ArvandHerd: Parallel Planning with a Portfolio

Richard Valenzano, Hootan Nakhost, Martin Müller, Jonathan Schaeffer
University of Alberta
{valenzan, nakhost, mmueller, jonathan}@cs.ualberta.ca

Nathan Sturtevant
University of Denver
sturtevant@cs.du.edu

Abstract

ArvandHerd is a satisficing parallel planner that has been entered in the 2011 International Planning Competition (IPC 2011). It uses a portfolio-based approach where the portfolio contains four configurations of the Arvand planner and one configuration of the LAMA planner. Each processor runs a single planner, and the execution is mostly independent from the other processors so as to minimize overhead due to communication. ArvandHerd also uses the Aras plan-improvement system to improve plan quality.

Introduction

If a planner is to be successful, it must be able to handle problems from a diverse set of domains. Unfortunately, no single algorithm can be expected to dominate all other challengers on all possible domains. Even within a single domain, it has been shown that to achieve the best possible performance, it is often necessary to use different *parameterizations* or *configurations* of an algorithm on different problems (Valenzano et al. 2010). These issues suggest the use of an *algorithm porfolio*. This means that instead of using a single strategy, problems should be tackled with a set of strategies that differ by their configuration or in the underlying algorithm.

For parallel planning, different members of the portfolio can be assigned to separate processors. This is a simple alternative to the difficult process of parallelizing a single-core algorithm and it mostly avoids overhead from communication and synchronization. These ideas form the backbone of our ArvandHerd planner.

In this paper, we begin with a description of the individual members of the ArvandHerd portfolio. This is followed by a description of the general architecture of ArvandHerd, including communication between processors, memory management, and the use of the Aras plan-improvement system.

The ArvandHerd Portfolio

The portfolio was selected so as to maximize the coverage of ArvandHerd by including different configurations of two significantly different planning approaches. More specifically, the portfolio contains four configurations of the random walk based Arvand planner (Nakhost and Müller 2009) and one configuration of the WA*-based LAMA planner (Richter and Westphal 2010). Below, these planners and their configurations are described in more detail.

The Arvand Planner

Arvand is a sequential satisficing planner that uses heuristically evaluated random walks as the basis for its search. The execution of Arvand consists of a series of *search episodes*. In the simplest version of Arvand, each search episode begins with n random walks, each being a sequence of m legal random actions originating from the initial state s_i, where n and m are parameters. The heuristic value of the final state reached by each random walk is also computed using some heuristic function. Once all n walks have been performed, the search *jumps* to the end of the walk whose final state, s, has the lowest heuristic value. This means that Arvand now runs a new set of n random walks of length m, only this time the walks originate from state s. This is followed by another jump to the end of the most promising walk from this new set of walks. This process repeats until either a goal state is encountered, or some number of jumps are made without any improvement in the heuristic values being seen. In the latter case, the current search episode is terminated and a new episode begins with random walks originating from s_i.

Arvand has been shown to be able to solve many difficult problems that traditional planners have been unable to solve (Nakhost and Müller 2009). This increase in coverage generally comes at the expense of solution quality, though the quality can be improved significantly by using the any-time and plan-improvement strategies described later in this paper. Arvand also requires very little memory which makes it ideal for running simultaneously in a shared-memory environment with other memory-intensive planners.

Configurations Four different Arvand configurations have been included in the ArvandHerd portfolio. Below, the parameters that differ between configurations in the portfolio are described in more detail. We omit any description of most of the other system parameters. For a more comprehensive discussion of all the parameters in Arvand, see (Nakhost and Müller 2009), (Nakhost, Hoffmann, and Müller 2010), (Nakhost et al. 2011).

Config	Bias Type	Initial Walk Length	Extending Rate
1	MDA	1	2.0
2	MDA	3	1.5
3	MHA	1	1.5
4	MHA	10	1.5

Table 1: `Arvand` configurations used in `ArvandHerd`.

The first important difference between configurations relates to the biasing of the random action selection. `Arvand` allows for random walks to either be unbiased, biased to avoid actions that have previously led to dead-ends (referred to as *MDA*), or biased to using *helpful actions* identified by the heuristic function (referred to as *MHA*). These different biasing strategies have been shown to be useful for different domains (Nakhost and Müller 2009). The bias used for each configuration in the portfolio is shown in Table 1.

The portfolio configurations also differ in parameters related to the random walk length. In `Arvand`, this length is adjusted online if little progress is being made in the heuristic values seen during a set of random walks. Such stagnation may occur if the current state is in a *heuristic plateau*. In an attempt to escape these plateaus, the walk length is increased over time. The initial walk length, the frequency with which walks are lengthened, and the factor by which they are lengthened (called the *extending rate*) are all parameters affecting this process. The initial walk length and the extending rate for each configuration in the portfolio is shown in Table 1. Note, the frequency with which walks were lengthened did not vary between configurations.

