Towards a Second Generation Random Walk Planner: an Experimental Exploration

Hootan Nakhost and Martin Müller
University of Alberta, Edmonton, Alberta, Canada
{nakhost,mmueller}@ualberta.ca

Abstract

Random walks have become a popular component of recent planning systems. The increased exploration is a valuable addition to more exploitative search methods such as Greedy Best First Search (GBFS). A number of successful planners which incorporate random walks have been built. The work presented here aims to exploit the experience gained from building those systems. It begins a systematic study of the design space and alternative choices for building such a system, and develops a new random walk planner from scratch, with careful experiments along the way. Four major insights are: 1. a high state evaluation frequency is usually superior to the endpoint-only evaluation used in earlier systems, 2. adjusting the restarting parameter according to the progress speed in the search space performs better than any fixed setting, 3. biasing the action selection towards preferred operators of only the current state is better than Monte Carlo Helpful Actions, which depend on the number of times an action has been a preferred operator in previous walks, and 4. even simple forms of random walk planning can compete with GBFS.

1 Introduction

The most common current technique for building satisficing planning systems is heuristic search [Bonet and Geffner, 2001]. In IPC-2011, it was used by 25 out of 27 planners in the deterministic satisficing track. Most of these planners use greedy search algorithms such as Greedy Best First Search (GBFS), weighted A* and Enforced Hill Climbing. These algorithms mainly exploit the heuristic and do not explore the search space much. This lack of exploration hurts performance in case of inaccurate heuristic values, which are very common in the automatically generated heuristic functions of domain independent planning. A search algorithm that is more robust with inaccurate or misleading heuristics is not only valuable, but essential to improve the state of the art. [Nakhost and Müller, 2009] introduced Monte Carlo Random Walks (MRW), an explorative, forward chaining local search method. MRW runs bounded length random walks (RW) to an endpoint which is evaluated by a heuristic h. After sampling, an endpoint with lowest h-value becomes the next state.

Previous systems that utilize RW include many Stochastic Local Search algorithms [Hoos and Stützle, 2004], Rapidly-Exploring Random Trees (RRT) in robot path planning [Alcázar *et al.*, 2011], and the planners Identidem [Coles *et al.*, 2007], Roamer [Lu *et al.*, 2011], and Arvand [Nakhost and Müller, 2009; Nakhost *et al.*, 2011; Xie *et al.*, 2012; Nakhost *et al.*, 2012; Valenzano *et al.*, 2012]. The current study uses the experience from this first generation of RW planners to build Arvand-2013, a second generation system, from scratch. Each design step is justified by careful experiments, which also address important open questions about RW planning. Does it pay off to evaluate more of the generated states, and if so, which fraction? What are effective ways to control the length of walks and restarting? Is it possible to improve random walks by using preferred operators, beyond MHA [Nakhost and Müller, 2009]? Like most experimental papers on AI planning, experiments are run on recent IPC benchmarks.

2 Building a Second Generation RW Planner

2.1 The Experimental Framework

Like many current systems, the new planner Arvand-2013 is built on top of the Fast Downward (FD) code base [Helmert, 2006]. All tested algorithms share the FD implementation of successor generator, heuristic function, and representation of states and actions. Tests are run on all IPC-2011 domains on a 2.5 GHz machine with 4GB memory and 30 minutes per instance. Results for randomized planners are averaged over 5 runs, which is mandated by computational resource limits but already quite reliable, especially in cases of frequent restarts within each run. The main focus is on *coverage* - the number of problems solved.

2.2 Baseline Arvand-2013: a Simple RW Planner

Despite many recent experiments on RW planning, basics such as controlling the length of random walks and the restarting strategy have not been further explored since the original work of [Nakhost and Müller, 2009]. Does the effectiveness of a restarting strategy depend on the walk length distribution? Is there a robust strategy performing well across all or at least most planning domains? If not, is it possible to dynamically learn the most effective strategies?

The planner Arvand-2013 is built bottom-up, starting with the simple RW planner shown in Algorithm 1. The first experiment studies the effects of restarting and the length of walks in this planner. In the forward chaining local search, each *run* consists of one or more *search episodes*, and terminates when the current episode meets a *termination condition*. Episodes start from initial state s_0 and perform a series of search steps until a *restart condition* or *termination condition* becomes true. Let h_{min} be the minimum heuristic value reached in the current episode. Each search step $step_i$ starts from state s_{i-1} and ends in s_i with $h(s_i) < h_{min}$. In $step_i$, the planner runs a series of random walks to select s_i. When a random walk reaches a state s with $h(s) < h_{min}$, the algorithm immediately jumps there, setting $s_i = s$. The partial plan sequence of actions from s_0 to s_i is tracked. The two termination conditions are:

1. reaching a goal state s_G. In this case the planner returns the sequence of actions from s_0 to s_G.
2. exceeding a time limit. No plan is returned (details omitted from the code).

