Bi-Directional Heuristic Planning in State-Spaces

Dimitris Vrakas and Ioannis Vlahavas
Department of Informatics
Aristotle University of Thessaloniki
[dvrakas,vlahavas]@csd.auth.gr

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 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 step. 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.

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.

These planners usually adopt the STRIPS [] notation. A planning problem in STRIPS is a tuple $<I,A,G>$ where I is the Initial state, A a set of available Actions and G a set of goals. States in STRIPS are represented as sets of atomic facts. Each action A has three lists of facts containing:

a) the preconditions of A (noted as *prec(A)*)
b) the facts that are added to the state (noted as *add(A)*) and
c) the facts that are deleted from the state (noted as *del(A)*).

- An action A is applicable to a state S' if prec(A) $\subseteq$ S'.
- If A is applied to S, the following formula holds for the successor state S':
  S' = S / del(A) $\cup$ add(A)
- The solution to such a problem is a sequence of actions, which if applied to I leads to a state S' such as $S' \supseteq G$.
- If no solution can be found for a problem, the problem is characterized as unsolvable.

Although the efficiency of these planners depends strongly on the accuracy of their heuristic function, the direction in which they search the space of states plays an important role as shown by experimental results. Planners as UNPOP, GRT and HSP/ASP progress the *Initial State* until they have reached the goals. On the contrary, HSP-R and AltAlt regress the *Goals* of the problem until they have reached the *Initial State*.

Hector Geffner in his recent work on HSP-2 studies the matter of search direction and the HSP-2 planning system enables the user to decide for the direction of the search. 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 step 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.

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 details and deals with certain issues that arise while regressing the goals of a problem. Section 4 describes the heuristic functions of BP, proposes a fast and efficient technique for the elimination of useless information from the problem's definition 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. Finally section 6 concludes the paper and poses future directions.

2. Related Work

Two of the most promising trends in domain-independent planning were presented over the last few years.

The first one consists of the transformation of the classical search in the space of states to other kinds of problems which can be solved more easily. Examples of this category are the SATPLAN [] and BLACKBOX [] planning system, the evolutionary GRAPHPLAN [] and certain extensions of GRAPHPLAN as the famous STAN [] planner.

SATPLAN and BLACKBOX transform the planning problem into a satisfiability problem, which consists of a number of boolean variables and certain clauses between these variables. The goal of the problem is to assign values to the variables in such a way that establishes all of the clauses.

GRAPHPLAN [] on the other hand creates a concrete structure, called the planning graph, where the nodes correspond to facts of the domain and edges to actions that either achieve or delete these facts. Then the planner searches for solutions in the planning graph. GRAPHPLAN has the ability to produce parallel plans, where the number of steps is guaranteed to be minimum.

Fox and Long developed STAN [], a powerfull planning system, extending GRAPHPLAN with *State Analysis* techniques. Apart from the *State Analysis* techniques, the efficiency of STAN is due to the construction of the planning graph in STAN, which is done very efficiently through bit-wise operators on vectors of bits. In its latest version, called Hybrid STAN [], the system is cable of identifying specific sub-problems (e.g. TSP sub-problems) from the definition of the original problem. The planner then uses specialized techniques to tackle each of the sub-problems separately.

The second category 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 []. Examples of planning systems in this category are UNPOP[], the ASP/HSP family [], GRT[], AltAlt[] and FF[], which was awarded for outstanding performance in the last AIPS-00 planning competition.

The planners of the latter category rely on the same idea to construct their heuristic function. They relax the planning problem by ignoring the delete lists of the domain operators and starting either from the Initial State or the Goals they construct a leveled graph of facts, noting for every fact f the level at which it was achieved $L(f)$. In order to evaluate a state S, the heuristic function takes into account the values of $L(f)$ for each $f \in S$.

