Monte-carlo search for deterministic planning

Hootan Nakhost and Martin Müller
Department of Computing Science
University of Alberta
{nakhost,mmueller}@cs.ualberta.ca

Abstract

A summary of Monte-carlo random walk (MRW) planning, and tools that are needed to use this method effectively are given. Some preliminary results that show MRW can be effective to solve resource constrained problems are presented, and potential directions for future work are discussed.

Introduction

Greedy search-based approaches, such as Hill-climbing, and Best-first Search are popular in classical planning. These methods exploit all the knowledge obtained from strong but slow heuristic functions such as planning graph-based heuristics (Hoffmann and Nebel 2001) and do not try to explore the search space. This lack of exploration can cause problems in case of misleading heuristic values.

For an example, Figure 1 shows a problem in the Pipesworld domain. The task is to move all oil products P3, P4, P5, and P6 from A to B through the connecting pipe. The main action in this domain is to push a product into a pipe, which forces the product inside the pipe to get out at the other end. A valid short plan should first move the product P2, which is in the pipe, to A, and then use P2 to push all the other four products through the pipe. However, this is not a plan that a good heuristic function like Fast Forward's (Hoffmann and Nebel 2001) suggests: ignoring delete lists makes it possible to move all products to B without using P2. Figure 2 shows parts of the search space of this problem. While easy paths to a goal can be found by little exploration in the search space (e.g., the first path on top), greedy search methods, misguided by heuristic values will waste lots of time exploring irrelevant search regions (the states inside the dashed oval).

The focus in this thesis is on designing planning algorithms that are more sophisticated than common greedy algorithms to recover from areas where the heuristic evaluation is poor. Monte-carlo random walk (MRW) planning (Nakhost and Müller 2009) can be considered as a significant step in this line of research. MRW uses random samples from local search space to balance the exploration and exploitation in the search.

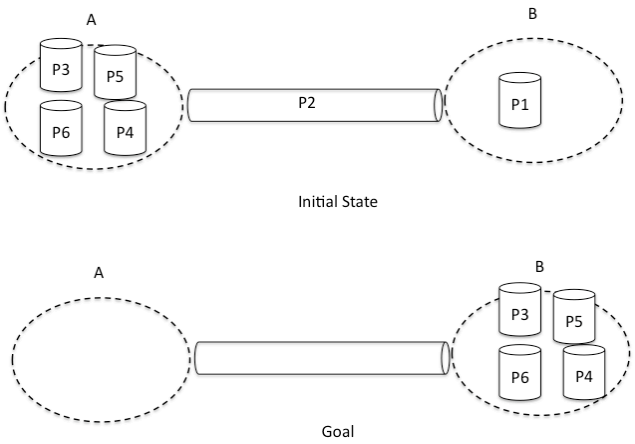

Figure 1: The initial state and goal of an example in Pipesworld domain.

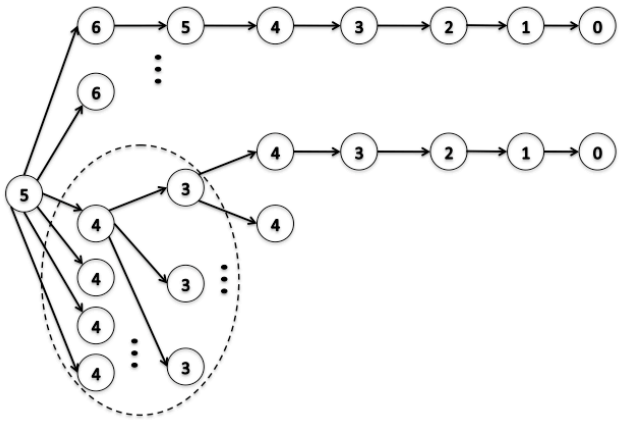

Figure 2: The search space of the problem in Figure 1. Circles signify states, arrows depict actions and numbers show the heuristic values.

More exploration in the search does come at a cost. Stochastic methods like MRW can generate unnecessary long and inefficient solutions. To address this limitation of strong but highly sub-optimal methods like MRW planning, fast post-processing tools are developed (Nakhost and Müller 2010) to improve the quality of solutions generated by these planners. These methods use the sum of costs of action in the plan as a measure of the quality. Another metric that can be used in make-span. The post-processing approaches of (Do and Kambhampati 2003; Veloso, Pérez, and Carbonell 1990) aim to reduce the make-span of a given totally ordered plan by converting it to a partially ordered plan.