Like Arvand, local search restarts using a *restart threshold* t_g, whenever t_g successive walks fail to improve h_{min}. To begin a new episode, the current state is reset to s_0 .

Within a random walk (Algorithm 2), baseline Arvand-2013 evaluates *all* visited states, unlike Arvand. A walk stops early if it reaches a goal state, achieves an h-value below h_{min}, or hits a dead end; otherwise it continues until terminating as in the restarting random walks model of [Nakhost and Müller, 2012], with fixed probability r_l at each step. r_l is called the *local restarting rate*. In the absence of early stops, the length of walks is geometrically distributed with mean $1/r_l$. As the heuristic function, Arvand-2013 uses the cost-sensitive version of h^{FF} [Hoffmann and Nebel, 2001] from the FD code base.

Algorithm 1 Monte Carlo Random Walk Planning

Input Initial State s_0, goal condition G, heuristic h
Output Solution plan
Parameters t_g, r_l

```
  c ← s0 {current state}
  h_min ← h(c)
  loop
    s ← randomWalk(c, G, h_min, r_l)
    if s ⊇ G then
      return plan from s0 to s
    else if s ≠ Deadend and h(s) < h_min then
      c ← s; h_min ← h(c)
    else if restart() then
      c ← s0; h_min ← h(c)
    end if
  end loop
```

Algorithm 2 Random Walk

Input current state c, goal condition G, h_{min}
Output sampled state s
Parameter r_l

```
  loop
    s ← c; A ← applicableActions(s)
    if A = ∅ or h(s) = ∞ then
      return Deadend
    end if
    a ← uniformlyRandomSelectFrom(A)
    s ← apply(s, a)
    if h(s) < h_min or s ⊇ G then
      return s
    end if
    with probability r_l: return s
  end loop
```

2.3 Parameters for Global and Local Restarts

The first experiment studies the coverage of Arvand-2013 as a function of the parameters t_g and r_l which control global and local restarts. Out of 14 IPC-2011 domains, ten with interesting results are shown in Figure 1. Not shown are BARMAN and TRANSPORT, where no configuration solved more than 10% of problems, and OPENSTACKS and PEGSOL, with more than 90% coverage in all configurations. Key observations are:

- This very basic RW planner already solves 126.6 of 280 IPC-2011 problems with the best tested fixed thresholds of $t_g = 100, r_l = 0.01$. Section 2.6 gives a more comprehensive comparison with other planners.
- No single setting performs well across all domains. Larger r_l values leading to shorter walks perform better in NOMYSTERY and WOODWORKING, but worse in TIDYBOT and VISITALL. In ELEVATORS, restarting rarely with $t_g = 10000$ increases the coverage compared with frequent restarting with $t_g = 100$. In NOMYSTERY, FLOORTILE, PARCPRINTER and TIDYBOT, such frequent restarts are better.
- The two parameters r_l and t_g are mostly independent. Larger r_l are always better in NOMYSTERY and WOODWORKING and always worse in TIDYBOT and VISITALL, independent of the choice of t_g. Higher t_g values are never worse in WOODWORKING or ELEVATORS, but detrimental in the other domains, independent of r_l.

In light of these results, finding robust settings for t_g and r_l that work well across all domains seems infeasible. This is not surprising considering the widely varying characteristics of IPC-2011 domains. A parameter learning system can help to fully realize the potential of RW.

2.4 Adaptive Global Restarting

Why does Arvand-2013 with a small restarting threshold perform well in ELEVATORS but fail in FLOORTILE and NOMYSTERY? Is it possible to learn an effective restarting strategy on the fly? Figure 2 shows details of two typical examples, contrasting ELEVATORS and FLOORTILE. The graphs plot h_{min} as a function of the number of RW for $t_g = 1000$ and $t_g = 10000$. In FLOORTILE, h_{min} decreases very quickly at first, then stalls in a dead end or very large local

