Using Backwards Generated Goals for Heuristic Planning

Vidal Alcázar, Daniel Borrajo, Carlos Linares López
Departamento de Informática
Universidad Carlos III de Madrid
Avda. de la Universidad, 30. Leganés (Madrid). Spain
valcazar@inf.uc3m.es, dborrajo@ia.uc3m.es, carlos.linares@uc3m.es

Abstract

Forward State Planning with Reachability Heuristics is arguably the most successful approach to Automated Planning up to date. In addition to an estimation of the distance to the goal, relaxed plans obtained with such heuristics provide the search with useful information such as helpful actions and look-ahead states. However, this information is extracted only from the beginning of the relaxed plan. In this paper, we propose using information extracted from the last actions in the relaxed plan to generate intermediate goals backwards. This allows us to use information from previous computations of the heuristic and reduce the depth of the search tree.

Introduction

The use of reachability heuristics along with a forward search algorithm has proved to be one of the most effective approaches in Automated Planning, as seen in the last International Planning Competitions. Most heuristics of this kind are based on the relaxation of the delete effects of the actions. In particular, the relaxed plan (RP) heuristic first used in FF (Hoffmann 2001) still remains a very effective way of guiding a forward search in the state space. It estimates the distance to the goal by solving a relaxed instance of the problem from the evaluated state and returning the number of actions of its solution plan.

Apart from the heuristic numeric value obtained, the actions that compose the RP offer additional information that can be exploited in the search process. Helpful actions (Hoffmann 2001) can be used to prune unpromising successors in domains with a high branching factor; look-ahead states (Vidal 2003) are an attempt to advance several steps at once in the search state; and macro-actions (Botea, Müller, and Schaeffer 2007) may be inferred online from the RP to redefine the domain and reduce its depth.

A common aspect of these techniques is that they all take into account only the first actions of the RP. This is reasonable, given that these techniques require legal sequences of actions in the evaluated state, and RPs have a higher possibility to include illegal actions the farther the actions are from the evaluated state. However, this ignores potential additional information the RP may contain.

In this work we propose a novel method to take advantage of information from the last actions of the RP. The motivation behind this work is that the last actions in the RP are often similar across different calls to the heuristic function. This fact allows to improve the search by: reusing information from previous computations of the heuristic; or getting more accurate heuristic estimations. The way of exploiting this observation is by generating intermediate goals backwards by using those actions from the last part of the RP. This way, in subsequent computations of the heuristic, the closest intermediate goal to the current state can be detected. We have called this approach the Backwards Generated Goals (BGG) heuristic.

Those goals provide additional information that can be used to reduce the number of levels used in the reachability heuristic or to get heuristic estimations closer to the real value than other heuristics. Besides, this technique allows to finish the search earlier when satisfying an intermediate goal, as the path to the original goal can be built by tracing back the generation of the intermediate goal. This relies on a careful generation of extra information together with the intermediate goal.

To ensure the validity of the backwards generated goals, actions must legally support the reached goal. This is done by using concepts from regression heuristic planners like HSPr (Haslum and Geffner 2000): considering delete effects when supporting goals and taking into account static mutexes between the preconditions of the actions and the goal propositions not satisfied by the supporting actions. Experimental evaluation of the techniques presented in this paper shows improvements in coverage and number of expanded nodes over the regular FF heuristic. In particular, the approach seems to improve performance substantially in traditionally hard domains to common reachability heuristics.

The structure of the paper is as follows: first, some background on propositional planning and reachability heuristics is given. Afterwards, the motivation of the work will be explained, describing then the details of the employed techniques both conceptually and in terms of implementation. An example in the Gold-Miner domain will be added to illustrate how the technique works. Finally, some results will be presented, followed by a discussion including related work and possible future developments.

Background

In this section, we will formally define propositional planning and define reachability heuristics, focusing on the RP heuristic.

Propositional Planning

Automated Planning can be described as the task of coming up with an ordered set of actions (in the common case a sequence) that achieves a set of goals from a given initial state. We consider then a standard formalization of a planning problem as a tuple $P=(S,A,I,G)$ where S is a set of atomic propositions (also known as facts), A is the set of grounded actions derived from the operators (action schemes) of the domain, $I \subseteq S$ is the initial state and $G \subseteq S$ the set of goal propositions. The actions that are applicable depend on several propositions being true and their effects can make propositions true of false, which is commonly known as *adding* and *deleting* the propositions, respectively. Thus, in terms of a STRIPS (Fikes and Nilsson 1971) planning instance, an action would be defined as a triple $\{pre(a), add(a), del(a)\}$ in which $a \subseteq A$ and $pre(a), add(a), del(a) \subseteq S$.

