Arvand: the Art of Random Walks

Hootan Nakhost
University of Alberta
Edmonton, Canada
nakhost@ualberta.ca

Martin Müller
University of Alberta
Edmonton, Canada
mmueller@ualberta.ca

Richard Valenzano
University of Alberta
Edmonton, Canada
valenzan@ualberta.ca

Fan Xie
University of Alberta
Edmonton, Canada
fxie2@ualberta.ca

Abstract

Arvand is a stochastic planner that uses Monte Carlo random walks (MRW) planning to balance exploration and exploitation in heuristic search. Herein, we focus on the latest developments of Arvand submitted to IPC'11: smart restarts, the online parameter learning system, and the integration of Arvand and the postprocessing system Aras.

Introduction

Most of the state of the art heuristic planners such as FF (Hoffmann and Nebel 2001), the top performer at IPC'00; Fast Downward (FD) (Helmert 2006), the top performer at IPC'04; and LAMA (Richter *et al.* 2008), the top performer at IPC'08, use variations of greedy search algorithms such as best first search and enforced hill climbing. These methods totally exploit the heuristic information and do not do much exploration in the search space. While this exploitive nature contributes to very fast performance in many IPC benchmarks, it can lead to serious inefficiencies where the heuristic values are misleading. For example, most of the current planners have poor performance on problems involving resource management (Nakhost *et al.* 2010).

Arvand (Nakhost and Müller 2009) is among the new wave of heuristic planners such as identidem (Coles *et al.* 2007) and the planner introduced by (Lòpez and Borrajo 2010), that try to use more explorative search algorithms to handle heuristic deficiencies. Arvand's main idea is to combine the exploration power of fast random walks with the strength of the available heuristic functions. Although still the heuristic function is the main guiding engine of the algorithm, local exploration using random walks and fast transition in the search space using long jumps are significantly helpful in regions such as plateaus where heuristic values do not help much.

The Arvand Planner

Arvand uses a forward-chaining local search algorithm. Each run of Arvand consists of one or more "search episodes". Each episode starts from the initial state, and transitions through the search space by jumping to the states found in the local neighborhood until a termination criterion is met. At each transition or search step the next state s is chosen from a set of random samples obtained by random walks. Before each transition, n bounded random walks, sequences of randomly selected actions, are run and the obtained states are evaluated. Since usually the heuristic evaluation is orders of magnitude slower than state generation, only the endpoints of random walks are evaluated using a heuristic function. The current Arvand uses the FF heuristic function. This enables the algorithm to get a large set of samples in a short period of time. After running n random walks, the planner jumps to the endpoint with minimum heuristic value. The episode terminates if either a goal state is found, or it fails to improve the minimum heuristic value reached for several jumps. After termination a new search episode starts by restarting from the initial state. For the details of the algorithms that control the number and length of random walks see (Nakhost and Müller 2009). Arvand is implemented based on Fast Downward (FD) code base.

Smart Restarts

In the original version of Arvand, the search always restarts from the initial state (basic restarts). However, it is shown that in more constrained problems a more evolved restarting mechanism called smart restarts (SR) is helpful (Nakhost *et al.* 2010). In SR a pool of most promising search episodes are kept in the memory and each time that the algorithm restarts, a state visited by one of the episodes in the pool is selected as the next restarting point. Search episodes are compared based on the minimum heuristic value they have reached. Each time an episode fails, it can replace the worst episode in the pool if the minimum heuristic value reached by the new episode is lower. To select the restarting point, first an episode e is selected from the pool randomly and then a state visited by e is randomly selected. Therefore, the states that are repeated in several episodes have a higher chance to be chosen. If SR is used from the beginning, then the information inside the pool get too biased to the first episode. In Arvand, SR gets activated after N restarts. Before that basic restarting is used. Therefore when SR is activated the pool contains N episodes. The parameter N and the size of the pool are both set to 50 for the competition.

Biasing the Random Walks

Two different methods are used to bias the random action selection inside Arvand: Monte Carlo Helpful Actions (MHA) and Monte Carlo Deadlock Avoidance (MDA) (Nakhost and Müller 2009). The main idea is to use the statistics from the earlier random walks to bias the action selection towards more promising actions and away from non-promising ones. MHA uses Gibbs sampling to give priority to actions that have been more often selected as a helpful action at an endpoint. Helpful actions are computed as a by product of the heuristic function at the endpoints. MDA keeps track of the number of times that each action has appeared in a failed random walk (a walk that reaches a state with no applicable actions) and tries to sample actions with higher failure rate less often.

