# Planning via Random Walk-Driven Local Search

**Fan Xie** and **Hootan Nakhost** and **Martin Müller**
Computing Science, University of Alberta
Edmonton, Canada
{fxie2,nakhost,mmueller}@ualberta.ca

**Abstract**

The ideas of local search and random walks have been used successfully in several recent satisficing planners. Random Walk-Driven Local Search (RW-LS) is a strong new addition to this family of planning algorithms. The method uses a greedy best-first search driven by a combination of random walks and direct node evaluation. In this way, RW-LS balances between exploration and exploitation. The algorithm has been implemented in the system Arvand-LS. Its performance is evaluated against other state of the art planners on problems from IPC-2011, as well as on scaled up instances from several IPC domains. The results show significant improvements in both coverage and plan quality.

## Introduction

Most successful current satisficing planners combine several complementary search algorithms. Examples range from portfolio planners such as Fast Downward Stone Soup (Helmert, Röger, and Karpas 2011) and loosely coupled parallel planners such as ArvandHerd (Valenzano et al. 2011) to systems which alternate several search strategies, such as FF (Hoffmann and Nebel 2001), FD (Helmert 2006), and ArvandHerd, and dual queue search algorithms as in LAMA (Richter and Westphal 2010). Other recent examples are Probe (Lipovetzky and Geffner 2011), which combines very sophisticated explorative probes with standard search techniques, Arvand (Nakhost and Müller 2009), a planner which alternates exploration by Random Walks with a greedy hill-climbing strategy, and Roamer (Lu et al. 2011), which combines a best-first strategy with the use of random walks to escape from plateaus and local minima. Another successful recent planning technique has been post-processing for plan improvement, as in the Aras system (Nakhost and Müller 2010).

While the most recent planning competition IPC-2011 had clear winners - LAMA 2011 (Richter, Westphal, and Helmert 2011) for the sequential satisficing track and ArvandHerd (Valenzano et al. 2011) for the parallel satisficing track - no single planner currently dominates all others for all planning domains. This is one reason that portfolio approaches (Helmert, Röger, and Karpas 2011; Valenzano et al. 2011) do well.

The main contribution of this paper is *Random Walk-Driven Local Search* (RW-LS), a new planning method which combines local search with random walks. In RW-LS, a greedy best-first search is driven by two kinds of evaluations: direct evaluation of tree nodes, and evaluation of random walk endpoints as in Arvand.

Another contribution is *IPC-2011-LARGE*, a set of scaled up test instances for several of these domains. Such scaled up instances are useful since some of the current IPC benchmarks have become too easy for the best planners.

This paper is organized as follows: after a brief review of the framework of Monte Carlo Random Walk (MRW) Planning, which is one basis for the current work, the new planning algorithm RW-LS is introduced and compared with related work. The RW-LS planner Arvand-LS is described next, followed by a section about the generation and selection of harder problems from existing IPC domains for which scalable problem generators are available. The experimental results for Arvand-LS show strong improvements over the state of the art in both coverage and plan quality for hard problems from several IPC domains. The paper concludes with a discussion of possible future work, including perspectives for a portfolio system containing Arvand-LS.

## Monte Carlo Random Walk (MRW) Planning

A STRIPS *planning task* is a tuple $(P, I, G, A)$ containing a set of propositions $P$, *initial state* $I \subseteq P$, *goal* $G \subseteq P$, and a set of *actions* $A$. A *state* is identified with the set of propositions $s \subseteq P$ that are true in the state. A *plan* is a sequence of actions leading to a state $s \supseteq G$.

In Monte Carlo Random Walk planning (Nakhost and Müller 2009), quick random walks are performed to explore the neighborhood of the current search state $s_0$. A random walk starting from $s_0$ is a sequence of states $s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_n$, in which each transition action $s_k \rightarrow s_{k+1}$ is randomly selected from among the legal actions in $s_k$. Several statistical techniques are used for biasing the random selection. Only the end state $s_n$ of the random walk is evaluated by a heuristic function $h$ such as the FF heuristic $h^{FF}$ (Hoffmann and Nebel 2001). After a number of random walks from the current start state $s_0$, the search greedily updates $s_0$ to an end state of minimum $h$-value. This process of performing a set of random walks, then jumping to a best endpoint is called a *search step*. For efficiency, MRW only evaluates end states of random walks. Computing a heuristic such as $h^{FF}$ is usually orders of magnitude more expensive than generating and executing a random action. Therefore, MRW can afford to explore a much larger neighborhood of the current search state than the more systematic searches in planners such as LAMA and FF. MRW iterates until either a goal state is reached, or it restarts from the initial state $I$. Restarts are scheduled when the minimum $h$-value doesn't change for a given number of search steps, or when all random walks from a state reach a dead end.

