A Heuristic for Planning based on Action Evaluation

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

Abstract

This paper proposes a domain independent heuristic for state space planning, which is based on action evaluation. The heuristic obtains estimates for the cost of applying each action of the domain by performing a forward search in a relaxed version of the initial problem. The estimates for the actions are then utilized in a backward search on the original problem. The heuristic, which has been further refined by a goal-ordering technique, has been implemented in AcE (Action Evaluation), a state space heuristic planner, and thoroughly tested on a variety of toy problems.

Introduction

State space planning is the simplest form of planning and has been an active research area for many years. However, a large part of the AI researchers in Planning, abandoned it and focused on other areas, since they regarded it to be non promising. This was totally justifiable, since even with the heuristics that were available at that time the search in the space of states was combinatorial explosive. However, 6-7 years ago, McDermott with his work on UNPOP and later Geffner with ASP/HSP, urged a large part of the planning community to re-consider state space planning. They showed that with the appropriate heuristics, state space planning can be very efficient.

A large number of scientists were interested in this promising area. As a result, during the last few years a quite large number of state-space planning systems showed with remarkable performance. These planners extended the ideas, proposed by McDermott and Geffner, and utilize the power of more refined heuristic functions, in order to traverse the state of space using variations of known search strategies. Most of these planners create their heuristic by solving a relaxed problem, from which they extract estimates for the distances between the facts of the domain and the goals. The distances of the independent facts are then combined to provide the search with estimates for the distances of whole states.

In this paper we propose a different approach in state-space, heuristic planning, which is based on estimated distances between the domain's actions, rather than facts, and the goals. Basing the estimates on actions rather than facts, enables the heuristic to keep better track of the various interactions between the facts, and therefore produce better estimates. The proposed heuristic is embodied in a regression planner employing a weighted A* search strategy, which is thoroughly tested on a large variety of problems, adopted from the last AIPS-00 planning competition. The efficiency of the new planner, called AcE, is assessed through comparative tests with 5 of the most efficient planners in this category.

The rest of the paper is organized as follows: Section 2 presents an overview of the research in the area of state space, heuristic planning. Section 3 describes the heuristic of AcE among with an extension based on goal orderings. Section 4 addresses certain implementation issues and section 5 presents experimental results of Ace and most of the state-of-the-art planners presented over the last years. Finally section 6 concludes the paper and poses future directions.

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 (Kautz and Selman 1996) and BLACKBOX (Selman, Levesque and Mitchell 1992) planning system, the evolutionary GRAPHPLAN (Blum and Furst 1995) and certain extensions of GRAPHPLAN as the famous STAN (Long and Fox 1998) 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 (Fox and Long 2000), 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 (Korf 1998). Examples of planning systems in this category are UNPOP (McDermott 1996), the ASP/HSP family (Bonet, Loerincs and Geffner 1997, Bonet and Geffner 1999), GRT (Refanidis and Vlahavas 1999, 2001), AltAlt(Nguyen, Kambhampati and Nigenda 2000) and FF (Hoffmann 2000), 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, Loerincs & Geffner's HSP (Bonet, Loerincs and Geffner 1997) 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 1999) 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 (Bonet and Geffner 2000), 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 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 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.

The latests planner in the category of domain-independent, heuristic, state- space planning is BP (Vrakas and Vlahavas 2001). BP is a bi-directional planner, which consists of two independent search modules (the progression and the regression) that interleave their execution. Each module is equipped with a simple heuristic, based on HSP, enhanced with a goal-ordering technique for further heuristic improvement. BP starts with the progression module and changes direction every time it perceives that the heuristic is no longer able to guide the search.

Evaluating actions

Most of the state-of-the-art planners in this category, such as GRT, HSP/ASP, HSPr and AltAlt, note for every fact of the domain its estimated distance from the goals (or from the initial state) and then use these values in order to evaluate a whole state (usually by summing up the values of the included facts).

Their main inefficiency sources from the facts that they consider the facts of the domain to be completely independent and the cost of achieving a set of facts is equal to the sum of the costs of achieving each one of them separately. However, this is rarely the case since in the attempt to achieve a certain fact many other facts are also achieved in the way.

GRT partially deals with this problem with the introduction of *related facts*. Two facts *p* and *q* are related with each other if:

a) The action that achieved *p*, during the construction of the heuristic, also achieves *q* or the opposite.

b) *q* is already noted as related with one of the preconditions of the action achieving *p*.

Although, the *related facts* are able to track only a small subset of the interactions between the facts of the domain, they manage to refine the heuristic of GRT and they prove to be useful in many domains (Refanidis and Vlahavas 2001).