Heuristic Function All `Arvand` configurations use the FF heuristic (Hoffmann and Nebel 2001). For this heuristic, a possibly suboptimal plan starting at the current state is found to a relaxed version of the problem. This relaxation corresponds to the removal of delete effects from operators.

Techniques used for solving the relaxed problem vary. The implementation used for `Arvand` is from the `Fast Downward` planning system (Helmert 2006). In this implementation, the heuristic value ignores operator costs and is given by the number of operators in the relaxed plan. This heuristic will be referred to as the FF_{FD} heuristic.

Smart Restarts If `Arvand` makes a number of jumps without seeing any progress in the heuristic values encountered, the current search episode is terminated. However, instead of always restarting from scratch, as is done in the simplest version of `Arvand`, the planner can build upon progress made by previous search episodes through the use of a *walk pool* (Nakhost, Hoffmann, and Müller 2010). For some a (called the *pool size*), the walk pool holds the a "best" trajectories seen in all search episodes performed thus far. A trajectory t_1 is preferred over a trajectory t_2 if the state with the lowest heuristic value in t_1 is lower than the state with the lowest heuristic value in t_2. Qualifying trajectories are added to the walk pool at the termination of the corresponding search episode. For each new search episode, a trajectory t is randomly selected from the walk pool. Instead of starting from s_i, the new search episode then begins from a state that has been randomly selected from t.

Note, for the first b search episodes — where b is a parameter called the *pool activation level* — the search begins from the initial state. It is only after the first b episodes are completed that partial trajectories from the walk pool are used to find new starting positions for search. This prevents the walk pool from becoming completely biased towards trajectories that are all similar to the very first trajectory.

Configuration Selection as a Bandit Problem `Arvand` has been enhanced with a system that, given a set of configurations C, selects a configuration for the next search episode from C based on the performance of the configurations during previous search episodes. This system views configuration selection as a multi-armed bandit problem in which C is the set of bandits and the search episodes correspond to arm pulls. This paradigm requires the definition of a payoff function for search episodes. For this system, the reward given to a search episode e performed with configuration c is given as follows: where s is the state on the trajectory of e that achieved the lowest heuristic value, the reward given to c is $\max(0, 1 - h(s)/h(s_i))$, where $h(r)$ is the heuristic value of state r.

Using this reformulation of the problem, configurations can be selected online using any of the multi-armed bandit algorithms. In `Arvand` , the UCB algorithm (Auer, Cesa-Bianchi, and Fischer 2002) is used.

Any-time Planning with `Arvand` The solutions found by `Arvand` are generally suboptimal and so this planner does not terminate once a solution is found. Instead, the solution is added to the walk pool and a new search episode is started. The cost of the best solution found thus far is used as a bound on all future trajectories. This planner can then be run indefinitely or until some resource limit is reached.

The `LAMA` Planner

`LAMA` is a WA*-based planner that won the sequential satisficing track of IPC 2008 (Helmert, Do, and Refanidis 2008). It uses both multiple heuristic functions and helpful action open lists. Given a set of k heuristics $H = \{h_1, ..., h_k\}$, `LAMA` will have two sets of k open lists, denoted $O = \{o_1, ..., o_k\}$ and $O^p = \{o_1^p, ..., o_k^p\}$. `LAMA` must also be given a second set of heuristics, denoted $H^p = \{h'_1, ..., h'_j\}$, for the generation of helpful actions. Note, we will let $pref_{h'_i}(s)$ denote the set of children corresponding to the helpful actions found with heuristic h'_i for state s.

When it is time to expand a state, one of the open lists from either O or O^p is selected in a process described below. This open list will return the state s it identifies as the best state it contains. If s is a goal state, the solution is extracted from the closed list and returned. If s is not a goal state, the children of s, denoted C, are then generated, as is the set of *preferred children* of s, given by $C' = pref_{h'_1}(s) \cup ... \cup pref_{h'_j}(s)$. The states in C are then added to each of the lists in O for which states in any $o_i \in O$ are sorted by the cost function $f_i(s') = g(s') + w * h_i(r)$, where r is the parent of s' and w is the weight used for the current WA* search. For example, the cost given to $c \in C$ in open list o_i is given by

$g(c) + w * h_i(s)$. This technique is called *delayed heuristic evaluation* and has been shown to be effective in planning. The states in C' are then added to each of the lists in O^p. For every $o_i^p \in O^p$, states in o_i^p are ordered using the same cost function as o_i. However, notice that o_i and o_i^p do not contain the same states as o_i contains all generated but not expanded states, while o_i^p only contains preferred children.