Reachability Heuristics and the RP Heuristic

When using a heuristic search algorithm, the estimation of the distance to a goal state is obtained by solving the same problem, but after introducing some kind of relaxation. The most common relaxation modifies the actions, removing their delete effects, actually becoming the tuple $\{pre(a), add(a), \emptyset\}$. The value of the heuristic function is computed from the solution of this problem. Computing the optimal solution is still an NP-complete problem (Hoffmann 2001) so in most cases the heuristic is non-admissible, making it impossible to ensure optimality.

The relaxed plan heuristic originally made use of the planning graph implemented by Graphplan (Blum and Furst 1995). It builds a layered directed graph alternating facts and actions until all goals appear at some level, ignoring the possible mutual exclusions. Once the graph has been generated, a plan is extracted by a backtrack-free search algorithm in which actions that make the goals true are greedily selected. The supported goals are removed and the preconditions of the actions are added to the goal set, until all the goals belong to the initial state. The actions are chosen taking into account the level they first appeared, choosing those with lower levels. The heuristic value returned is the number of actions in the relaxed plan.

While planning as heuristic search is the paradigm used by most state-of-the-art planners, it has some disadvantages. Even though reachability heuristics can be computed in polynomial time, they still require heavy computation. Besides, due to the complex interactions that often appear in planning problems, heuristics must be recomputed for every state from scratch. In fact, it has been reported that the computation of the heuristic takes up to 80% of the time to find a solution (Liu, Koenig, and Furcy 2002). While this has been addressed by earlier works (Refanidis and Vlahavas 2001; Liu, Koenig, and Furcy 2002), reducing the impact of this heavy computation required for heuristic evaluation still remains an open question.

One way to alleviate this is to extract additional information from the heuristic computation. Reachability heuristics solve a relaxed version of the problem, and thus yield a relaxed plan as a solution. The RP may contain actions in common with a possible solution, which can favorably bias the search. Helpful actions and look-ahead states are the most prominent examples of this kind of techniques. In particular, helpful actions are sometimes more relevant than the quality of the heuristic used (Richter and Helmert 2009). Since these actions must have the possibility of belonging to a real solution, they must be legal actions even in the relaxed case. This is ensured by checking that the action is applicable in the evaluated state in the case of helpful actions, and that none of the preconditions of the actions that lead to the look-ahead state is deleted by another action. Therefore, this limits the actions to those close to the evaluated state.

However, there are ways of checking the legality of the actions when they are close to the goal as well. This is shown by the existence of planners that do regression search. This gives the opportunity to extract similar information from actions far from the beginning state. In forward state search, goals are static. Thus, in many cases the last actions that lead to the goal in the RP will be the same for different heuristic computations. In fact, they may even belong to a solution. Hence, generating sets of intermediate goals as done in regression search using the information derived from the RP may allow reusing information from relaxed plans obtained in the evaluation of other nodes. This can benefit the search in several ways like: decreasing the depth of the search in both the heuristic computation and the forward state search; and capturing constraints appearing in areas of the search space around the goal. As it will be discussed in the Related Work section, this is similar to the work on landmarks (Richter, Helmert, and Westphal 2008), but it also presents some important differences.

Intermediate Goals and Reachability Heuristics

Reachability heuristics are computed using a relaxed Graphplan. This is actually a breadth-first search in the space of relaxed problems. There is an important observation about this: breadth-first search exhibits a wave-like behavior. Therefore, it not only gives an estimation of the distance to a goal, but also estimates that the first reached goal is the closest to the state in case multiple goal states are present. To better illustrate this, let us consider the example shown in Figure 1. There is a grid in which some cells are labeled as goal cells, and a key is needed to open a chest that contains the gold. There is only one key, and it is available in the initial state. An agent can only move to adjacent cells, and there are obstacles. The key is lost when going through them. In this case, in the heuristic computation, the first level of the relaxed planning graph will check whether any of the cells at distance one is a goal cell; at the second level, all the cells at distance 2 will be checked, and so on. The heuristic computation will find the closest goal cell in the relaxed

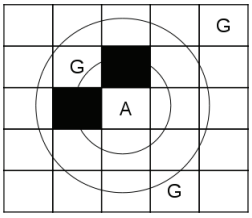

Figure 1: The agent is located in A and cells marked with G are goal cells. In the heuristic computation, only two levels are expanded estimating that the goal cell to the left and above the agent is the closest one.

problem, independently of how many goal cells exist in the grid. The relaxed plan may be illegal (if for example there is an obstacle in the way to a gold cell, since the key would be lost), but it still gives an estimation to the closest goal.

