Combining Progression and Regression in State-Space Heuristic Planning

Dimitris Vrakas and Ioannis Vlahavas
Department of Informatics
Aristotle University of Thessaloniki
54006, GREECE
[dvrakas,vlahavas]@csd.auth.gr
Fax: ++3031 998419

Abstract. One of the most promising trends in Domain Independent AI Planning, nowadays, is state-space heuristic planning. The planners of this category construct general but efficient heuristic functions, which are used as a guide to traverse the state space either in a forward or in a backward direction. Although specific problems may favor one or the other direction, there is no clear evidence why any of them should be generally preferred.

This paper proposes a hybrid search strategy that combines search in both directions. The search begins from the *Initial State* in a forward direction and proceeds with a weighted A* search until no further improving states can be found. At that point, the algorithm changes direction and starts regressing the *Goals* trying to reach the best state found at the previous search. The direction of the search may change several times before a solution can be found. Two domain-independent heuristic functions based on ASP/HSP planners enhanced with a *Goal Ordering* technique have been implemented. The whole bi-directional planning system, named BP, was tested on a variety of problems adopted from the recent AIPS-00 planning competition with quite promising results. The paper also discusses the subject of domain analysis for state-space planning and proposes two methods for the elimination of redundant information from the problem definition and for the identification of independent sub-problems.

Keywords: Planning, Heuristic Search, Bi-Directional Search

1. Introduction

Motivated by the work of Drew McDermott in the mid 90's on heuristic state-space planning, a number of researchers turned to this direction. During the last few years a great amount of work has been done in the area of domain-independent, state-space, heuristic planning and a significant number of planning systems with remarkable performance were developed.

Hector Geffner in his recent work on HSP-2 [3] studies the matter of search direction and the HSP-2 planning system is able to plan in both directions. It is clear from the experimental results that there are specific problems, which favor one or the other search directions, but in general there is no clear evidence why any of the two directions should be preferred.

In this paper we propose a hybrid search strategy for domain-independent, state-space heuristic planning that combines both progression (forward chaining) and regression (backward chaining). The search begins from the *Initial State* and proceeds with a weighted A* search until no further improving states can be found from the

Goals. At that point the algorithm changes direction and regress the *Goals* trying to reach the best state found at the previous search. The direction of the search may change several times before a solution can be found.

Two domain-independent heuristic functions based on ASP/HSP enhanced with a *Goal Ordering* technique were implemented and the whole bi-directional planning system, named BP, was tested on a variety of problems adopted from the recent AIPS-00 planning competition with quite promising results.

This paper also discusses the subject of automatic domain analysis and the utilization of the information extracted from it during the planning process. We propose two methods, based on the planning graphs created by the heuristic functions of BP, which can identify valuable information about the internal structure of the problems. The methods are utilized in BP for eliminating redundant information from the definition of the domain and for dividing the problem in a number of easier sub-problems that can be tackled in parallel.

The rest of the paper is organized as follows: Section 2 provides a brief review of the work related to state-space, heuristic planning and other approaches to bi-directional planning. Section 3 describes the bi-directional search strategy in detail and deals with certain issues that arise while regressing the goals of a problem. Section 4 describes the heuristic functions of BP and describes the adoption of a Goal Ordering technique to heuristic state space planning. Section 5 presents experimental results that illustrate the efficiency of BP on a variety of problems adopted from the AIPS-00 planning competition. Section 6 discusses a number of domain analysis techniques that result from the construction of planning graphs and section 7 concludes the paper and poses future directions.

2. Related Work

One of the most promising trends in domain-independent planning was presented over the last few years. It is based on a relatively simple idea where a general domain independent heuristic function is embodied in a heuristic search algorithm such as Hill Climbing, Best-First Search or A*. A detailed survey of search algorithms can be found in [9]. Examples of planning systems in this category are UNPOP[13], the ASP/HSP family [2,3,4], GRT[16], AltAlt[14] and FF[7].

The first planner in this category was UNPOP[13], a regression planner that constructs at each step a graph, named *greedy regression-match graph*. The graph is used in the search, for creating the heuristic function and cutting down the branching factor by pruning certain actions.