FF, adopts a different strategy for keeping track of the possible interactions between the facts of the domain. In order to construct its heuristic, FF builds a graph similar to that of GRAPHPLAN and uses this graph to extract a relaxed partially ordered relaxed plan, the size of which is the estimated distance between the current state and the goals. From the way in which the relaxed plan is constructed FF is able to discover a large portion of the interactions. However, the fact that FF has to search in both directions in order to construct its heuristic, enforces it to reconstruct it at each state (or at least in a large number of states) during the actual search.

This paper proposes a different approach in the construction of the heuristic function, based on action evaluation, which is able to keep track of the great majority of interactions between the domain's facts and yet needs to be constructed only once at the beginning.

The heuristic of AcE is constructed in the forward direction, starting from the initial state and proceeds towards the goals, calculating the distances between the initial state and all the actions of the domain (or at least all the actions that can be achieved from the initial state). The distance, noted as *dist*, for a given action *A* is calculated by the following rules:

$$dist(A) = \begin{cases} 1, \ if \ \ prec(A) \subseteq I \\ 1 + \sum_{X \in MPS(\,prec(A))} dist(X), if \ \ prec(A) \not\subset I \end{cases}$$

where *MPS(S)* is a function returning the set of actions $\{A_p\}$ achieving all the facts of *S* with the minimum accumulated cost of $dist(A_p)$. Note that MPS never returns actions with undefined dist.

In order to find the minimum set of actions achieving a specific state, *MPS(S)* has to calculate all the possible combinations of actions achieving *S*, and this process is combinatorial explosive. In AcE, *MPS(S)* is approximated using a greedy algorithm, which is outlined in figure 1.

Function MPS(S)
Input: a set of facts *S*
Output: a set of actions achieving *S* with near minimum accumulated *dist*

Set $G = \varnothing$
$S = S - S \cap I$
Repeat
  f is the first fact in S
  Let *act(f)* be the set of actions achieving *f*
  for each action *A* in *act(f)* do
    $val(A)=dist(A) / |add(A) \cap S|$
  Let *A'* be an action in *act(f)* that maximizes *val*
  Set $G = G \cup A'$
  Set S = S - $add(A') \cap S$
Until $S = \varnothing$
Return *G*

Figure 1. *MPS* Function

In the worst case, each action A' selected by MPS will only achieve one fact of S and therefore the complexity of MPS will be |S|*N, where N is the number of the domain's actions. Since |S|<<N the order of the complexity is $O(N^2)$.

In the average case, the actions achieving a fact f are a small subset of the domain's actions and the size of the subset is N/K, where K is comparable to |S| for each state S

of the domain. Therefore the complexity of MPS is |S|*N/K, which is in the order of O(N).

We will illustrate the heuristic function of AcE with a concrete example of the Blocks-world domain. Suppose that the initial state of the problem is the one shown in figure 2.

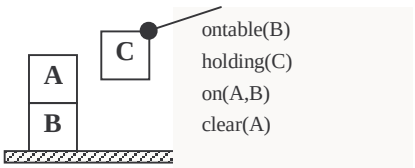

Figure 2

The actions that can be applied to the initial state of the problem in figure 2 are: put-down(C) and stack(C,A) and therefore: dist(put-down(C))=1 and dist(stack(C,A))=1.
In order to calculate dist(un_stack(A,B)) we need *MPS(prec(un_stack(A,B)):*
S= {handempty,on(A,B),clear(A)}
$S = S - S \cap I$ = {handempty}
f=handempty
act(f) = {put_down(C), stack(C,A)}, *note here that all the other actions achieving f have undefined dist, so they are not taking into account.*
A' = put_down(C) both actions in act(f) have the same val, so one of them is arbitrarily selected.
G = {put_down(C)}
$S = \varnothing$
MPS({handempty,on(A,B),clear(A)})={put_down(C)} and dist(un_stack(A,B))=1+dist(put_down(C)) =1+1 = 2.

Similarly the algorithm proceeds with the rest of the domain's actions. When distances have been assigned to all the domain's actions, AcE starts searching the state-space starting from the goals. During the search, the estimated distances of the actions are used to evaluate all the intermediate states.

In order to calculate the heuristic value a given state S_1 ($h(S_1)$), AcE uses $MPS(S_1)$ to find the near minimum set of actions achieving the facts of S_1 and then sums up the distances of the actions in $MPS(S_1)$. This can be seen as the process of evaluating an action A_{S1} for which: prec(A_{S1}) = S_1.

