ArvandHerd 2014

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

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

Abstract

ArvandHerd is a sequential satisficing planner that uses a portfolio consisting of LAMA and Arvand. This planner won the multi-core track of the 2011 International Planning Competition. In this paper, we describe the various components of ArvandHerd, the updates made for the 2014 competition, and the modifications that allow ArvandHerd to compete in the single-core sequential satisficing tracks.

1 Introduction

In the 2011 International Planning Competition, the winner of the multi-core track was a planner called ArvandHerd. This planner uses a portfolio-based approach to combine the strengths of the complementary approaches of random-walk and best-first search based planning. This is accomplished by simultaneously using the LAMA (Richter and Westphal 2010) and Arvand (Nakhost and Müller 2009) planners.

An updated version of this planner has been submitted to the 2014 competition. ArvandHerd 2014 is very similar to the planner which competed in 2011 and uses the same code base. However, it has been updated in several ways. These updates include the addition of techniques to LAMA including ϵ-greedy node selection, aggressive restarting, and diverse any-time search. We have also modified the planner so that it could compete in the single-core sequential satisficing tracks. In this paper, we will briefly consider the various components of ArvandHerd, describe the newly added techniques, and look at how the standard multi-core version of this planner has been modified so that it can compete in the single-core tracks.

2 The Components of ArvandHerd

ArvandHerd is a portfolio-based planner that uses a variety of planning techniques. In this section, we will briefly describe those components that have remained mostly the same from the 2011 version of this planner.

2.1 The ArvandHerd Code Base

The version of ArvandHerd submitted to the 2011 competition ran multiple threads from a single C++ binary. This is because ArvandHerd used both Arvand and LAMA, and Arvand had been built on top of LAMA 2008. Though there have been updates to each of these planners in LAMA 2011 (Richter, Westphal, and Helmert 2011) and Arvand 2013 (Nakhost and Müller 2013), the version of ArvandHerd submitted to the 2014 competition is still based on the LAMA 2008 code base.

The only component of this code base which was updated is the PDDL to SAS+ (Bäckström and Nebel 1995) translator, and the knowledge compilation step needed for the landmark count heuristic (Richter, Westphal, and Helmert 2011) used in LAMA. In the 2011 competition, ArvandHerd used the translator and knowledge compilation code in LAMA 2008. These pieces crashed on some of the problems in the 2011 competition, and so we have used the translator and knowledge compilation code from the version of Fast Downward used in IPC 2011. For details on how this translation is performed see (Helmert 2009). Details on the knowledge compilation step can be found in (Helmert 2006).

2.2 LAMA

LAMA is the winner of both the 2008 and 2011 IPC competitions, and is therefore a natural candidate for use as the greedy best-first search planner included in the portfolio. This planner uses a number of different techniques including multiple heuristics, preferred operators, deferred heuristic evaluation, and Restarting Weighted A* (RWA*). For a more complete description of this planner, see (Richter and Westphal 2010).

For the 2011 competition, a few additions were made to LAMA for its use in ArvandHerd. In particular, the planner was set to use random operator ordering, to cache the heuristic values of states in between iterations of the RWA* search, and the planner was modified so that a single call for the computation of the FF heuristic could be used to return both the action-cost aware or action-cost unaware versions of this heuristic. ArvandHerd also added a memory usage estimator to LAMA. This system estimates how much memory LAMA is using, and it allows the search to be restarted whenever a given memory limit is reached so that another parameterization of LAMA can be tried. These additions were also used in the 2014 version of ArvandHerd. For more information regarding how LAMA is used in ArvandHerd see (Valenzano et al. 2012) and (Valenzano et al. 2011).

* Now at Google.