The direct descendant of UNPOP was HSP[4], which searches the state space in the forward direction though and constructs a more sophisticated heuristic function from a similar graph. HSP was followed by HSP-R[2], a similar planer with two main differences from HSP. The search is performed in a backwards manner and the heuristic is created in the opposite direction, which enables HSP-R to create the planning graph only once. HSP and HSP-R were later embodied in a unifying planning system called HSP-2[3]. GRT [18] is another extension to HSP that searches in the forward direction and creates the heuristic backwards once at the beginning.

Nigenda, Nguyen and Kambhampati presented a hybrid planning system, named AltAlt [14], which was created using programming modules from STAN [10] and HSP-R. In the first phase, AltAlt uses the module from STAN to create a planning

graph, from which it extracts a heuristic function that is used to guide the backward hill-climbing search, which is performed in an HSP-R manner.

One of the latest planners in this category and the most effective according to the results of the AIPS-00 planning competition[1] is Hoffmann's FF planning system [7]. FF constructs a graph similar to this of GRAPHPLAN from which it extracts a sketch plan. The sketch plan is then used in a forward enforced hill-climbing search in two ways. Firstly, the length of the sketch plan is used as an estimate for the distance between the *Initial* state and the *Goals* and secondly a set of *helpful* actions is extracted which helps in cutting down the branching factor of the search.

Bi-directional search is a well-known search strategy mentioned in almost any textbook about Artificial Intelligence. However, it has not been broadly adopted as a search strategy. Especially in planning, there are only a few systems performing a combined search in both directions. The only bi-directional planners that have been developed, to the knowledge of the authors, are PRODIGY [19], NOLIMIT[18], FLECS[20] and RASPUTIN[6]. All of these planners have been developed by researchers of the Carnegie Mellon University's PRODIGY project and are based on the combination of goal-directed backward chaining with simulation of plan execution [18]. Although these planners perform some kind of search in both directions, they are actually forward-direction planners, which utilize the backward search as an action selection mechanism.

3. The Search Strategy of BP

The planners presented in the previous section have shown quite impressive performance and they have proved to be able to handle a large variety of difficult problems. However, they usually present variations in their efficiency among different domains or even between problems of the same domain.There are two main reasons that justify this behavior:

a) Although the heuristic functions constructed by all the planners are general, they seem to work better with specific domains.
b) There are domains and problems that clearly favor one of the two search directions (forward or backward).

The first argument, which is also a conclusion drawn from the experience of the authors, has been stated by Stone, Veloso and Blythe in [17]. The second argument is the main conclusion drawn by Bart Massey in an extensive study in the directions of planning presented in [11]. Bonet and Geffner have pushed the same argument one step further: "*Although we don't fully understand yet when HSP will run better than HSP-R, the results suggest nonetheless that in many domains a bi-directional planner combining HSP-R and HSP could probably do better than each planner separately*" [2]. The answer to the question posed by Bonet and Geffner above has been answered by Massey in [11], where the planning problems are discriminated into forward and backward problems, in the sense that strongly directed planners will find the problems of the opposite direction intractable.

Motivated by the conclusions stated above we developed BP, a heuristic state-space planner, which combines search in both directions. A part of the plan is

[1] A complete review of the participating systems, the domains and the results of the AIPS-00 competition can be found at the URL: http://www.cs.toronto.edu/aips2000/

constructed with the progression module (forward chainer) and the rest is constructed with the regression module (backward chainer). The sub-plan of the regression module is inversed and merged with that of the progression module in order to produce the final plan. However the case is not always that simple, because usually BP interleaves the execution of both modules several times before a solution is found. Details about the search strategy are presented later in this section, but we will first describe the progression and the regression search modules.

3.1 The Progression Module

The progression module employs a best-first search method starting from the initial state and moving forward trying to reach the goals with two main differences:

a) the size of the planning agenda is limited by an upper limit *SOF_AGENDA*. This means that if there are N states (N>SOF_AGENDA) only the SOF_AGENDA most promising (according to *h*) states will be stored and the rest will be pruned. This fact sacrifices completeness but it is necessary, since otherwise a lot of problems would become intractable, due to memory limitations.

b) The progression module will stop the search when it is not further possible to move to a state, the distance of which is not greater than the distance of the current state plus *T*. This part of the algorithm is crucial to the unified bi-directional search strategy, since the value of *T* determines how frequently will the algorithm change the search direction.

The progression module takes five arguments, which are: a) the initial state *I'* of the sub-problem, b) the goals *G'* of the sub-problem, c) the maximum size *SOF_AGENDA* of the planning agenda d) a threshold *T* declaring when the search should stop and e) a heuristic function *h* capable of estimating distances between states. The progression module returns a new state *S*, which is the state closer to *G'* that the module could find.