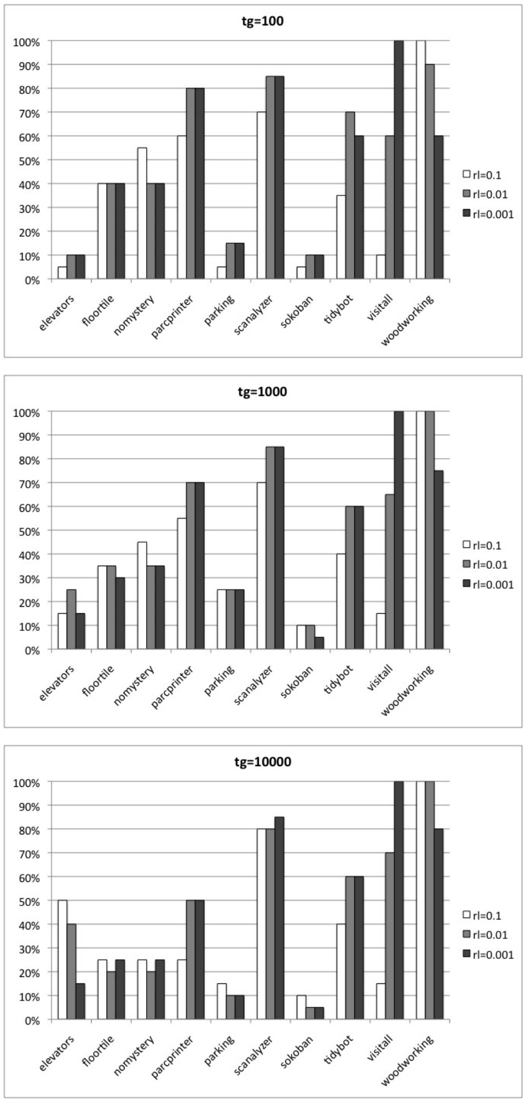

Figure 1: Coverage of BRW for $r_l \in \{0.1, 0.01, 0.001\}$, $t_g \in \{100, 1000, 10000\}$.

minimum. Here, a large t_g wastes lots of time exploring those dead ends and local minima, while restarting more often with a small t_g increases exploration and the chance of reaching a goal. ELEVATORS shows the opposite behaviour: the plots show steady, slow progress towards $h_{min} = 0$. Fast restarts terminate the search before it can reach a goal.

Let V_w (V for velocity, w for walks) be the *average heuristic improvement per walk*, so on average, about $h(s_0)/V_w$ walks should reach $h = 0$. Algorithm 3, *Adaptive global restarting (AGR)*, adjusts t_g by continually estimating V_w and setting $t_g = h(s_0)/V_w$. AGR initializes $t_g = 1000$ and updates both t_g and the estimated V_w after each episode. Before the i-th episode, AGR measures V_w^i, the average number of random walks to reach h_{min} and sets $V_w = \text{avg}_{j \le i} V_w^j$.

Figure 3 compares AGR with restarting with fixed t_g. While not always best, AGR is a robust, close to best choice in all domains. With AGR and $r_l = 0.01$, Arvand-2013

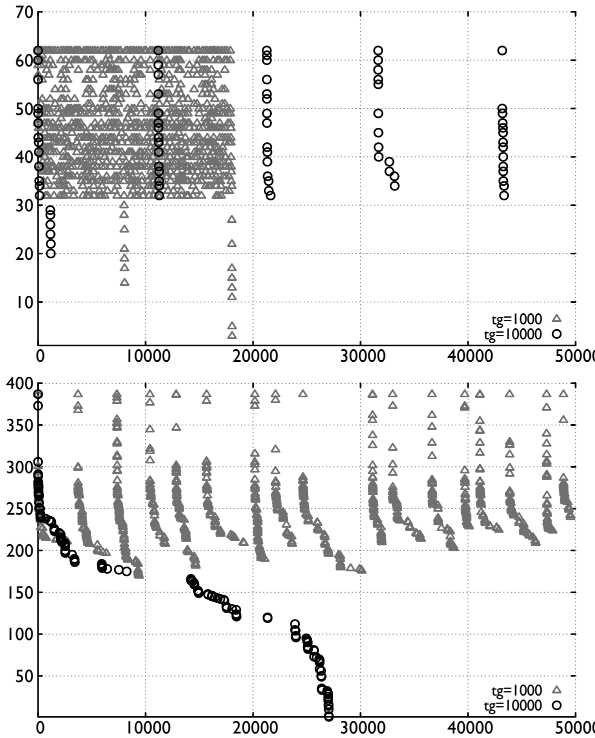

Figure 2: Progress of h_{min}, depending on t_g, in FLOORTILE-01 (top) and ELEVATORS-03(bottom).