## Monte Carlo Random Walks Local Greedy Best-First Search

The MRW algorithm emphasizes fast exploration by random walks, while leaving plan improvement to a postprocessor such as Aras. In contrast, RW-LS focuses on plan quality during search by performing a local Greedy Best First Search in each step. Figure 1 compares the search strategies of MRW and RW-LS. Both algorithms use random walks to explore the search space near a starting point $s_0$. After each exploration phase, both algorithms update $s_0$ to an explored state with minimum h-value and start the next search step from this new $s_0$. Unlike MRW, RW-LS performs a local Greedy Best-First Search starting from state $s_0$ during exploration. The search uses well-known enhancements such as delayed evaluation and a second open list containing only states reached via preferred operators (Helmert 2006). RW-LS evaluates the best state $s$ retrieved from an open list, and also performs a random walk starting from $s$ ending in a state $r$. A linear combination of the heuristic values $h(s)$ and $h(r)$ is used to order the nodes in the open list.

Algorithm 1 shows an outline of the MRW framework (Nakhost and Müller 2009), adapted to RW-LS. The only difference in the top-level algorithm is the call to *IteratedRW-LS* instead of doing pure exploration by random walks. A successful search returns the sequence $s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_n$ of states along a plan, with $s_0 = I$ the initial state, $s_n \supseteq G$ a goal state and the partial paths $s_k \rightarrow s_{k+1}$ obtained by search steps starting from $s_k$. The main search loop fails if *IteratedRW-LS* reports no progress, or if the current search state becomes a dead end. In these cases, the algorithm simply restarts from the initial state $I$.

*IteratedRW-LS*, shown in Algorithm 2, attempts to find an improved state $s'$ by calling RW-LS a limited number of times. Each call constitutes one search step. If the minimum h-value does not improve, *IteratedRW-LS* returns no progress.

Unlike MRW, RW-LS performs a random walk-guided greedy best-first search as its search step. This method is shown in Algorithm 3. In the algorithm, $RandomWalk(s, G)$ performs one random walk from a state $s$. As in (Nakhost and Müller 2009), each random walk either returns the end state $r$ reached after a given maximum number of random actions, or terminates early on encountering a goal or dead end.

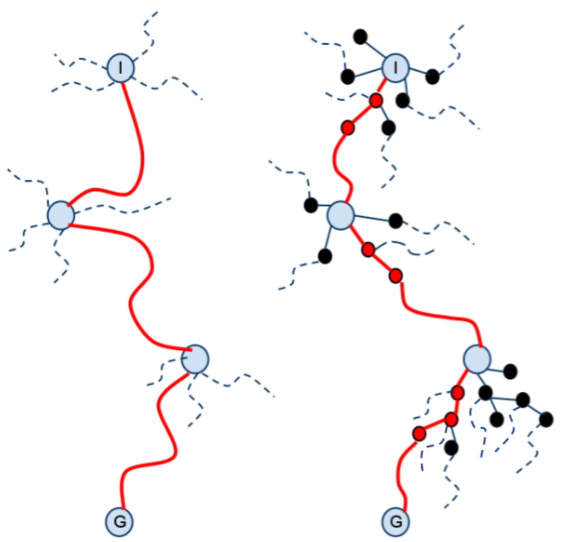


Figure 1: The search strategies of MRW (left) and RW-LS (right).

**Algorithm 1** The MRW-like main loop of RW-LS

**Input** Initial State $I$ and goal condition $G$
**Output** A solution plan

```
$(s, h_{min}) \leftarrow (I, h(I))$
while $G \not\subseteq s$ do
    $(s, status) \leftarrow$ IteratedRW-LS$(s, G)$
    if $status$ = NO_PROGRESS or DeadEnd$(s)$ then
        $(s, h_{min}) \leftarrow (I, h(I))$ {restart from initial state}
    else
        $h_{min} \leftarrow h(s)$
    end if
end while
return the plan from $I$ to $s$
```

**Algorithm 2** Iterated RW-LS

**Input** State $s$ and goal condition $G$
**Parameter** MAXSTEPS
**Output** New $s$ and progress status

```
$open \leftarrow [s]$
$closed \leftarrow []$
for $i = 1$ to MAXSTEPS do
    $s' \leftarrow$ RW-LS$(s, G, open, closed)$
    if $s' \neq$ DEAD_END and $h(s') < h(s)$ then
        return $(s'$, PROGRESS)
    end if
end for
return $(s$, NO_PROGRESS)
```

### Random Walk-Driven Local Search

Algorithm 3 shows the random walk-guided greedy best-first search algorithm. Open and closed lists are shared between iterations. RW-LS performs a random walk from each

