The Roamer Planner
Random-Walk Assisted Best-First Search

Qiang Lu
University of Science and Technology of China
Hefei, Anhui, China
rczx@mail.ustc.edu.cn

You Xu, Ruoyun Huang, Yixin Chen
Washington University in St. Louis
St. Louis, MO, USA
{yx2,ruoyun.huang,chen}@cse.wustl.edu

Abstract

Best-first search is one of the most fundamental techniques for planning. A heuristic function is used in best-first search to guide the search. A well-observed phenomenon on best-first search for planning is that for most of the time during search, it explores a large number of states without reducing the heuristic function value. This phenomenon, called "plateau exploration", has been extensively studied for heuristic search algorithms for satisfiability (SAT) and constraint satisfaction problems (CSP).

In planning, plateau exploration consists of most of the search time in state-of-the-art best-first search planners. Therefore, their performance can be improved if we can reduce the plateau exploration time by finding an exit state (a state with better heuristic value than the best one found so far). In this paper, we present a random-walk assisted best-first search algorithm for planning, which invokes a random walk procedure to find exits when the best-first search is stuck on a plateau. The resulting planner, Roamer, building on the LAMA and Fast-Downward planning system, uses a best-first search in first iteration to find a plan and a weighted A* search to iteratively decreasing weights of plans. Roamer is an anytime planner which continues to search for plans with better quality until exhausting the whole state space or being terminated because of time limits.

Introduction

Roamer is a best-first state space search planner based on Fast-Downward and LAMA planners (Helmert 2006; Richter and Westphal 2010). The core feature of Roamer is the use of Monte-Carlo Random Walks assisted heuristic search to escape from plateaus. The Monte-Carlo random exploration method for deterministic planning is introduced in (Nakhost and Msller 2009).

One of the most successful approaches to planning is best-first search. Best-first search typically employs a heuristic function that maps any state to a real number that estimates the distance to goal. The number of states expanded by best-first search depends largely on the quality of the heuristic function. Best-first search with a perfect heuristic function only needs to expand $O(|L|)$ states where L is the solution path from the initial state to a goal state (Russell and Norvig 2003). On the other hand, best-first search for planning with almost perfect heuristic may still explore an exponential number of states before finding a goal (Helmert and Röger 2008). In practice, since the length of the solution path L is much smaller than the number of expanded states, it is easy to see that most of the states explored by a best-first search are not on the solution path.

During the best-first search, for any state s explored, we define the incumbent heuristic value $h^*(s)$ as the smallest heuristic function value of all states explored so far till s. Evidently, h^* decreases monotonically during search and finally reaches 0 when a goal is found.

For a given planning problem, let $\mathcal{S}$ be the set of all generated states, since we have $|\mathcal{S}| \gg h^*(s_0)$, and $h^*(s)$ is a monotonic mapping from $\mathcal{S}$ to $[0, h^*(s_0)]$, we see that for most of the state pairs (s, s') where s' is explored right after s during the search, $h^*(s) = h^*(s')$.

The reasoning above shows that most of the time a best-first search for planning explores states without reducing h^*. This phenomenon is named *plateau exploration* as it involves state exploration without changing h^*. Therefore, to improve the performance of best-first search for planning, it is important to find a way that can reduce plateau exploration.

To address this challenge, in the Roamer planner, we use random walks to assist best-first search for planning to escape from plateau more quickly. Specifically, when the best-first search makes no progress on h^* for an extended period, we use a random walk algorithm to explore the search space and help escape from the plateau.

There are three advantages of using random walks to assist best-first search for planning. First, a random walk has the potential to directly and quickly jump out of a local minima region where it is not likely to find an "exit" state that reduces h^*, whereas a best-first search will have to explore all possible states around the local minima. Second, comparing to best-first search in which heuristic functions are evaluated at each state, the random walk algorithm can skip heuristic evaluations of most of the intermediate states during exploration, making space exploration more efficient. Third, random walks require little memory, and therefore do not add space complexity to the original best-first search.

Plateau Explorations

Plateaus during search have been well studied for both SAT and CSP problems (Hampson and Kibler 1993). In SAT and CSP problems, a plateau is defined as a set of neigh-

boring variable assignments that lead to the same number of unsatisfied constraints or clauses (Frank, Cheeseman, and Stutz 1997; Russell and Norvig 2003). Plateau structures have also been studied in planning under the context of local search. A detailed analysis on why some planning problems are simple and how long the maximum exiting distance is in enforced hill-climb are presented in (Hoffmann 2002). G-value plateau in planning has also been studied in (Benton et al. 2010).

Many works have been done to accelerate plateau exploration for local search algorithms. In CSP and SAT, tabu search (Glover and Laguna 1997) can be used to avoid falling back to the same states on a plateau. WalkSAT (Kautz and Selman 1996) is a random-walk based algorithm that can find an exit to escape from a local minima.