Algorithm 3 Monte Carlo Random Walks using AGR

Input Initial State s_0, goal condition G
Output A solution plan
Parameters t_g, r_l

```
c ← s0; hmin ← h(s0)
r ← 0; w ← 0 {number of restarts; walks}
li ← 0 {last improving walk}
Vw ← 0
loop
   s ← RandomWalk(c, G, hmin, rl){sampled state}
   ++w
   if s ⊇ G then
      return plan reaching s
   else if s ≠ Deadend and h(s) < hmin then
      c ← s
      hmin ← h(s)
      li ← w
   else if w − li > tg then
      Vw^i ← (h(s0) − hmin)/li
      Vw ← (Vw^i − Vw)/r + Vw {update estimate}
      tg ← h(s0)/Vw {update tg}
      c ← s0{restart from initial state}
      hmin ← h(s0)
      w ← 0; ++r
   end if
end loop
```

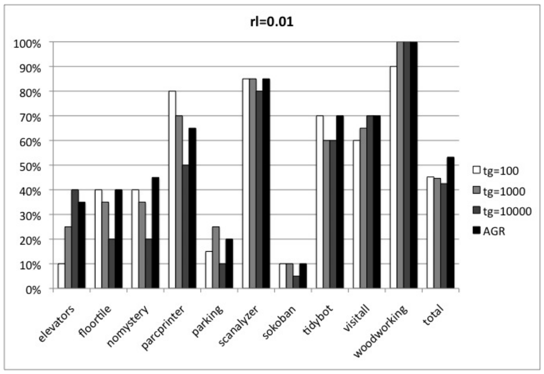

Figure 3: Coverage of AGR versus fixed threshold restarting with $t_g \in \{100, 1000, 10000\}$, with fixed $r_l = 0.01$.

solves 149 of 280 problems, 22 more than the best tested fixed thresholds of $t_g = 100, r_l = 0.01$.

2.5 Adaptive Local Restarting

As for global restarting, an adaptive algorithm can improve local restarting. As motivation, Figure 4 plots h_{min} against number of evaluated nodes in VISITALL-15 and ELEVATORS-05, for $r_l = \{0.1, 0.01, 0.001\}$ and $t_g = 10000$. Let $V_e(r)$ (e for evaluations) be the *average heuristic improvement per evaluation* when $r_l = r$. Larger V_e indicate faster progress towards a goal. In VISITALL, smaller r_l settings achieve faster progress (larger V_e). The opposite happens in ELEVATORS.

Adaptive local restarting (ALR) is a multi-armed bandit method [Gittins *et al.*, 2011] that estimates $V_e(.)$ to learn the best r_l. Before each random walk, ALR selects $r_l = r_i$ from a candidate set $C = \{r_1, \ldots, r_n\}$. Each r_i can be considered one arm of the bandit. For each r_i, ALR tracks the average number of evaluations $\text{avg}_e(r_i)$ and the average heuristic improvement $\text{avg}_h(r_i)$, which is bounded below by 0. $\text{avg}_h(r_i)/\text{avg}_e(r_i)$ is used as estimate for $V_e(r_i)$. ALR samples arms in an ϵ-greedy manner [Sutton and Barto, 1998]: an arm is selected uniformly at random with probability $\epsilon \geq 0$, and with probability $1 - \epsilon$, an arm with largest estimated $V_e(r_i)$ is chosen.

Figure 5 compares ALR using $\epsilon \in \{0.1, 1\}$ with three fixed settings $r_l \in \{0.1, 0.01, 0.001\}$. To ensure comparable results, the ALR candidate set is the same, $C = \{0.1, 0.01, 0.001\}$. In all configurations, AGR is used for *global* restarting. Key observations are:

- ALR performs robustly across all domains: the gap between ALR and the best fixed setting for a domain is never more than 10%, except in ELEVATORS where $r_l = 0.1$ solves 15% more problems.
- Sampling based on $V_e(.)$ gives a small advantage over uniform sampling: Setting $\epsilon = 0.1$ solves 6 (2%) more problems than uniform sampling with $\epsilon = 1$.