McDermott's UNPOP [] was the first planner in the area of state space heuristic planning. UNPOP extended the well-known Means-ends analysis by building a graph, named *greedy regression-match graph*, consisting of subgoals and actions that achieve these subgoals. The subgoals of a planning problem are the goals of the problem and the preconditions of actions that achieve other subgoals. The creation of the greedy regression-match graph starts from the goals of the problem and proceeds backwards until all the subgoals at the last level exist in the Initial state of the problem. The information drawn from this graph is then used in the search phase in order to: a) estimate the distance between a given state S' and the Initial state and b) prune the actions that do not appear in the graph. The search starts from the goals and proceeds backwards, reconstructing at each intermediate state a new greedy regression-match graph.

The direct ancestor of UNPOP was Bonet & Geffner's HSP[] planning system. Given an Initial state *I*, HSP constructs a graph of facts starting from *I* by adding the facts that are added by actions whose preconditions already exist in the graph. A value *v(f)* is assigned to each fact *f* in the graph corresponding to the number of actions needed to achieve this fact starting from *I*. If all the preconditions of an action *a* already exist in the graph, HSP assigns a value *v(a)* to action *a*, where $v(a) = \sum_{} v(f_i)$ for each $f_i \in$ prec(*a*). The value of v(*a*) is then inherited to the facts in the add list of *a* using the following formula:

$v(q_i) = \min(v(q_i), v(a))$ for each qi$\in$ add(*a*).

The expansion of the graph stops when all the goals of the domain are included in the graph. In the search phase HSP starts from the Initial state and proceeds forwards with a Hill Climbing strategy (A* in the case of ASP) constructing the graph from scratch at each intermediate state.

In [] Bonet and Geffner present a variation of HSP called HSP-R. HSP-R uses the same heuristic function and the same search strategy as HSP, but searches the state-space backwards, starting from the goals and regressing them until it reaches the Initial state. The graph is still constructed in the same direction as in HSP and this enables HSP-R to compute the heuristic function only once and thus speed up the planning process.

The latest member of the HSP/ASP family is the HSP2 planner[1], which integrates HSP and HSP-R under a common environment from which apart from the direction of the search, the user can also select the heuristic function that will guide the search

GRT is another extension to HSP, which was developed by Refanidis and Vlahavas []. GRT creates a graph, similar to the one created by HSP, starting from the goals of the problem and proceeding backwards. The graph is created only once and it is used to extract a heuristic function that will be later used to guide the search. The search starts from the Initial state and proceeds forwards, using a best first search strategy. The main innovation of GRT is the use of Related Facts, which monitor the interactions between the facts in the graph. GRT has been also improved with a number of techniques for enriching incomplete goal states, eliminating irrelevant objects from the problem.

Nigenda, Nguyen and Kambhampati presented a hybrid planning system, named AltAlt [], which was created using programming modules from STAN [] and HSP-R []. In the first phase, AltAlt uses the module from STAN to create a planning graph similar to the one created by GRAPHPLAN[]. From the planning graph AltAlt creates an admissible heuristic function. The heuristic function is used in the second phase 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[2] is Hoffmann's FF planning system []. The construction of the heuristic function in FF is done in a process very similar to GRAPHPLAN. FF starts from the Initial state and constructs a leveled graph with the facts of the domain, noting for each fact the level at which it was achieved. In the next phase FF performs a relaxed backward search on the fixpoint (the

[1] The HSP-2 planning system can be found at the URL: http://www.ldc.usb.ve/~hector/

[2] 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/

graph of the facts) trying to find a sketch plan containing parallel steps. The sketch plan, which may not be valid, 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, i.e. the actions at the first level of the sketch plan, 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 [], NOLIMIT[],FLECS[] and RASPUTIN[]. All of these planners have been developed by researchers of the Carnegie Mellon University's PRODIGY project[3] and are based on the combination of goal-directed backward chaining with simulation of plan execution, a technique developed by Veloso et al [], which extends means-end analysis. 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, their performance is unstable and they frequently present precessions in their efficiency between 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 [].

The second argument is the main conclusion drawn by Bart Massey in an extensive study in the directions of planning presented in []. 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*". The answer to the question posed by Bonet and Geffner above has been answered by Massey in []. Massey discriminates planning problems 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 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 will be presented later in this section but first we have to 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. It is worth noting here that the initial state and the goals refer to the specific sub-problem that is passed to the progression module and not necessary to the

---

More information about the PRODIGY project can be found at the URL: http://www.cs.cmu.edu/~prodigy/

initial problem. This means that as the execution of the two search modules interleave, the initial state and the goals change in a matter that will be explained in more details later in this section.