2.3 Arvand

Arvand is a random-walk based planner that has been shown to be effective in certain domains that are difficult for best-first search based planners (Nakhost and Müller 2009). The execution of Arvand consists of a series of *search episodes*. Each episode starts by performing a set of random walks from the initial state and using the heuristic function to evaluate the endpoint of each of these random walks. Once a state with a low heuristic value is found, or after a certain number of such walks, the search jumps to the end of the walk on which the best heuristic value was found. The search episode then continues with a set of random walks from this state. This process then repeats until either a goal is found, or enough jumps are made without heuristic progress, in which case the search starts with a new search episode from the initial state. For a more thorough description of Arvand see (Nakhost 2013), (Nakhost and Müller 2009), (Nakhost and Müller 2013), and (Nakhost, Hoffmann, and Müller 2012).

Arvand has a number of parameters that allow the user to control the length of the random walks, the frequency with which the algorithm jumps during a search episode, and the frequency with which a search episode is terminated and the algorithm restarts. Since different parameterizations of the algorithm are best for different problems, Arvand has been designed so that a single instance of this planner can use different configurations in different search episodes. In the version of Arvand used in the 2011 competition, a *configuration selection* system was used to determine the parameterization to use on the next search episode. This system, which is based on the idea of a multi-armed bandit algorithm and is also used in ArvandHerd 2014, biases Arvand to more frequently use those parameterizations which have previously made the most heuristic progress.

Arvand also uses a second technique for sharing information across search episodes. This feature, which is called a *walk pool* (Nakhost, Hoffmann, and Müller 2012), stores those search trajectories which made the most search progress. When starting a new search episode, these trajectories are used to suggest an alternative starting point for the episode that is deeper into the state-space than the initial state. For a more in-depth description of how Arvand uses these features in ArvandHerd see (Valenzano et al. 2012).

2.4 Plan Improvement

In the sequential satisficing and sequential multi-core tracks, planner evaluation is based on the quality of solutions found. As such, when competing in these tracks it is critical to use the time remaining after a first solution is found to find better solutions. ArvandHerd uses multiple techniques for improving solution quality, and in this section we describe those which remain mostly the same from the version of this planner that was submitted to IPC 2011.

Aras. ArvandHerd uses a plan post-processing system called Aras (Nakhost and Müller 2010). The execution of this system consists of two phases. The first is a linear scan of a given solution path that looks for actions that can be removed such that the remaining plan is still valid. The second phase involves the construction of a neighbourhood graph around the solution path using a combination of forward search and a backwards, regression-based search. This graph is built until the number of nodes it contains reaches a given node limit. A search is then performed which finds the shortest path in this neighbourhood graph from the initial state to a goal state.

Aras runs by iterating between these two phases until some time or memory limit is reached, such that the limit on the nodes in the neighbourhood graph is increased each time the neighbourhood graph phase begins. All solutions found by ArvandHerd using either LAMA or Arvand are fed to Aras in an effort to improve solution quality.

Restarting Weighted A* (RWA*). RWA* was a feature introduced in the original version of LAMA that was later analyzed in (Richter, Thayer, and Ruml 2010). When using this technique, LAMA restarts and begins a less greedy search from scratch each time a solution is found. For example, on the first iteration of LAMA, the planner uses a greedy best-first search for finding the first solution. On the second iteration, LAMA then runs WA* with a weight of 10. If a second solution is found, WA* is run again but with an even smaller weight. This process then repeats until the time limit is reached.

In the standard version of RWA*, no information is shared between the iterations of RWA* except for the best solution found thus far, and the heuristic values of nodes that have already been expanded. This means that the search will not consider any nodes whose g-cost is as large as the best solution found thus far. This was the approach taken by ArvandHerd in 2011, though we used a different technique in ArvandHerd 2014 as described in Section 3.3.

Any-Time Arvand. Arvand was also set to continuously look for solutions even after a first solution is found. This simply means that Arvand continues to perform search episodes. As in LAMA, the cost of previously found solutions were also used to bound the search. This means that episodes with a g-cost that is larger than the bound are forced to restart. However, unlike how LAMA was used in ArvandHerd 2011, the bound used for the search episodes is only based on the best solution found by Arvand. As a result, the solutions found by LAMA or Aras are not factored into how search episodes are bound. This type of bounding was employed because Arvand is often unable to find any new solutions if the bound is too tight. By restricting the bound to only consider solutions found by Arvand means that the bound is looser than it would be if the other solutions were also factored in. This often allows Arvand to produce more plans, thereby increasing the chance that a plan will be found that will be greatly improved by Aras.