Parameter Learning

(Nakhost and Müller 2009) showed that different configurations of Arvand perform well in different types of domains. In the current version of Arvand an online learning algorithm is used to find the best configuration of the parameters for the given problem. The problem of selecting the best configuration from a set of possible configurations C can be viewed as a multi-armed bandit problem, where pulling an arm corresponds to using the corresponding configuration for the next search episode of Arvand. Each time that Arvand restarts, a bandit algorithm is used to select one of the configurations and then based on the minimum heuristic value (h_{min}) obtained in the search episode a reward is assigned to the corresponding arm. Let h_i be the heuristic value of the initial state, then the reward r is computed as follows: $r = max(0, 1 - (h_i/h_{min}))$. Arvand uses the upper confidence bounds (UCB) algorithm (Auer *et al.* 2002) to balance the exploration and the exploitation in the configuration selection.

Currently, C includes three configurations: two MHA versions with initial length of random walks 1 and 10; and one MDA with initial length 1. Since in some problems running a search episode might be quite slow, and in the initial phase of UCB all the configurations are tried once, the best configuration might not be selected enough times to be able to solve the task. To remedy this problem, for the initial episodes a smaller number of random walks n per search step is used to speed up the learning process. Specifically, for the first three episodes n is set to 100 and for the next episodes n is doubled up until it reaches the maximum 2000.

Integration with Aras

Since IPC'08 there has been an emphas on the quality of the generated plans. This has been perfectly reflected in the competition's scoring function: the cost of the best known plan divided by the cost of the plan. Aras (Nakhost and Müller 2010) is a fast postprocessing tool that is able to improve the quality of a given plan generated by any planner. Aras uses simple fast techniques to locally search the neighborhood of the given plan and works well for a wide range of planners including LAMA, which is designed to generate high-quality plans. The most effective technique in Aras, called Plan Neighborhood Graph Search (PNGS), builds a graph encapsulating the search space close to the original plan and then finds the lowest-cost path inside the graph. The anytime version of Aras starts with a small initial size for the graph and then iteratively increases the size of the graph until the memory limit is reached.

An alternation of Aras and Arvand is used to obtain high-quality plans. First Arvand is run until a solution is found. The solution is saved and fed into anytime Aras to be improved. After Aras reaches the memory limit, which is set to 2 GB, a new search episode of Arvand is used to find another plan and again it is fed into Aras. This process continues until the time limit is reached. In the whole process, as soon as a plan with better quality is found it is saved.

Arvand also uses a bounding mechanism to stop episodes or random walks that already exceed the cost of a previously found solution. However, the solution bound is only updated by plans generated by Arvand itself. The reason is that usually the bounds obtained from Aras' solution are too tight for Arvand and significantly lower the probability of reaching any solution. As the result the number and diversity of the plans that are fed into Aras gets much lower and this has a detrimental effect in the best quality plan reached by the system.

Acknowledgements

We would like to thank Malte Helmert for giving us access to the Fast Downward code. We would also like to acknowledge the support of NSERC and Alberta Ingenuity.

References

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47:235–256, May 2002.

Andrew Coles, Maria Fox, and Amanda Smith. A new local-search algorithm for forward-chaining planning. In *Proc. ICAPS'07*, pages 89–96, 2007.

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

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

Carlos Linares Lòpez and Daniel Borrajo. Adding diversity to classical heuristic planning. In *Proc. SOCS'10*, pages 73–80, Atlanta, USA, 2010.

Hootan Nakhost and Martin Müller. Monte-Carlo exploration for deterministic planning. In *Proc. IJCAI'09*, pages 1766–1771, 2009.

H. Nakhost and M. Müller. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In *Proc. ICAPS'10*, pages 137–144, 2010.

H. Nakhost, J. Hoffmann, and M. Müller. Improving local search for resource-constrained planning. Technical Report TR 10-02, Dept. of Computing Science, University of Alberta, Edmonton, Alberta, Canada, 2010.

Silvia Richter, Malte Helmert, and Matthias Westphal. Landmarks revisited. In *Proc. AAAI'08*, pages 975–982, 2008.