2.6 First Comparison with Systematic Search

This experiment studies how Arvand-2013 at this stage of development compares with GBFS, a popular systematic search planner. To keep the playing field level at this point, GBFS uses the same single heuristic h^{FF} and no preferred operators. Figure 6 compares the coverage:

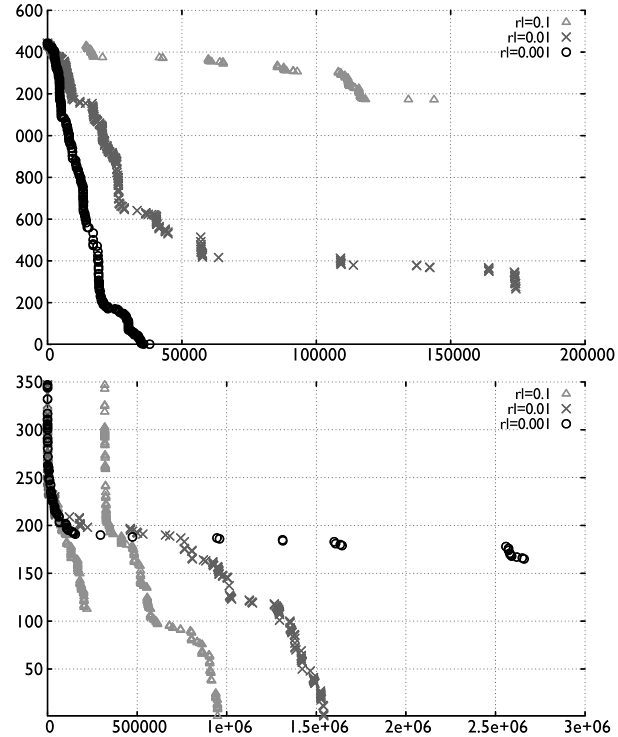

Figure 4: h_{min} progress with number of evaluated states in VISITALL-14 (top) and ELEVATORS-05 (bottom).

- Arvand-2013 and GBFS have very different strengths and weaknesses: Arvand-2013 on average solves 5 (25%) to 16 (80%) more problems in ELEVATORS, PARCPRINTER, and VISITALL, GBFS solves 7 (35%) to 15 (75%) more in BARMAN, PARKING, and SOKOBAN.
- Overall, Arvand-2013 is about level with GBFS, solving 5 (2%) more problems.

3 The Rate of Heuristic Evaluation

While heuristic state evaluations provide key information to guide search, computing a strong heuristic such as h^{FF} is costly. Two methods which reduce the number of evaluations are *deferred evaluation* [Helmert, 2006], which uses the parent's evaluation for a node, and MRW [Nakhost and Müller, 2009], which evaluates only the endpoint of a random walk. The next experiment varies the frequency of state evaluations: Instead of all states as in the baseline, Arvand-2013 evaluates:
1. the endpoint of each random walk as in Arvand, and
2. intermediate states with probability p_{eval}.
This interpolates between the baseline algorithm with $p_{eval} = 1$ and MRW with $p_{eval} = 0$. ALR with $\epsilon = 0.1$ and AGR are used as in previous versions. Figure 7 shows the coverage in IPC-2011 when varying p_{eval}. Four categories of domains emerge:

- Domains where more evaluation always hurts: Arvand-2013 solves 100% of OPENSTACKS, VISITALL and

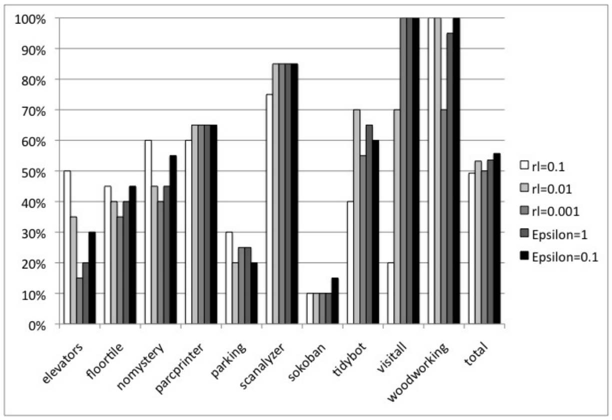

Figure 5: Coverage of ALR and local restarting with fixed rate, $\epsilon \in \{0.1, 1\}$, $r_l \in \{0.1, 0.01, 0.001\}$.

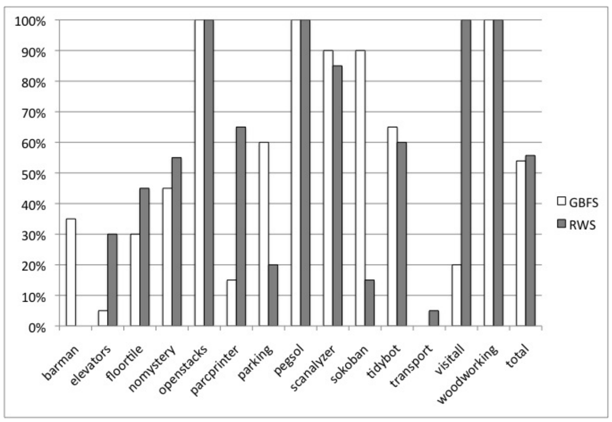

Figure 6: Coverage of simple versions of GBFS and Arvand-2013.