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 and d) a threshold *T* declaring when should the search 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. Figure 3.1 illustrates the algorithm of the progression module.

```
Progression_Module
Input: I', G', SOF_AGENDA, T, h , Output: S
Set Agenda = [I'], K=I'
While (Agenda ≠ ∅ )
begin
   G = the first state in the Agenda
   If G'⊆G
       Return G
   If G∉Closed List
   begin
      If h(G)>h(K)+T
          Return (K)
      If h(G)<h(K)
          K=G
      For all actions A' : ∀q∈prec(A') →q∈G
      begin
            S'=G∪add(A')/del(A')
            Add S' to the Agenda sorted by h(S')
            If size of Agenda>SOF_AGENDA
                  remove the last element from Agenda
            Add G to the Closed List
      end
   end
   Remove G from Agenda
End
Return K
```

The progression module employs a simple forward best first search strategy 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) that should be stored in the Agenda, only the SOF_AGENDA most promising (according to *h*) states will be stored and the rest will be pruned. As a consequence, the algorithm is not complete and it may stop without returning a solution even if a solution exists. However, the memory requirements in order to maintain completeness may become unrealistic in hard real world problems. Moreover, the overhead in computational workload increases with the size of the Agenda, as the last one must be kept sorted.

b) The search may stop before reaching the goals or ending up in a deadlock, as it usually happens with search algorithms. The progression module also stops the search when it is not further possible to move to a state with a smaller distance from the goals than the one of the current state. In fact, the policy adopted by BP is a little bit more lenient and the search will not stop even if it can't find any improving state, as long as there is at least one successor state, which distance is not greater than the distance of the current state plus *T*, a constant passed as an argument to the progression module. 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.

3.2 The Regression Module

The algorithm of the regression module is quite similar to that of the progression module described in the previous section, since the search strategy is symmetric. However, there are certain key points that need to be clarified and all these points refer to common problems caused to the regression planners by the representation of the planning problems. The main idea behind regression planners is that they don't deal with states, as progression planners do, but with sets of states. This originates from the fact that the goals do not usually form a complete state description and therefore a more sophisticated technique than simply reversing the actions, as done in GRT [], is necessary in order to regress the goals.

The regression module makes extensive use of binary mutual exclusions between facts []. Two facts *p* and *q* are mutual exclusive, we note *mx(p,q)*, if no valid state contains both of them at the same time. For example, in the BLOCKS domain the facts clear(A) and on (B,A) are mutual exclusive. Mutual exclusions are calculated in a way similar to the one they are calculated in GRAPHPLAN. BP progressively builds a graph of facts noting at the same time the mutual exclusions between them using the following formulae:

mx(*p*,*q*) if :
$\forall$ action *A'* : $p \in$ add(*A'*) $\rightarrow q \in$ del(*A'*) and
$\forall$ action *A'* : $q \in$ add(*A'*) $\rightarrow p \in$ del(*A'*)

mx(*p*,*q*) if :
$\forall$ action A': $p \in$ add(*A'*) and $\forall$ action *B'*: $q \in$ add(*B'*) $\in (x,y)$: $x \in$ prec(*A'*), $y \in$ prec(*B'*), mx(*x*,*y*)

It is worth noting here that BP does not detect mutual exclusions of order higher than two and this may cause problems with certain domains. Consider, for example, a BLOCKS problem and a state *S'=[on(A,B), on(B,C), on(C,A))]*. Although there is no binary mutual exclusion between any two of the facts in *S'*, it is quite clear that *S'* is not a valid state. However, since the detection of mutual exclusions is a hard process, this was a necessary compromise.