In most domains the set of goal propositions does not describe a complete state (an exception being the N-Puzzle domain, for instance), so the reachability analysis deals with multiple goal states implicitly. The goal (set of goal propositions in G) does not describe a complete state when the value of some proposition in the instantiated problem is not defined. For example, if S=$\{a,b\}$ and G=$\{a\}$, both s_1=$\{a,b\}$ and $s_2 = \{a, \neg b\}$ are goal states. However, this fact does not change the behavior of the heuristic, meaning that regular reachability heuristics effectively take into account multiple potential goals. This interesting property can be extended to the case in which there are different sets of goal propositions as well.

Taking advantage of this fact, the key observation of this work is to generate multiple intermediate goals that lead to the original goal. Actions known to lead to the problem goals are used to generate the intermediate goals. At each call to the heuristic function, a new intermediate goal set G' is obtained so it can be added to the set of goals $\mathcal{G}$. Initially, $\mathcal{G} = \{G\}$. In the first call to the heuristic function, there will only be a set of goals G. Then, G' is generated by choosing an action a from the RP that supports one of the goal propositions $g \in G$. We remove the propositions in G supported by a and add the preconditions of the action (technique known as goal regression, which determines the qualification of a *backwards generated goal*). Then, this new set G' may be added to $\mathcal{G}$, a list of goals relevant to subsequent computations of the heuristic. Whether a backwards generated goal is added or not to the list depends on its potential usefulness. Usually, the intermediate goal obtained from a node with a bad heuristic value is probably not useful and hence discarded, although the criteria may vary depending on the forward search algorithm used. An example of a possible criteria when using greedy best-first search will be given in the experimentation section. Lastly, the chosen action a and the originally reached goal proposition g are also stored together with G' with two purposes: allowing reconstructing a path by tracing back the chosen actions; and obtaining the distance from the intermediate goal to the original goal, that is the cost of the intermediate goal. Optimality on this distance cannot be ensured, as there may be shorter paths from the intermediate goal to the original goal.

In the following iterations, there will be several goal sets in $\mathcal{G}$. So, instead of finishing the expansion of the relaxed planning graph when all goal propositions $g \in G$ appear in a given layer P_i, the expansion finishes in a propositional layer P_i when $\exists G_j \in \mathcal{G}$ such that $G_j \subseteq P_i$. In that case, the steps in the previous paragraph are executed to generate a new goal set G' to be added to $\mathcal{G}$.

The implementation of this technique is closely related to how actions are known to be applicable at a given level, as the preconditions of any action form a subgoal themselves. Each proposition maintains a list of indexes of the sets of goals $G_i \in \mathcal{G}$ they appear in. Also, we define a counter that keeps the number of unsatisfied propositions in each intermediate goal set G_i. Whenever a goal proposition p is satisfied by a new action in the relaxed planning graph, the counter of the goal sets G_i where it appears ($p \in G_i$) is decreased. When the counter of any G_i reaches zero, there is a RP that can reach G_i.

A key difference with respect to the standard computation of the RP heuristic is that we require at least one action that reached the intermediate goal set in the last step to be a *legal support*. The concept of legal support is based on the application of the action in the search space: it must be able to appear as the last action in a valid solution plan. There are three conditions that must be fulfilled to ensure that a supporting action a is legal for supporting G_i (thus, being able to generate a new G'):

- its delete effects must not include a proposition appearing in the reached goal ($del(a) \cap G_i = \emptyset$);
- the preconditions of the action must not be mutually exclusive with any proposition of the goal not supported by the action; and
- the new set of goal propositions must not have been previously generated ($G' \notin \mathcal{G}$)

The first constraint is straightforward: an action that deletes a goal proposition can never be the last action of a plan, as it necessarily leads to a non-goal state. In fact, this constraint ensures that there is a legal path from a given intermediate goal to the original goal, although it might not be enough since sets of unreachable goal propositions may be generated.

The second constraint is related to the concept of mutual exclusivity between propositions as it was originally described in (Haslum and Geffner 2000). A static relation of mutual exclusivity between propositions (or static mutex) is a set of propositions $M = (p_0, \ldots, p_n)$ for which there is no reachable state $s \subseteq S$ such that two or more elements from M are true. Static mutexes may not suffice to detect all the unreachable sets of propositions in the search space, but in many domains they are able to prune most unreachable states, as shown by regression planners. The complexity of the computation of these sets grows exponentially with the number of elements in the set. Thus, only mutexes between pairs of propositions will be computed, as such sets can be computed in reasonable time using the h^2 (Haslum and Geffner 2000) heuristic. h^2 gives a lower bound of the

distance from the initial state to a state in which any two propositions are true. If this bound is infinite, those two propositions are a static mutex.