WOODWORKING even with $p_{eval} = 0$. RW search is so effective that higher evaluation rates only increase the runtime. In TIDYBOT, $p_{eval} = 0$ is also best, but coverage is below 100%. Here, h^{FF} is both very costly and misleading. For reference, even a *blind* random walk search, using only goal checks but no evaluation at all, can solve 90% of TIDYBOT instances! Only one planner in the IPC-2011 competition, BRT [Alcázar and Veloso, 2011], surpassed that number.

- Domains where more evaluation always pays off: PARCPRINTER and NOMYSTERY. Running time decreases with increasing p_{eval}, so spending more time on evaluation is worth it.
- An intermediate evaluation rate works best in ELEVATORS, FLOORTILE, and PARKING. $p_{eval} = 0$ provides too little information and $p_{eval} = 1$ is too slow.
- In SCANALYZER, PEGSOL, SOKOBAN, and TRANSPORT, the coverage is the same for all tested $p_{eval} > 0$.

These results challenge the previous practice of always setting $p_{eval} = 0$ as in MRW, and show that for a significant number of domains a higher evaluation rate is suitable.

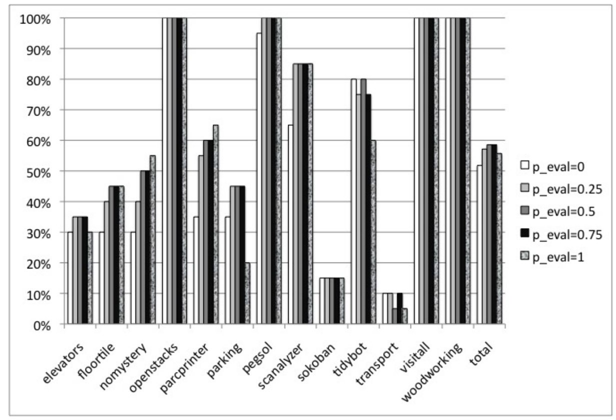

Figure 7: Coverage varying $p_{eval} \in \{0, 0.25, 0.5, 0.75, 1\}$.

4 Biased Action Selection for Random Walks

In the versions so far, Arvand-2013 used the heuristic computation only to obtain the h-value. Preferred operators can be obtained from common heuristic functions at no additional cost. Using them to guide random walks is the next step.

The GRIPPER example in [Nakhost and Müller, 2012] demonstrates that biased action selection can lower the regress factor, and greatly decrease the runtime of RW. Monte Carlo Helpful Actions (MHA) [Nakhost and Müller, 2009] bias action selection using information gathered from random walks. $Q(a)$ scores are updated for each possible action a, and actions are sampled from a Gibbs distribution with temperature T. For current state s with applicable actions $A(s)$, action a is chosen with probability

$$p(a, s) = \frac{e^{Q(a)/T}}{\sum_{b \in A(s)} e^{Q(b)/T}} .$$

T determines the strength of the bias towards actions with larger scores: lowering T gives less uniform distributions.

In the MHA implementation of [Nakhost and Müller, 2009], $Q(a)$ counts the number of times an action is preferred in the current search step. $Q(a)$ is computed only from statistics gathered from endpoints of walks starting from the current state s. Preferred operators of intermediate states are not computed. Scores are reinitialized at every jump to another state. Preferred operators of s itself are not treated separately, which seems counterintuitive: they could at least be given a higher priority.

Arvand-2013 extends MHA to exploit the extra information from its more frequent state evaluations, and give higher priority to current preferred operators, when known. Let $n(a)$ be the number of times that a was a preferred operator, $N = \max_{a \in A(s)} n(a)$, and PO($s$) the set of preferred operators in s. Then:

$$Q(a) = \begin{cases} N \times W + n(a)(1 - W) & \text{if } a \in \text{PO}(s) \\ n(a) & \text{Otherwise} \end{cases}$$

The parameter $W \in [0, 1]$ controls the relative weight of the operators in PO(s): larger W favor them more. In states where PO(s) is not computed and therefore the empty set, the result is the same as in classical MHA.

Figure 8 shows the coverage of MHA, varying the temperature T and weight W, against the version without MHA. In all runs, $p_{eval} = 0.5$, ALR($\epsilon = 0.1$) and AGR are used.

