The Roamer-p Planner

You Xu
Washington University in St. Louis
St. Louis, USA
youxu@wustl.edu

Qiang Lu
USTC
Anhui, China
rczx@mail.ustc.edu.cn

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

Abstract

Roamer-p is a search-based planner designed for parallel computers. In addition to a deterministic heuristic search, it uses multiple instances of *random walks* to explore the search space. The idea is inspired by the following well-observed phenomenon on best-first search: when the heuristic function is misleading or not informative, a heuristic search has to explore a large number of states without making progress in reducing the heuristic function value. This phenomenon, called "plateau exploration", has been observed and extensively studied in solving satisfiability (SAT) and constraint satisfaction problems (CSP) using search algorithms.

In circumstances where the heuristic search is trapped in a plateau exploration, instead of exploring the state space in an order designated by heuristic functions, we believe that other alternative ways of exploring search space can help accelerate search. In the Roamer-p planner, we adopt a deterministic search guided by heuristic functions to explore the search space. In addition, we use random walks as alternatives to explore the state space when the deterministic search is trapped in a plateau explorations. By using random walks in parallel with the deterministic search, the Roamer-p planner increases the chance of finding states that can help exit the plateau, and therefore decrease the overall time for solving planning problems.

Introduction

One of the most successful approaches to planning is best-first search. Best-first search employs a heuristic function that maps any state to a real number that estimates the distance to goal. A data structure called the *open* list is also used during search. Initially, the initial state s_0 of a planning problem is added to $open$. Then, at each step of the best-first search, a state s with the smallest heuristic value is fetched from $open$. If s is not a goal state, all of its successors are then inserted into the *open* list for later exploration. Otherwise, a solution path from the initial state s_0 to the just discovered goal is reported as the solution path.

During the 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. The decreasing of h^* during search can be used to estimate the progress of search, since it measures how far away the best node in the search space is from a goal state. Ideally, we want h^* to decrease quickly. In practice, a best-first search can explore a vast number of states without reducing h^*. This phenomenon, called *plateau exploration*, is not desired because it reflects the stalled progress of search.

Fundamentally, for heuristic search, plateau exploration happens because of non-informative or misleading heuristic functions. Much work has been done to improve the quality of heuristic functions (Hoffmann 2001; Helmert 2006; Richter, Helmert, and Westphal 2008). However, depending on the problem, there are still cases where heuristic functions are misleading or not informative. In these cases, instead of trusting heuristic functions, we believe that alternative ways of exploring the search space can help, especially in finding states that can reduce the incumbent heuristic function value h^*. Our planner, Roamer-p, is built on this idea. It runs a deterministic best-first search and several random walks in parallel to explore the search space. Specifically, when the best-first search is doing plateau exploration, we use random walks, starting from states on that plateau, to find states that reduces h^* (it is called an "exit" state).

There are three advantages of employing 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, compared to best-first search in which heuristic functions are evaluated at each state, in random walk, we skip heuristic evaluations of most of the intermediate states during exploration, making space exploration more efficient. Third, random walks require little memory, and can run in parallel with their deterministic counterpart, adding little time or space overhead to the overall search.

Background

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

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, in the sense that state space can be explored not just according to one arrangement of heuristics. First, multiple heuristic functions can be used to sort states in the *open* list in different orders (Helmert 2006). Since different heuristic functions have 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 calculations and extra open lists can increase the overall time and space complexity of the search algorithm.

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 planner is inspired by both the MRW approach and multiple 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.

Planner Structure

Figure 1 presents the structure of the Roamer-p planner. It first uses a PDDL-SAS$^+$ translator to convert PDDL-encoded planning problems to SAS$^+$ formalisms. Then, it adopts a two-round search strategy to find solution paths. The first-round search uses one best-first search adapted from the LAMA planner (Richter and Westphal 2010) and three random walks to find a solution to a planning problem. By the rules of IPC 2011, all planner in the multicore track runs on quad-core Linux boxes. We subsequently tailor our planner to use four threads in the first-round search. We shared data structures to communicate between threads. The second round search is essentially a weighted A* search. This design is also inspired by the LAMA planner. It uses weighted heuristic functions $f = g + wh$ where w gradually reduces to 1 from an initial 10.

Algorithms and Implementation

In this section, we explain algorithmic details of our planner. We also explain some tweaks used in the planner. We only

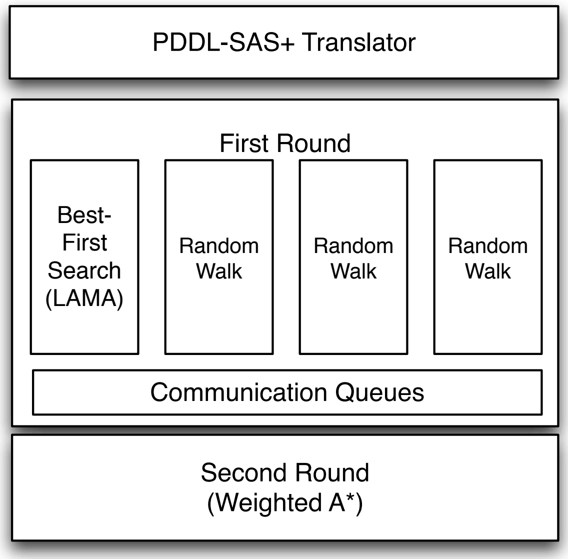

Figure 1: Planner Structure

discuss algorithms used in the first-round search.

