph to that level, and search it to find the solution directly, thereby avoiding costly searches on shorter planning graphs that do not contain any solution. Specifically, our heuristics allow Graphplan to solve a planning problem with a n-step solution without first proving that no m-step solution is possible for $m < n$. This is in contrast to the standard Graphplan algorithm, which, as mentioned in [2], cannot improve its performance even if given additional steps for planning.
- Our work presents an interesting way of incorporating the "distance based heuristics" popularized by McDermott's UNPOP planner [11] and Bonet & Geffner's HSP and HSP-R planners [4; 3] into Graphplan, while keeping the other advantages of Graphplan–such as the ability to generate parallel plans, and to exploit other CSP search techniques (such as local consistency enforcement strategies, EBL and DDB)

The rest of the paper is organized as follows. Section 2 re-

*Authors' names listed alphabetically. Binh Minh Do, Jillian Nottingham, Xuan Long Nguyen, Biplav Srivastava and Terry Zimmerman have all contributed significantly to this work through discussions and feedback. This research is supported in part by NSF young investigator award (NYI) IRI-9457634, ARPA/Rome Laboratory planning initiative grant F30602-95-C-0247, Army AASERT grant DAAH04-96-1-0247, AFOSR grant F20602-98-1-0182 and NSF grant IRI-9801676.

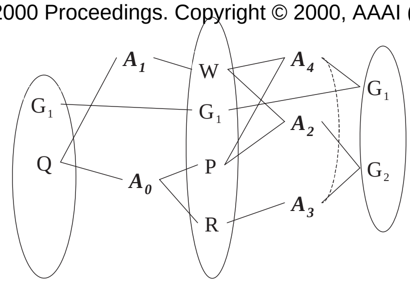


Figure 1: An example planning graph. To avoid clutter, we do not show the no-ops corresponding to the persistence of propositions Q, W, P and R.

views Graphplan, and discusses the inadequacy of traditional dynamic variable ordering heuristics, as well as the "noop first" heuristic used as default by the standard Graphplan implementations. Section 3 review the variable ordering heuristics that have been considered in the literature and discuss their shortfalls. Section 4 describes the distance-based variable and value ordering heuristics, and presents the empirical evaluation of these heuristics. Section 5 discusses the relations between our heuristics and the distance based heuristics used in UNPOP and HSP. Section 6 summarizes the contributions of the paper.

2 Background on Graphplan

Graphplan algorithm [2] can be seen as a "disjunctive" version of the forward state space planners [8]. It consists of two interleaved phases – a forward phase, where a data structure called "planning-graph" is incrementally extended, and a backward phase where the planning-graph is searched to extract a valid plan. The planning-graph (see Figure 1) consists of two alternating structures, called "proposition lists" and "action lists." Figure 1 shows a partial planning-graph structure. We start with the initial state as the zeroth level proposition list. Given a k level planning graph, the extension of structure to level $k+1$ involves introducing all actions whose preconditions are present in the k^{th} level proposition list. In addition to the actions given in the domain model, we consider a set of dummy "persist" actions, one for each condition in the k^{th} level proposition list. A "persist-C" action has C as its precondition and C as its effect. Once the actions are introduced, the proposition list at level $k+1$ is constructed as just the union of the effects of all the introduced actions. Planning-graph maintains the dependency links between the actions at level $k+1$ and their preconditions in level k proposition list and their effects in level $k+1$ proposition list. The planning-graph construction also involves computation and propagation of "mutex" constraints. The propagation starts at level 1, with the actions that are statically interfering with each other (i.e., their preconditions and effects are inconsistent) labeled mutex. Mutexes are then propagated from this level forward by using two simple propagation rules. In Figure 1, the curved lines with x-marks denote the mutex relations.

The search phase on a k level planning-graph involves checking to see if there is a sub-graph of the planning-graph that corresponds to a valid solution to the problem. This involves starting with the propositions corresponding to goals at level k (if all the goals are not present, or if they are present but a pair of them are marked mutually exclusive, the search is abandoned right away, and planning-graph is grown another level). For each of the goal propositions, we then select an action from the level k action list that supports it, such that no two actions selected for supporting two different goals are mutually exclusive (if they are, we backtrack and try to change the selection of actions). At this point, we recursively call the same search process on the $k-1$ level planning-graph, with the preconditions of the actions selected at level k as the goals for the $k-1$ level search. The search succeeds when we reach level 0 (corresponding to the initial state).