3.2 The Regression Module

The algorithm of the regression module is quite similar to the one of the progression module, since the search strategy is symmetric. The only differences are in the way of finding the applicable actions and forming the successor states. The regression module makes extensive use of binary mutual exclusions between facts. Two facts *p* and *q* are mutual exclusive, denoted as *mx(p,q)*, if no valid state contains both of them at the same time. Mutual exclusions are calculated in a way similar to the one they are calculated in GRAPHPLAN [1].

An action *A* is backward applicable to state *S* if *S* contains at least one add and no del effects of *A* and there are no mutual exclusions between facts of *S* and the facts added by *A*. State *S'* is produced from the backward application of action *A* to state *S* by removing the del effects of *A* from *S* and adding the add ones.

3.3 Combining the two Search Modules

The underlying framework of the bi-directional search strategy is based on a relatively simple idea. Usually, single-directional planners reach a point in the search process where the heuristic function becomes less informative. Two of the main reasons that justify this behavior are: a) the branching factor of the current sub-problem is too large for the heuristic to produce accurate estimates b) the sub-problem is much too complex and the heuristic function becomes obsolete as the search goes on, especially when it is constructed only once at the beginning.

In contrast, BP constructs the heuristic function in the backward direction and starts performing a forward directed search until it reaches a state S_B from where it is difficult to proceed. Then it reconstructs the heuristic function in the opposite (forward) direction and starts searching, in the opposite direction (backward), from the *Goals* towards S_B. If the backward search is also blocked after some steps in a state S_{B2}, BP will restart the planning process replacing the *Initial* state with S_B and the *Goals* with S_{B2}. The value of the *Threshold* is increased each time a search module returns without improving its initial state. The bi-directional search strategy of BP is outlined in Figure 1, where *St.plan* represents the plan from the *Initial* state to *St* for the progression module and the plan from the *Goals* to *St* for the regression one.

```
Search Algorithm of BP
Input I, G, Output Plan
Plan1=Plan2=[],S = I,F = G,Direction=Forward,Threshold=Init_Thr
While F⊄S
Begin
  If Direction = Forward
  begin
    Create backward heuristic function h_B
    St=Progression_module(S,F,MAX_SOF_AGENDA,Threshold,h_B)
    If St≠S Plan1=Plan1+St.plan, Threshold=Init_Thr, S=St
     Else Threshold=Threshold+STEP
     Direction = Backward
  end
  Else
  begin
    Create forward heuristic function h_F
    St=Regression_module(S,F,MAX_SOF_AGENDA,Threshold,h_F)
    If St≠F Plan2=Plan2+ St.plan,Threshold=Init_Thr,F=St
     Else Threshold=Threshold+STEP
     Direction = Forward
  End
end
Return Plan1+reversed(Plan2)
```

Figure 1: The Search algorithm of BP

There are two reasons that enable BP to face the problems stated above: a) the change in the direction enables BP to update the heuristic function, b) due to the adaptive way in which BP changes directions, it tends to solve the major part of the problem in the direction, which best fits it.

4. BP's Heuristic Functions

In order to test the efficiency of the bi-directional search strategy, we developed two, relatively simple, domain independent heuristic functions that were embedded in the BP planning system. The two heuristic functions are quite similar and are based on exactly the same idea, but the first one is used for the progression module and the other for the regression one. Note here, that both search modules of BP adopt a weighted A* search strategy, where the total cost of a state S is calculated as: $w_1*L(S)+w_2*h(S)$. In this formula, $L(S)$ is the number of steps needed to reach state S

starting from the Initial state (Goals in case of regression), $h(S)$ is the value returned by the heuristic function and w_1 and w_2 are user-defined constants.

4.1 The Progression Heuristic Function

The heuristic function used for the progression module is similar to the one of the GRT planning system. The heuristic function is extracted from a leveled graph, similar to the one built by GRAPHPLAN. The graph consists of all the subgoals of the domain (the action levels of the GRAPHPLAN are omitted) that are generated from the Initial state, tagging them with a number K, identifying the minimum number of steps needed to generate them starting from the Goals.