**Algorithm 3** *RW-LS*, random walk-driven local search

```
Input state $s$, goal condition $G$, open, closed
Parameter NUMWALKS
Output Best state $s_{min}$ or DEAD_END

  $(s_{min}, h_{min}) \leftarrow (s, h(s))$
  for $i = 1$ to NUMWALKS do
    if $open.empty()$ then
      return DEAD_END
    end if
    $s \leftarrow open.remove\_min()$ {best open node in tree}
    if $G \subseteq s$ then
      return $s$
    end if
    $r \leftarrow RandomWalk(s, G)$
    if $G \subseteq r$ then
      return $r$
    end if
    if $h(s) < min(h_{min}, h(r))$ then
      $(s_{min}, h_{min}) \leftarrow (s, h(s))$
    else if $h(r) < h_{min}$ then
      $(s_{min}, h_{min}) \leftarrow (r, h(r))$
    end if
    closed.insert($s$)
    open.insert($s$.children() - closed, $w \times h(s) + h(r)$)
  end for
  return $s_{min}$
```

state retrieved from the open list, and evaluates the endpoint $r$. The algorithm keeps track of, and returns the best state seen overall, either from *closed* or from a random walk endpoint. States in $open$ are ordered by a linear combination $w \times h(s) + h(r)$. The parameter $w$ controls the trade-off between exploration and exploitation. The variable *open* in the algorithm manages both the normal and the preferred operator open lists. The method also keeps track of the path (action sequence) leading from $s$ to the best state $s_{min}$. In case $s_{min}$ was a random walk endpoint, this path consists of an in-tree prefix followed by the random walk. For simplicity, the details of path handling are omitted in the pseudocode.

If RW-LS fails to find an improvement, it returns the input state. Unlike Arvand, there is no forced transition to an end point in this case, and *IteratedRW-LS* repeatedly calls RW-LS from the same start state, while keeping the open and closed lists and increasing the parameter *NUMWALKS*.

### Comparison to Other Related Algorithms

The RW-LS algorithm has some similarities to Enforced Hill Climbing (Hoffmann and Nebel 2001). Both algorithms perform a local search before transitioning to the next search state. The major differences are:

- EHC uses a uniform, breadth first exploration strategy. RW-LS combines a selective GBFS with random walks to explore a larger neighborhood.
- EHC does not use heuristic values to guide the search, until it finds a state with improved h-value. RW-LS uses a linear combination of $h(s)$ and $h(r)$ to guide the greedy best-first search.

Random-Walk Assisted Best-First Search (RWA-BFS) (Lu et al. 2011) is more closely related to RW-LS, as it combines best-first search with random walks. However, random walks in RWA-BFS are added to a global, complete best-first search. In contrast, RW-LS is a stochastic local search algorithm. As a complete global search algorithm, RWA-BFS outperforms RW-LS in domains which seem to require some exhaustive search, such as Sokoban and Parking. RW-LS shines in solving large problems for which its combination of local search and exploration are effective. Furthermore, while RW-LS runs random walks from every expanded node, RWA-BFS uses them only when the search is considered to be stuck in a local minimum.

## Arvand-LS: A Simple Planner based on RW-LS

Arvand-LS is an implementation of the RW-LS algorithm based on the IPC-2011 version of Arvand (Nakhost et al. 2011), called Arvand-2011 here. As outlined in Algorithm 1, Arvand-LS replaces the pure exploration by random walks of Arvand-2011 by RW-LS. For most other parts of the planner, the Arvand-2011 default settings are used, such as $MAXSTEPS = 7$ in Algorithm 1. Arvand-LS first runs each search step with $NUMWALKS = 100$ random walks. For each restart from $I$, this value is doubled, up to a maximum of 3200. There are two reasons for this doubling strategy: as long as no plan to a goal state is found, larger local searches help increase the probability to find a solution; after a solution is found, larger searches increase the fraction of in-tree actions, as opposed to random walk actions, in the solution, which helps improve plan quality. The weight $w$ in the combined heuristic function of RW-LS is set to a large value, $w = 100$, after some initial experiments on IPC-2011. $h(r)$ is often used only for tie-breaking, but it seems to be very successful in that because it makes the search more informed in large plateaus. In our testing, the algorithm was robust against changes to MAXSTEP in the interval [1..14]. Performance declined slowly for larger values of this parameter.

Many of the details of how random walks of Arvand-LS are performed are identical to Arvand-2011 (Nakhost et al. 2011). Parameters are expressed as a tuple ($len\_walk$, $e\_rate$, $e\_period$, $WalkType$) (Nakhost and Müller 2009). Random walk length scaling is controlled by an initial walk length of $len\_walk$, an extension rate of $e\_rate$ and an extension period of $NUMWALKS * e\_period$. The choices for $WalkType$ are MHA and MDA, which use online statistics of $helpful\ actions$ and $deadend\ states$ respectively to bias the action selection in random walks. For example, a configuration of $(1, 2, 0.1, MHA)$ means that all random walks use the $MHA$ enhancement, and if $h_{min}$ does not improve for $NUMWALKS * 0.1$ random walks, then the length of walks, $len\_walk$, which starts at 1, will be doubled. The three configurations used are:

- *config-1*: $(10, 2, 0.1, MHA)$

- *config-2*: $(1, 2, 0.1, MDA)$
- *config-3*: $(1, 2, 0.1, MHA)$

The algorithm cycles through these configurations, starting with *config-1* and changing at each restart.

Integration with the Aras postprocessor is also identical to Arvand-2011: Aras is run on each new plan found until it reaches the memory limit. Compared to the original postprocess-once method proposed in (Nakhost and Müller 2010), this applies Aras to a larger variety of input plans.