Previous work [8; 17; 9] had explicated the connections between this backward search phase of Graphplan algorithm and the constraint satisfaction problems (specifically, the dynamic constraint satisfaction problems, as introduced in [12]). Briefly, the propositions in the planning graph can be seen as CSP variables, while the actions supporting them can be seen as their potential values. The mutex relations specify the constraints. Assigning an action (value) to a proposition (variable) makes variables at lower levels "active" in that they too now need to be assigned actions.

3 Variable and Value ordering in backward search

The order in which the backward search considers the (sub)goal propositions for assignment is what we term the "goal ordering" heuristic. The order in which the actions supporting a goal are considered for inclusion in the solution graph is the "value ordering" heuristic. In their original paper, Blum & Furst [2] argue that the goal ordering heuristics are not particularly useful for Graphplan. Their argument can be paraphrased as follows–in general, Graphplan conducts backward search on a planning graph for several iterations–searching for the solution, failing, extending the planning graph by a single level, and searching for the solution again. In other words, a significant part of the search is conducted on planning graphs that *do not* contain a solution. When solving a CSP problem that does not contain a solution, variable ordering strategies, especially the static variable ordering strategies, are not expected to have much of an impact on the search efficiency, since the whole search space has to be visited anyway.[1] Blum & Furst argue, in essence, that since a variable (goal) ordering strategy is not expected to help much in failing levels, its impact on the overall efficiency of Graphplan is likely to be minimal.

There are however reasons to pursue goal ordering strategies for Graphplan:

- In many problems, the search done at the final level does account for a significant part of the overall search. Thus, it will be useful to pursue variable ordering strategies, even if they improve only the final level search.
- There may be situations where one might have lower bound information about the length of the plan, and us-

[1]Techniques like EBL [7] are however likely to help even in failing levels, as they let the search terminate faster in failing branches.

Problem	Noops first		without Noops first	
	Length	Time	Length	Time
BW-large-A	12/12	.008	12/12	.009
BW-large-B	18/18	.76	18/18	.81
BW-large-C	-	>30	-	>30
huge-fct	18/18	1.88	18/18	3.16
bw-prob04	8/20	33.5	8/18	27.96
Rocket-ext-a	7/30	1.51	7/30	.043
Rocket-ext-b	-	>30	7/29	.043
Att-log-a	-	>30	11/56	.17
Gripper-6	11/17	.076	11/17	.03
Gripper-8	-	>30	15/23	1.08
Ferry41	27/27	.66	27/27	.42
Ferry-5	-	>30	33/33	1.40
Tower-5	31/31	.67	31/31	1.00

Table 1: Experiments establishing the fallibility of the "noops-first" heuristic. Times are in minutes on a Pentium 500MHz machine with 256M ram, running linux.

ing that information, the planning graph search may be started from levels at or beyond the minimum solution bearing level of the planning graph.

3.1 Fallibility of the noops-first heuristic

The original Graphplan algorithm did not commit to any particular goal or value ordering heuristic. The implementation however does default to a value ordering heuristic that prefers to support a proposition by a noop action, if available. Although the heuristic of preferring noops seems like a reasonable heuristic (in that it avoids inserting new actions into the plan as much as possible), and has mostly gone unquestioned,[2] it turns out that it is not infallible. Our experiments with Graphplan implementations show that using noops first heuristic can, in many domains, drastically worsen the performance. Table 1 shows the results from our experiments. As can be seen, in most of the problems, considering noops first worsened performance over not having any specific value ordering strategy (and default to the order in which the actions are inserted into the planning graph). The differences were particularly striking in the rocket world and logistics domain–in both of which, performance worsens significantly with noops-first heuristic.

3.2 Ineffectiveness of "most constrained variable" first heuristic

In CSP literature, the standard heuristic for variable ordering involves trying most constrained variables first. A variable is considered most constrained if it has least number of actions supporting it. Although some implementations of Graphplan, such as SGP [17] include this variable ordering heuristic, empirical studies elsewhere have shown that by and large this heuristic leads to at best marginal improvements. In particular, the results reported in [7] show that the most constrained first heuristic leads to about 4x speedup at most.

[2] In fact, some of the extensions of Graphplan search, such as Koehler's incremental goal sets idea [10] explicitly depend on Graphplan using noops first heuristic.