The graph construction begins from the Goals of the problem (level 0) and proceeds backwards adding iteratively a new level L with all the subgoals that are generated by actions that are applicable at level L-1. An action A is applicable at level L-1, if at least one add-effect of A exists in L-1. For each action A that is applicable at level L-1, the algorithm computes a value V as the sum of the tags of all the facts in add(A). The facts in its precondition list are then added at level L and tagged with V+1 if they have not already been tagged with a smaller value than V+1. The expansion of the graph iterates until the graph reaches level L_{MAX}, where no more subgoals can be generated with a cost smaller than the one in their tags.

After the creation of the graph, which is done only once as long as the planner does not change direction, the tags of the facts are used to produce estimates for the distance between any state S in the domain and the goals, just by summing up the tags of the facts in S.

4.2 The Regression Heuristic Function

As stated earlier in this section, the regression heuristic function is similar to the progression one and just differs in the direction in which the graph is created. The graph for the regression is built starting from the facts in the Initial state (level 0) and proceeds forwards until it reaches a level L_{MAX}, where no more facts can be added to the graph with a cost smaller than the one in their tags. Here, an action A is applicable at level L, if all the preconditions of A exist in L.

4.3 Refining the heuristic functions with Goal Ordering

Goal ordering for planning has been an active research topic over the last years and a number of techniques have been successfully adopted by state-of-the-art planning systems. The research so far has been focused on two tasks: a) how to automatically extract as much information as possible about orderings among the goals of the problem, with minimum computational cost and b) how to use this information during planning. McCluskey and Porteous with their work on PRECEDE[12] proposed a method for identifying goal orderings between pairs of atomic facts, based on direct domain analysis. The more recent work of Koehler and Hoffman on GAM [8] have resulted in two techniques for identifying goal orderings, one based on domain analysis and another utilizing the information gained by the construction of a planning graph. The simplest and yet quite effective orderings extracted by these techniques have been described as *reasonable orders* and are based on the following idea:

> *"A pair of goals A and B can be ordered so that B is achieved before A if it isn't possible to reach a state in which A and B are both true, from a state in which A is true, without having to temporarily destroy A."* [15].

IPP [8] and FF[7] make use of reasonable orderings during planning through the construction of a goal agenda that divides the goals into an ordered set of sub-goals. The planners sequentially achieve the first sub-goal in the agenda, which has not yet been achieved. Experimental results have shown that the use of the goal agenda yields in significance improvement in terms of both planning time and plan quality.

BP adopts a slightly different method to compute reasonable orderings between goals, which is based on mutual exclusions between facts of the domain. Since the planner calculates the set of binary mutual exclusions, in order to use them for the regression phase, the overhead imposed by the calculation of reasonable orderings is negligible. Function *OB* (Ordered Before), which is outlined in Figure 2, is iteratively ran on every pair of goals in order to identify the possible orderings between the goals of the problem.

```
Function OB
Input: Goals a and b
Output: True (a should be ordered before b) or False
For each action O: a∈add(O)
begin
    MutexPre=false
    For each fact f: f∈prec(O)
        If mx(b,f)=true MutexPre=true
    If MutexPre = false return false
end
Return true
```

Figure 2: The OB Function

The orderings extracted by OB are used in the planning phase, in order to refine the results of the heuristic functions and not to divide the goals into sub-sets. More specifically, after the evaluation of a state *S* by one of the two heuristic functions, as exemplified by sub-sections 4.1 and 4.2, BP searches state *S* for possible violations of the goal orderings. Fact f of a state S is violating the goal ordering if:

$f \in$ Goals and $\exists$ goal g: $g \notin S$ and OB(g,f)=true

For every ordering violation in state S, the estimated distance between S and the Goals is increased by a constant number, since at a later point the ordering breaches will have to be destroyed and re-achieved after the correct ordering has been reinstated.

5. Experimental Results

In order to test the efficiency of BP we implemented two additional planners: a) PMP (Progression Module Planner), a progression planner using the progression module and heuristic function of BP and b) RMP (Regression Module Planner), a regression planner using the regression module and heuristic function of BP. The search modules in PMP and RMP were slightly modified, so as to continue their search until a solution is found. The three planning systems were tested on a large variety of problems adopted by the recent AIPS-2000 planning competition.