One major difference between Arvand-2011 and Arvand-LS is less pruning: Arvand-LS turns off the pruning in the search or random walks for states whose cost so far exceeds the cost of the best found solution. This pruning does not work well for Arvand-LS since its high quality initial plans often prevent the algorithm from finding any other alternate plans. In conjunction with the Aras postprocessor, allowing greater plan diversity is more important for achieving good final results than stronger pruning.

## IPC-2011-LARGE: Scaling up IPC-2011 Domains

In several domains, the test instances used at IPC-2011 have become too easy for the current top planners. Harder instances are needed to show that a new approach advances the state of the art in coverage as well as in plan quality. The collection *IPC-2011-LARGE*, available at `http://code.google.com/p/ipc2011-large-domain`, represents a new, more challenging test set.

Four out of fourteen IPC-2011 domains have been scaled up to harder problems in *IPC-2011-LARGE* to help illustrate the performance of Arvand-LS: *Woodworking*, *Openstacks*, *Elevators* and *Visit-All*. Regarding the other IPC-2011 domains, *Transport*, *Tidybot*, *Nomystery* and *Floortile* seem hard enough for current planners, so scaling can wait. In *Parking*, *Sokoban* and *Barman*, neither Arvand nor Arvand-LS scale very well - see the discussion in the IPC-2011 results section below. Depending on the needs of the community, *IPC-2011-LARGE* could be expanded in the future to include harder instances from those domains. For the remaining domains of *Parcprinter*, *Pegsol* and *Scanalyzer*, no scalable generators were available.

In *Woodworking*, wood must be processed using different tools to produce parts. Planners should minimize the processing cost. *Openstacks* is a combinatorial optimization problem where products must be temporarily stacked. Planners need to minimize the maximum number of stacks in simultaneous use. In *Elevators*, passengers need to be transported using two types of elevators with different cost characteristics. In *Visit-All*, an agent located in the center of a $n \times n$ grid must visit all cells of the grid while minimizing the number of moves.

### IPC-2011-LARGE Parameter Settings

Parameters used in most *IPC-2011-LARGE* instances are larger than the hardest problems in IPC-2011. To describe a range of parameters, the notation $p \in [L..U; I]$ is used to indicate that parameter $p$ varies from a lower limit of $L$ to an upper limit of $U$ in increments of $I$. For example, $p \in [10..30; 5]$ means that the generated instances have values of $p \in \{10, 15, 20, 25, 30\}$.

**Elevators** The hardest IPC-2011 problem has $f = 40$ floors and $p = 60$ passengers. The new problems also fix $f = 40$, and vary $p \in [20..305; 15]$.

**Openstacks** The largest number of products in IPC-2011 is $p = 250$. New problems use $p \in [250..630; 20]$.

**Visit-All** The hardest IPC-2011 problem has grid size $g = 50$. New problems use $g \in [48..86; 2]$.

**Woodworking** The largest number of parts in IPC-2011 is $p = 23$. For new problems, $p \in [25..105; 4]$.

The new test set is designed to be a clear step up in difficulty from IPC-2011. Only planners that were able to solve most of the IPC-2011 instances for a given domain are likely to solve some of the new *IPC-2011-LARGE* instances.

## Experiments

Experiments include tests on all IPC-2011 and IPC-2011-LARGE benchmarks. They were run on an 8 core 2.8 GHz machine with a time limit of 30 minutes and memory limit of 2 GB per problem. Results for planners which use randomization are averaged over 5 runs.

### Planners and Evaluation

The experiments compare Arvand-LS with the best known version of seven other planners: three top performers from IPC-2011, LAMA-2011, FDSS-2 and Probe; the random walk-based planner Arvand; the well-known local search planner LPG-td (Gerevini, Saetti, and Serina 2003); FF and FD-Autotune-2 (FD-AT-2), which both use Enforced Hill Climbing as the first try to solve the problem – while FF is not tuned to generate high-quality solutions, FD-AT-2, after finding the first solution runs WA* to improve the quality; and Roamer, which is included for comparison with Random-Walk Assisted Best-First Search. LAMA-2011-FF is LAMA-2011 in FF-only mode, with the Landmark Heuristic disabled. Since Arvand and Arvand-LS use FF only, this version of LAMA-2011 helps to make a better comparison by isolating the influence of of the LM heuristic.

The IPC-2011 versions of Roamer and Arvand were affected by a bug in the PDDL-to-SAS+ translator used. Arvand also had a memory management problem. These issues have been fixed in the versions of the planners used in the current experiments.

The evaluation method is the same as in IPC-2011: the score for an unsolved problem is 0. For each solved problem, the planner receives a score between 0 and 1, scaled relative to the best plan.

### Results on IPC-2011 Benchmarks

The coverage and quality results on IPC-2011 are shown in Tables 1 and 2. In addition, Figure 2 plots the number of problems solved for the first 5 minutes and for the full 30 minutes. Arvand-LS solves more problems quickly than the other tested planners. The overall performance of Arvand-LS shows a significant improvement compared to the baseline planner Arvand and exceeds that of several other top planners.