4 Level-based heuristics

One reason for the ineffectiveness of most-constrained-first variable ordering is that it does not adequately capture the structure of the planning graph. In contrast to normal CSP problems, in Graphplan's backward search, the search process does not end as soon as we find an assignment for the the current level variables. Instead, the current level assignments activate specific goal propositions at the next lower level and these need to be assigned; this process continues until the search reaches the first level of the planning graph. What we need to improve this search is a heuristic that finds an assignment to the current level goals, which is likely to activate fewer and easier to assign variables at the lower levels. A strategy such as "most-constrained-variable first" that quickens the process of assigning values to the variables at the current level, may not be effective as it doesn't concern it self with the question of what types of variables get activated at the lower levels based on the current level assignments. Specifically, since a variable with fewer actions supporting it may actually be much harder to handle than another with many actions supporting it, if each of the actions supporting the first one eventually lead to activation of many more and harder to assign new variables.

A more appropriate class of heuristics are those that choose among goals based on the "distance" of those goals from the initial state where distance is interpreted as the number or actions required to go from the initial state to a state where that goal is true; under some relatively strong relaxation assumptions. It turns out that the planning graph structure itself provides a significant leverage in gauging the distances of various goal propositions. The main idea is simply this:

> Propositions that are very easy to achieve would have come into the planning graph at early levels, while those that are harder to achieve come in at later levels.

We talk about propositions "coming in to the planning graph" since once a proposition enters the planning graph at some level l, it will then be present in all subsequent levels. We formalize this intuition as follows:

> *The level of the proposition p is defined as the earliest level l of the planning graph that contains p.*

Consider the planning graph in Figure 1. In this planning graph, the level of the propositions G_1 and Q are 0, that of W, P and R are 1, and finally the level of G_2 is 2. It is easy to see that the level information can be calculated as an inexpensive by-product of planning graph expansion.

To support value ordering, i.e., the order in which actions supporting a proposition are to be considered during search, we need to define the cost of an action A supporting a proposition p. Obviously this cost will be related to the costs of the preconditions of A. This raises the question of how to define the cost of a *set* of propositions that constitute an action's preconditions. The following lists three possible alternatives, with each alternative leading to a different heuristic. All of them define the cost of a proposition the same way–*as the index of the level at which that proposition first occurs in the planning graph*.

Problem	Normal GP		Mop GP		Lev GP		Sum GP		Speedup		
	Length	Time	Length	Time	Length	Time	Length	Time	Mop	Lev	Sum
BW-large-A	12/12	.008	12/12	.005	12/12	.005	12/12	.006	1.6x	1.6x	1.3x
BW-large-B	18/18	.76	18/18	.13	18/18	.13	18/18	.085	5.8x	5.8x	8.9x
BW-large-C	-	>30	28/28	1.15	28/28	1.11	-	>30	>26x	>27x	-
huge-fct	18/18	1.88	18/18	.012	18/18	.011	18/18	.024	156x	171x	78x
bw-prob04	-	>30	8/18	5.96	8/18	8	8/19	7.25	>5x	>3.7x	>4.6x
Rocket-ext-a	7/30	1.51	7/27	.89	7/27	.69	7/31	.33	1.70x	2.1x	4.5x
Rocket-ext-b	-	>30	7/29	.003	7/29	.006	7/29	.01	10000x	5000x	3000x
Att-log-a	-	>30	11/56	10.21	11/56	9.9	11/56	10.66	>3x	>3x	>2.8x
Gripper-6	11/17	.076	11/15	.002	11/15	.003	11/17	.002	38x	25x	38x
Gripper-8	-	>30	15/21	.30	15/21	.39	15/23	.32	>100x	>80	>93x
Ferry41	27/27	.66	27/27	.34	27/27	.33	27/27	.35	1.94x	2x	1.8x
Ferry-5	-	>30	33/31	.60	33/31	.61	33/31	.62	>50x	>50x	>48x
Tower-5	31/31	.67	31/31	.89	31/31	.89	31/31	.91	.75x	.75x	.73x

Table 2: Effectiveness of level heuristic in solution-bearing planning graphs. The columns titled Level GP, Mop GP and Sum GP differ in the way they order actions supporting a proposition. Mop GP considers the cost of an action to be the maximum cost of any if its preconditions. Sum GP considers the cost as the sum of the costs of the preconditions and Level GP considers the cost to be the index of the level in the planning graph where the preconditions of the action first occur and are not pair-wise mutex.

Mop heuristic: The cost of a set of propositions is the maximum of the cost (distance) of the individual propositions. For example, the cost of A_4 supporting G_1 in Figure 1 is 1 because A_4 has two preconditions W and P, and both have level 1 (thus maximum is still 1).

Sum heuristic: The cost of a set of propositions is the sum of the costs of the individual propositions. For example, the cost of A_4 supporting G_1 in Figure 1 is 2 because A_4 has two preconditions W and P, and both have level 1.

