Monte-Carlo Exploration for Deterministic Planning

Hootan Nakhost and Martin Müller
Department of Computing Science
University of Alberta
{nakhost,mmueller}@cs.ualberta.ca

Abstract

Search methods based on Monte-Carlo simulation have recently led to breakthrough performance improvements in difficult game-playing domains such as Go and General Game Playing. Monte-Carlo Random Walk (MRW) planning applies Monte-Carlo ideas to deterministic classical planning. In the forward chaining planner ARVAND, Monte-Carlo random walks are used to explore the local neighborhood of a search state for action selection. In contrast to the stochastic local search approach used in the recent planner *Identidem*, random walks yield a larger and unbiased sample of the search neighborhood, and require state evaluations only at the endpoints of each walk. On IPC-4 competition problems, the performance of ARVAND is competitive with state of the art systems.

1 Introduction

Local search methods are among the most effective techniques for satisficing planning. Search steps in a local search strategy are determined based on local knowledge obtained from neighbors in the search space. These techniques typically scale much better than systematic search methods, even though their sophisticated heuristic evaluation functions, such as FF's relaxed graphplan heuristic, are relatively slow to compute.

While they perform very well on "easy" search topologies, current local search methods have trouble dealing with extensive local minima and plateaus. They fail to provide guidance for the search if the heuristic values of neighboring states do not improve the current state's value. Most local search methods are greedy - they try to immediately exploit their local knowledge instead of exploring the neighborhood of a state. Following a misleading heuristic function can quickly lead to a much worse state than what could be achieved with a little exploration. This *exploitation - exploration trade-off* has been extensively studied in so-called bandit problems [Auer *et al.*, 1995]. The resulting search algorithms such as UCT [Kocsis and Szepesvári, 2006] have been hugely successful in difficult adversarial domains such as Go [Gelly and Silver, 2008] and General Game Playing [Finnsson and Björnsson, 2008].

Monte-Carlo Random Walk (MRW) is an algorithm for deterministic planning that uses random exploration of the local neighborhood of a search state for selecting a promising action sequence. MRW is built on two premises:

1. In typical planning problems, action generation is orders of magnitude faster than computing a state of the art heuristic evaluation. Therefore, increased randomized exploration of the local search neighborhood is computationally feasible as long as the number of heuristic evaluations can be limited.
2. The experience from game-playing domains shows that exploration can be a good thing, if carefully balanced with the exploitation of focusing most effort on states with good evaluation.

Monte-Carlo methods have already been applied to several other areas of automated planning, such as sampling possible trajectories in probabilistic planning [Bryce *et al.*, 2006] and robot motion planning [LaValle, 2006]. In the context of deterministic planning, [Fern *et al.*, 2004] used exploration by random walks to learn domain-specific control knowledge.

The remainder of this paper is organized as follows: Subsection 1.1 briefly reviews different techniques used in AI planning to escape from local minima and plateaus. Section 2 describes the main idea of MRW planning. Section 3 explains details of the basic MRW algorithm and two refinements. Section 4 introduces two techniques that use learned action value estimates to address the two main weak points of the basic algorithm. Section 5 discusses the strengths and weaknesses of the new approach and evaluates its performance on standard planning benchmarks from the IPC-4 competition. Sections 6 and 7 contain concluding remarks and some potential directions for future work.

1.1 The Problem of Local Minima and Plateaus

In AI planning, many different approaches have been proposed to tackle the problem of escaping from local minima and plateaus. Fast Forward (FF) [Hoffmann and Nebel, 2001] uses a local search method called Enforced Hill Climbing (EHC), which includes a breadth first search to try to escape from local minima. While EHC is very effective in many planning benchmarks, its systematic search can be slow to escape from extensive local minima/plateaus. Intuitively, a planner should use bigger jumps to get away from such

traps. Several FF-inspired systems address this issue: Marvin [Coles and Smith, 2007] and YAHSP [Vidal, 2004] build macro action sequences, and Identidem [Coles *et al.*, 2007] uses stochastic local search.

Coles and Smith's planner Marvin adds machine-learned plateau-escaping macro-actions to EHC. Marvin's learning system memoizes successful action sequences that led to an *exit* from a plateau, and attempts to escape from similar plateaus via these macros. This strategy is most effective in search spaces that contain many repeating structures.