The coverage of both Arvand and Arvand-LS is weak in the domains of *Sokoban*, *Parking* and *Barman*. The main reason for the bad performance of random walks in these domains is that specific action sequences need to be discovered in order to make progress. The probability of repeatedly discovering such sequences through random walks is very small. In contrast, best-first search planners work much better in these domains, since they store the states that they have already tried, and can discover such paths by a complete search. The local search in Arvand-LS improves upon Arvand's pure exploration: The average *Sokoban* coverage improves from 2.2 to 8.4, *Parking* improves from 3.8 to 8.6 and *Barman* from 0 to 9.6.

Another interesting data point shown is LAMA-2011-FF, the version of LAMA without the LM heuristic. It solves 39 fewer problems than LAMA-2011. *Visit-All* shows the biggest drop, from 20 to 4. This domain is also very hard for many other planners that don't use the LM heuristic, such as FD-AT-2 and FDSS-2. Interestingly, the random walks planners Arvand and Arvand-LS, which use only the FF heuristic, solve all problems in *Visit-All*. These question remain as future work: why do random walks perform well in *Visit-All*? And can adding the LM heuristic improve Arvand and Arvand-LS?

While local search improves the coverage of random walk planners most of the time, it fails for *Nomystery* (Nakhost, Hoffmann, and Müller 2012). This is a truck-package transportation domain similar to *Transport* in IPC-2011. Its key feature is that trucks only have a limited amount of fuel. In such resource constrained planning domains, delete-relaxation heuristics are not informative, since they ignore resource consumption. Random walk-based methods work very well here (Nakhost, Hoffmann, and Müller 2012). In Arvand-LS, the local greedy best first search guided by the FF heuristic seems to suffer from similar problems as other GBFS planners, following paths of great heuristic improvement but running out of fuel in the process. While Arvand-LS is weaker than Arvand here because of its greedy best-first search, thanks to random walks, it still achieves the third best score among all tested planners.

Compared to Arvand, Arvand-LS shows a significant improvement on plan quality in *Elevator*, *Openstacks*, *Scanalyzer* and *Visit-All*. In these domains, the local search finds much shorter plans. In *Elevator*, the plan quality of Arvand-LS is even better than that achieved with systematic search planners.

Arvand-LS ties with at least one other planner in solving all the tasks in 7 out of 14 domains. To be able to make meaningful comparisons on these domains, harder instances were created for domains where scalable generators are available.

## Results on *IPC-2011-LARGE* Problems

Table 3 shows the coverage of all tested planners on the four scaled domains. The key observations are:

- Arvand-LS significantly outperforms the other tested planners in all domains except *Large-Openstacks*, where Arvand has a slight advantage. Arvand-LS solves twice as many problems as LAMA-2011, the winner of IPC-2011. Many other planners have problems scaling up to these large instances.
- Both RW-LS and MRW scale better than EHC implemented both in FF and FD-AT-2.
- Relying on random walks to escape from the plateaus in a global best-first search framework is not enough to scale to these large problems: both Arvand and Arvand-LS solve between 5 to 20 problems more than Roamer in each of the domains.
- In three out of four domains, the local search in Arvand-LS increases coverage compared to Arvand.

Figure 3 shows IPC-style scores for each domain. To keep the figures readable, only the results of the four planners with highest score are shown in each domain [1]. The results show:

- Arvand-LS always generates better quality plans compared to Arvand. In a few cases such as problems 17-20 in *Large-Openstacks*, Arvand-LS gets lower scores since it is not able to consistently solve the problem in all 5 runs.
- Except for *Large-Visit-All*, Arvand-LS creates plans of better (*Large-Elevators*) or competitive quality compared to more systematic planners such as LAMA-2011.

Focusing on the *Large-Elevators* results in Figure 3(a), the difference in plan quality of Arvand-LS and Arvand decreases with problem size. There are two reasons for this: For problems 10 to 20, solutions are found only by the *config-1* configuration using long random walks, so the generated solutions consist mostly of random actions. Furthermore, these solutions are so long that Aras has memory problems. The best solution for problem 1 is found by FD-Autotune-2. However, this planner did not scale to larger instances.

In *Large-Openstacks*, Figure 3(b), both LAMA-2011 and Arvand-LS generate good quality solutions initially, but Arvand-LS scales better. Arvand-LS finds better quality solutions than Arvand until problem 16, and then Arvand takes over for larger instances. Beyond problem 16, Arvand-LS does not solve the problems in all 5 trial runs.

For smaller *Large-Visit-All* instances, both Arvand and Arvand-LS generate lower quality plans than LAMA. Only Arvand-LS scales beyond problem 13. To avoid making *bubbles* - unvisited squares surrounded by visited squares - is a good strategy to generate good quality plans in this domain. Random walks visit unvisited squares quickly in the beginning, but create a lot of *bubbles* on the way, leading to bad plan quality.