Level heuristic: The cost of a set of propositions is the first level at which that set of propositions are present and are non-mutex. For example, the cost of A_4 supporting G_1 in Figure 1 is 1 because A_4 has two preconditions W and P, and both occur in level 1 of the planning graph, for the first time, and they do not have any mutexes between them.

Given this background, the level-based variable and value ordering heuristics are stated as follows:

Propositions are ordered for consideration in decreasing value of their levels. Actions supporting a proposition are ordered for consideration in increasing value of their costs.

The distance heuristics can be seen as using a "hardest to achieve goal (variable) first/easiest to support action (value) first" idea, where hardness is measured in terms of the level of the propositions.

It is easy to see that the cost assigned by level heuristic to an action A is just 1 less than the index of the level in the planning graph where A first occurs in the planning graph. Thus, we can think of level heuristic as using the uniform notion of "first level" of an action or proposition to do value and variable ordering.

In general, the Mop, Sum and Level heuristics can give widely different costs to an action. For example, consider the following entirely plausible scenario: an action A has preconditions $P_1 \cdots P_{10}$, where all 10 preconditions appear individually at level 3. The first level where they appear without any pair of them being mutually exclusive is at level 20. In this case, it is easy to see that A will get the cost 3 by Mop heuristic, 30 by the Sum heuristic and 20 by the Level heuristic. In general, we have: $Mop(A) \leq Sum(A)$ and $Mop(A) \leq Level(A)$, but depending on the problem $Level(A)$ can be greater than, equal to or less than $Sum(A)$. We have experimented with all three heuristics.

4.1 Evaluation of the effectiveness of level-based heuristics

We have implemented the three level-based heuristics for Graphplan backward search and evaluated its performance as compared to normal Graphplan. Our extensions were based on the version of Graphplan implementation bundled in the Blackbox system [9], which in turn was derived from Blum & Furst's original implementation. Tables 2 and 3 show the results on some standard benchmark problems. The columns titled "Mop GP", "Lev GP" and "Sum GP" correspond respectively to Graphplan armed with the Mop, Level and Sum heuristics for variable and value ordering. Cpu time is shown in minutes. For our Pentium Linux machine with 256 Megabytes of RAM, Graphplan would normally exhaust the physical memory and start swapping after about 30 minutes of running. Thus, we put a time limit of 30 minutes for most problems (if we increased the time limit, the speedups offered by the level-based heuristics get further magnified).

Table 2 compares the effectiveness of standard Graphplan (with noops-first heuristic), and Graphplan with the three level-based heuristics in searching the planning graph containing minimum length solution. As can be seen, the final level search can be improved by 2 to 4 orders of magnitude with the level-based heuristics. Looking at the Speedup

Problem	Normal GP		Mop GP		Lev GP		Sum GP		Speedup		
	Length	Time	Length	Time	Length	Time	Length	Time	Mop	Lev	Sum
BW-large-A	12/12	.008	12/12	.006	12/12	.006	12/12	.006	1.33x	1.33x	1.33x
BW-large-B	18/18	.76	18/18	0.21	18/18	0.19	18/18	0.15	3.62x	4x	5x
huge-fct	18/18	1.73	18/18	0.32	18/18	0.32	18/18	0.33	5.41x	5.41x	5.3x
bw-prob04	8/20	30	8/18	6.43	8/18	7.35	8/19	4.61	4.67x	4.08x	6.5x
Rocket-ext-a	7/30	1.47	7/26	0.98	7/27	1	7/31	0.62	1.5x	1.47x	2.3x
Rocket-ext-b	-	>30	7/28	0.29	7/29	0.29	7/28	0.31	>100x	>100x	>96x
Tower-5	31/31	0.63	31/31	0.90	31/31	0.89	31/31	0.88	.70x	.70x	.71x

Table 3: Effectiveness of level-based heuristics for standard Graphplan search (including failing and succeeding levels).

columns, we also note that all level-based heuristics have approximately similar performance on our problem set (in terms of cpu time).

Table 3 considers the effectiveness when incrementally searching from failing levels to the first successful level (as the standard Graphplan does). The improvements are more modest when you consider both failing and succeeding levels (see Table 3). This is not surprising since in the failing levels, we have to exhaust the search space, and thus we will do the same amount of search no matter which heuristic we actually use.