When selecting which open list to remove a state from, the strategy in use is to alternate between all lists in $O \cup O^p$. The alternation is supplemented with a preferred children open list bonus. Whenever a state is seen such that for at least one of the heuristics it is the state with the lowest heuristic value seen so far, the open lists in O^p are all given a bonus of j state expansions. This means that the alternation will be restricted to only the open lists in O^p until each has expanded j nodes (or more if additional bonuses are accrued during this phase), at which point alternation will continue among all lists in $O \cup O^p$.

Heuristics Two heuristics were used in the version of the LAMA planner entered in IPC 2008: the landmark-count heuristic and a variation of the FF heuristic. These will be denoted as LM and FF^+, respectively. Both heuristics were also used for helpful action generation.

While the LM heuristic was one of the major advances introduced in the LAMA planner, the heuristic was used as is in ArvandHerd and so interested readers are referred to the journal paper on LAMA (Richter and Westphal 2010). However, instead of using FF^+, ArvandHerd uses two related heuristics. To explain why, we briefly describe FF^+.

Just as in FF_{FD}, LAMA's version of the FF heuristic computes a plan for the relaxed problem. This plan yields two obvious heuristics. The first, denoted by FF_{size}, is given by the number of actions in the relaxed plan just as is done in FF_{FD}. This heuristic is intended to capture the expected depth of the solution from the current state. The second, denoted by FF_{cost}, is given by the sum of the cost of the actions in the relaxed plan and is designed to capture the expected cost of the solution from the current state. FF^+ is given by the sum of FF_{size} and FF_{cost} as a way to balance between the two heuristics. Note, as Fast Downward and LAMA compute the relaxed plan differently, the values of FF_{FD} and FF_{size} are often different, as are the set of generated helpful actions.

In our experiments, we found that coverage was increased if, instead of using FF^+, we used both FF_{cost} and FF_{FD} as a way to balance between these metrics. This means that three heuristics are used in the version of LAMA used in ArvandHerd: LM, FF_{cost}, and FF_{FD}. However, only FF_{FD} and FF_{cost} were used to generate helpful actions.

Any-time Planning Once a solution is found with LAMA, the search is restarted from the initial state with a lower weight value. The previous best solution found is then used to prune all future searches. Changing the weight introduces *diversity* into the search which helps the planner avoid making the same early mistakes it has made previously. In the version of LAMA used in ArvandHerd, the first iteration runs greedy best-first search which means open lists are ordered by heuristic values alone. This is then followed by iterations with weights 10, 5, and 2, followed by 4 iterations with a weight of 1, and a final iteration with a weight of 0. A similar strategy has been shown to significantly outperform other forms of any-time planning (Richter, Thayer, and Ruml 2010). The WA* iterations have been further diversified effectively by randomizing the order in which generated children of the same parent are added to any one open list. This causes ties between children of the same state to be broken differently in different iterations.

The caching of heuristic values, helpful actions, and the best path found for each state in the closed list has also been shown to increase the speed of LAMA since many heuristic values will not need to be re-computed during future iterations (Richter, Thayer, and Ruml 2010). This feature was not part of LAMA as submitted to IPC 2008, but has been added to LAMA as used in ArvandHerd.

The ArvandHerd Architecture

For the sequential satisficing multi-core track of IPC 2011, 4 processors are allotted for each planner. As both Arvand and LAMA are built on top of Fast Downward, ArvandHerd is run from a single binary. When problem-solving begins, this binary spawns threads for different members of the portfolio. However, before this can begin, the planner first requires a *translation* from PDDL to a SAS+-like formalism, and a *knowledge compilation* step that builds data structures necessary for the LM heuristic. We have not parallelized these components and simply use this portion of the original LAMA code as is. For more information on this process, see the work on LAMA (Richter and Westphal 2010) or the work on Fast Downward (Helmert 2006) on which this process is based.

Once the translation and knowledge compilation stages are complete, one of the processors is assigned to run LAMA while the other three are each given one of the four Arvand configurations to run. Most of the communication between the processors is limited to those running Arvand. Specifically, the three processors share a walk pool and a single UCB configuration selection system. When a processor has completed a search episode, it submits the corresponding trajectory to the shared walk pool, and gets a new trajectory in return, or the empty trajectory if the activation level has not yet been reached. The processor then submits the reward for its current configuration to the UCB system and in return is given a configuration to use in its next search episode. This sharing of the UCB system among the processors running Arvand allows them to more quickly identify strong configurations than they would be able to with independent UCB systems. The walk pool, for which both the activation level and size are set to 100, is shared for similar reasons. LAMA is also given the ability to add walks to the solution pool, though in the submitted planner it only adds solution trajectories.