Bonet and Geffner in HSP-R [] define two criteria for identifying whether an action *A* is backwards applicable to a state *S'*:

a) relativeness: add(*A*) $\cap$ *S'* $\neq \varnothing$
b) consistency: del(A) $\cap$ *S'* $= \varnothing$

The consistency check is not very strict and frequently HSP-R encounters invalid states. In order to prune these states, HSP-R checks to see if they contain pairs of facts that contain mutual exclusions. BP extends the consistency check of HSP-R in order to prune the actions that lead to invalid states before they are applied to the current state. So the criteria for backwards applicability in BP can be formed as:

Regressibility test
An action A is backwards applicable to state S if:

1. add(*A*) $\cap$ *S'* $\neq \varnothing$
2. del(A) $\cap$ *S'* $= \varnothing$
3. $\neg\exists$ (q,p): q $\in$ add(A), p $\in$ *S'*, mx(q,p)
4. $\neg\exists$ (q,p): q $\in$ (prec(A)-del(A)), p $\in$ *S'*, mx(q,p)

State *S'* that is produced after the backward application of action *A* to state *S* by the following formula:

State Regression
S'=*S*-add(*A*) $\cup$ prec(*A*)

Note however that there may be common elements in *S* and prec(*A*), so the double entries have to be eliminated from *S'*.

The regression module takes the same arguments with the progression module: 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 and d) a threshold *T* declaring when should the search stop and e) a heuristic function *h* capable of estimating distances between states. The module returns a set of facts (not necessarily a complete state), which resulted from the regression of *G'* and are the closer to *I'* that the module could find. Figure 3.2 illustrates the algorithm of the regression module.

```
Regression_Module
Input: I', G', SOF_AGENDA, T, h, Output: S
Set Agenda = [G'], K=G'
While (Agenda ≠ ∅)
begin
   G = the first state in the Agenda
   If G⊆I'
      Return G
   If G∉Closed List
   begin
      If h(G)>h(K)+T
         Return (K)
      If h(G)<h(K)
         K=G
      For each fact f in G
         For each action A' that has f in its add list
            If (del(A')∩G=∅)
            begin
               Set E=prec(A')-del(A')∪add(A')
               If there are no mutual exclusions between facts in E and G
               Begin
                  S'=S∪prec(A')-del(A')
                  Remove double entries from S'
                  Add S' to the Agenda sorted by h(S')
                  If size of Agenda>SOF_AGENDA
                     remove the last element from Agenda
               end
            end
      Add G to the Closed List
   end
   Remove G from Agenda
End
Return K
```

The only differences between the algorithms of the two search modules lie in the way they identify the applicable actions and produce the successor states. The other parts of the algorithm are identical. They both employ a best first search strategy with limited agenda, which stops and returns the best state (or set of states in the regression module) found so far, when the search cannot proceed any further to states that do not transcend the distance limit of *T*.

3.3 Combining the two Modules into an integrated Search Strategy

The underlying framework of the bi-directional search strategy is based on a relatively simple idea. Very frequently, single-directional planners reach a point in the search process where the heuristic function becomes less informative and they proceed with blind search. 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 less informative as the search goes on. The second reason though applies only to planners that construct the heuristic function only once at the beginning.

In a few words, 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 towards the goals. 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}. In order to avoid infinite loops between the two search modules, if one of the modules returns without improving its initial (or final in the case of regression) state, the *Threshold* of the search is increased by a constant number. The bi-directional search strategy of BP is outlined in Figure 3.3.

```
Search Algorithm of BP
Input I, G, Output Plan
Set Plan1=Plan2=[], S = I, F = G, Direction = Forward, Threshold = Init_Thr
While S⊇F
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
            Set 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=Progression_module(S,F,MAX_SOF_AGENDA,Threshold,h_F)
        If St≠F
            Set Plan2=St.plan+Plan2, Threshold=Init_Thr, F=St
        Else
            Threshold=Threshold+STEP
        Direction = Forward
    End
end
Return Plan1+Plan2
```

The changes in the direction of BP aim to deal with the problems stated above and this can be understood with the following two arguments: a) The change in the direction enables BP to reconstruct the heuristic function and thus make it more informative. b) The adaptive way in which BP changes directions tends to solve the major part of the problem following the search direction, which best fits the specific problem.