The impressive effectiveness of the level-based heuristics for solution bearing planning graphs suggests an alternative (inverted) approach for organizing Graphplan's search– instead of starting from the smaller length planning graphs and interleave search and extension until a solution is found, we may want to start on longer planning graphs and come down. One usual problem is that searching a longer planning graph is both more costly, and is more likely to lead to non-minimal solutions. To see if the level-based heuristics are less sensitive to these problems, we investigated the impact of doing search on planning graphs of length strictly larger than the length of the minimal solution.

Table 4 shows the performance of Graphplan with the Mop heuristic, when the search is conducted starting from the level where minimum length solution occurs, as well as 3, 5 and 10 levels *above* this level. Table 5 shows the same experiments with with the Level and Sum heuristics. The results in these tables show that Graphplan with a level-based variable and value ordering heuristic is surprisingly robust with respect to searching on longer planning graphs. We note that the search cost grows very little when searching longer planning graphs. We also note that the quality of the solutions, as measured in number of actions, remains unchanged, even though we are searching longer planning graphs, and there are many non-minimal solutions in these graphs. Even the lengths in terms of number of steps remain practically unchanged–except in the case of the rocket-a and rocket-b problems (where it increases by one and two steps respectively) and logistics problem (where it increases by two steps). (The reason the length of the solution in terms of number of steps is smaller than the length of the planning graph is that in many levels, backward search armed with level-based heuristics winds up selecting noops alone, and such levels are not counted in computing the number of steps in the solution plan.)

This remarkable insensitivity of level-based heuristics to the length planning graph means that we can get by with very rough information (or guess-estimate) about the lower-bound on the length of solution-bearing planning graphs.

A way of explaining this behavior of the level-based heuristics is that even if we start to search from arbitrarily longer planning graph, since the heuristic values of the propositions remain the same, we will search for the same solution in the almost the same route (modulo tie breaking strategy). Thus the only cost incurred from starting at longer graph is at the expansion phase and not at the backward search phase.

It must be noted that the default "noops-first" heuristic used by Graphplan implementations does already provide this type of robustness with respect to search in non-minimal length planning graphs. In particular, the noops-first heuristic is biased to find a solution that winds up choosing noops at all the higher levels–thereby ensuring that the cost of search remains the same at higher length planing graphs. However, as the results in Table 1 point out, this habitual postponement of goal achievement to earlier levels is an inefficient way of doing search in many problems. Other default heuristics, such as the most-constrained first, or the "consider goals in the default order they are introduced into the proposition list", worsen significantly when asked to search on longer planning graphs. By exploiting the structure of the planning graph, our level-based heuristics give us the robustness of noops-first heuristic, while at the same time avoiding its inefficiencies.

5 Relation to the HSP & UNPOP heuristics

The level heuristic, as described in the previous section, holds some obvious similarities to the distance based heuristics that have been popularized by McDermott's UNPOP planner [11] and Geffner & Bonet's HSP and HSP-r planners [4; 3]. In all these cases, the heuristic can be seen as computing the number of actions required to reach a state where some proposition p holds, starting from the initial state of the problem. The computation is done in a top-down demand-driven fashion (starting from the goals) in UNPOP, and in a bottom-up fashion starting from the initial state in the case of HSP and HSP-r. To make the computation of the heuristic cheap, all these systems assume that every pair of goals and subgoals are strictly *independent*. This independence assumption implies that the heuristic is not admissible. It is neither a lower bound on the true distance (since we are ignoring positive interactions between subgoals), nor is it an upper bound on the true distance (since the negative interactions are also being ignored). The fact that the heuristic is inadmissible means that

Problem	Normal GP		MOP GP		+3 levels		+5 levels		+10 levels	
	Length	Time	Length	Time	Length	Time	Length	Time	Length	Time
BW-large-A	12/12	.008	12/12	.005	12/12	.007	12/12	.008	12/12	.01
BW-large-B	18/18	.76	18/18	.13	18/18	.21	18/18	.21	18/18	.25
BW-large-C	-	>30	28/28	1.15	28/28	4.13	28/28	4.18	28/28	7.4
huge-fct	18/18	1.88	18/18	.012	18/18	0.01	18/18	.02	18/18	.02
bw-prob04	-	> 30	8/18	5.96	-	>30	-	>30	-	>30
Rocket-ext-a	7/30	1.51	7/27	.89	8/29	0.006	8/29	0.007	8/29	.009
Rocket-ext-b	-	> 30	7/29	.003	9/32	0.01	9/32	.01	9/32	.01
Att-log-a	-	>30	11/56	10.21	13/56	8.63	13/56	8.43	13/56	8.58
Gripper-6	11/17	.076	11/17	.002	11/17	0.003	11/17	.003	11/15	.004
Gripper-8	-	> 30	15/23	.30	15/23	0.38	15/23	0.57	15/23	0.33
Ferry41	27/27	.66	27/27	.34	27/27	0.30	27/27	0.43	27/27	.050
Ferry-5	-	> 30	33/31	.60	31/31	0.60	31/31	.60	31/31	.61
Tower-5	31/31	.67	31/31	.89	31/31	0.91	31/31	.91	31/31	.92