For example, in order to evaluate state S_a of figure 3, we calculate MPS(S_a):
S= {ontable(B),ontable(C),holding(A),clear(B),clear(C)}
$S = S - S \cap I$ *={ontable(C),holding(A),clear(B),clear(C)}*
f=ontable(C)
act(f)={put_down(C)}
A' = put_down(C)
G={put_down(C)}
S={ holding(A),clear(B)}
f=holding(A)
act(f)={pick-up(A),unstack(A,B),unstack(A,C)}
A'=unstack(A,B)
G={put_down(C), unstack(A,B)}
$S = \varnothing$
MPS(S_a)={put_down(C), unstack(A,B)} and
h(S_1)=dist(*put_down(C)*) + dist(*unstack(A,B)*) = 1 + 2 =3

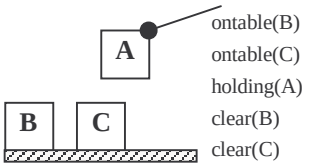

Figure 3

Similarly, $h(S_b)$ for S_b={*ontable(A), ontable(C), holding(B), clear(A), clear(C)*} is equal to dist(put_down(C)) + dist(put_down(A)) + dist(pick_up(B)) = 1 + 3 + 4 = 8

It is clear from the above examples that the heuristic of AcE still produces overestimates. This can be overcome by keeping track of the MPS for all the domain's actions, apart from the distances, and taking also into account part of the delete lists. It remains in our direct future plans to investigate this type of heuristic refinement. However, the heuristic as it is behaves satisfactory and it generally succeeds in guiding the search to the most promising states.

Refining the heuristic 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 (McCluskey and Porteous 1997) 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 (Koehler and Hoffmann 2000) 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."* (McCluskey and Porteous 1997).

IPP (Koehler and Hoffmann 2000) and FF(Hoffmann 2000) 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.

AcE 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 4, 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 4: 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 the heuristic function, AcE searches state *S* for possible violations of the goal orderings. Fact f of a state S is violating the goal ordering if:

f∈ Goals and ∃ goal g: g∉ 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 (10 at the current implementation), since at a later point the ordering breaches will have to destroyed and re-achieved after the correct ordering has been reinstated.

Consider, for example, the case of figure 5:

$S_1 - S_1 \cap I$ ={ontable(A),clear(B)}

$S_2 - S_2 \cap I$ ={ontable(A),ontable(C),clear(A),clear(B)}

Since $(S_2 - S_2 \cap I) \supseteq (S_1 - S_1 \cap I)$, $MPS(S_2) \supseteq MPS(S_1)$ and therefore $h(S_2) \geq h(S_1)$. However, it is clear that *I* is closer to S_2 than to S_1. The goal-ordering technique, however, will notice that *OB(on(A,B),on(C,A))*=true and although *on(C,A)* exists in S_1, *on(A,B)* does not, so S_1 will be penalized and therefore S_2 will be preferred from S_1.

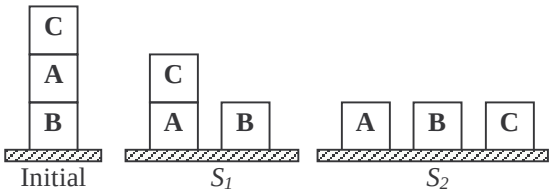

Figure 5

Implementation issues

This section discusses certain implementation issues of the AcE planning system, concerning the adopted search strategy and the internal representation of the important information. It also addresses certain problems that arise during the application of the planner in specific domains and presents two techniques for enriching and simplifying the problems definitions that deal with the above problems.

Search strategy

AcE employs a weighted A* search strategy which starts from the goals of the problem and moves backwards until it reaches the initial state. A state *S* in weighted A* is evaluated using the following formula:

$$e(S) = w*h(S) + d(S),$$

where h(*S*) is the value of the heuristic function for state *S*, d(*S*) is the number of steps that were performed to reach S and w is a constant real number between 0 and 1. If we set w to 1, then the algorithm becomes best-first and if we set w equal to 0, the algorithm becomes breadth first. The value of *w* in Ace is 0.75, meaning that the evaluation of state is 75% based on the result of the heuristic function and 25% based on the already walked distance.

Ace has a closed list, which is index by a hash table, in which it keeps all the already expanded states, in order to avoid revisiting them in the future.

The states that have not yet been expanded are kept in an agenda, the size of which is limited to a constant number N. If the size of the agenda grows larger than *N* at some point during search, only the *N* best states (according to the heuristic function) will remain in the agenda, while the rest will be pruned.

By pruning states from the agenda, AcE risks its completeness (i.e. the pruned states may be leading to a solution), but this is a necessary concession since otherwise the memory requirements may grow outside the available resources. In AcE *N* is 5000, which seemed to be more than enough. In fact, in our tests with smaller values of *N*, the only problems faced by AcE were in the large MIC-10 problems, since all the other problems could be very easily solved even with *N*=200.

As stated above, AcE is a regression planner. Although in progression the task of finding if an action is applicable and produce the successor states is trivial, this is hardly the

case with regression. The goals rarely constitute a complete state and they describe a set of states rather than a single state. Therefore, the criterion for action applicability must be more sophisticated.