The remainder of this paper is organized as follows: The next section describes the main idea of MRW planning. Then, two methods to improve the quality of sub-optimal solutions are introduced, and finally potential directions for future work are given.

Algorithm 1 Local Search Using Monte Carlo Random Walks (Nakhost and Müller 2009)

Input Initial State s_0, goal condition G and available actions A
Output A solution plan

$s \leftarrow s_0$
$h_{min} \leftarrow h(s_0)$
$counter \leftarrow 0$
while s does not satisfy G **do**
  if $counter > MAX\_STEPS$ **or** $DeadEnd(s)$ **then**
    $s \leftarrow s_0$ {restart from initial state}
    $counter \leftarrow 0$
  end if
  $s \leftarrow MonteCarloRandomWalk(s, G)$
  if $h(s) < h_{min}$ **then**
    $h_{min} \leftarrow h(s)$
    $counter \leftarrow 0$
  else
    $counter \leftarrow counter + 1$
  end if
end while
return the plan reaching the state s

Monte-carlo random walks

Algorithms 1 and 2 illustrate an outline of the MRW method. The basic idea is to use a local heuristic search, and at each search step, instead of choosing the next state from the immediate neighbors, choose the successor from a set of sampled states from the local neighborhood. Each sample is obtained by running a random walk, which is a sequence of randomly selected actions. The sampled state is the endpoint of the walk. Therefore, just the endpoints need to be evaluated. The endpoint that has the minimum heuristic value is chosen as the next state in the local search. The number and length of random walks are modified according to local characteristics of the search space (For more details, see (Nakhost and Müller 2009)). MRW search fails when the minimum obtained h-value does not improve within a given number of search steps, or when it gets stuck in a dead-end state. In such cases the search simply restarts from the initial state, s_0.

Algorithm 2 $MonteCarloRandomWalk$ (Nakhost and Müller 2009)

Input current state s, goal condition G
Output s_{min}

$h_{min} \leftarrow INF$
$s_{min} \leftarrow NULL$
for $i \leftarrow 1$ to NUM_WALK **do**
  $s' \leftarrow s$
  for $j \leftarrow 1$ to $LENGTH\_WALK$ **do**
    $A \leftarrow ApplicableActions(s')$
    if $A = \phi$ **then**
      break
    end if
    $a \leftarrow UniformlyRandomSelectFrom(A)$
    $s' \leftarrow apply(s', a)$
    if s' satisfies G **then**
      return s'
    end if
  end for
  if $h(s') < h_{min}$ **then**
    $s_{min} \leftarrow s'$
    $h_{min} \leftarrow h(s')$
  end if
end for
if $s_{min} = NULL$ **then**
  return s
else
  return s_{min}
end if

Two other variations of MRW methods are explored: Monte-Carlo Deadlock Avoidance (MDA) and Monte-Carlo with Helpful Actions (MHA). In contrast to the base algorithm that uses uniformly random action selection, these methods use statistics from earlier random walks to bias the action selection towards previously successful actions, or away from unsuccessful ones. MDA tries to avoid actions that frequently appear in failed walks, and MHA biases the action selection towards the actions that often are among helpful actions according to FF relaxed plan. Helpful actions like the heuristic function are just computed at the endpoints.

The performance of MRW planning, is competitive with state of the art systems. The main drawback of the planner is that it can generate very inefficient solutions.

Two Approaches to Plan Improvement

This section introduces the main ideas behind two methods for plan improvement: *Action Elimination* improves an existing plan by repeatedly removing sets of irrelevant actions. *Plan Neighborhood Graph Search* finds a new, shorter plan by creating a neighborhood graph NG(P) of a given plan P, and then solving a shortest path problem in NG(P).

Action Elimination

As it is explained in (Nakhost and Müller 2010), "Action Elimination iteratively improves a given plan $\pi = (a_1, \ldots, a_n)$ by computing a plan reduction in each iteration. The details are given in Algorithm 3. Starting from a_1, the algorithm tries to remove each action in turn. After removing the action, other actions that consequently lose their support - at least one of their preconditions becomes unsatisfied - are removed from the plan. If the reduced sequence remains a solution, the algorithm commits to it as a new plan, otherwise, the removed actions are restored, and the plan remains unchanged. The process terminates when the last action in the remaining plan is tried".