4. BP's Heuristic Functions

In order to test the efficiency of the bi-directional search strategy, we developed two, relatively simple, domain independent heuristics functions that were embedded in 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 the previous formula $L(S)$ is the number of steps needed to achieve state S, starting from the Initial state, $h(S)$ the value returned by the heuristic function and w_1 and w_2 user-defined constants. Sub-section 4.1 presents the progression heuristic function, while the regression one is presented in sub-section 4.2. Finally sub-section 4.3 describes the adoption of a goal ordering technique in order to refine the estimates of the heuristic functions.

4.1 The Progression Heuristic Function

The heuristic function used for the progression module is similar to the one of the GRT planning system. As in GRT the heuristic function is extracted from a leveled graph, similar to the one built by GRAPHPLAN. The graph consists of all the facts of the domain (the action levels of the GRAPHPLAN are omitted) that are achievable from the Initial state, tagging them with a number K, identifying the minimum number of steps needed to achieve 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 facts that are added by actions that are applicable at level L-1. Note however, that since the creation of the graph proceeds backwards and the Goals do not necessarily form a complete state, a level L is built from level L-1 through relaxed regression and not by progression, as done in GRT.

Relaxed regressibility test
An action *A* is backwards applicable to level *M* of the graph if
$\exists$ fact *f*: $f \in$ add(*A*), $f \in$ level *M* of the graph

The relaxed regressibility test is similar to the first criterion of the regressibility test of section 3.2, if we treat the levels of the graph as states. However, the last three criteria of the test have been omitted since the graph is just used for estimates of the real distances between each fact and the goals.

Graph Expansion
For each action *A* that passes the relaxed regressibility test for 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 reiterates until the graph reaches level L_{MAX}, where no more facts can be achieved with a cost smaller than the one in its 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 they just differ 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 proceeding 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. The progressibility test is not relaxed and an action *A* is selected for expanding level *L*+1 of the graph if all the preconditions of *A* exist in level *L* of the graph. Again a value *V* summing up the tags of the preconditions of *A* is calculated and the facts in add(*A*) are added to level *L*+1 of the graph and tagged with *V*+1 if they have not already been tagged with a smaller value than *V*+1.

The graph of the regression is built only once each time the planner selects the backward direction for its search. In order to estimate the distance between each state *S* in the domain and the Initial state, the heuristic function just sums up the tags of all the facts in *S*.

4.3 Refining the heuristic functions with Goal Ordering

Goal ordering for planning has been an active research topic over the last years and although the technology is not yet mature, goal-ordering techniques have been successfully used in 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[] 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 [] 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."* [].

IPP [] and FF[] make use of reasonable orderings during planning through the construction of a goal agenda that divides the goals into an order set of sub-goals. The planners sequentially try to 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*, which is outlined in figure X, 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 (a should not be ordered before b)
For each action O: a∈add(O)
begin
    Result =true
    For each fact f: f∈prec(O)
    begin
        If mx(b,f)=true
            Result=false
    end
    If result = true return false
end
Return true
```

Function OB is not complete and it may miss certain goal orderings (e.g. in problems with mutual exclusions of higher order than two). However it is a fast method for identifying a respectable number of goal orderings.

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 S for possible breaches of the goal orderings.

Ordering breach

Fact f of a state S is an ordering breach if:

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

For every ordering breach found in state S, the latter is penalized (i.e. the estimated distance between S and the Goals is increased by a constant number), since at some point later the ordering breaches will have to destroyed and re-achieved after the correct ordering has been reinstated.

It is straightforward that function OB can only be used be the progression module, so there is a slightly different function called OB-R (figure X) that is used by the regression one.