The regressibility criterion adopted by AcE makes excessive use of mutual exclusive relations between pairs of facts (Blum and Furst 1995). Two facts *p* and *q* are said to be mutually exclusive, denoted as mx(p,q), if no valid state can contain both of them. More specifically, an action *A* can be applied (backwards) to a state *S*, if:

1) $add(A) \cap S \neq \varnothing$: *A* must have at least one add-effect in S.
2) $del(A) \cap S = \varnothing$: *A* must have no delete effects in S.
3) $\neg\exists(p,q),\ p \in add(A), q \in S : mx(p,q)$: There must be no mutual exclusions between add-effects of *A* and facts of *S*.
4) $\neg\exists(p,q),\ p \in (prec(A) - del(A)),\ q \in S : mx(p,q)$: There must be no mutual exclusions between the facts in *S* and the implied add-effects of *A*. A fact *f* is said to be implied by an action *A*, if it belongs to the preconditions but not in the delete list of *A*.

If we apply (backwards) action *A* to state *S*, the resulting state *S'* will be the following:

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

Representation

After the parsing of the input files (domain file and problem file) and in order to speed up the planning process, AcE creates all the (grounded) facts of the domain and the (grounded) actions that can be achieved from the initial state of the problem. These facts and actions are stored in tables and AcE assigns to each fact and actions a unique integer number, Based on these numbers Ace makes all the necessary links between facts and actions. For example, for each fact *f* it creates four lists of actions containing:

a) The actions that have *f* in their preconditions
b) The actions that have f in their add-effects
c) The actions that have f in their delete lists
d) The actions that have f in their preconditions and not in their delete lists.

These integers and the appropriate lists are then utilized during the search. A state is represented as a one-dimensional table of integers and in order to find the actions that are applicable to a state *S*, AcE performs a limited search in the appropriate lists of actions of the facts in *S*.

Enrichment of initial states

During the experiments, we noticed that AcE faced unreasonable difficulties in handling the problems of the MIC-10 domain. The problems originated from the fact that the board action had an add-effect (boarded), which was never deleted and there was no other predicate (e.g. not_boarded) noting its negation. These resulted in situation were the planner would select to board a passenger even though the latest was already boarded. Ace would, frequently, come to endless loops selecting in each iteration to add the board action at its plan.

In order to deal with this problem, which has also been faced by GRT, we enhanced AcE with a method that checks for predicates that appear in the add lists of an action, are never deleted and there is no other precondition that serves as the negation of these predicates. In this case the method automatically enriches the domain encoding with a predicate (playing the role of negation) and adds this predicate to the initial state and to the preconditions of all the appropriate actions.

For example, in the MIC-10 domain it adds to the initial state one fact for every passenger noting that he is initially not boarded. It then modifies the operator board to include not-boarded in its preconditions.

Simplifications of Initial states

A second case, where AcE modifies the problem's definition is the one where the initial state contains redundant information. For example, in the Blocks world domain with the 3 operators definition used in (Refanidis and Vlahavas 2001) the fact *handempty* remains at the initial state of he problems, although no operator is making use of it. Another example is certain problems of the Logistics domain, where there are packages at the initial state, with initial locations, that are absent from the goals. These facts not only may confuse the planners, but they can also increase the branching factor with unnecessary actions.

In order to cope with this problem, AcE is equipped with another technique that identifies whether a fact is useful, in order to achieve the goals, or not. The technique, which is executed before the actual search, starts from the goals and recursively proceeds backwards identifying useful facts, until no more useful facts can be found.

A fact *f* is noted as useful if:

a) *f* belongs to the goals of the problem or
b) *f* is a precondition to an action achieving an other useful fact.

After the identification of useful facts, the technique can safely remove all the facts from the problem definition that are not identified as useful.

Experimental Results

In order to evaluate the efficiency of AcE we compared it, on a large variety of problems used in the last AIPS-00 planning competition, with 5 state-of-the-art planning systems that were distincted in the AIPS-00 competition. The planners are (in alphabetical order): AltAlt, FF, GRT, HSP-2 and STAN. We used problems from all the domains (Blocks world, Logistics, Mic-10 and FreeCell) used in th competition.

The results of the tests are presented in tables: 1, 2, 3 and 4 for the Blocks world, the Logisics, the MIC-10 and the Freecell domain respectively. For each planner we present the number of steps in the solution (column with L) and the time, in seconds, needed to solve the problem (column with T). A dash in a cell indicates that the specific problem coud not be solved in the 180 seconds CPU time limit, or that the planner ran out of memory. Plan legths typed in bold indicate the shortest plan found by all planners.