Algorithm 3 A greedy algorithm to remove irrelevant actions (Nakhost and Müller 2010)

Input Initial State s_0, plan $\pi = (a_1, \ldots, a_n)$, and goal condition G
Output A plan reduction

$s \leftarrow s_0$
$i \leftarrow 1$
repeat
  mark a_i {try to remove a_i}
  $s' \leftarrow s$
  for $j \leftarrow i+1$ to $length(\pi)$ **do**
    if a_j is not applicable to s' **then**
      mark a_j
    else
      $s' \leftarrow apply(s', a_j)$
    end if
  end for
  if s' satisfies G **then**
    remove marked actions from π {commit}
  else
    unmark all actions
    $s \leftarrow \Gamma(s, a_i)$
  end if
  $i \leftarrow i+1$
until $i > length(\pi)$
return π

Plan Neighborhood Graph Search

Plan Neighborhood Graph Search (PNGS) tries to find the optimal solution in a small subset of the original state space of the problem. This subset that is built by using the information in an initial plan is called the neighborhood graph. The smallest neighborhood graph of a plan just includes the states and actions that are visited and executed in the plan. A given neighborhood graph can be expanded by using a search method M. L new states are expanded by running the search method M starting from each node that is already in the graph, and then adding these states and the actions leading to them to the graph. Algorithm 4 gives pseudocode.

One good option for method M is simply the baseline uniform cost search algorithm from the optimal track of IPC-2008. This algorithm implements an A* search with the following heuristic h: $h = 0$ for goal states and h is equal to the minimum action cost in the problem otherwise.

Algorithm 4 Computation of Neighborhood Graph

Input A neighborhood graph (V, E) of a state space with $V = \{v_0, \ldots, v_n\}$, $E \subseteq V \times V$, nonnegative integer L, and search method M
Output The graph $NG(V, M, L)$

$V' \leftarrow V$
for $i \leftarrow 0$ to n **do**
  $M.initialize(v_i)$ {search neigborhood of v_i}
  for $j \leftarrow 1$ to L **do**
    $s \leftarrow M.get\_next\_state()$
    if $is\_null\_state(s)$ **then**
      return (V', E)
    end if
    $V' \leftarrow V' \cup s$
    $E \leftarrow E \cup M.edgeto(s)$
  end for
end for
return (V', E)

After building a neighborhood graph, the lowest-cost path from initial state to goal is computed.

Both Action Elimination and PNGS were implemented in a postprocessor and were empirically shown to improve the result of several planners, including the Monte-carlo planner above and the top four planners from IPC-2008, under competition conditions.

Future and Ongoing Work

Planning with Resources

Most of the state of the art planners have problems dealing with tasks with constrained resources. The main reason goes back to the heuristic functions that are used. Most powerful domain-independent heuristic functions ignore negative effects of actions, and consequently, can not capture the interaction between actions and resources. An interesting future work, proposed by Jörg Hoffmann, is to see if algorithms like MRW, which use more exploration in the search, can handle these heuristic deficiencies.

Some preliminary tests were run for two domains with highly constrained resources: Transport (Hoffmann et al. 2007) and Trucks, which is used in IPC-2006. Both domains are transportation domains in which some packages are to be delivered to target locations. In Transport, a truck with an initial amount of fuel is available to move packages. The amount of fuel used by the truck depends on the lengths of the roads traversed. No refueling action is available in this domain. In contrast to Transport, the critical resource in Trucks is not the fuel but time: there are deadlines for delivering packages.

Table 1 shows the percentage of solved tasks for each pair of tested domains and methods. All the Transport instances were generated randomly and each of them contains 8 locations and 8 packages. The initial fuel for each task was set to 1.1 times the minimum fuel that is needed to solve the task.

FF data are obtained by running best first search with the implementation of Fast Forward's heuristic that is available in Fast Downward's (Helmert 2006) code. Tests were run on a 2.5 GHz machine using 4GB memory and 30 minutes timeout.

In both domains, at least one of the variations of MRW performs better than FF. A surprising result is that MHA performs much better than MDA in Transport. This domain contains many dead-end states, and it was expected that MDA that tries to avoid dead-ends performs better. This initial result shows a great potential.