The *Large-Woodworking* instances, Figure 3(d), seem relatively easier since more planners can solve problems here. The plan quality on the first 10 problems is very close for the tested planners. Only Arvand-LS scales beyond problem

[1]In *Large-Visit-All*, only three planners solve any problem.

| Domains | Arvand-LS | Arvand | Roamer | Probe | LAMA-2011 | LAMA-2011-FF | FD-AT-2 | FDSS-2 |
|---|---|---|---|---|---|---|---|---|
| barman(20) | 9.6 | 0 | 16.8 | **20** | **20** | 17 | 4 | 14 |
| elevators(20) | **20** | **20** | 13.4 | **20** | **20** | 18 | 16 | 18 |
| floortile(20) | 1.2 | 1.8 | 1 | 4 | 5 | 4 | **9** | 6 |
| nomystery(20) | 14 | **19** | 10 | 7 | 11 | 8 | 16 | 13 |
| openstacks(20) | **20** | **20** | **20** | 12 | **20** | 17 | 19 | 14 |
| parcprinter(20) | **20** | **20** | 7 | 13 | **20** | **20** | 14 | **20** |
| parking(20) | 8.6 | 3.8 | 5 | 15 | 15 | 11 | 4 | **19** |
| pegsol(20) | **20** | **20** | 19 | **20** | **20** | **20** | **20** | **20** |
| scanalyzer(20) | **20** | 17.6 | **20** | 18 | **20** | **20** | 17 | **20** |
| sokoban(20) | 8.4 | 2.2 | 12.4 | 15 | **18** | **18** | 15 | **18** |
| tidybot(20) | 17 | 17.8 | 15 | **18** | 16 | 11 | 16 | 16 |
| transport(20) | **18** | 14 | 16 | 13 | 14 | 12 | 9 | 13 |
| visitall(20) | **20** | **20** | 18 | 18 | **20** | 4 | 4 | 6 |
| woodworking(20) | **20** | **20** | 18.6 | **20** | **20** | **20** | 13 | **20** |
| Total(280) | 216.8 | 196.2 | 192.2 | 213 | **239** | 200 | 176 | 217 |

Table 1: Number of tasks solved in IPC-2011. Total number of tasks shown in parentheses after each domain name.

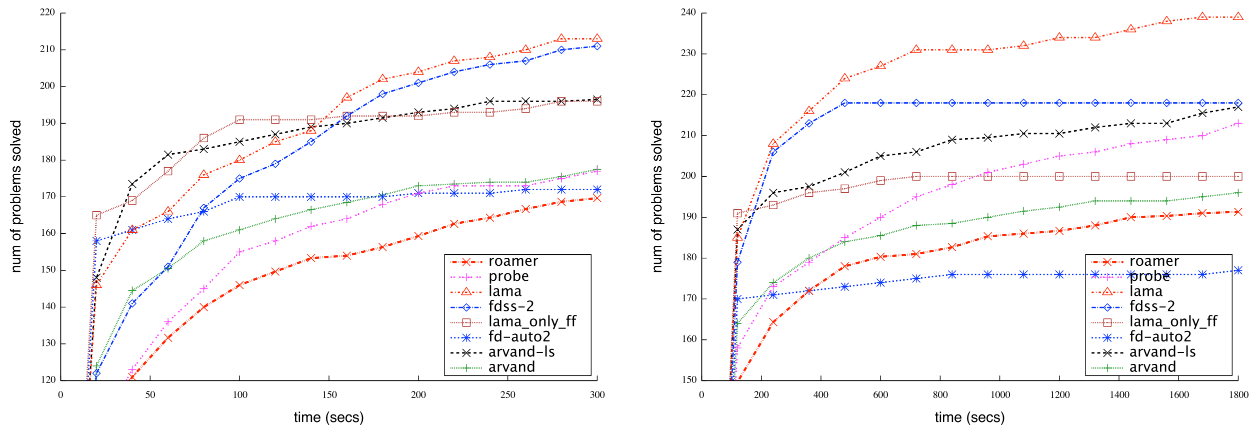


Figure 2: The number problems solved in the first 5 minutes and 30 minutes

13. The low score for problem 17 is due to 3 out of 5 trial runs of Arvand-LS failing.

To summarize, Arvand-LS is the strongest planner overall in these tests in terms of both coverage and quality. In some problems with moderate plan length, Arvand has better performance than Arvand-LS. Because of the post-processing system Aras, for small problems, the diversity of input plans for Aras is more important than the quality of initial plans. Arvand produces more, and also more diverse, plans than Arvand-LS because of its faster pure exploration strategy. For larger problems, the very long plans generated by Arvand often deviate too far from good plans, and the local optimization of Aras is not sufficient to substantially improve these.

## Conclusions and Future Work

The primary contribution of this paper is introducing the new algorithm RW-LS, which uses a local Greedy Best-First Search driven by both direct node evaluation and Random Walks. The planner Arvand-LS improves both coverage and quality significantly over the IPC-2011 version of Arvand. In domains which contain sufficiently many paths from the initial state to a goal the algorithm scales better than other state of art planners.

Because of its complimentary strengths compared to planners based on more systematic search, Arvand-LS is a strong candidate component for inclusion in future portfolio planners.