The platform used for the results is a SUN ENTERPRISE 3000 unix station with 256 MB of shared memory and 2 SPARC-1 processors running at 167 MHz. The underlying operating system is SUN Solaris 2.7. The codes of all the planners were available through the World Wide Web.

Blocks

AcE managed to solve more problems than any other planner in the Blocks domain. It solved 39 out of 42 problems, 1 more than HSP-2 and 6 more than FF. The plans created by AcE were quite good and in 24 problems they were the shortest found of all planners. Concerning planning time, AcE needed less than 2 seconds for each problem up to 14-1 and less than one minute (except for 15-0) for the rest of the problems.

Logistics

The problems of the Logistics domain were quite easy and the majority of them were solved by all planners. Only AltAlt missed 5 out of the 37 problems and GRT missed one. Concerning planning time and solution length, AcE did better than HSP-2 and AltAlt but was outperformed from FF, STAN and GRT.

	AltAlt		FF		GRT		HSP-2		STAN		AcE	
Prob	**L**	**T**	**L**	**T**	**L**	**T**	**L**	**T**	**L**	**T**	**L**	T
4-0	**6**	0.1	**6**	0	**6**	0	10	0	**6**	0.1	**6**	0
4-1	**10**	0.1	**10**	0	**10**	0	**10**	0.1	**10**	0	**10**	0
4-2	**6**	0.1	**6**	0	**6**	0	**6**	0.1	**6**	0	**6**	0
5-0	**12**	0.1	**12**	0	**12**	0.1	18	0.1	**12**	0.1	**12**	0
5-1	**10**	0.1	**10**	0	18	0.1	14	0.1	**10**	0.1	**10**	0
5-2	18	0.1	**16**	0	20	0.1	20	0.1	**16**	0.1	**16**	0
6-0	**12**	0.1	20	0	40	0.2	24	0.1	**12**	0.1	**12**	0
6-1	**10**	0.4	**10**	0	14	0.1	14	0.1	**10**	0.3	**10**	0
6-2	32	0.3	**20**	0	32	0.1	28	0.1	**20**	0.3	**20**	0
7-0	26	0.3	**20**	0	22	0.1	22	0.1	20	0.2	24	0.1
7-1	24	0.9	**22**	0.1	56	0.4	32	0.2	**22**	1	**22**	0.1
7-2	20	0.7	22	0.3	48	0.4	28	0.3	**20**	0.7	**20**	0.1
8-0	**18**	0.7	**18**	0	38	0.6	26	0.3	**18**	0.9	**18**	0.2
8-1	**20**	0.4	-	-	44	0.4	30	0.5	**20**	1	**20**	0.2
8-2	**16**	0.6	**16**	0	50	0.4	28	0.3	**16**	0.6	**16**	0.2
9-0	40	6.8	-	-	116	3.1	48	0.9	**30**	1.5	**30**	0.4
9-1	36	1.2	**28**	0.1	104	3.1	42	2.9	**28**	1.5	38	0.4
9-2	34	3.1	**26**	0	44	0.3	40	1.1	**26**	0.9	**26**	0.3
10-0	108	53.5	**34**	0	108	1.3	46	0.7	**34**	5	38	0.5
10-1	40	2.3	38	5.8	108	11	44	1.1	**32**	136	52	0.6
10-2	-	-	**34**	0.3	124	11	48	1.1	35	9	36	0.5
11-0	36	27.0	**34**	0.1	50	0.6	54	1.9	-	-	36	0.7
11-1	36	2.8	-	-	134	4.5	46	2.3	-	-	**30**	0.8
11-2	56	37.4	**34**	0	128	3.7	48	2.8	-	-	38	0.9
12-0	36	3.1	44	1.1	114	12	58	11	**34**	99	**34**	0.9
12-1	38	11.8	**34**	0.1	68	11	56	9	**34**	12	44	1.2
13-0	-	-	**42**	0.1	-	-	64	10	-	-	44	1.6
13-1	-	-	-	-	128	90	70	6	-	-	**44**	1.5
14-0	**40**	6.6	**40**	0.1	88	26	68	62	-	-	**40**	1.8
14-1	48	16.2	**42**	0.7	-	-	68	72	-	-	**42**	1.9
15-0	-	-	-		-	-	70	23	-	-	**46**	65
15-1	-	-	**52**	0.2	-	-	72	36	-	-	64	2
16-1	-	-	-	-	-	-	86	16	-	-	**62**	11
16-2	-	-	**52**	0.1	-	-	86	16	-	-	-	-
17-0	**60**	37.4	-	-	-	-	90	97	-	-	64	28
17-1	-	-	-	-	-	-	84	61	-	-	**58**	29
18-0	-	-	-	-	-	-	90	149	-	-	-	-
18-1	-	-	**64**	0.2	-	-	94	58	-	-	76	52
19-0	-	-	**62**	0.2	-	-	-	-	-	-	-	-
19-1	-	-	**58**	0.1	-	-	-	-	-	-	64	19
20-0	-	-	**60**	0.2	-	-	-	-	-	-	62	22
20-1	-	-	**72**	0.2	-	-	-	-	-	-	82	45