The basic idea of the third constraint is that when a duplicated goal is generated, that goal was necessarily already supported in the relaxed graphplan. Then, if the heuristic computation did not stop at an earlier level when supporting that goal, this means that it may be an unreachable goal as no legal action was found for it so far. This can be explained taking into account the leveled approach of the heuristic computation. Intermediate goals are generated by taking away the supported propositions and adding the actions preconditions. Preconditions must be satisfied at earlier levels than the propositions the action supports. If the new set of propositions that includes the preconditions of the action was already an intermediate goal, this means that this goal had already been satisfied in earlier levels of the heuristic computation. This is so because the already existing intermediate goal only differs from the reached goal in the preconditions and effects of the action, and these preconditions belong to earlier levels.

Implementation Details

It is possible for a given set of goals to be legally supported in several ways. In this case, one action must be selected. First, a goal proposition to be supported must be chosen. In the current implementation, propositions are heuristically chosen by the level in which they first appear, preferring propositions farther from the initial state. The intuition behind this decision is that these propositions seem more difficult to satisfy and thus they are probably achieved in later stages of the solution plan. This concept is loosely related to the way actions are heuristically chosen when generating the RP, although in a reversed way. Second, if the proposition has several supporting actions, those appearing closer to the initial state are chosen first, as their preconditions are deemed easier to achieve. The reachability analysis is described in Algorithm 1, while the way a legal action is sought and chosen to generate a new intermediate goal is described in Algorithm 2.

As depicted in Algorithm 1, it is possible for the relaxed graphplan to level out before a supporting action is found. This is due to the constraints on the legality of the supporters. In this case, the goals are unreachable and the state is then a dead end.

Intermediate goals also affect the forward search stopping criterion. Because of the first constraint on the legality of the supporters, a legal path can always be built from any intermediate goal to the original goal by tracing back the actions used in the intermediate goal generation. Consequently, once the forward search satisfies a goal, be it an intermediate one or the original one, the search stops and returns a valid plan. Stemming from this fact, two different heuristic values can be extracted. Since intermediate goals store the distance to the original goal, the value extracted in the heuristic computation may be either the distance to the intermediate goal, which is the length of the RP extracted from that goal, or the distance to the original goal as the former value plus the stored computation of the distance from

Algorithm 1: Heuristic Computation with Intermediate Goals.

```
Data: Current State s ⊆ S, G
Result: Intermediate Goal igoal
begin
    SatisfiedPropositions ⟵ s
    while not leveledOut(G) do
        NewlySatisfied ⟵ ∅
        foreach p ∈ SatisfiedPropositions do
            foreach GoalReference ∈ p do
                Unsatisfied ⟵ Unsatisfied − 1
                if Unsatisfied = 0 then
                    igoal ⟵
                    LegalGoal(GoalReference)
                    if igoal ≠ NULL then
                        return igoal
            foreach ActionReference ∈ p do
                Preconds ⟵ Preconds − 1
                if Preconds = 0 then
                    addEffectstoNewlySatisfied
            SatisfiedPropositions ⟵
            NewlySatisfied
    return NULL
end
```

Algorithm 2: LegalGoal Function.

```
Data: Intermediate Goal igoal
Result: New Intermediate Igoal igoal
begin
    /* Propositions are sorted by the
       level they were satisfied in,
       those at later levels first   */
    foreach Proposition ∈ igoal do
        foreach SupportingAction ∈ Proposition do
            newIgoal ⟵ igoal
            take out AddEffects from newIgoal
            if SupportingAction deletes newIgoal then
                Continue
            if IsMutex(Preconds, newIgoal) then
                Continue
            add Preconds to newIgoal
            if newIgoal already exists then
                Continue
            return newIgoal;
    return NULL;
end
```

the intermediate goal to the original one. If the main objective is finding a solution regardless of its quality, the distance to the intermediate goal is good enough, as the forward state search only has to reach an intermediate goal to build a valid plan. If plan length is important, adding both values may give a more accurate estimation than the regular RP heuristic thanks to delete effects being taken into account to some degree. Just like the FF heuristic, both cases can be extended to a metric different from plan length too, being cost the most common one.