The codes of the planners were based on the publicly available code of the second version of GRT and were implemented in C++. All the tests were run on a SUN ENTERPRISE 3000 parallel computer, with two SPARC-1 processors at 167 MHz and 256 MB of RAM. The underlying operating system was SUN Solaris 2.6

and the programs were compiled by GNU C++ compiler. For the tests we have chosen the following configuration for the three planners:
`MAX_SOF_AGENDA`=200, `Init_Thr` = 2, `STEP` = 2, w_1=0.4 and w_2=1.0.

The three planners (PMP, RMP and BP) were tested on all problems of the *blocks world*, the *logistics*, the MIC-10 and the *freecell* domains used in the AIPS-00 planning competition. Tables 1, 2, 3 and 4 present the results of the tests. Columns 1,2 and 3 present the number of steps of the plan found by each planner and the time needed to solve the problem (in brackets). Note that short dashes mean that the problem could not be solved within the 180 seconds limit in CPU time set on all planners. Plan lengths written in bold note the minimum plan length found by the three planners. Note that due to space limitations, tables 1, 2, 3 and 4 present only a part of the tested problems.

5.1 Blocks world

It is clear from table 1 that the specific problems favor regression planners. RMP was able to solve 47% more problems than PMP producing in all problems shorter plans in much less time, while BP presented results quite similar to RMP. Specifically, BP solved 1 problem less than RMP, producing 16% longer plans, spending though 45% less time on average. BP clearly outrivaled PMP, producing 67% shorter plans and spending almost 20 times (1930%) less time on average.

5.2 Logistics

In the *logistics* problems the choice of planning direction didn't seem to play a very important role. RMP and BP solved more problems than PMP, but produced slightly worse plans than the latter. Specifically, BP produced 3% longer plans than PMP but solved 32% more problems in 7% less time on average. RPM solved the same number of problems with BP and was quite faster (9%) than the latter. However the plans it produced were 5% longer than those of BP.

Prob	PMP	RMP	BP
4-0	**6** (40)	**6** (40)	**6** (60)
5-0	**12** (790)	**12** (100)	**12** (110)
6-0	42 (14790)	**12** (140)	18 (270)
6-2	54 (15820)	24 (640)	**22** (260)
7-0	44 (12930)	**20** (410)	22 (390)
7-1	70 (13990)	**22** (430)	24 (440)
7-2	106 (42890)	**20** (530)	22 (410)
8-1	88 (27360)	**20** (490)	30 (850)
8-2	18 (250)	**16** (410)	**16** (390)
9-1	-	**30** (5150)	**30** (3570)
9-2	-	**26** (5130)	28 (1740)
10-1	-	**38** (21350)	42 (7490)
10-2	-	-	114 (40200)
11-0	62 (5270)	**34** (4730)	78 (8490)
11-1	-	**30** (2080)	-
11-2	-	-	**220** (125030)
12-0	-	**34** (5040)	48 (8680)
12-1	-	**38** (18380)	-
13-1	-	-	-
14-0	-	**38** (8170)	-
14-1	-	**36** (5910)	-
17-0	-	-	**50** (59280)

Table 1: Plan length and solution time (in msec) for Blocks world problems

Prob	PMP	RMP	BP
4-0	**20** (150)	23 (240)	**20** (240)
5-0	**27** (220)	32 (350)	**27** (350)
6-0	**25** (210)	33 (440)	28 (450)
7-1	62 (10730)	51 (1340)	**50** (1830)
8-0	**31** (640)	41 (1230)	37 (1340)
9-1	**32** (590)	34 (1230)	**32** (1320)
10-0	74 (29650)	54 (3790)	**50** (5010)
12-0	**45** (4590)	51 (3820)	**45** (4960)
13-1	-	83 (17620)	**81** (24160)
14-0	-	**78** (15490)	79 (31140)
14-1	**76** (5970)	93 (20410)	87 (30120)
15-0	112 (70170)	**95** (21480)	106 (38620)
15-1	**76** (20240)	85 (20070)	82 (24410)
16-0	-	**109** (35820)	112 (47130)
16-1	**83** (10330)	108 (40890)	85 (16430)
17-0	-	116 (41350)	**114** (53130)
17-1	-	**120** (45650)	**120** (85490)
18-1	104 (90900)	**101** (46410)	102 (59810)
19-1	-	119 (70160)	**113** (89260)
20-0	-	**127** (73580)	138 (116440)
20-1	**110** (22620)	123 (66510)	112 (33480)

Table 2: Plan length and solution time (in msec) for Logistics problems

5.3 MIC-10

MIC-10 is a domain that clearly favored progression planners, as shown by the experimental results. RMP was unable to solve problems harder than s4-1, while PMP and BP solved almost every problem of the domain. We tried to increase the size of the planning agenda for RMP and as a result the planner was able to solve a few more problems (up to s5-2) but this had a negative impact on planning time. Concerning the other two planners, BP clearly outperformed PMP by solving 7.5% more problems in 35% less time on average and by producing 10% shorter plans.