Table 1: Blocks world domain

	AltAlt		FF		GRT		HSP-2		STAN		AcE	
Prob	**L**	**T**	**L**	**T**	**L**	**T**	**L**	**T**	**L**	**T**	**L**	**T**
4-0	24	0.1	**20**	0	**20**	0.1	26	0.2	**20**	0.1	23	0
4-1	21	0.1	**10**	0	19	0.1	25	0.2	19	0.1	25	0
4-2	17	0.1	**15**	0	**15**	0.1	**15**	0.2	**15**	0	17	0.1
5-0	31	0.1	**27**	0.1	**27**	0.1	36	0.3	**27**	0	32	0.1
5-1	19	0.1	**17**	0	**17**	0.1	21	0.2	**17**	0	20	0.1
5-2	**8**	0.1	**8**	0	**8**	0.1	10	0.1	**8**	0	**8**	0
6-0	29	0.2	**25**	0	**25**	0.1	34	0.3	**25**	0	32	0.1
6-1	16	0.1	**14**	0	**14**	0.1	16	0.2	**14**	0.1	16	0.1
6-9	31	0.1	**24**	0.1	25	0.1	28	0.2	25	0	30	0.1
7-0	-	-	**36**	0.1	38	0.2	52	2	37	0.2	47	0.4
7-1	-	-	**44**	0.1	**44**	0.2	56	1	**44**	0.1	51	0.3
8-0	39	0.5	**31**	0.1	32	0.2	40	0.9	**31**	0.1	39	0.3
8-1	55	0.5	**44**	0.1	45	0.2	60	1.1	45	0.1	54	0.4
9-0	50	0.5	**36**	0.1	38	0.2	51	1.2	**36**	0.1	47	0.4
9-1	36	0.4	**30**	0.1	31	0.2	43	1.2	**30**	0.1	34	0.4
10-0	57	1.2	**46**	0.1	47	0.4	59	2.8	**46**	0.1	54	1
10-1	52	1.1	**42**	0.1	43	0.3	57	2.8	**42**	0.3	54	0.9
11-0	63	1.4	**49**	0.1	52	0.4	69	4.4	**49**	0.1	63	1
11-1	74	1.5	**60**	0.2	66	0.5	86	5.1	61	0.5	73	1
12-0	54	1.3	**42**	0.1	-	-	57	4.4	**42**	0.2	54	1
12-1	95	2.2	73	0.2	74	0.6	98	4.5	**68**	0.5	78	1.1
13-0	-	-	**75**	0.5	85	1.1	102	14.2	**75**	1.4	94	3
13-1	-	-	**66**	0.4	73	0.9	82	11.6	**66**	0.6	83	3.5
14-0	74	3.6	**58**	0.2	68	0.9	82	11.9	59	0.7	77	3.9
14-1	93	4.7	83	0.7	75	0.9	98	14.5	**71**	0.8	93	3.7
15-0	98	4.9	82	0.4	86	1.2	107	15.9	**80**	1.5	103	4.5
15-1	89	4.9	70	0.3	72	1	98	14.3	**68**	0.8	85	3.8
16-0	114	10	93	0.8	97	1.8	121	32.7	**89**	2.3	106	6.9
16-1	109	9.9	85	0.6	91	1.6	116	32.5	**83**	1.8	97	6.1
17-0	116	10	99	0.9	107	2.1	125	30.6	**93**	1.7	116	8
17-1	120	11	**93**	1	102	2	133	37.6	95	2.4	117	7.4
18-0	154	15	126	3.5	139	2.6	172	52	**117**	4	145	12
18-1	-	-	87	0.9	89	1.7	105	42.8	**78**	0.8	97	9.2
19-0	130	20	103	1.3	113	2.8	144	75.8	**101**	3.1	127	15
19-1	118	18	93	1	100	2.6	124	60.3	**89**	1.4	119	16
20-0	141	21	118	1.6	137	3.5	154	118	**110**	4.1	131	13
20-1	134	21	111	1.1	114	2.9	155	72.9	**106**	3.6	127	14

Table 2: Logistics domain

Mic-10

AcE did clearly better then HSP-2 and AltAlt in the Mic-10 domain, concerning both planning time and plan length. AcE produced plans of, at least, equal quality to those produced by GRT, although the latter was quite faster on average. AltAlt and STAN produced near optimal plans in all problems in very little time (FF needed less than a second for each problem).