An Example of Backwards Goal Generation

To show how the process of backwards generating goals works, we will include an example from a challenging domain. The domain is the Gold-Miner from the learning track of the International Planning Competition held in 2008[1]. This is a domain similar to the aforementioned grid domain. The goal in this case is picking up the gold, which appears at only one cell. On the grid there are rocks that block the way, that can be either hard or soft. There is a laser and unlimited bombs which can be used to clear the rocks. Soft rocks can be destroyed with either one, but hard rocks require the laser. Besides, when using a bomb it is lost and another one must be picked up from the pile of bombs if needed. The drawback of the laser is that it deletes the gold proposition if used to clear up the way in the cell where the gold is, leading to a dead end. The difficulty of this domain lies in the fact that since delete effects are ignored, using the laser has no drawbacks when solving the relaxed problem. Therefore, once the laser is picked up (which is often mandatory to clear the way through a hard rock) picking up a bomb to destroy the last rock is seldom considered in the relaxed plan. This is further aggravated if helpful actions are used, as the action of picking the bomb up is never considered as helpful.

The first steps of the backwards generation are the following (assume that c1 and c2 are adjacent cells):

```
Goal: (holds-gold)

Step 1: (pick-gold c2) -> (holds-gold)
-New goal: (robot-at c2), (gold-at c2), (arm-empty)

Step 2: (move c1 c2) -> (robot-at c2)
-New goal: (robot-at c1), (clear c2), (arm-empty),
           (gold-at c2)

Step 3: (fire-laser c1 c2) -> (clear c2)
                        ;; deletes (gold-at c2)
        (detonate-bomb c1 c2) -> (clear c2), (arm-empty)
-New goal: (robot-at c1), (holds-bomb), (gold-at c2)
```

The original goal is $G_1 = \{$*(holds-gold)*$\}$. There is only one action that supports it with no other effect, *(pick-gold c2)*, so the proposition it satisfied is substituted with its preconditions, generating a new set of goals, $G_2 = \{$*(robot-at c2), (gold-at c2), (arm-empty)*$\}$. After this step, there are two sets of goal propositions: the original one, G_1 and the new one, G_2. In the second step, G_2 is satisfied in an earlier step when expanding the relaxed plan graph. Both *(robot-at c2)* and *(arm-empty)* can be legally supported (*(gold-at c2)* is already true in the initial state). As it was explained before, ties among actions that legally support goal propositions are broken choosing those that support propositions first satisfied at later levels. Because of this, the action *(move c1 c2)* is the one chosen to generate an additional intermediate set of goals, $G_3 = \{$*(robot-at c1), (clear c2), (arm-empty), (gold-at c2)*$\}$.

In the third step, G_3 can be supported by several actions again. In particular, to clear the way towards the goal the robot can either fire the laser or use a bomb. Nevertheless, since delete effects are taken into account in the backwards goal generation (first constraint of legal support), *(fire-laser c1 c2)* appears as illegal and is discarded. Thus, *(detonate-bomb c1 c2)* is chosen instead. This captures the hardest difficulty in the domain and allows easily solving the problem. Actually, if $\{$*(robot-at c1), (holds-bomb), (gold-at c2)*$\}$ was given to a planner using the RP heuristic instead of the original goal, the instance would be most likely trivial, as the proposition *(holds-bomb)* strongly bias the generation of the RP and feeds the forward search with the right helpful actions.

Experimentation

Both the RP heuristic and the technique presented in this work (which we called BGG, Backwards Generated Goals) have been implemented on top of JavaFF (Coles et al. 2008). Because of the characteristics of BGG, it has been implemented as a forward state search algorithm with a dual queue as in FastDownward (Helmert 2006). One of the queues uses the BGG heuristic, while the other uses the regular RP heuristic. The reason behind this is that when computing the BGG heuristic, extracting a RP from the original goal takes little additional effort – the reachability analysis is already done and the best supporters are already computed – and may be helpful for the cases in which the BGG heuristic is strongly guided towards an unreachable intermediate goal.

Greedy best-first search has been used as the search algorithm. Greedy best-first search expands in every step the most promising node determined by the function *f(n) = h(n)* in which *h(n)* is the heuristic function of the node. The use of intermediate goals requires some modifications, though. For every evaluated node, an intermediate goal is obtained. States with bad heuristic values are less likely to be on the path of a possible solution plan. Hence, they may yield intermediate goals that may mislead the search. To avoid this, intermediate goals are associated with the state along with its heuristic value in the open list instead of being added to the goal list. When a state is expanded its intermediate goal is added to the list. This way, all the goals are generated from the best state at every iteration.

Helpful actions for the BGG heuristic are the union of the sets of helpful actions obtained using the BGG RP and the regular RP from the original goal. Helpful action pruning has been enabled, but no restarts with the full set of successors have been done for the cases in which the planner was not able to find a solution with helpful action pruning under the time limit. This is done in order to analyze the impact of

[1] IPC 2008 website: http://ipc.informatik.uni-freiburg.de/

the extra helpful actions provided by the more informed RPs from intermediate goals.