Vidal's YAHSP constructs an additional macro action in each search step, using actions in FF's relaxed planning graph. This macro-action allows YAHSP to make a big step in the search space, which can help to quickly escape from local minima. The YAHSP look-ahead strategy requires a heuristic function that can also produce the actions of a relaxed plan. Its effectiveness depends on the quality of these actions.

FF, Marvin and YAHSP do not use much exploration. FF and Marvin explore only when the local search gets stuck. While the single macro-action built from the relaxed plan allows YAHSP to jump out of traps, this planner does not explore the neighborhood.

Coles and Smith's Identidem introduces exploration by stochastic local search (SLS) [Hoos and Stützle, 2004]. Identidem's exit strategy consists of trying successively longer *probes*, action sequences chosen probabilistically from the set of all possible actions in each state. Identidem evaluates the FF heuristic after each action of each probe, and immediately jumps to the first state that improves on the start state.

2 Monte-Carlo Random Walks in Planning

In MRW planning, fast Monte-Carlo random walks are used for exploring the neighborhood of a search state. A relatively large set of states S in the neighborhood of the current state s_0 is sampled before greedily selecting a most promising next state $s \in S$. Each state in S is reached through a new random walk starting from s_0. All states in S, *but no states along the walks*, are evaluated by a heuristic function h, for example by the FF heuristic. When a stopping criterion is satisfied, the algorithm replaces s_0 by the minimum h-value element of S. This method uniformly deals with both problems of local search methods: it quickly escapes from local minima, and can recover from areas where the evaluation is poor. The method does not rely on any assumptions about the local properties of the search space or heuristic function. It locally explores the state space before it commits to an action sequence that leads to the best explored state.

Compared to Identidem's probing strategy, ARVAND is able to find better neighbors because of increased sampling and delayed commitment to an action sequence. However, if Identidem finds a very good neighbor, it jumps there immediately, while ARVAND will waste time on more exploration. This issue is partially resolved by the *acceptable progress* stopping criterion of Section 3.2.

Experiments in Section 5 show that this simple planning method significantly outperforms FF on hard planning tasks, and is competitive with recent systems that are built on top of FF's relaxed graphplan heuristic. A main benefit of MRW planning is that increased exploration through fast random walks can overcome many biases introduced by heuristic evaluation functions, while retaining much of their guiding power.

3 Local Search using Monte Carlo

Algorithm 1 shows an outline of the MRW method. For a given planning problem, a forward-chaining search in the state space of the problem is used to find the solution. Search builds a chain of states $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots \rightarrow s_n$ such that s_0 is the initial state, s_n is a goal state, and each transition $s_j \rightarrow s_{j+1}$ uses an action sequence found by MRW exploring the neighborhood of s_j. MRW search fails when the minimum obtained h-value does not improve within a given number of search steps, or when it gets stuck in a dead-end state. In such cases the search simply restarts from s_0.

Algorithm 1 Local Search Using Monte Carlo Random Walks

Input Initial State s_0, goal condition G and available actions A
Output A solution plan

```
s ← s_0
h_min ← h(s_0)
counter ← 0
while s does not satisfy G do
  if counter > MAX_STEPS or DeadEnd(s) then
    s ← s_0 {restart from initial state}
    counter ← 0
  end if
  s ← MonteCarloRandomWalk(s, G)
  if h(s) < h_min then
    h_min ← h(s)
    counter ← 0
  else
    counter ← counter + 1
  end if
end while
return the plan reaching the state s
```

Three variations of MRW methods are explored: The base algorithm uses pure random walks, where all applicable actions are equally likely to be explored. The two other methods use statistics from earlier random walks to bias the random action selection towards previously successful actions, or away from unsuccessful ones.

3.1 Pure Random Walks

The main motivation for MRW planning is to better explore the current search neighborhood. The simplified pseudocode in Algorithm 2 illustrates how random walks are used to generate samples. A walk stops at a goal state, a dead-end, or when its length reaches a bound $LENGTH\_WALK$. The end state of each random walk is evaluated by a heuristic h. The algorithm terminates either when a goal state is reached, or after NUM_WALK walks. The sampled state with minimum h-value and the action sequence leading to it are re-

turned. If all random walks stop at dead-end states, the start state and an empty sequence are returned.

These limits on the length and the number of random walks have a huge impact on the performance of this algorithm. Good choices depend on the planning problem, and the characteristics of the local search space. While they are constant in the basic algorithm shown here, the next subsection describes how these parameters are adapted dynamically in ARVAND.