FreeCell

For some reason, AltAlt did not manage to solve any ot the FreeCell problems. FreeCell proved to be a quite hard domain also for STAN and HSP-2. STAN solved only 8 problems and HSP-2 only 11 out of the total of 25. FF solved all the problems and was faster than GRT and AcE, but produced quite lengthy plans. GRT and AcE produced plans of almost the same quality (GRT did slightly better), with AcE being faster on average.

	AltAlt		FF		GRT		HSP-2		STAN		AcE	
Prob	**L**	**T**	**L**	**T**	**L**	**T**	**L**	**T**	**L**	**T**	**L**	T
2-0	**7**	0	**7**	0	**7**	0	8	0	**7**	0	**7**	0
2-1	**7**	0	**7**	0	**7**	0	**7**	0	**7**	0	**7**	0
3-0	**10**	0	**10**	0	11	0	12	0.1	11	0	11	0
3-1	**11**	0	**11**	0	**11**	0.1	**11**	0.1	**11**	0	**11**	0
4-0	15	0.1	**14**	0	**14**	0.1	15	0.1	**14**	0	**14**	0
4-1	15	0.1	**13**	0	**13**	0.1	14	0.1	**13**	0	14	0
5-0	18	0.1	**17**	0	18	0.1	20	0.1	**17**	0	18	0.1
5-1	18	0.1	**17**	0	**17**	0.1	18	0.1	**17**	0	19	0.1
6-0	21	0.2	**19**	0.1	20	0.2	23	0.3	20	0.1	22	0.2
6-1	22	0.3	**19**	0	20	0.2	23	0.2	**19**	0	20	0.2
7-0	**23**	0.4	**23**	0.1	25	0.3	27	0.4	**23**	0.1	**23**	0.3
7-1	24	0.4	**23**	0.1	27	0.3	28	0.4	24	0.1	24	0.3
8-0	28	0.7	**27**	0.1	29	0.4	32	0.7	**27**	0.2	28	0.5
8-1	31	0.8	**27**	0.1	29	0.4	31	0.6	**27**	0.1	29	0.5
9-0	33	1.1	**31**	0.1	33	0.6	36	0.9	**31**	0.2	33	0.9
9-1	31	1.1	**30**	0.1	33	0.6	34	0.9	**30**	0.3	31	0.9
10-0	35	1.9	**33**	0.1	34	0.7	39	1.5	**33**	0.4	34	1.1
10-1	37	1.9	**32**	0.1	33	0.7	39	1.3	34	0.3	33	1.1
11-0	41	2.9	**37**	0.2	41	1.1	42	1.9	**37**	0.6	41	2
11-1	38	2.8	**34**	0.1	35	1	44	1.9	35	0.7	37	1.7
12-0	41	3.5	**40**	0.2	41	1.3	48	2.7	**40**	0.8	41	2.5
12-1	43	3.5	**40**	0.2	43	1.4	47	2.6	41	0.8	43	2.6
13-0	46	4.9	**44**	0.3	45	1.7	50	3.6	**44**	0.4	46	4
13-1	45	4.7	**42**	0.2	**42**	1.8	51	5.6	**42**	0.8	43	3.2
14-0	49	8.1	**45**	0.2	47	2.1	55	4.9	**45**	1	48	4
14-1	50	6.7	**47**	0.3	52	2.3	55	5.2	**47**	0.9	50	4.5
15-0	52	11	**46**	0.3	49	2.6	58	6.5	47	1.1	51	6.9
15-1	52	9	**50**	0.3	56	2.9	59	6.7	52	1.2	58	7
16-0	55	12	**53**	0.4	55	3.3	63	8.3	54	1.3	56	9
16-1	59	14	**52**	0.4	**52**	3.4	63	8.2	**52**	1.6	56	10
17-0	59	15	**56**	0.5	60	4.2	68	11.8	59	1.4	59	13
17-1	57	19	**54**	0.4	58	4	67	11.5	55	1.6	56	15
18-0	63	20	**59**	0.5	63	5.1	71	14.5	**59**	1.8	63	15
18-1	63	24	**58**	0.6	62	4.9	71	13.1	59	2	62	14
19-0	68	26	**62**	0.7	69	6.3	75	20.2	63	2.7	68	17
19-1	69	25	**65**	0.7	69	6.3	76	16.5	**65**	2.7	69	19
20-0	71	33	**64**	0.7	70	7.3	79	20.2	65	3.1	71	21
20-1	74	40	**66**	0.7	69	7.1	80	25.7	67	2.6	73	24