While this would suggest that perhaps no bounding should be used, experimentation with this system did indicate that some bounding was useful in certain domains in which Aras was ineffective at improving the solution quality. Using the best solution found by Aras was experimentally found to be an effective compromise between finding enough solutions for Aras while still adding useful bounding for domains in which Aras was not as successful. Note

that similar behaviour has been seen when using LAMA (Xie, Valenzano, and Müller 2013), and so we consider bounding in LAMA in Section 3.3.

3 Additions to ArvandHerd 2014

In this section, we describe the main changes that have been made to ArvandHerd for its submission to IPC 2014. Note that several of these techniques require parameters to be set, and we will describe the parameter values used in each track in Section 4.

3.1 ϵ-Greedy Node Selection

ϵ-greedy node selection is a simple technique that effectively introduces random exploration into the search (Valenzano et al. 2014). This technique, which requires the user to set a parameter ϵ in the range from 0 to 1, works as follows. With probability $1 - \epsilon$, the search acts exactly as the search algorithm ordinarily would. For example, if the search being used is GBFS, then with probability $1 - \epsilon$ the search will select the node from the open list with the smallest heuristic value as the next node to be expanded. However, with probability ϵ, the search is forced to use a different policy. Specifically, the search with select a node uniformly at random from amongst those in the open list. ϵ therefore determines how often the algorithm exploits heuristic information, and how often it explores.

Despite its simplicity, ϵ-greedy node selection has been shown to improve the coverage of planners like LAMA, even though this planner is already using multiple techniques for introducing variation into its search (Valenzano et al. 2014). However, we use ϵ-greedy node selection slightly differently in LAMA than as explained above. This is because LAMA uses $2k$ open lists where there are k heuristics in use, with k of the open lists holding all open nodes (each ordered by a different heuristic), and k open lists holding only those nodes achieved with a preferred operator (again, with each ordered by a different heuristic). For each node expansion, LAMA must first select one of the $2k$ open lists, and then using the corresponding heuristic to select a node from that open list. In our implementation of ϵ-greedy node selection for LAMA, we have left the open list selection mechanism the same, but have modified each open list to return a randomly selected node from that open list with probability ϵ. For example, if LAMA selects one of the preferred operator open lists as the next to be used and $\epsilon = 0.3$, then there is a 70% chance that the next node to be expanded will correspond to the node achieved using a preferred operator which has the lowest heuristic value and a 30% chance that the node will be randomly selected from the set of all nodes achieved using a preferred operator. This approach was taken due to the known effectiveness of the LAMA open list selection policy.

3.2 Aggressive Restarting

While the version of LAMA used in ArvandHerd in the 2011 competition would restart and use a different parameterization whenever the memory estimator indicated that a given memory limit was reached, an investigation that was performed after the competition suggested that this was not an effective restarting policy (Valenzano et al. 2012). In particular, if the search does not use up the memory quickly enough, it may spend all its time using an ineffective planner parameterization or an unlucky operator ordering. Moreover, if it does quickly use up the memory, the fact that it caches the heuristic values may mean that the other parameterizations do not have much memory to work with.

To remedy these problems, the restarting policy was modified in the 2014 version of ArvandHerd in two ways. In the first, we made the policy perform restarts much more often. This policy requires the user to set two parameters: an initial node expansion limit L_i and a limit factor L_f. The execution of LAMA in ArvandHerd 2014 begins with a node expansion limit of L_i. When this limit (or a memory limit) is hit, LAMA will restart and use a different configuration. Once the limit is reached with all of the configurations, the expansion limit is increased by a factor of L_f. This process then repeats until the time limit is reached.

So as to avoid the problem by which the additional parameterizations do not have enough memory for their search, we have set LAMA to clear its heuristic value cache if it reaches the memory limit too many times.