Table 4: Results showing that level-based heuristics are insensitive to the length of the planning graph being searched.

a state-space search using such heuristics does not guarantee minimal solutions. Although one can ensure admissibility by severely underestimating the cost of achievement of subgoals (for example, considering the cost of achievement of a set of propositions to be the maximum of the distances of the individual propositions, [4]), this makes the heuristic value stray too far from the true distance, and makes it ineffective in controlling search.[3] Some researchers [14] have started looking at ways of improving the differential between the distance computed by the heuristics using independence assumptions, and true distance (cost) of achieving a (sub)goal.

Given the context above, the use of level-based heuristics in Graphplan presents several interesting contrasts:

- Distance-based heuristics have hither-to been used only in the context of state-space planners. In fact, some earlier papers suggested that the very existence of distance based heuristics attests to the supremacy of state-space search over other ways of organizing the search for plans. Our work foregrounds the fact that distance heuristics, while computed from the state-space structure, are by no means restricted to a state-space planners. In fact, the distance values can potentially be used to improve search in partial order planners such as UCPOP too–we can rank a partial plan with a set of open conditions as a function of the distances of the individual open conditions of that partial plan [15].
- The use of level (distance) based heuristics in Graphplan do not have any obvious impact on the optimality of the plans generated by Graphplan. In particular, since it is used as a variable ordering heuristic for controlling CSP search, the heuristic does not directly determine the optimality.
- Using the level-based heuristics in Graphplan does not in any way compromise our ability to use other types of CSP search improvement ideas. In particular, we can still use EBL/DDB type analysis [7], or local consistency enforcement ideas [16], to further improve search. Our preliminary experiments show that EBL/DDB and level-heuristic complement each other. For example, for the 8-ball problem in the Gripper domain (Gripper-8), normal Graphplan is unable to solve the problem at all, the level heuristic allows Graphplan to solve the problem in 10 minutes, EBL/DDB alone allows Graphplan to solve it in 2 minutes, and the two techniques together solve it in 0.13 minutes. (At the time of this writing, we have not yet ported the EBL/DDB code to the C version of Graphplan. Thus, the results being discussed are from a Lisp implementation of Graphplan.)
- Unlike HSP, HSP-r and UNPOP planners, Graphplan with level-based heuristics still retains the ability to find parallel action plans. In fact, the "level" of a proposition (and action, in the case of level heuristic) already implicitly takes into account the potential parallel structure of the solution plans. If a proposition p can be made true by a single action A, which requires preconditions $q_1 \cdots q_j$, and each of these preconditions can be given by $A_1 \cdots A_j$ respectively from the initial state, then the level of p will be computed as 2. Parallelism is possible to achieve with state-space search too, but will significantly increase the branching factor, as it will force the planner to consider all subsets of non-interfering actions [8]. Empirical studies in [6] seem to suggest that it is hard to make state-search competitive with Graphplan in generating parallel plans.
- Our heuristics already exploits the mutex-propagation phase of the planning graph to improve the distance estimates. This happens in two ways. First, recall that Graphplan does not introduce actions into the planning graph if their preconditions are marked mutex in the previous level. This ensures that the level of a proposition is more truly indicative of the distance of the proposition from the initial state. In fact, mutexes ensure that the distance heuristics take a better account of the negative interactions between subgoals. While we still ignore

[3] Even taking positive interactions into account completely makes the heuristic computation intractable [4].