Table 3: MIC-10 domain

	AltAlt		FF		GRT		HSP-2		STAN		AcE	
Prob	**L**	**T**	**L**	**T**	**L**	**T**	**L**	**T**	**L**	**T**	**L**	T
2-1	-	-	**9**	0.6	**9**	3	**9**	10	**9**	0.4	**9**	1.8
2-2	-	-	**8**	0.7	**8**	3	**8**	9.1	**8**	0.5	9	2.1
2-3	-	-	9	0.6	9	3	9	11	**8**	0.3	**8**	1.8
2-4	-	-	9	0.6	9	3	**8**	9.9	**8**	0.2	9	1.8
2-5	-	-	**9**	0.6	**9**	3	**9**	10	10	0.6	10	1.8
3-1	-	-	21	1.2	**16**	5.5	18	58.6	-	-	17	4
3-2	-	-	20	1.3	14	5.3	15	40.2	16	0.4	**13**	3.4
3-3	-	-	17	1.2	**14**	5.3	15	75.2	-	-	**14**	3.8
3-4	-	-	22	1.1	14	5.3	16	133	14	0.9	**13**	3.6
3-5	-	-	20	1.3	16	5.3	17	67	18	0.6	**13**	3.6
4-1	-	-	26	2.8	24	8	-	-	-	-	**23**	5.8
4-2	-	-	26	2.3	**19**	8	-	-	-	-	20	6.1
4-3	-	-	32	2.5	**20**	7.9	-	-	-	-	24	6.5
4-4	-	-	23	2	21	8	-	-	-	-	**19**	5.9
4-5	-	-	33	4.1	22	7.8	25	134	-	-	**19**	6.1
5-1	-	-	37	4.3	**29**	11	-	-	-	-	30	11
5-2	-	-	33	3	**24**	11	-	-	-	-	30	12
5-3	-	-	44	4.9	**27**	11	-	-	-	-	32	12
5-4	-	-	40	3.5	**27**	11	-	-	-	-	29	11
5-5	-	-	34	3.9	**30**	11	-	-	-	-	32	14
6-1	-	-	43	9.5	**38**	15	-	-	-	-	**38**	16
6-2	-	-	42	4.6	31	15	-	-	-	-	**30**	11
6-3	-	-	43	4.7	**32**	15	-	-	-	-	34	13
6-4	-	-	48	9.8	34	15	-	-	-	-	**33**	15
6-5	-	-	52	7	**38**	15	-	-	-	-	**38**	12

Table 4: FreeCell domain

Conclusion and Future Work

This paper presented a different approach to domain-independent heuristic planning in state spaces. The proposed heuristic is not based on distances between independent facts and the goals, but on distances between actions and the goals. This enables the heuristic to keep track of more interactions and yet remain simple enough to be executed with little computational cost.

The proposed heuristic has been embodied in a weighted A* regression planner, called AcE, and the planner has been tested on a variety of toy problems and compared with state-of-the-art planning systems. The results are quite promising, since they show that AcE is at least comparable to the planners excelling in the planning competitions.

It is in our direct future planners to investigate ways of further refining the heuristic. One possible way of doing this is by keeping track of more information for every action, than just its estimated distance, and taking also into account part of the information provided by the delete lists of the actions. Furthermore, we plan to extend AcE to handle many of the new features in PDDL 2.1, such as time and resources.

References

Blum, L., and Furst M., 1995. Fast planning through planning graph analysis, In Proeedings of the 14th Int. Joint Conference on Artificial Intelligence, 636-642. Montreal, Canada.

Bonet, B. and Geffner, H. 1999. Planning as Heuristic Search: New Results, In Proceedings of the 5th European Conference on Artificial Intelligence. Durham UK.

Bonet, B. and Geffner, H. 2000. Planning as Heuristic Search, Artificial Intelligence, Forthcoming.

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

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

Fox, M. and Long, D., 2000. Hybrid STAN: Identifying and Managing Combinatorial Subproblems in Planning. *In Proceedings of 19th UK Planning and Scheduling SIG Workshop,*.

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

Kautz, H. and Selman, B., 1996. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. *In Proceedings of the Thirteen National Conference on Artificial Intelligence (AAAI-96).* Portland, Oregon, USA.

Koehler, J. and Hoffmann, J. 2000. On Reasonable and Forced Goal Orderings and their Use in an Agenda-Driven Planning Algorithm, *Journal of Artiicial Intelligence Research* 12.

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

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

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

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

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.

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

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.

Refanidis, I. and Vlahavas, I. The GRT Planner: Backward Heuristic Construction in Forward State-Space Planning, *Journal of Artificial Intelligence Research* 15:115-161.

Selman, B., Levesque, H. and Mitchell, D., 1992. A New Method for solving Hard Stisfiability Problems. *In Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, 440-446. San Jose, California, USA.

Vrakas, D. and Vlahavas, I, 2001. Combining Progression and Regression in State-Space Heuristic Planning, *In Pre-Proceedings of the 6th European Conference on Planning*, 1-12. Toledo, Spain.