5.4 Freecell

Like the *logistics* domain, *freecell* does not clearly favor a specific planning direction. However PMP seemed to perform better than RMP and this is probably due to the fact that there is too much implied information that is omitted from the goals. In this domain BP solved 3 problems more than RMP and 2 less than PMP, producing plans of lower quality (approx. 6% longer plans) than both RMP and PMP. However, concerning planning time, BP clearly outperformed the other two, needing 35% less time than PMP and 614% less time than RMP on average.

Prob	PMP	RMP	BP
S1-0	**4 (0)**	**4 (10)**	**4 (20)**
S3-0	12 (40)	**10** (320)	11 (40)
S4-0	16 (100)	-	**15** (100)
S10-0	40 (1980)	-	**36** (1390)
S11-0	41 (2400)	-	**39** (1580)
S12-0	48 (4280)	-	**41** (2500)
S15-0	60 (8710)	-	**52** (5130)
S17-0	67 (12840)	-	**62** (7880)
S17-4	65 (14280)	-	**57** (8040)
S18-4	70 (16050)	-	**62** (9140)
S19-0	-	-	**66** (11520)
S19-2	74 (26090)	-	**64** (11990)
S19-3	-	-	**67** (12100)
S19-4	77 (21910)	-	**66** (11570)
S20-0	-	-	**70** (16560)
S20-1	84 (28640)	-	**71** (13450)
S20-2	79 (23800)	-	**65** (12850)
S20-3	-	-	**72** (14770)
S20-4	-	-	**70** (15570)

Table 3: Plan length and solution time (in msec) for MIC-10 problems

Prob	PMP	RMP	BP
2-1	**9** (3920)	**9** (37660)	11 (4240)
2-2	**8** (3910)	**8** (34110)	9 (4150)
2-3	**8** (3510)	**8** (38860)	9 (3940)
2-4	**8** (4110)	**8** (32920)	9 (4150)
2-5	**9** (3930)	**9** (33770)	11 (4390)
3-1	18 (18990)	**15** (88360)	18 (43870)
3-2	**17** (20190)	19 (100650)	19 (15000)
3-3	16 (30130)	19 (90870)	**15** (10580)
3-4	15 (14950)	**13** (71910)	**13** (9990)
3-5	**16** (38760)	**16** (83530)	17 (19310)
4-1	**27** (106590)	-	28 (93750)
4-2	24 (25090)	**21** (150020)	-
4-3	**28** (78620)	-	38 (86140)
4-4	26 (67300)	**19** (127580)	-
4-5	30 (100620)	-	**24** (27880)
5-1	-	-	-
5-2	**28** (159790)	-	-
5-3	-	-	**46** (152930)
5-4	**29** (85080)	-	39 (153630)

Table 4: Plan length and solution time (in msec) for Freecell problems

6. Domain Analysis through Planning Graphs

The relaxed planning graphs built by the heuristic functions of BP, as described in sections 4.1 and 4.2, can be used as a means for extracting valuable information about the structure of the domain. This information can then be used in various ways, such as a) removing redundant information from the definition of the problem and thus cutting down the branching factor and b) identifying independent sub-goals that can be solved in parallel.

6.1 Simplifying the Definition of the Problem

The simplification of the problem's definition concerns facts of the *Initial* state that are useless for the planning process. For example, in the *logistics* domain the *Initial* state may contain facts noting the initial location of packages, which do not need to be

moved. This information is present in the description of the world, for means of completeness, but increases the branching factor of the problem with useless actions.

BP employs a simple but efficient method for eliminating useless facts from the *Initial* state, which is based on the backward graph built by the progression heuristic function. After the initial construction of the backward graph, the method eliminates all the facts of the *Initial* state that do not appear in the graph. If a fact f does not appear in the backward graph, there is no way to reach a state S where $f \in S$, by regressing the *Goals*. This means that the facts added by the actions that have f in their preconditions, are not present in the graph too and therefore f does not contribute at all in the process of reaching the *Goals* of the problem. So it is safe to remove it from the *Initial* state without jeopardizing completeness.