So as to maintain the correctness of the walk pool and the configuration learner, each system uses a lock that limits access to one processor at a time. As the search episodes dominate the Arvand execution time and LAMA is not expected to find solutions very often, there is little synchronization or contention overhead caused by sharing these resources.

The final shared value is the cost of the best solution found by any planning method thus far. This value is used to prune LAMA's WA* search.

Plan Improvement with Aras

While Arvand usually performs well in terms of coverage, it often finds low quality solutions. To address this issue, the Aras plan improvement system was created (Nakhost and Müller 2010). Aras involves two phases: *action elimination (AE)* and *plan neighbourhood graph search (PNGS)*. AE involves a scan of the current solution and the removal of unneccesary actions. For PNGS, a *plan neighbourhood graph* is built around the current solution using a breadth-first search. The plan neighbourhood graph is then searched for a shorter path between the start and any goal states.

The execution of Aras alternates between iterations of AE and PNGS until some time or memory limit is hit. However, instead of rebuilding the neighbourhood graph on each new PNGS iteration, the previous bread-first search is simply continued so as to grow the neighbourhood graph.

In ArvandHerd, whenever a solution is found by any processor, an instance of Aras is created and run on the current solution. If the initial solution was found by Arvand, Aras is given a 60 second time-limit. If the initial solution was found by LAMA, Aras is given a 40 second time-limit. This limit is lower for LAMA since that planner already has a fairly effective plan improvement scheme.

Recall that LAMA uses the cost of the best solution found by any method for pruning. Such pruning is ineffective for Arvand which instead only uses the best cost of a solution found strictly with Arvand as a bound. This is because bounds given by LAMA or Aras solutions are often too tight for Arvand in which case Arvand is unable to find any new solutions. As such, it was generally found to be more effective to create a diverse set of plans with Arvand and improve them with Aras, than to force Arvand to create low cost plans directly by using the global bound.

Memory Management

As the memory requirements of Arvand are limited to space for the current trajectory, the best random walk seen thus far, the walk pool, and the UCB configuration selection, Arvand is expected to almost never hit the 6 GB memory limit given to planners for IPC 2011. This is not the case for Aras and LAMA. As such, these processes need to be prevented from exhausting all the memory given to the planner, thereby crashing the whole system, and preventing further search by the processors running Arvand . To address this problem, the PNGS phase of each Aras instance is limited to using only 500 MB, and the total memory of the open and closed lists in LAMA is set as 2.7 GB. If the Aras limit is hit, Aras quits and returns the best solution found thus far. If the LAMA limit is hit, the current search iteration is ended and the open lists are emptied. The next iteration of LAMA then begins with the possibility that the diversity introduced by changing the weight and tie-breaking may avoid the mistakes made on the previous iterations. If the final 0-weight iteration also runs out of memory, the processor running LAMA will run another copy of Arvand instead.

Conclusion

We have described the main features of the ArvandHerd parallel planner which uses a portfolio containing the Arvand and LAMA planners. Due to the use of the porfolio, ArvandHerd is expected to have strong coverage, while the use of Aras and LAMA's any-time strategies should lead to good solution quality.

Acknowledgments

We would like to thank Sylvia Richter for allowing us to use the LAMA planner in the ArvandHerd portfolio, and 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

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning* 47(2-3):235–256.

Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds. 2010. *Proceedings of the 29th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*. AAAI.

Helmert, M.; Do, M.; and Refanidis, I. 2008. IPC 2008 Deterministic Track.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research (JAIR)* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research (JAIR)* 14:253–302.

Nakhost, H., and Müller, M. 2009. Monte-Carlo Exploration for Deterministic Planning. In Boutilier, C., ed., *IJCAI*, 1766–1771.

Nakhost, H., and Müller, M. 2010. Action Elimination and Plan Neighborhood Graph Search: Two Algorithms for Plan Improvement. In Brafman et al. (2010), 121–128.

Nakhost, H.; Müller, M.; Valenzano, R.; and Xie, F. 2011. Arvand: The Art of Random Walks. *IPC 2011 Deterministic Track Planner Reports*.

Nakhost, H.; Hoffmann, J.; and Müller, M. 2010. Improving Local Search for Resource-Constrained Planning. Technical Report TR 10-02, Dept. of Computing Science, University of Alberta, Edmonton, Alberta, Canada.

Richter, S., and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research (JAIR)* 39:127–177.

Richter, S.; Thayer, J. T.; and Ruml, W. 2010. The Joy of Forgetting: Faster Anytime Search via Restarting. In Brafman et al. (2010), 137–144.

Valenzano, R. A.; Sturtevant, N. R.; Schaeffer, J.; Buro, K.; and Kishimoto, A. 2010. Simultaneously Searching with Multiple Settings: An Alternative to Parameter Tuning for Suboptimal Single-Agent Search Algorithms. In Brafman et al. (2010), 177–184.