The focus of the experimentation is the coverage (percentage of solved problems out of the total number of problems) of the proposed technique. Consequently, we have avoided some benchmark domains in which state-of-the-art planners are able to solve most problems in little time. In particular, the selected domains were Gold-Miner, Matching-BW, N-Puzzle and Sokoban from the learning track of IPC-6 and Storage, Peg-Solitaire, Scananalyzer and Transportation from the deterministic track of the same competition. For the sake of completeness, two other older domains were chosen as well, the well-known Driverlog Domain from IPC-3 and the Pipesworld domain from IPC-4.

Experiments have been done on a Dual Core Intel Pentium D at 3.40 Ghz running under Linux. The maximum available memory for the planner was set to 1GB, and the time limit was 1800 seconds. The heuristic value of BGG is the distance to the closest intermediate goal without adding the distance from that goal to the original one. No parameter setting is necessary. Table 1 shows the results for these domains in terms of percentage of solved problems. Geometric mean and median of the ratio between states evaluated by BGG and RP are also displayed, with values above one meaning that BGG evaluates fewer nodes. Overall, the coverage improves for BGG, but there is a notable trend: the harder a domain is, the greater the difference is between both techniques. Not taking into account Gold-Miner, which has a very hard constraint close to the goal and hence is the ideal case for BGG, domains in which BGG performs better have dead-ends and strong order interactions (Matching-BW, Sokoban) whereas in those without them there is no improvement. Regarding the number of evaluated nodes, BGG shows a similar behavior. Also, as the size of the problem increases, BGG tends to expand fewer nodes compared to the regular RP heuristic in spite of having a higher branching factor because of the additional helpful actions. This means that BGG is consistently more informed on the long run due to the increasing number of intermediate goals.

To better understand the differences between both approaches, some features of the domains that affect the performance of BGG can be highlighted. Matching-BW has numerous dead-ends which can be pruned earlier thanks to the constraints on the legality of the supporting actions; besides, towers of blocks impose order restrictions that can be found more easily using regression. Sokoban is characterized by having tunnels, which are implicitly exploited by the backwards goal generation, leading to a significant decrease in the depth of the search. Storage and Scananalyzer have similar characteristics to Matching-BW, but in Storage the search does not benefit from BGG as much as expected, probably because the correct goal ordering needs longer sequences of actions to be found when doing goal regression. N-Puzzle has no particularities that BGG may benefit from, and so coverage is not improved, but for those problems solved by both approaches BGG expands fewer nodes. Pipesworld, Driverlog and Transportation are domains in which the RP heuristic is relatively well informed, and hence the margin of improvement for BGG is small. Peg-Solitaire seems to be a similar case to N-Puzzle, although in this domain coverage decreases significantly for BGG.

Domain	RP	BGG	Mean-S	Median-S
Driverlog	70	**80**	0.89	0.84
Gold-Miner	50	**100**	5.59	4.27
Matching-BW	3	**33**	9.72	9.72
N-Puzzle	10	**13**	4.13	4.25
Peg-Solitaire	**97**	80	1.67	1.71
Pipesworld	**28**	22	0.94	0.62
Scananalyzer	33	**67**	1.47	1.09
Sokoban	13	**17**	5.52	7.22
Storage	53	**60**	1.91	0.49
Transportation	63	**70**	0.7	0.68
Average	42	**57.5**	3.25	3.09

Table 1: Comparison between the RP heuristic and the BGG heuristic in terms of coverage and number of evaluated nodes (geometric mean and median of the ratio between evaluated states). Instances that were solved by both approaches expanding fewer than 100 nodes have not been considered.

Regarding quality, in those domains in which BGG improves the number of evaluated nodes it tends to improve plan quality as well, as seen in Table 2. In domains in which the number of evaluated nodes is similar, plan quality is worse for BGG, although on average BGG finds slightly better plans. Analyzing the solution plans, it can be observed that the sequence of actions traced back from the reached intermediate goal tends to be of better quality than the part derived from the nodes expanded in the forward search. This was to be expected: taking delete effects into account yields a more accurate heuristic value and in the beginning of the search there are no backwards generated goals which may improve the quality of RPs. Still, experimentation has been done without adding the distance from the reached goal to the original one to the heuristic value of the evaluated nodes, which may prove useful when aiming for better quality plans.

Also, theoretically, the higher the average length of the part of the solution plan traced back from the reached intermediate goals is, the more relevant the backwards generation of goals is. Therefore, this can be considered a measure of how appropriate BGG is for some domains. Table 2 shows that this holds for most domains with the exceptions of N-Puzzle and Transportation. On average, around a fifth of the plan is retrieved from the reached intermediate goal. This also explains why BGG expands fewer nodes: the search space that must be explored to find a solution is potentially much smaller due to the reduced depth.