Let MHA(w, t) denote a version of Arvand-2013 as above enhanced with MHA, with $W = w$ and $T = t$. Key observations are:

- MHA($1, 10$) is very effective. Coverage in BARMAN improves from 0 to 18 (90%), in TRANSPORT from 2 (10%) to 20 (100%), in ELEVATORS from 7 (35%) to 20 (100%) and in PARKING from 9 (45%) to 17 (85%). In total, MHA($1, 10$) solves 56 more problems (20%).
- $W = 1$, using only the current preferred operators when available, works best. MHA($1, t$) consistently outperforms MHA($0.5, t$), which outperforms MHA($0, t$).
- With $W = 1$, for most domains lower temperatures of $T = 10$ and $T = 100$ are preferable. Exceptions are PARCPRINTER with $T = 1$ and TIDYBOT with $T = 1000$.

5 Comparison with Other Planners

Table 1 shows total coverage in IPC-2011 for the successive versions of Arvand-2013, compared with the top three competition planners in terms of coverage, LAMA2011, FDSS2 [Helmert *et al.*, 2011] and Probe [Lipovetzky and Geffner, 2011], as well as the RW planner Roamer. The last version of Arvand-2013 using MHA and $p_{eval} = 1$ is very competitive. Overall, it only lags behind LAMA2011. This is mainly due to the results in SOKOBAN: Arvand-2013 solves 17 fewer problems in this domain, which is considered to be hopeless for RW search [Xie *et al.*, 2012].

Arvand-2013 Version	solved	Ref. Planner	solved
Baseline	127	Roamer	215
+AGR	149	FDSS2	220
+ALR, $p_{eval} = 0.5$	164	Probe	226
+ALR, $p_{eval} = 1$	156	LAMA2011	**250**
+MHA, $p_{eval} = 0.5$	219		
+MHA, $p_{eval} = 1$	**226**		

Table 1: Number of solved tasks out of 280 in IPC-2011. Left: Arvand-2013 versions. Right: IPC reference planners.

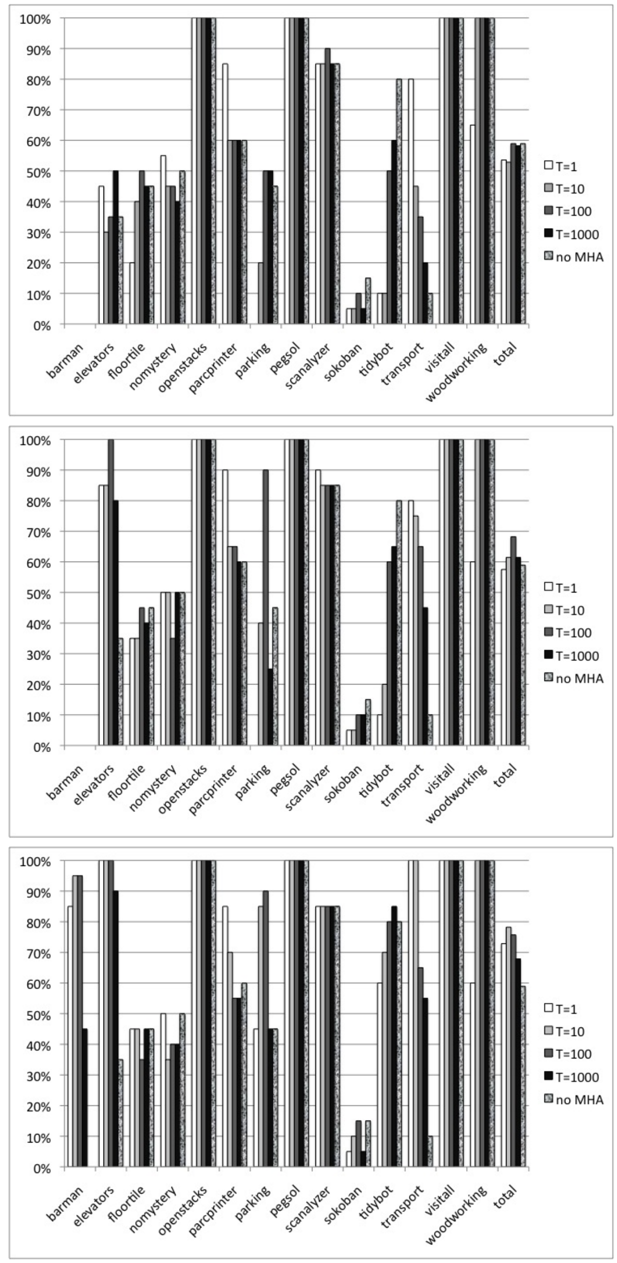

Figure 8: Coverage of MHA versus uniform action selection (no MHA), with $T \in \{1, 10, 100, 1000\}$ and $w = 0$ (top), $w = 0.5$ (middle), $w = 1$ (bottom).