Another contribution of this paper is motivating the need for, and providing scaled up test instances from IPC planning domains in form of the *IPC-2011-LARGE* collection. Since several of the IPC-2011 test sets have become too easy for the best planners, the limits of current planning technology on standard domains can be explored only by scaling up to larger instances.

One remaining weakness of Arvand-LS is in problems that require exhaustive search of large regions of the state space, such as *Sokoban* and *Parking*. Planners such as LAMA-2011 and Probe are much stronger in solving those problems. While a limited local GBFS can escape from

| Domains | Arvand-LS | Arvand | Roamer | Probe | LAMA-2011 | LAMA-2011-FF | FD-AT-2 | FDSS-2 |
|---|---|---|---|---|---|---|---|---|
| barman(20) | 9.56 | 0.00 | 14.22 | **17.98** | 16.78 | 14.12 | 2.55 | 13.15 |
| elevators(20) | **18.79** | 7.63 | 9.57 | 7.57 | 9.20 | 8.87 | 13.99 | 11.76 |
| floortile(20) | 1.20 | 1.80 | 0.89 | 2.13 | 4.22 | 3.42 | **8.96** | 5.21 |
| nomystery(20) | 13.16 | **18.11** | 9.65 | 6.79 | 10.84 | 7.89 | 15.61 | 12.28 |
| openstacks(20) | 15.25 | 9.78 | 17.20 | 10.61 | **18.59** | 15.93 | 18.08 | 11.94 |
| parcprinter(20) | 18.56 | 17.61 | 6.05 | 11.29 | **19.82** | 18.34 | 13.64 | 18.24 |
| parking(20) | 6.63 | 2.85 | 3.04 | 7.32 | 12.82 | 6,79 | 3.84 | **17.14** |
| pegsol(20) | **19.96** | 19.91 | 17.69 | 18.41 | 19.90 | 19.23 | 19.78 | 14.46 |
| scanalyzer(20) | **18.94** | 15.41 | 16.63 | 14.83 | 17.60 | 16.59 | 15.14 | 17.81 |
| sokoban(20) | 7.89 | 2.17 | 12.15 | 11.83 | **16.64** | 17.61 | 14.92 | 16.08 |
| tidybot (20) | 15.01 | 15.85 | 12.95 | **16.40** | 14.04 | 9.63 | 13.78 | 13.19 |
| transport(20) | **16.26** | 10.54 | 13.36 | 6.76 | 11.02 | 7.10 | 7.51 | 7.67 |
| visitall (20) | 11.51 | 7.59 | 17.43 | 17.57 | **18.07** | 1.40 | 3.35 | 1.73 |
| woodworking(20) | 16.36 | 16.09 | 11.05 | 17.87 | 15.26 | 15.28 | 9.71 | **19.14** |
| Total(280) | 189.08 | 145.34 | 161.86 | 167.36 | **204.78** | 162.21 | 160.87 | 179.80 |

Table 2: Score in IPC-2011. The maximum possible score is shown in parentheses after each domain name.

| Domains | Arvand-LS | Arvand | Roamer | Probe | LAMA-2011 | FD-AT-2 | FDSS-2 | FF | LPG |
|---|---|---|---|---|---|---|---|---|---|
| large-elevator(20) | **20.0** | 18.8 | 1.2 | 4.0 | 4.0 | 1.0 | 2.0 | 3.0 | 0.4 |
| large-openstacks (20) | 16.0 | **17.6** | 10.0 | 0.0 | 8.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| large-visitall(20) | **20.0** | 12.6 | 0.0 | 0.0 | 13.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| large-woodworking(20) | **18.0** | 10.4 | 5 | 4.0 | 11.0 | 3.0 | 11.0 | 3.0 | 8.6 |
| total(80) | **74.0** | 59.4 | 16.2 | 8.0 | 36.0 | 4.0 | 13.0 | 7.0 | 9.0 |

Table 3: Number of tasks solved in IPC-2011-LARGE. Total number of tasks shown in parentheses after each domain name.

small local minima or plateaus with narrow exit points, it fails for large cases. Some interesting topics for future work include:

- Build a portfolio planner containing Arvand-LS as well as other planners such as LAMA and Arvand.
- Add efficient complete global search algorithms such as Beam-Stack Search (Zhou and Hansen 2005) to the RW-LS framework, in order to improve performance in problems where exhaustive search is important.
- Review the generators for more planning domains, for example those used in previous competitions, and identify further domains that can also be scaled up and added to the collection of large problems.
- Study the scaling behavior of current planners under other parameters such as increased memory and time.

## References

García-Olaya, A.; Jiménez, S.; and Linares López, C., eds. 2011. *The 2011 International Planning Competition*. Universidad Carlos III de Madrid.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs in LPG. *J. Artif. Intell. Res. (JAIR)* 20:239–290.

Helmert, M.; Röger, G.; and Karpas, E. 2011. Fast Downward Stone Soup: A baseline for building planner portfolios. In *Proceedings of the ICAPS-2011 Workshop on Planning and Learning (PAL)*, 28–35.