There is no actual overhead imposed by the above method, since the backward graph is built by the progression heuristic function no matter if the method for eliminating useless facts is applied or not. As far as the efficiency of the method is concerned, the method is not complete, in the sense that it does not identify all the forms of information that could be safely removed from the definition of the problem.

6.2 Identifying Independent Sub-Goals

Motivated by the results of the method for eliminating useless facts from the *Initial* state, we developed a second method for identifying independent sub-goals that can be solved in parallel. Given the definition of a problem $<I,A,G>$, the method iteratively builds N (size of G) backward planning graphs, removing at each time one of the goals in G. It follows from the method in the previous sub-section that if we remove goal G_k from G, then the facts in I (noted as I_k), that do not appear in the backward graph of G-$\{G_k,\}$ are closely related to G_k and they are not needed for the achievement of the rest of the goals. After the creation of the graphs, the algorithm comes up with a number L of sets of the form $<I_k,G_k>$, which indicate that the set I_K of initial facts is only needed for the achievement of goal G_K.

Consider, for example, a *logistics* problem with a city consisting of two locations (`airport` and `center`), two trucks (`tr1` and `tr2`) and three packages (P_1, P_2 and P_3) that have to be moved. The initial state and the goals of the problem are:

I={at(P_1,center),at(P_2,center),at(P_3,center),at(tr1,center),at(tr2,center)}

G={at(P_1,airport),at(P_2,airport),at(P_3,center)}

The independent goals and the sub-sets of G used for the graphs are the following:

G_1=at(P_1,airport), G-{G_1}={at(P_2,airport),at(P_3,center)}

G_2=at(P_2,airport), G-{G_2}={at(P_1,airport),at(P_3,center)}

G_3=at(P_3,center), G-{G_3}={at(P_1,airport),at(P_2,airport)}

The three subsets of the *Initial* state that are extracted from the backward graphs are the following:

I_1={at(P_1,center)}

I_2={at(P_2,center)}

I_3={at(P_3,center)}

So the sets extracted by our method are the following:

S_1=<{at(P_1,center)}, at(P_1,airport)>

S_2=<{at(P_2,center)}, at(P_2,airport)>

S_3=<{at(P_3,center)}, at(P_1,center)>

At a first step, these sets can be used for further elimination of redundant facts from the problem's definition. If for a given set $<I_k,G_k>$, the size of I_k is equal to 1 and

$I_k \equiv \{G_k\}$, it is safe to remove G_k from G and I_k from I and discard $<I_k,G_k>$. This is exactly the case with set S_3 of the previous example, where it is obvious that fact `at(P3,center)` could be safely removed from I and G.

The second use of the sets of the form $<I_k,G_k>$ is for dividing G in sub-goals the plans of which that can be combined in a parallel plan. The benefits of such a method are: a) a speedup in the planning process since tackling each sub-problem independently is usually easier than tackling the whole problem as one and b) a parallel plan which will probably be executed in less time than any sequential one.

The first argument is also supported by experimental results. We developed a planner, named BP-SP (Bi-directional Planner for Sub Problems), which uses the above method for dividing the initial problem into a number of sub-problems and then uses BP for solving the sub-problems sequentially. Supposing that we have L sets of the form $<I_k,G_k>$ and I and G are the initial state and the goals respectively, BP-SP starts with the sub-problem $<I',A,G'>$ where:

$$I' = I - \bigcup_{k=2}^{L} I_k \ , \ G' = \{G_1\}$$

If BP manages to solve $<I',A,G'>$ it will encounter a state S' where $G' \subseteq S'$. S' will be used for the next iteration of BP-SP to form the new sub-problem $<I'',A,G''>$ as follows:

$I''=S'-\{G_1\} \cup I_2$, $G''=\{G_2\}$

This process is repeated L times and the overall plan returned by BP-SP is constructed by concatenating the plans of the sub-problems.

The two planners (BP and BP-SP) were tested on all 80 *logistics* problems of the AIPS-00 planning competition. BP-SP managed to solve 66 problems, 20 more than BP, and it was almost 4 times faster on average. The resulted plans were approximately 30% longer but this could be probably overcome by a planner utilizing the fact that the sub-plans of BP-SP can be combined in a parallel plan. It lies in our future plans, to enhance BP with the ability to handle parallel problems.