There are several lines of work to accelerate plateau exploration in best-first search. First, space reduction techniques like preferred operations (Richter and Helmert 2009) and partial order reduction (Chen and Yao 2009; Chen, Xu, and Yao 2009) can effectively reduce the number of states explored by the search algorithm, and subsequently reduce the number of states on a plateau. However, space reduction approaches are indirect approaches to accelerate plateau exploration. These approaches cannot efficiently accelerate plateau exploration when preferred operators or partial order reductions are not effective. Second, Monte Carlo random walk (MRW) algorithms have been used to solve planning problems with good performance (Nakhost and Msller 2009). It is capable of escaping from local minima. However, it is slower comparing to deterministic best-first search when heuristic functions are informative.

Our proposed random-walk assisted best-first search (RW-BFS) for planning is inspired by both the MRW approach and best-first heuristic search approach. We use a best-first search procedure for planning to conduct state space search for most of the time, as best-first search gives good performance when the heuristic functions are informative. In addition, under certain conditions, a random-walk procedure is invoked to assist the best-first search.

Random-Walk Assisted Best-First Search Algorithm

Now we introduce our random walk assisted best-first search (RW-BFS) algorithm framework.

Our proposed RW-BFS algorithm is presented in Algorithm 1. It is a variant of a standard best-first search procedure. In addition to the original best-first search algorithm, RW-BFS adds a *detect plateau* check after expanding a new state (Line 13 in Algorithm 1). If a plateau is detected, the *random walk exploration* procedure will be called to explore the search space in order to find a state that can reduce h^*. Meanwhile, h^*, the incumbent heuristic value, is updated whenever a state with a smaller heuristic value is found (Line 6-7 in Algorithm 1).

Algorithm 2 presents the *random walk exploration* procedure. It essentially adopts the Monte-Carlo exploration algorithm proposed in (Nakhost and Msller 2009). Given a state s and h^*, it explores s's neighbors using a sequence of

Algorithm 1: The RW-BFS Algorithm

```
   input : Initial state s_0
1  open ← s_0 ;
2  while open is not empty do
3      s ← fetch(open);
4      if s is goal then
5          return solution found ;
6      if h(s) ≤ h* then
7          h* ← h(s) ;
8      if s is not a dead end then
9          closed ← s;
10         foreach s_i ∈ successor(s) do
11             evaluate h(s_i);
12             open ← (s_i, h(s_i)) ;
13     if detect plateau then
14         open ← random walk exploration (s, h*);
15 return no solution
```

Algorithm 2: Random Walk Exploration

```
   input : a state s, the incumbent heuristic value h*
1  s' ← s;
2  for j ← 1 to t do
3      decide parameters l, n ;
4      s' ← walk(s', l, n) ;
5      if s' is dead-end then
6          s' ← s;
7      else if h(s') < h* then
8          return s';
```

Algorithm 3: Walk

```
   input : a state s, the parameter l, the parameter n
1  c ← 0 ;
2  s' ← s ;
3  for c ← 1 to n do
4      for j ← 1 to l do
5          o ← a random applicable action in s' ;
6          s' ← apply(s', o) ;
7      if h(s') < h_min then
8          s_min ← s' ;
9  return s_min ;
```

(t) random walks. A state is returned if its heuristic function value is lower than h^* (Line 7-8 in Algorithm 2).

At each iteration, it initializes parameters l, n that are used in *walk*. Then, it invokes *walk* to visit a new state s' (Line 4). If s' is a dead-end, this algorithm will restart from the input state s (Line 5-6). If s' has even smaller heuristic value than h^*, this state will be returned to Algorithm 1. Otherwise, s' will be used as a new starting state for the next walk. The number of walks is bounded by t, so that this algorithm always returns in finite time, whether a better state s' is found or not.

Algorithm 3 gives a detailed view of the *walk* procedure. Given a starting state s and two parameters l and n, Algorithm 3 will try n paths, where each path is a random sequence of l actions. The procedure will return the best ending state among the n paths. Note that for any path yielded by Algorithm 3, heuristic functions are evaluated only at the end of the l actions (Line 7).

Plateau Detection. The performance of our algorithm also depends on the performance of the *detect plateau* procedure used in Algorithm 1.

This plateau detection test can neither be too sensitive nor too unresponsive. If it is too sensitive, the *random walk exploration* procedure will be invoked frequently and the progress of the best-first search may be hindered by constant interruption. On the other hand, if this detection is unresponsive, our designed random walks cannot help the best-first search as desired. Therefore, a balanced plateau detection mechanism is needed. In our proposed algorithm, the best-first search is currently on a plateau if the value of h^* is not reduced for m consecutive states. In Roamer, we set $m = 3000 + (n_p - 1) * 1000$, where n_p is the number of plateaus found so far.

Parameter settings. The Random Walk Exploration algorithm has a few parameters affecting its performance, among which n and l are the most important since they directly control the process of escaping from extensive local minima and plateaus. If n and l are too small, the local search method is greedy as it tries to immediately exploit their local knowledge instead of exploring the neighborhood of current state. Following a misleading heuristic value may quickly lead to a much worse state than what could be achieved with a little more exploration. On the other hand, if they are too large, the search may take a long time on exploring the neighborhood of the current state. In our algorithm, we set $t = 4$ and let $l = 1 + (10 - 1) * j/(t - 1), n = 200 + (1000 - 200) * j/(t - 1)$ for $j = 1 \cdots t$.