Problem	Lev GP						SUM GP					
	+3 levels		+5 levels		+10 levels		+3 levels		+5 levels		+10 levels	
	Length	Time	Length	Time	Length	Time	Length	Time	Length	Time	Length	Time
BW-large-A	12/12	.007	12/12	.008	12/12	0.01	12/12	.007	12/12	.008	12/12	.008
BW-large-B	18/18	0.29	18/18	0.21	18/18	0.24	20/20	0.18	20/20	0.28	20/20	0.18
BW-large-C	28/28	4	28/28	3.9	28/28	4.9	-	>30	-	>30	-	>30
huge-fct	18/18	.014	18/18	.015	18/18	.019	18/18	.014	18/18	.015	18/18	.019
bw-prob04	11/18	18.88	-	>30	-	>30	-	>30	-	>30	-	>30
Rocket-ext-a	9/28	.019	9/28	0.02	9/28	0.02	8/28	.003	8/28	.004	8/28	.006
Rocket-ext-b	9/32	.007	9/32	.006	9/32	0.01	7/28	.011	7/28	.012	7/28	.014
Att-log-a	13/56	8.48	14/56	8.18	13/56	8.45	13/56	8	13/56	8.18	13/56	8.45
Gripper-6	11/15	.003	11/15	.003	11/15	.003	11/15	.004	11/15	.004	11/15	.004
Gripper-8	15/21	0.4	15/21	0.47	15/21	0.4	15/21	0.47	15/21	0.47	15/21	0.4
Ferry41	27/27	0.30	27/27	0.30	27/27	0.34	27/27	0.30	27/27	0.30	27/27	0.34
Ferry-5	31/31	0.60	31/31	0.60	31/31	0.60	31/31	0.59	31/31	0.60	31/31	0.61
Tower-5	31/31	0.89	31/31	0.89	31/31	0.89	31/31	0.87	31/31	0.87	31/31	0.87

Table 5: Performance of Level and Sum heuristics in searching longer planning graphs

positive interactions, the damage is much less given that the planning graphs already take parallel action execution into account. Second, mutex propagation ensures that during backward search, the infeasible goal sets are detected earlier and pruned (instead of being expanded further).

While we concentrated on the use of distance-based heuristics in Graphplan, Geffner & Bonet [3] attempt to transfer the advantages of Graphplan into a state-space search setting, while using the distance heuristics. In particular, their planner, HSP-r, does a regression search in the space of (sub)goal sets, and estimates the distance of these subgoal sets from the initial state in terms of the distance heuristics augmented with static mutex analysis.

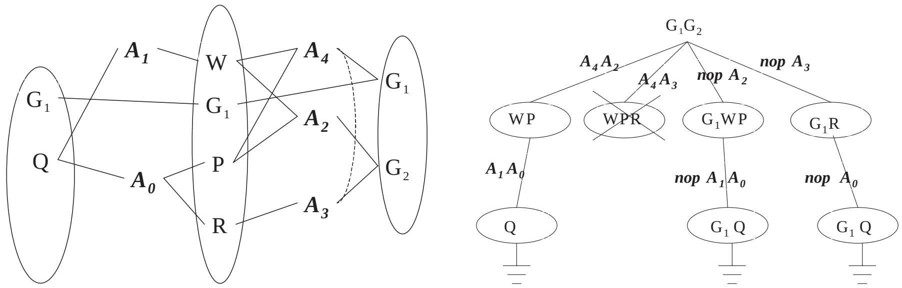
In fact, it is easy to see that modifying the Graphplan algorithm in the following ways gets us a Graphplan-variant that is essentially isomorphic to a state-space regression planner using dis-based heuristics:

1. Mark every pair of non-noop actions as mutex. This ensures that only serial plans will be produced [8] by Graphplan.
2. Modify the backward search so that it generates and searches directly in the space of subgoal sets. As illustrated in Figure 2, this is done by picking a set of actions to support all top level goals simultaneously, and then considering the union of preconditions of the chosen actions as the new subgoal set. Once all subgoal sets at a level a generated, they are ranked using one of the level-based heuristics, and the most promising subgoal set is expanded into the next level. (The tree shown in Figure 2 allows multiple non-noop actions to be applied together. Once we mark every pair of non-noop actions to be mutex however, each branch in this tree will correspond to a set of actions, all but one of which will be noops. At that point, the tree corresponds to the search tree generated by a state-space planner using regression).
3. Search in a planning graph whose length is considerably longer than the minimal length solution.

Using the normal backward search allows us to continue exploiting other CSP techniques such as EBL/DDB, and allow generation of parallel plans, while using this variant of backward search makes the whole process much closer to regression search, and thus closer to HSP-R type planners.