7. Conclusions and Future Work

It is a generally accepted fact that there are certain domains or certain problems of domains that can be tackled more efficiently by forward planners and others that can be tackled more efficiently by backward ones. The matter of direction in planning is an active field of research and yet no clear answer has been given to the question of which direction should be generally preferred. This paper proposed BP, a hybrid planning system that combines search in both directions. BP changes the search direction in an adaptive way, which enables it to solve the major part of the problems in the direction that best fits them.

BP has been tested on a variety of problems used in the AIPS-00 competition and has been compared to two single-direction planners (a forward and a backward) that utilize the same heuristic function and optimization techniques. Experimental results have shown that BP has a stable performance on all domains, outperforming, in general, both of the single-direction planners.

It is in our future plans to develop a more sophisticated heuristic function and embody it in BP, along with several optimization techniques extracted from automatic domain analysis. Our experience with BP has shown that a large amount of useful

information can be extracted from the combination of planning graphs in both directions and this information can be used to construct efficient optimization techniques, such as the two methods discussed in section 6.

8. References

[1] Blum, L., and Furst M., 1995, Fast planning through planning graph analysis, In *Proc., 14th Int. Joint Conference on Artificial Intelligence*, Montreal, Canada, pp. 636-642.

[2] Bonet, B. and Geffner, H. 1999, Planning as Heuristic Search: New Results, In Proceedings, ECP-99, Durham UK.

[3] Bonet, B. and Geffner, H. 2000, Planning as Heuristic Search, Artificial Intelligence, forthcoming.

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

[5] Fikes, R., and Nilsson, N., 1971, STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, **2**: 189-208.

[6] Fink, E. and Blythe, J. 1998, A complete bidirectional planner, In proceedings, 4th International Conference on AI Planning Systems.

[7] Hoffmann, J. 2000, A Heuristic for Domain Independent Planning and its Use in an Enforced Hill-climbing Algorithm, 12th Int. Symposium on Methodologies for intelligent Systems.

[8] Koehler, J. and Hoffmann, J. 2000, On Reasonable and Forced Goal Orderings and their Use in an Agenda-Driven Planning Algorithm, JAIR (12).

[9] Korf, R. 1998, Artificial intelligence search algorithms**,** CRC Handbook of Algorithms and Theory of Computation**,** Atallah, M. J. (Ed.), CRC Press, Boca Raton, FL, pp. 36-1 to 36-20

[10] Long, D. and Fox, M. 1998. Efficient Implementation of the Plan Graph in STAN, JAIR, 10, pp. 87-115.

[11] Massey, B. 1999, Directions In Planning: Understanding the Flow of Time in Planning, Available as a Technical Report from the University of Oregon.

[12] McCluskey, T. and Porteous, J. 1997, Engineering and Compiling Planning Domain Models to Promote Validity and Efficiency, Artificial Intelligence, 95.

[13] McDermott, D. 1996, A Heuristic Estimator for Means-End Analysis in Planning, In Proceedings, AIPS-96

[14] Nguyen, X., Kambhampati, S. and Nigenda, R. 2000, AltAlt: Combining the advantages of Graphplan and Heuristics State Search, In Proceedings, 2000 International Conference on Knowledge-based Computer Systems, Bombay, India.

[15] Porteous, J. and Sebastia, L., 2000, Extracting and ordering Landmarks for Planning, In Proceedings, 18th Workshop of the UK Planning and Scheduling SIG

[16] Refanidis, I., and Vlahavas, I., 1999, *GRT:* A Domain Independent Heuristic for STRIPS Worlds based on Greedy Regression Tables, In *Proceedings, 5th European Conference on Planning*, Durham, UK, pp. 346-358.

[17] Stone, P., Veloso, M. and Blythe, J. 1994, The Need for Different Domain-Independent Heuristics, In proceedings, AIPS-94, Chicago, USA.

[18] Veloso, M. 1994, Planning and learning by Analogical Reasoning, PhD Dissertation, Springer-Verlag.

[19] Veloso, M., Carbonell, J., Perez, A., Borrajo, D., Fink, E. and Blythe, J. 1995, Integrating Planning and Learning: The PRODIGY Architecture, Journal of Experimental and Theoretical Artificial Intelligence, 7(1).

[20] Veloso, M. and Stone, P. 1995, FLECS: Planning with a Flexible Commitment Strategy, JAIR (3).