Multiple Heuristic Evaluations

Using multiple heuristic functions in search usually gives better performance than a single one. Since different heuristic functions sort states in *open* lists in different orders (Helmert 2006), it involves different search topologies. When one heuristic function becomes uninformative on its value plateau, other heuristics may give informative guidance and find exits on a plateau. However, extra heuristic function calculation and extra open lists may increase the overall time and space complexity of the search algorithm.

The default heuristic evaluation of LAMA planner adapts action costs, which estimates the minimum cost of a relaxed plan from current state to goal. In our experimental results, it performs not as good in domains which some actions have large costs while others are small as other domains. In our planner, we add a heuristic evaluation which estimates the length of the relaxed plan besides estimating the minimum cost of the relaxed plan. Respectively, we add an *open* list in our planner which sorts states by the order of the length of the relaxed plan. This synthetic heuristic evaluations achieve a good trade-off performance in our experimental results.

Experimental Results

In this section, we report experimental results of our planner. We evaluate performances for four planners, i.e., a baseline planner - LAMA, Roamer with random walk (Roamer^{rw}), Roamer with multiple heuristic evaluations (Roamer^{mh}), and Roamer which integrating these two techniques. We test all domains in IPC-6 (The Sixth International Planning Competition 2008), including Elevators (Elevator), Openstacks (Open), PARC printer (Parc), Peg solitaire (Peg), Scanalyzer-3D (Scan), Sokoban (Sokoban), Transport (Trans) and Woodworking (Wood). All experiments are ran on a PC workstation with a 2.40 GHz CPU and 2GB memory. The running time limit for each instance is set to 300 seconds.

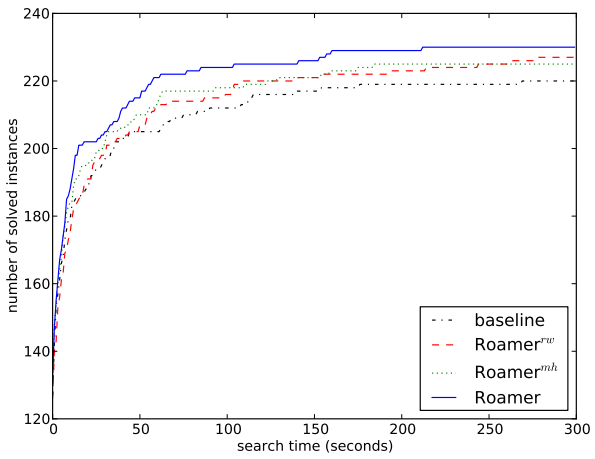

Figure 1: Number of instances (out of all the instances in the testing domains) that are solvable for a given time limit.

In Figure 1, we present the number of instances that are solvable in the testing domains with respect to a given time limit. Clearly, both Roamer^{rw} and Roamer^{mh} both solve more problem instances than the baseline planner. Roamer gives the best performance in these three algorithms.

Conclusions

In this paper, we have presented a random walk assisted best-first search algorithm, which can improve the efficacy

of heuristic search, and a multiple heuristic evaluations technique to balance the performance in different problem domains. Comparing to the baseline planner, the experimental results show that our planner, Roamer, outperforms in number of solved instances in testing domains.

Acknowledgments

This research was supported by China Scholarship Council, NSF grants IIS-0535257, DBI-0743797, IIS-0713109, and Microsoft Research New Faculty Fellowship.

References

Benton, J.; Talamadupula, K.; Eyerich, P.; Mattmüller, R.; and Kambhampati, S. 2010. G-value plateaus: A challenge for planning. In *Proc. ICAPS*, 259–262.

Chen, Y., and Yao, G. 2009. Completeness and optimality preserving reduction for planning. In *Proc. IJCAI*.

Chen, Y.; Xu, Y.; and Yao, G. 2009. Stratified planning. In *Proc. IJCAI*.

Frank, J. D.; Cheeseman, P.; and Stutz, J. 1997. When gravity fails: Local search topology. *Journal of Artificial Intelligence Research* 7:249–281.

Glover, F., and Laguna, M. 1997. *Tabu Search*. Norwell, MA, USA: Kluwer Academic Publishers.

Hampson, S., and Kibler, D. 1993. Plateaus and plateau search in boolean satisfiability problems: When to give up searching and start again. In *The 2nd DIMACS Implementation Challenge*, 437–456.

Helmert, M., and Röger, G. 2008. How good is almost perfect. In *Proc. AAAI*, 944–949.

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

Hoffmann, J. 2002. Local search topology in planning benchmarks: A theoretical analysis. In *AIPS*, 92–100.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. AAAI*.

Nakhost, H., and Msller, M. 2009. Monte-carlo exploration for deterministic planning. In *Proc. IJCAI*, 1766–1771.

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

Richter, S., and Westphal, M. 2010. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*.

Russell, S. J., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Pearson Education.

The Sixth International Planning Competition. 2008. http://ipc.informatik.uni-freiburg.de/.