Algorithm 2 Pure Random Walk

Input current state s, goal condition G
Output s_{min}

```
1: h_min ← INF
2: s_min ← NULL
3: for i ← 1 to NUM_WALK do
4:    s' ← s
5:    for j ← 1 to LENGTH_WALK do
6:       A ← ApplicableActions(s')
7:       if A = φ then
8:          break
9:       end if
10:      a ← UniformlyRandomSelectFrom(A)
11:      s' ← apply(s', a)
12:      if s' satisfies G then
13:         return s'
14:      end if
15:   end for
16:   if h(s') < h_min then
17:      s_min ← s'
18:      h_min ← h(s')
19:   end if
20: end for
21: if s_min = NULL then
22:    return s
23: else
24:    return s_min
25: end if
```

3.2 Adapting the Length and Number of Random Walks

Random Walk Length

Each run of the Random Walk Algorithm from a new start state uses an *initial length bound*, and successively extends it by *iterative deepening*, similar to the way probes are extended in Identidem. If the best seen h-value does not change quickly enough, the length bound is increased and the sample space iteratively expands. If the algorithm encounters better states frequently enough, the length bound remains unchanged.

Number of Random Walks

The *acceptable progress* mechanism stops exploration if a state with small enough h-value is reached. This algorithm is much more efficient than using a fixed number of walks.

Let h_{min} be the minimum h-value encountered so far. The *progress* of walk n measures the decrease (if any) in h_{min} caused by this walk: $P(n) = \max(0, h^{old}_{min} - h_{min})$. As soon as *progress* exceeds an *acceptable progress* threshold, exploration is stopped and the corresponding state is immediately selected. *Acceptable progress*, $AP(n)$, is defined as a weighted combination of *progress* made in earlier search steps, $AP(1) = P(1)$, $AP(n+1) = (1-\alpha)AP(n)+\alpha P(n)$, with a parameter α, $0 \leq \alpha \leq 1$. Higher α values put more emphasis on recent *progress*.

The algorithm requires several parameters such as α and the length extension schedule for random walks. Fortunately, fixed settings for these parameters, as used in the experiments in Section 5, seem to work well.

4 MDA and MHA: Using Online Statistical Action Value Estimates

Pure random walks work poorly in two types of planning problems. First, in problems with a high density of dead-end states, most of the walks are aborted and fail to yield new evaluated end states. The other source of trouble are problems with a large average branching factor of 1000 or more. If most actions are detrimental in such a search space, almost all walks will fail to find a better state. These types of situations are generally challenging for local search methods. Most planners use pruning techniques to avoid exploring all possible actions. A popular technique limits the search to *preferred operators* [Helmert, 2006], which are believed to likely lead to progress. Such operators can sometimes be found in the process of computing a heuristic evaluation, at no significant additional cost. The best known example of preferred operators are *helpful actions* from the relaxed planning graph. They are used in planners such as FF, YAHSP, Marvin and Identidem.

4.1 Gibbs Sampling for Probabilistic Action Selection

The Monte-Carlo Deadlock Avoidance (MDA) and Monte-Carlo with Helpful Actions (MHA) enhancements address the problems of MRW planning with dead-end states and large branching factors. Their common basic idea is to extract more information than just a single heuristic value from the sampled states, and use this information to improve future random walks. This enhancement is inspired by the history-based techniques in games, such as the history heuristic [Schaeffer, 1989] and rapid action value estimate [Gelly and Silver, 2008]. Both MDA and MHA continuously update an action value $Q(a)$ for each possible action a, and use these values to bias random walks by Gibbs sampling [Finnsson and Björnsson, 2008]: The probability $P(a, s)$ that action a is chosen among all applicable actions $A(s)$ in state s is set to

$$P(a, s) = \frac{e^{Q(a)/\tau}}{\sum_{b \in A(s)}^{n} e^{Q(b)/\tau}} \quad (1)$$

The parameter τ stretches or flattens the probability distribution. MDA and MHA compute $Q(a)$ differently.