The results suggest that one direction that is worth exploring is to combine MHA and MDA. The combination might be better than each of them separately: while MDA keeps random walks away from dead-ends, MHA guides the walks towards the goal.

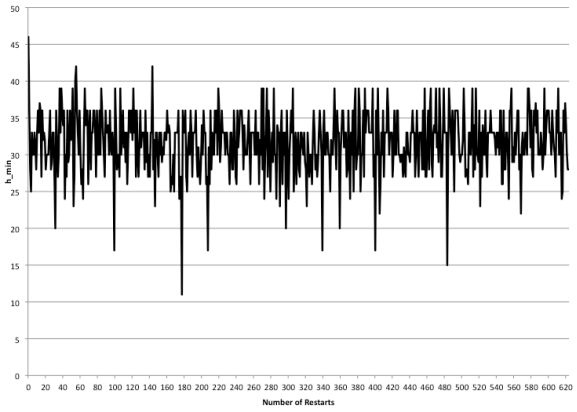

Figure 3: The minimum heuristic value obtained before each restart of MDA for Trucks-18.

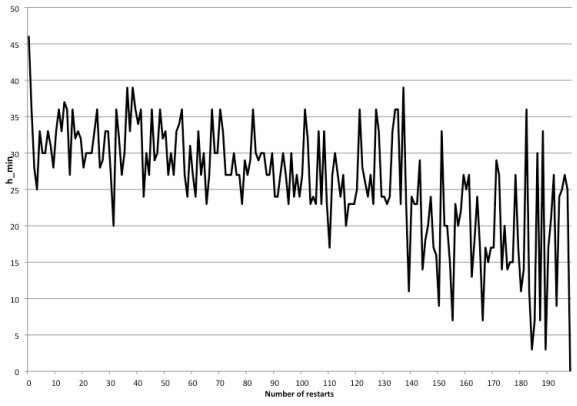

Figure 4: The minimum heuristic value obtained before each *smart restart* of MDA for Trucks-18.

Smart Restarts

Figure 3 plots the minimum heuristic value reached in each run of MDA in Trucks-18. MDA with the default restarting mechanism did not solve this instance. The interesting point is some runs of MDA are much more promising than other: before they get stuck in a local minima or hit a dead-end, they reach heuristic values that are much lower than the average values reached in each run. This might be an indication that they contain useful action sequences. In basic restarting, no such information from previous runs are used.

Domain	MDA	MHA	FF
Trucks(30)	**60**	7	40
Transport(100)	76	**100**	43

Table 1: Percentage of tasks solved for MDA, MHA and FF in two challenging domains. Total number of tasks shown in parentheses after each domain name.

Domain	Basic Restarts	Smart Restarts
Trucks(30)	60	**70**

Table 2: Percentage of tasks solved for MDA with basic and smart restarts in Trucks.

A simple method to benefit from good runs, which reached relatively small values, is to use the states visited in n best previous runs as starting points of later runs. At each restart, it is enough to select one of these states randomly and use it as the starting point of the next run. Figure 2 shows the data obtained from running this method, called *smart restarts*, to solve Trucks-18. The value n was set to 10 and *smart restarts* was activated after run 50. Before that the basic restarting is used. The results show that on average the heuristic values obtained in later runs are smaller. Trucks-18 was solved after 197 restarts, which confirms that the information kept from previous runs is helpful. Table 2 shows that *smart restarts* solves three more problems in the Trucks domain.

References

Do, M. B., and Kambhampati, S. 2003. Improving temporal flexibility of position constrained metric temporal plans. In *ICAPS*, 42–51.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Hoffmann, J.; Kautz, H.; Gomes, C.; and Selman, B. 2007. Sat encodings of state-space reachability problems in numeric domains. In *IJCAI*, 1918–1923.

Nakhost, H., and Müller, M. 2009. Monte-Carlo exploration for deterministic planning. In *IJCAI*, 1766–1771.

Nakhost, H., and Müller, M. 2010. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. Accepted for ICAPS10.

Veloso, M. M.; Pérez, M. A.; and Carbonell, J. G. 1990. Nonlinear planning with parallel resource allocation. In *DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, 207–212. Morgan Kaufmann.