Algorithm 1 represents the *best-first search* algorithm used in Roamer-p. It is a variant of the standard best-first search procedure. In addition to the original best-first search algorithm, we add a *plateau detected* check after exploring any state s. If a plateau is detected (see below for details), it sends s and the incumbent heuristic value $h^*(s)$ to *random walk* procedures for further exploring.

The *random walk* procedure, as shown in Algorithm 2, adopts the Monte-Carlo exploration algorithm proposed in (Nakhost and Msller 2009). It is parameterized by l, m and n. Parameter l denotes the length of a random exploration, n denotes the maximum number of random explorations from a state s'. The random walk procedure then "walks" to s that has the smallest heuristic function value after n explorations. Parameter m limits the number of total walks from s' to a new s, and therefore bound the total running time of this procedure for any starting state. This procedure sends state s back to the best-first search if $h(s)$ is smaller than the incumbent heuristic value.

Plateau Detection. To make search efficient, the plateau detection test can neither be too sensitive nor too unresponsive. If it is too sensitive, the random walk 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 search as desired. In our planner, the following two conditions are used to decide if the best-first search is currently on a plateau.

- The value of h^* is not reduced for K consecutive states; or
- The moving average of the heuristic values of W recent states is higher than λh^*.

Algorithm 1: The Modified Best-First Search in Roamer-p

```
input : Initial state $s_0$
1  $open \leftarrow s_0$ ;
2  while $open$ is not empty do
3      $s \leftarrow$ fetch($open$);
4      check goal $s$ ;
5      if $s$ is not a dead end then
6          add $s$ to $closed$ ;
7          foreach $s_i \in$ successor($s$) do
8              add $(s_i, h(s_i))$ to $open$ ;
9      if plateau detected then
10         send $(s, h^*(s)$ to random walk;
11 return no solution
```

Algorithm 2: Random walk^{l,m,n}

```
input : a state $s$, the incumbent heuristic value $h^*$
1  $s' \leftarrow s$;
2  $t \leftarrow 0$;
3  Randomly explore $n$ states $l$ steps away from $s'$ ;
4  Pick the one with the minimal heuristic value as $s$ ;
5  if $s'$ is not dead-end and $t < m$ then
6      $s' \leftarrow s$;
7      $t \leftarrow t + 1$ ;
8  else if $h(s') < h^*$ then
9      Send $s'$ to best-first search ;
```

The first condition is intuitive. We let $K = 3000$. The second one is devised to pass more states to random walks when the best-first search is in a local-minimal. We let $W = 10$ and $\lambda = 1.2$ in our planner.

Parameter Settings. We initialize different sets of parameters for random walks. This brings diversity in exploration search spaces. Since there is no general theory to estimate the l, m, n values, we set these values empirically. Table 1 lists all the parameter settings of l, m and n used in Roamer-p.

Communications between Threads. Roamer-p uses a shared memory architecture for inter-thread communications. For states sent to random walks from best-first search, a shared queue Q^1 is used. Whenever a plateau is detected, a state is pushed in to Q^1. All three random walk threads fetch states from Q^1. States in Q^1 also have priorities. The ones with lower heuristic values have higher priorities. Random walks always pick states with highest priorities in Q^1.

For a random walk starting from s, when a new state s' is found with less heuristic value than h^*, all states alone the path from s to s' are inserted into a shared queue Q^2. *Best-first search* fetches paths from Q^2, evaluate heuristic values of states in those paths (as our random walks skip heuristic evaluations of some states along the path), and insert them into its $open$ list. It is easy to prove that this preserves the completeness of the search. During the search, there are

Threads	l	m	n
1	1	5	200
2	4	6	1000
3	10	7	2000

Table 1: Parameter settings for each random walk thread

cases where random walks are making slow progress while the best-first search is advancing quickly (or vice versa). In these cases, we lose the "synergy" between threads. In these cases, we force threads to check Q^1 or Q^2. Since we use the same heuristic function (FF) across all threads, the synergy check can be done by comparing the local best heuristic value of a thread and the globally best heuristic value found by all threads.

We also use a global flag *goal-found* to coordinate the termination of threads. Initially it is set to false. Each thread will check it periodically. If it is true, the best-first search procedure will trace the solution path using the $closed$ list. Meanwhile, all random walks will stop.

References

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. 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.

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 Westphal, M. 2010. The lama planner: guiding cost-based anytime planning with landmarks. *J. Artif. Int. Res.* 39:127–177.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proc. AAAI*, 975–982.

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