Timewise, the increasing number of intermediate goals makes the heuristic more expensive to compute. A priori, since the number of goals grows exponentially, so should do the time needed to compute the heuristic. In practice, however, the time needed increases linearly because intermediate goals tend to appear at earlier levels as the search progresses and those at later levels are not checked. In gen-

Domain	Mean-Q	Median-Q	Ratio-BGG
Driverlog	1.08	1.08	0.17
Gold-Miner	2.35	2.76	0.27
Matching-BW	1.26	1.26	0.24
N-Puzzle	1.25	1.25	0.01
Peg-Solitaire	0.98	1	0.32
Pipesworld	1.04	1.04	0.14
Scananalyzer	0.94	0.85	0.25
Sokoban	1.09	1.11	0.19
Storage	0.93	0.93	0.21
Transportation	0.99	1	0.04
Average	1.19	1.24	0.18

Table 2: Comparison between the RP heuristic and the BGG heuristic in terms of quality. Again, geometric mean and median of the ratio between evaluated states has been used. Ratio-BGG is the percentage of the length of the solution part that belongs to the sequence of actions traced back from the reached intermediate goal.

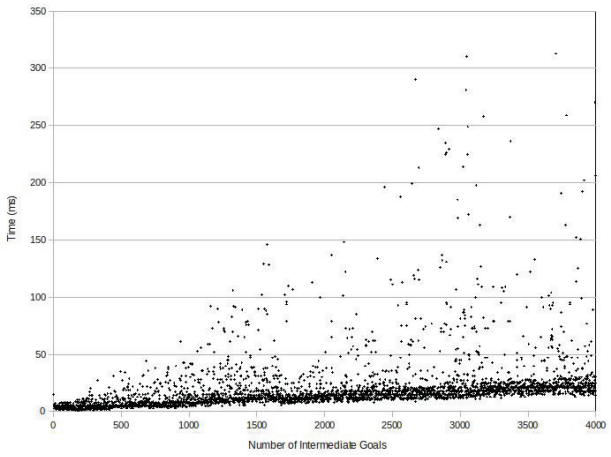

Figure 2: Time needed to compute the heuristic in milliseconds as the number of goals increase.

eral, in those domains in which the number of expanded nodes decreases when using BGG, it tends to find the solution in a similar time than when using the RP heuristic; in domains in which BGG does not improve the number of nodes, planning time may increase up to an order of magnitude. Figure 2 shows the time needed to compute the heuristic as the number of intermediate goals increases in the eleventh problem of the Pipesworld domain. As it can be observed, the main tendency shows that time increases linearly and only slightly, going from around 5 milliseconds at the beginning to around 25 milliseconds when 4000 intermediate goals have been generated. Still, there are two important factors that make BGG more costly in terms of time: first, the number and magnitude of outliers increase as intermediate nodes are generated; and second, heuristic computation accounts for most of the time spent by forward search heuristic planners, so even small increases in time in the heuristic computation have a great impact on the overall performance.

Related Work

A conceptually close idea, RRT-Plan (Burfoot, Pineau, and Dudek 2006), relies on generating intermediate states by incrementally choosing subsets of the goal and satisfying them. Thus, the intermediate goals generation is completely different, being a random selection of subsets of the original goal set in each iteration in their case. Interestingly, the authors proposed implementing a bidirectional search algorithm as future work, but this idea has not been developed further.

A field under intense study, landmarks for automated planning (J. Hoffmann 2004; Richter, Helmert, and Westphal 2008) also has some similarities with this work. Although being computed in a preprocessing step, landmarks are intermediate goals themselves and thus provide some valuable information to the search. In particular, they capture some constraints in the problem as well as orders among propositions, in a similar fashion to what backwards generated goals do in areas close to the original goal.

From a procedural point of view, a recent work on low-conflict RPs (Baier and Botea 2009) shares important aspects with this approach. In their work, they take into account delete effects and pairs of mutually exclusive propositions in the backwards search done to retrieve a RP from the reachability analysis to provide more accurate RPs. Furthermore, they expand additional levels in the computation of the heuristic, being the main difference that the number of extra levels are expanded based on a preliminary estimation whereas the expansion of levels in this work is systematic.

Conclusions and Future Work

The main contribution of this work consists on understanding the relation of the last actions in RPs to potential solution plans, and how this idea can be used to improve search performance.