```
Function OB-R
Input: Initial facts a and b
Output: True (a should be ordered before b) or False (a should not be ordered before b)
For each action O: a∈del(O)
begin
    Result =true
    For each fact f: f∈(prec(O)-del(O)∪add(O))
    begin
        If mx(a,f)=true
            Result=false
    end
    If result = true return false
end
Return true
```

RESULTS OF THE GOAL-ORDERING

5. Domain Analysis through Planning Graphs

6. Experimental Results

In order to test the efficiency of BP we implemented two additional planners: a) PMP, a progression planner using the progression module and heuristic function of BP and b) RMP, a regression one 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[4] and were implemented in C++. All the tests were run on a SUN ENTERPRISE 3000 parallel computer, with a SPARC-1 processor 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 chose the following configuration for the three planners:

1. The size of the planning agenda is limited to 200 states
2. The initial value for the search threshold (variable `Init_Thr` in the search algorithms) is 2
3. The step for the increases in the search threshold (variable STEP in the search algorithms) is also set to 2
4. The values of the weights for the weighted A* search algorithm are set as: 0.4 for the accumulated cost of the actions applied so far (w_1=0.4) and 1.0 for the estimated remaining cost (w_2=1.0).

The three planners (PMP, RMP and BP) were thoroughly 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 6.1, 6.2, 6.3 and 6.4 present the results of the tests. Columns 2,4 and 6 of all tables present the length of the produced plans (number of actions) and columns 3,5 and 7 the time (in milliseconds) needed to solve the problems. Note that a dash in cell means that the problem could not be solved, within the 180 seconds limit in CPU time set on all planners and plan lengths written in bold note the minimum plan length found by the three planners.

6.1 Blocks world

It is clear from table 6.1 that the specific problems of the Blocks world used in the competition favor regression planners. RMP was able to solve 47% more problems than PMP producing in all problems shorter plans in much less time. BP presented results quite similar to RMP. Specifically, it

[4] The code of GRT is available at the URL: http://www.csd.auth.gr/~lpis/GRT/main.html

solved 1 problem less than RMP, producing 16% lengthier 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.

Problem	PMP (length)	PMP (time)	RMP (length)	RMP (time)	BP (length)	BP (time)
4-0	6	40	6	40	6	60
4-1	10	40	10	50	10	70
4-2	6	30	6	40	6	50
5-0	12	790	12	100	12	110
5-1	10	810	10	90	20	180
5-2	24	880	16	330	16	140
6-0	42	14790	12	140	18	270
6-1	10	80	10	110	10	150
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
8-0	122	54310	18	430	122	19220
8-1	88	27360	20	490	30	850
8-2	18	250	16	410	16	390
9-0	92	37750	60	61010	84	8160
9-1	-	-	30	5150	30	3570
9-2	-	-	26	5130	28	1740
10-0	-	-	-	-	-	-
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	-	-
13-0	-	-	-	-	-	-
13-1	-	-	-	-	-	-
14-0	-	-	38	8170	-	-
14-1	-	-	36	5910	-	-
15-0	-	-	-	-	-	-
15-1	-	-	-	-	-	-
16-0	-	-	-	-	-	-
16-1	-	-	-	-	-	-
17-0	-	-	-	-	50	59280

Table 6.1: Blocks world problems

6.2 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% lengthier 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% lengthier than those of BP.

Problem	PMP (length)	PMP (time)	RMP (length)	RMP (time)	BP (length)	BP (time)
4-0	20	150	23	240	20	240
4-1	19	180	25	300	19	260
4-2	17	410	17	220	20	300
5-0	27	220	32	350	27	350
5-1	17	150	20	270	21	340
5-2	8	100	8	200	10	240
6-0	25	210	33	440	28	450
6-1	14	150	16	270	16	400
6-2	25	210	32	440	26	400
7-0	38	720	44	1340	41	1210
7-1	62	10730	51	1340	50	1830
8-0	31	640	41	1230	37	1340
8-1	44	810	54	1710	46	1490
9-0	37	700	47	1510	39	1430
9-1	32	590	34	1230	32	1320
10-0	74	29650	54	3790	50	5010

10-1	43	1570	53	2890	45	3140
11-0	52	3070	62	4370	54	4010
11-1	66	6330	75	5190	66	7040
12-0	45	4590	51	3820	45	4960
12-1	70	2600	82	5480	72	4070
13-0	79	7500	97	17000	93	38460