3.3 Diverse Any-Time Search

As mentioned above, if Arvand is set to bound its search episodes using the best solution found thus far including those from Aras, the bound often makes it too difficult to find any further plans. In that case, it was experimentally found to be better to use a looser bound so that Arvand finds more solutions and thus there is a greater chance that Aras will greatly improve at least one of them.

In (Xie, Valenzano, and Müller 2013) it was shown that similar behaviour was found when using Aras along with RWA* in LAMA. To remedy this situation, a new technique was developed called *Diverse Any-time Search (DAS)*. When using this approach, the planner runs RWA* as it typically does, but once a given time limit is hit, it starts a new RWA* search that begins again with the greediest of the configurations. This new RWA* search also ignores the cost of all previous solutions found. For example, if the time limit is five minutes, then a new RWA* search will begin again with GBFS every five minutes, and the bound used at any time is given by the best solution found during the current five minute RWA* phase. While Aras is also used on all solutions found, the cost of the solutions found by Aras are never used for bounding so as not to make it too difficult for LAMA to find new plans.

DAS was added to the RWA* search of LAMA in the version of ArvandHerd submitted to IPC 2014. The main difference with how it is described in (Xie, Valenzano, and Müller 2013) is how the RWA* time limit is set. In ArvandHerd 2014, we simply use the restarting policy described in the previous section to determine when a new DAS phase should begin. This means that a new DAS phase will begin once each configuration being used finds a solution or hits the current node expansion limit. The bound used during a DAS phase is given by the best solution found of all configurations tried with the same node expansion limit. Once all configurations have been tried with a particular

node expansion limit, the limit is increased according to the node limit factor, and a new RWA* search is started with a higher limit but no bound. Note that when a solution is found for a first time, we delay the increase of the expansion limit for one more RWA* phase.

4 Multi-Core and Single-Core ArvandHerd

While the version of ArvandHerd submitted in 2011 was solely a multi-core planner, the current version has been modified so that it can also compete in the sequential satisficing and the sequential agile tracks. In this section, we describe the differences between the versions used in these tracks including the parameterizations used.

4.1 Multi-Core ArvandHerd

The multi-core version of ArvandHerd runs almost identically to the way it did in 2011. From the single binary, ArvandHerd runs four threads. One of these threads runs LAMA while the other three run a parellelized version of Arvand. This parallel Arvand essentially has each thread run an independent search episode, although the threads share a single walk pool and a single configuration selector. For more information on this architecture, see the description of ArvandHerd given in (Valenzano et al. 2012). The only difference between the 2011 and 2014 versions of this system is that the Arvand threads no longer share the best solution they have found thus far with the thread running LAMA in the 2014 system. This is because, as described in Section 3.3, LAMA does not use the best solution found for bounding the search.

Multi-Core Parameters. In the multi-core version of ArvandHerd, there are four configurations made available for the configuration selector of Arvand. Two of these configurations bias the random walks to avoid using actions that have previously lead to dead-end states, while the other two bias the random walks to use helpful actions (Hoffmann and Nebel 2001) suggested by the heuristic. All configurations use a version of the FF heuristic (Hoffmann and Nebel 2001) which is not aware of action costs, but the configurations differ slightly in the initial length of the random walks, and how quickly the random walk length is increased. Multi-core ArvandHerd also uses a walk pool which holds a maximum of 100 search trajectories.

LAMA's RWA* has been set to run GBFS, then WA* with weights of 5 and 1. During the GBFS search, it uses a version of the FF heuristic that ignores action costs, while it uses a version which is aware of action costs when performing WA*. Both types of search also use the landmark count heuristic (Richter and Westphal 2010) and ϵ-greedy node selection with $\epsilon = 0.3$. Regarding the restart policy, the initial expansion limit is set at 100 while the node limit factor is set at 10.

4.2 Single-Core ArvandHerd

ArvandHerd did not compete in any single-core tracks in 2011, though it has been submitted to the sequential satisficing and sequential agile tracks in the 2014 competition. The single-core version of this planner does not use multiple threads. Instead, it runs Arvand first until a time limit is hit, and then it switches to LAMA. This is similar to the approach taken by Fast Downward Stone Soup (Helmert and Röger 2011), which also runs different planners in sequence.