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

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

Lipovetzky, N., and Geffner, H. 2011. Searching for plans with carefully designed probes. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS-2011)*, 154–161.

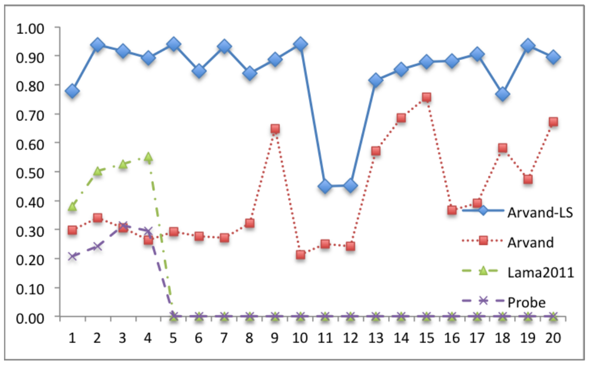
Lu, Q.; Xu, Y.; Huang, R.; and Chen, Y. 2011. The Roamer planner random-walk assisted best-first search. In García-Olaya et al. (2011), 73–76.

Nakhost, H., and Müller, M. 2009. Monte-Carlo exploration for deterministic planning. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI'09)*, 1766–1771.

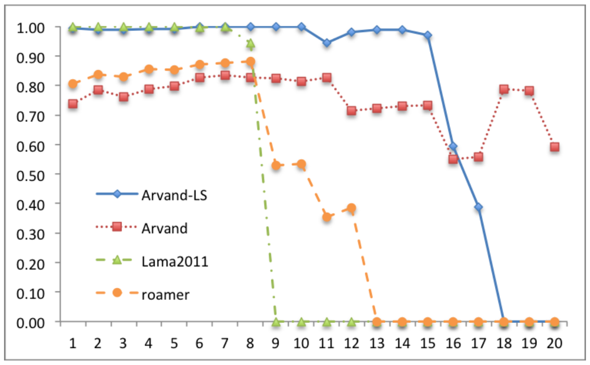
Nakhost, H., and Müller, M. 2010. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In Brafman, R.; Geffner, H.; Hoffmann, J.; and Kautz, H., eds., *Proceeedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS-2010)*, 121–128. Toronto, Canada: AAAI Press.

Nakhost, H.; Müller, M.; Valenzano, R.; and Xie, F. 2011. Arvand: the art of random walks. In García-Olaya et al. (2011), 15–16.

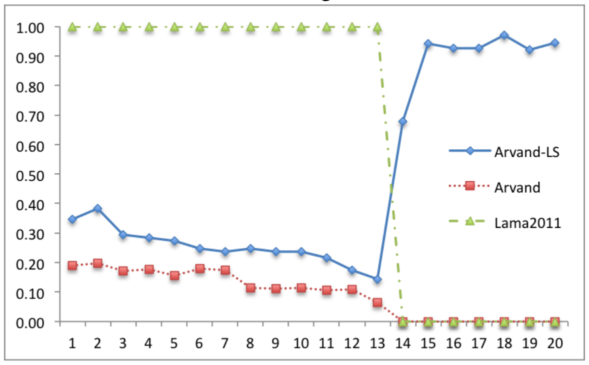
Nakhost, H.; Hoffmann, J.; and Müller, M. 2012. Resource-constrained planning: A Monte Carlo random walk ap-



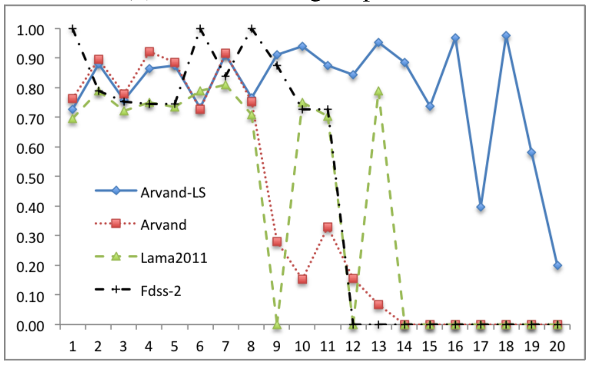
(a) IPC-2011-Large-Elevators



(b) IPC-2011-Large-Openstacks



(c) IPC-2011-Large-Visit-All



(d) IPC-2011-Large-Woodworking

Figure 3: Scores of top four planners on IPC-2011-LARGE. The x-axis presents the problem numbers. The size of problems grows from problem 1 to problem 20.

proach. In *Proceeedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS-2012)*.

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.; Westphal, M.; and Helmert, M. 2011. LAMA 2008 and 2011. In García-Olaya et al. (2011), 50–54.

Valenzano, R.; Nakhost, H.; Müller, M.; Schaeffer, J.; and Sturtevant, N. 2011. ArvandHerd: Parallel planning with a portfolio. In García-Olaya et al. (2011), 113–116.

Zhou, R., and Hansen, E. 2005. Beam-stack search: Integrating backtracking with beam search. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICPAS-2005)*, 90–98.