6 Conclusions and Future Work

The systematic bottom-up reconstruction of the new RW planner Arvand-2013 challenges several assumptions and design choices made in previous systems, and shows that strong improvements to RW systems are still possible. Two observations stand out:

1. The importance of adaptive systems: this becomes more important for search algorithms like RW search, which instead of systematically exploring all states, selectively sample parts of the search space: the effective distribution of samples depends on the search space characteristics of the input problem. ALR and AGR provide practical guidelines for developing such adaptive systems.
2. The big effect of action selection biasing: as the theory developed in [Nakhost and Müller, 2012] predicts and experiments in Section 4 confirm, action selection biasing can significantly improve the performance of RW search. MHA is one successful example of a biasing technique.

The latest version of Arvand-2013 is still relatively simple, with much room for adding features, but already has strong performance. Future work includes investigating heuristic functions other than h^{FF}, using multiple heuristics in RW planning, and tuning for plan quality as opposed to coverage.

References

[Alcázar and Veloso, 2011] V. Alcázar and M. Veloso. BRT: Biased rapidly-exploring tree. In *The 2011 International Planning Competition, IPC 2011, Universidad Carlos III de Madrid*, pages 17–20, 2011.

[Alcázar *et al.*, 2011] V. Alcázar, M. Veloso, and D. Borrajo. Adapting a rapidly-exploring random tree for automated planning. In *Proceedings of the Forth Annual Symposium on Combinatorial Search, SOCS 2012, Barcelona, Spain, July 15-16, 2011*, 2011.

[Bonet and Geffner, 2001] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[Coles *et al.*, 2007] A. Coles, M. Fox, and A. Smith. A new local-search algorithm for forward-chaining planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, pages 89–96, 2007.

[Gittins *et al.*, 2011] J. Gittins, K. Glazebrook, and R. Weber. *Multi-armed bandit allocation indices*. Wiley, 2011.

[Helmert *et al.*, 2011] M. Helmert, G. Röger, and E. Karpas. Fast Downward Stone Soup: A baseline for building planner portfolios. In *ICAPS 2011 Workshop on Planning and Learning*, pages 28–35, 2011.

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

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

[Hoos and Stützle, 2004] H. Hoos and T. Stützle. *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[Lipovetzky and Geffner, 2011] N. Lipovetzky and H. Geffner. Searching for plans with carefully designed probes. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011*, 2011.

[Lu *et al.*, 2011] Q. Lu, Y. Xu, R. Huang, and Y. Chen. The Roamer planner random-walk assisted best-first search. In *The 2011 International Planning Competition, IPC 2011, Universidad Carlos III de Madrid*, pages 73–76, 2011.

[Nakhost and Müller, 2009] H. Nakhost and M. Müller. Monte-Carlo exploration for deterministic planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009, Pasadena, California, USA, July 11-17, 2009*, pages 1766–1771, 2009.

[Nakhost and Müller, 2012] H. Nakhost and M. Müller. A theoretical framework for studying random walk planning. In *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS 2012, Niagara Falls, Canada, July 19-21, 2012*, 2012.

[Nakhost *et al.*, 2011] H. Nakhost, M. Müller, R. Valenzano, and F. Xie. Arvand: the art of random walks. In *The 2011 International Planning Competition, IPC 2011, Universidad Carlos III de Madrid*, pages 15–16, 2011.

[Nakhost *et al.*, 2012] H. Nakhost, J. Hoffmann, and M. Müller. Resource-constrained planning: A Monte Carlo random walk approach. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*, pages 181–189, 2012.

[Sutton and Barto, 1998] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[Valenzano *et al.*, 2012] R. Valenzano, H. Nakhost, M. Müller, J. Schaeffer, and N. Sturtevant. Arvand-Herd: Parallel planning with a portfolio. In *Proceedings of the 20th European Conference on Artificial Intelligence, ECAI 2012, Montpellier, France, August 27-31, 2012*, pages 113–116, 2012.

[Xie *et al.*, 2012] F. Xie, H. Nakhost, and M. Müller. Planning via random walk-driven local search. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*, pages 315–322, 2012.