Sequential Satisficing Parameters. In the sequential satisficing track, ArvandHerd runs Arvand for the first 15 minutes of the available runtime. The remaining time is then used by LAMA. The rest of the parameters are set just as they were for the multi-core track, except for the size of the walk pool which has been decreased to hold only a maximum of 50 trajectories.

Agile Satisficing Parameters. In the agile satisficing track, ArvandHerd runs Arvand for the first 3 minutes and LAMA for the remaining time. The parameters used are the same as in the sequential satisficing track, except that the walk pool size is decreased to hold a maximum of 20 trajectories, ϵ-greedy node selection is used with $\epsilon = 0.2$, and there is no weight 1 WA* configuration included in the set of LAMA configurations. Since solution quality is not counted in measuring performance in this track, Aras is not used, and ArvandHerd terminates once a first solution is found.

5 Conclusion

In this paper we have described the ArvandHerd planner submitted to the 2014 International Planning Competition. In particular, we have described the various components of this planner, the new techniques added since the 2011 competition, and how the planner has been made sequential for use in the single-core sequential satisficing tracks.

Acknowledgments

We would like to thank Fan Xie for the helpful discussions regarding techniques for improving plan quality in ArvandHerd 2014. This research was supported by GRAND and the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

Bäckström, C., and Nebel, B. 1995. Complexity Results for SAS+ Planning. *Computational Intelligence* 11:625–656.

García-Olaya, A.; Jiménez, S.; and López, C. L. 2011. The 2011 International Planning Competition. Technical report, Universidad Carlos III de Madrid, Madrid, Spain. http://hdl.handle.net/10016/11710.

Helmert, M., and Röger, G. 2011. Fast Downward Stone Soup: A Baseline for Building Planner Portfolios. In *The Proceedings of the 2011 ICAPS Workshop on Planning and Learning*, 28–35.

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

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173(5-6):503–535.

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

Nakhost, H., and Müller, M. 2009. Monte-Carlo Exploration for Deterministic Planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 1766–1771.

Nakhost, H., and Müller, M. 2010. Action Elimination and Plan Neighborhood Graph Search: Two Algorithms for Plan Improvement. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling*, 121–128.

Nakhost, H., and Müller, M. 2013. Towards a Second Generation Random Walk Planner: An Experimental Exploration. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*.

Nakhost, H.; Hoffmann, J.; and Müller, M. 2012. Resource-Constrained Planning: A Monte Carlo Random Walk Approach. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling*.

Nakhost, H. 2013. *Random Walk Planning: Theory, Practice, and Application*. Ph.D. Dissertation, University of Alberta. http://hdl.handle.net/10402/era.31939.

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

Richter, S.; Thayer, J. T.; and Ruml, W. 2010. The Joy of Forgetting: Faster Anytime Search via Restarting. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling*, 137–144.

Richter, S.; Westphal, M.; and Helmert, M. 2011. LAMA 2008 and 2011. In *The 2011 International Planning Competition* (2011) 50–54. http://hdl.handle.net/10016/11710.

Valenzano, R. A.; Nakhost, H.; Müller, M.; Schaeffer, J.; and Sturtevant, N. R. 2011. ArvandHerd: Parallel Planning with a Portfolio. In *The 2011 International Planning Competition* (2011) 113–116. http://hdl.handle.net/10016/11710.

Valenzano, R. A.; Nakhost, H.; Müller, M.; Schaeffer, J.; and Sturtevant, N. R. 2012. ArvandHerd: Parallel Planning with a Portfolio. In *20th European Conference on Artificial Intelligence*, 786–791.

Valenzano, R. A.; Sturtevant, N. R.; Schaeffer, J.; and Xie, F. 2014. A Comparison of Knowledge-Based GBFS Enhancements and Knowledge-Free Exploration. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*.

Xie, F.; Valenzano, R.; and Müller, M. 2013. Better Time Constrained Search via Randomization and Postprocessing. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling*, 269–277.