We have presented a new approach that combines forward state search and backwards goal generation with reachability heuristics and relaxed plans as the common core. BGG presents several advantages: it reduces the depth in both the relaxed graph used in the heuristic computation and the state space search; it detects constraints close to the goal, which also provides an advantage for the forward state search by generating more informed RPs; and it uses additional information, such as delete effects and mutually exclusive pair of propositions, to enrich the reachability analysis.

There are some disadvantages inherently derived from the use of intermediate goals: the heuristic must take into account an increasing number of goals, which is exponential in the worst case (though this effect has not been found in the experiments); and the distance to an intermediate goal used as the heuristic value by the nodes in the open list becomes obsolete as new intermediate goals are generated. Summarizing the results of the experiments, it usually pays off to generate intermediate goals during search. Out of the ten domains used, it improves the coverage in all but two of

them. Besides, it is competitive in terms of time and quality as well. It also seems to work better in domains traditionally hard for regular reachability heuristics, allowing planners to scale not only in size but also in complexity of the problems.

Future work can be: on a further analysis of the impact of the techniques developed in this work; and implementing other known techniques to improve the performance. It is still unclear how much every technique helps to improve the traditional RP heuristic. An insightful example of this is the aforementioned work on low-conflict plans (Baier and Botea 2009), which uses a similar approach by taking into account delete effects and mutexes, but in a less ambitious way. This way, ideas like forcing the reachability analysis to legally support every proposition instead of only one might greatly affect the performance, which requires a deeper understanding.

On the other hand, some known techniques intuitively seem to be well suited to the backwards generation of goals. Look-ahead techniques are a prominent example, which are known to work well in many domains with forward state planners. The twist in this case is to combine actions from the last part of the RP to generate intermediate goals more than one step farther from the reached goal. Another technique that offers a great potential is the combination of this work with that of landmarks. Or, similarly, the work on reasonable orders, such as the one that creates a goal agenda (Hoffmann 2001) or to add precedence constraints in heuristic computations (Cai, Hoffmann, and Helmert 2009). In this case, they can be used to impose additional constraints in the backwards goal generation, taking advantage of the strong order interactions among goals.

It would also be interesting to analyze the potential links between backwards goal generation and bidirectional/perimeter search (Dillenburg and Nelson 1994). In fact, this approach can be seen as a sort of imbalanced bidirectional algorithm in which the backwards search uses the last actions of the RPs as a guide and the forward search estimates the distance to the frontier of the backwards search instead of to the original goal.

Acknowledgments

This work has been partially supported by a FPI grant from the Spanish government associated to the MICINN project TIN2008-06701-C03-03.

References

Baier, J. A., and Botea, A. 2009. Improving planning performance using low-conflict relaxed plans. In *19th International Conference on Automated Planning and Scheduling*, 10–17.

Blum, A. L., and Furst, M. L. 1995. Fast planning through planning graph analysis. *Artificial Intelligence* 90:1636–1642.

Botea, A.; Müller, M.; and Schaeffer, J. 2007. Fast planning with iterative macros. In *IJCAI'07: Proceedings of the 20th international joint conference on Artifical intelligence*, 1828–1833. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Burfoot, D.; Pineau, J.; and Dudek, D. 2006. RRT-Plan: a randomized algorithm for STRIPS planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 362–365.

Cai, D.; Hoffmann, J.; and Helmert, M. 2009. Enhancing the context-enhanced additive heuristic with precedence constraints. In *ICAPS*.

Coles, A. I.; Fox, M.; Long, D.; and Smith, A. J. 2008. Teaching forward-chaining planning with JavaFF. In *Colloquium on AI Education, Twenty-Third AAAI Conference on Artificial Intelligence*.

Dillenburg, J. F., and Nelson, P. C. 1994. Perimeter search. *Artif. Intell.* 65(1):165–178.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3-4):189–208.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *In Proceedings of the 5th International Conference on AI Planning and Scheduling (ICAPS)*, 140–149. AAAI Press.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J. 2001. FF: The fast-forward planning system. *AI magazine* 22:57–62.

J. Hoffmann, J. Porteous, L. S. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research* 22:215–287.

Liu, Y.; Koenig, S.; and Furcy, D. 2002. Speeding up the calculation of heuristics for heuristic search-based planning. In *In Proceedings of the National Conference on Artificial Intelligence (ICAPS)*, 484–491.

Refanidis, I., and Vlahavas, I. 2001. The GRT planning system: Backward heuristic construction in forward state-space planning. *Journal of Artificial Intelligence Research* 15:115–161.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In *ICAPS*.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-2008*, 975–982. AAAI Press.

Vidal, V. 2003. A lookahead strategy for solving large planning problems. In *IJCAI'03: Proceedings of the 18th international joint conference on Artificial intelligence*, 1524–1525. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.