While the current paper discusses ways in which level information can be used to devise variable and value ordering heuristics, in [13], we consider the issue of using level information to devise state-search heuristics. In that work, we investigate the admissibility and effectiveness of a large family of state-search heuristics that compute the cost of a set of propositions using the structure of the planning graph. Three of the heuristics we investigated compute the cost of a set of propositions the way Mop, Level and Sum heuristics do. In contrast to the nearly equivalent performance of Mop, Level and Sum heuristics in backward search, in state-space search, we found these three heuristics to have significant differences in performance. Indeed, we found that the most effective (albeit inadmissible) heuristic is a combination heuristic that sums the Level and Sum heuristics together! Our best explanation for this difference is that the subgoal sets evaluated in state-space search have a more "global" character than those evaluated in guiding Graphplan's backward search. Specifically, in the normal backward search, we are evaluating subgoal sets that correspond to set of preconditions of single actions. In contrast, the subgoal sets encountered in regression search correspond to union of preconditions of the set of actions that can simultaneously support all goals in the previous level. These larger sets thus encapsulate more global interactions among the goals of the previous level, thereby presenting opportunities for the three heuristics show case their different talents.

One of the lessons of the work in [13] is that the mutex constraints in the planning graph structure are critical in gener-



(a) Planning Graph (b) Searching with subgoal sets

Figure 2: Example illustrating searching the planning graph in the space of sub-goal sets. The search in the space of subgoal sets is closely related to the search done by a state-space regression planner.

ating effective heuristics. One can view HSP-r as attempting to mimic the advantages of mutex propagation in Graphplan, without generating the planning graph. HSP-r does this with a pre-processing stage where the "eternal mutexes", i.e, propositions that are infeasible together, are computed. This computation can be seen as an indirect way of using the structure of the planning graph. An important difference is that HSP-r seems to concentrate on propositions that are never feasible together, while Graphplan's mutex relations give more finer information–in that they tell us whether two propositions are feasible together in a state reachable from the initial state in a certain number of moves (levels). As we show in [13], these level-specific mutexes can be used to significantly improve the informedness of the distance heuristics for state-space planners.

6 Conclusion

In this paper, we described a family of distance-based variable ordering heuristics, that are surprisingly effective in improving Graphplan's search. The heuristics are based on the index of the first level in which a proposition enters the planning graph, and thus uses the already existing structure of the planning graph. Empirical results demonstrate that these heuristics can speedup backward search by several orders in solution-bearing planning graphs. The results also show that the heuristics retain the efficiency of search even when searching non-minimal length planning graphs. Our heuristics, while quite simple, are nevertheless significant in that previous attempts to devise effective variable ordering techniques for Graphplan's search have not been successful. We have also discussed the deep connections between our level-based variable ordering heuristics and the use of distance-based heuristics in controlling state-space planners.

References

[1] F. Bacchus and P. van Run. Dynamic variable ordering in CSPs. In *Proc. Principles and Practice of Constraint Programming (CP-95)*, 1995. Published as Lecture Notes in Artificial Intelligence, No. 976. Springer Verlag.

[2] A. Blum and M. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2), 1997.

[3] B. Bonet and H. Geffner. Planning as heuristic search: New results. In *Proc. 5th European Conference on Planning*, 1999.

[4] B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In *In Proc. AAAI-97*, 1999.

[5] D. Frost and R. Dechter. In search of the best constraint satisfactions earch. In *Proc. AAAI-94*, 1994.

[6] P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *Proc. 5th Intl. Conference on AI Planning and Scheduling*, 2000.

[7] S. Kambhampati. Planning Graph as a (dynamic) CSP: Exploiting EBL, DDB and other CSP search techniques in Graphplan. *Journal of Artificial Intelligence Research*, 12:1–34, 2000.

[8] S. Kambhampati, E. Parker, and E. Lambrecht. Understanding and extending graphplan. In *Proceedings of 4th European Conference on Planning*, 1997. URL: rakaposhi.eas.asu.edu/ewsp-graphplan.ps.

[9] H. Kautz and B. Selman. Blackbox: Unifying sat-based and graph-based planning. In *Proc. IJCAI-99*, 1999.

[10] J. Koehler. Solving complex planning tasks through extraction of subproblems. In *Proc. 4th AIPS*, 1998.

[11] D. McDermott. Using regression graphs to control search in planning. *Aritificial Intelligence*, 109(1-2):111–160, 1999.

[12] S. Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. In *Proc. AAAI-90*, 1990.

[13] X. Nguyen and S. Kambhampati. Extracting effective and admissible state-space heuristics from the planning graph. Technical Report ASU CSE TR 00-03, Arizona State University, 2000.

[14] I. Refanidis and I. Vlahavas. GRT: A domain independent heuristic for strips worlds based on greedy regression tables. In *Proc. 5th European Planning Conference*, 1999.

[15] D. Smith. Private Correspondence, August 1999.

[16] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, San Diego, California, 1993.

[17] D. Weld, C. Anderson, and D. Smith. Extending graphplan to handle uncertainty & sensing actions. In *Proc. AAAI-98*, 1998.