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
18-0	-	-	145	61470	152	99020
18-1	104	90900	101	46410	102	59810
19-0	-	-	128	71670	129	98540
19-1	-	-	119	70160	113	89260
20-0	-	-	127	73580	138	116440
20-1	110	22620	123	66510	112	33480

Table 6.2: Logistics problems

6.3 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. Although in the tests we used 100 problems of the MIC-10 domain (s1-0 to s20-4), due to limitation in space table 6.3 only presents part of the problems.

Problem	PMP (length)	PMP (time)	RMP (length)	RMP (time)	BP (length)	BP (time)
S1-0	4	0	4	10	4	20
S1-4	4	10	4	10	4	0
S2-0	8	20	7	40	8	20
S2-4	8	20	7	40	7	30
S3-0	12	40	10	320	11	40
S3-4	11	40	10	350	10	40
S4-0	16	100	-	-	15	100
S4-1	16	120	16	1230	13	90
S4-2	16	100	-	-	16	100
S4-4	16	110	-	-	15	100
S5-0	20	190	-	-	19	190
S5-4	20	180	-	-	18	170
S9-0	36	1420	-	-	32	920
S9-4	33	1000	-	-	29	950
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
S17-2	65	9950	-	-	60	8050
S17-3	-	-	-	-	61	7870
S17-4	65	14280	-	-	57	8040
S18-0	70	19000	-	-	65	11150
S18-4	70	16050	-	-	62	9140
S19-0	-	-	-	-	66	11520
S19-1	-	-	-	-	69	12740
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 6.3: MIC-10 problems

6.4 Freecell

Like the Logistics domain, Freecell does not clearly favor a specific planning direction. However PMP seemed to perform a little bit 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% lengthier 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.

Problem	PMP (length)	PMP (time)	RMP (length)	RMP (time)	BP (length)	BP (time)
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 6.4: Freecell problems

7. Conclusions and Future Work

8. References

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

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

[3] Kautz, H. and Selman, B. 1996, Pushing the envelope: Planning, propositional logic, and stochastic search. In Proceedings, AAAI-96, Portland Oregon, pp. 1194-1201

[4] Kautz, H. and Selman, B. 1999, Unifying SAT-based and Graph-based Planning. In Proceedings, IJCAI-99, Stockholm.

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

[6] Long, D. and Fox, 2000, M. *Hybrid STAN: Identifying and Managing Combinatorial Sub-problems in Planning*, In Proceedings, 18th Workshop of the UK Planning and Scheduling SIG

[7] 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

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

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

[10] 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.
[11] 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.
[12] Bonet, B., Loerincs, G., and Geffner, H., 1997, A robust and fast action selection mechanism for planning, In *Proceedings, 14th International Conference of the American Association of Artificial Intelligence (AAAI-97)*, Providence, Rhode Island, pp. 714-719.
[13] Bonet, B. and Geffner, H. 1999, Planning as Heuristic Search: New Results, In Proceedings, ECP-99, Durham UK.
[14] 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).
[15] Veloso, M. and Stone, P. 1995, FLECS: Planning with a Flexible Commitment Strategy, JAIR (3).
[16] Veloso, M. 1994, Planning and learning by Analogical Reasoning, Springer-Verlag.
[17] Fink, E. and Blythe, J. 1998, A complete bidirectional planner, In proceedings, 4th International Conference on AI Planning Systems.
[18] Stone, P., Veloso, M. and Blythe, J. 1994, The Need for Different Domain-Independent Heuristics, In proceedings, AIPS-94, Chicago, USA.
[19] Massey, B. 1999, Directions In Planning: Understanding the Flow of Time in Planning, Available as a Technical Report from the University of Oregon.
[20] McCluskey, T. and Porteous, J. 1997, Engineering and Compiling Planning Domain Models to Promote Validity and Efficiency, Artificial Intelligence, 95.
[21] Koehler, J. and Hoffmann, J. 2000, On Reasonable and Forced Goal Orderings and their Use in an Agenda-Driven Planning Algorithm, JAIR (12).
[22] Porteous, J. and Sebastia, L., 2000, Extracting and ordering Landmarks for Planning, In Proceedings, 18th Workshop of the UK Planning and Scheduling SIG