4.2 MDA: Monte-Carlo Deadlock Avoidance

MDA tries to avoid dead-end states by penalizing actions that appear in failed walks. Let $S(a)$ and $F(a)$ be the number of

Domain	ARVAND		FF	Marvin	YAHSP	SGPlan
	Average	10 runs				
Airport(50)	85	92	74	74	72	86
Satellite(36)	92	92	100	83	100	83
Tankage(50)	86	92	46	40	86	66
NoTankage(50)	94	100	76	54	100	100
Opt.Telegraph(48)	10	10	27	19	27	29
Philosophers(48)	81	94	31	63	60	60
PSR-Small(50)	100	100	96	82	96	94
PSR-Large(50)	44	48	-	16	-	22

Table 1: Percentage of tasks solved. "Average" results are over ten runs. "10 runs" data indicates solved at least once in ten runs. Total number of tasks shown in parentheses after each domain name.

successful and failed random walks that contained action a, respectively. Then set $Q(a) = 0$ if $F(a) + S(a) = 0$, and $Q(a) = -F(a)/(S(a) + F(a))$ otherwise.

4.3 MHA: Monte-Carlo with Helpful Actions

MHA defines $Q(a)$ through helpful actions. Ideally, helpful actions would be computed at each step of each random walk. For efficiency, MHA only computes them at endpoints as a byproduct of the heuristic evaluation. $Q(a)$ counts how often action a was a helpful action at an evaluated endpoint in all random walks so far.

Both MDA and MHA are used as fall-back strategies in ARVAND. The planner always starts with pure random walks, and starts using one of these enhancements only when a threshold is exceeded. MDA is used whenever more than 50% of random walks hit a dead-end. MHA is switched on if the average branching factor exceeds 1000.

5 Experiments

ARVAND is built on top of Fast Downward (FD) [Helmert, 2006]. FD contains an efficient successor generator function and an implementation of the FF heuristic. It supports full propositional PDDL 2.2, including ADL-style conditions and derived predicates. Since FD uses the SAS+ [Bäckström and Nebel, 1995] formalism, STRIPS problems are run through FD's translator first.

ARVAND was tested on all the supported domains from the fourth international planning competition (IPC-4): Pipesworld Tankage, Pipesworld NoTankage, Promela Optical Telegraph, Promela Dining Philosophers, Airport, Satellite, and Power Supply Restoration-PSR. This test suite contains a diverse set of problems, including those that are challenging for both EHC and MRW planning.

ARVAND is compared with FF, Marvin, YAHSP and SGPlan [Chen *et al.*, 2006]. All these planners participated in IPC-4 and are based on the FF heuristic. ARVAND and FF were run on a 2.5GHz machine. The results for the other three planners [Edelkamp, 2004] were obtained on a 3GHz machine. All tests used 1GB memory and a 30 minute time limit.[1]

The results are summarized in Table 1. For each planner/domain pair, the percentage of solved tasks is shown.

[1]The memory limit is for the planner itself. The SAS+ translation in FD required up to 1.5 GB memory for some of the largest tasks.

Domain	PURE RW		MDA		MHA	
	Avg.	10 runs	Avg.	10 runs	Avg.	10 runs
Satellite(36)	81	83	-	-	92	92
Opt.Telegraphs(48)	10	10	10	10	-	-
Philosophers(48)	21	23	81	94	-	-

Table 2: Percentage of tasks solved for pure random walk, MDA and MHA in three challenging domains.

Missing entries in the table signify that a planner either does not support the domain description, or did not compete in that domain in IPC-4. The results for ARVAND are reported in two columns. In the first column the results are averaged over ten runs, and the second column shows the percentage of tasks that are solved at least once in ten runs. The ten run results can indicate the performance of a basic parallel planner that simply runs ten independent instances of ARVAND. Marvin and ARVAND used the ADL description with derived predicates in the Promela domains Optical Telegraph and Philosophers.

Over ten runs, the results for ARVAND show relatively little variation in terms of the number of solved tasks. Out of the eight tested domains, ARVAND solved the largest number of problems in four under the average measure, and is top in two more domains when given 10 tries. It is also competitive in Satellite, but lags behind in Optical Telegraph. ARVAND outperforms FF and Marvin in terms of total number of solved problems. The most challenging tests for MRW planning are the two Promela domains and Satellite. The Promela problems contain a high density of dead-end states, while big Satellite tasks have a very large branching factor.

Table 2 illustrates how MDA and MHA contribute to the performance of ARVAND in these domains. Missing entries indicate that this configuration is not used for the corresponding domain. Pure random walk without any fall-back strategy is the weakest configuration in these three domains. In both Promela domains MDA reduces the number of failed random walks by a factor of 10. This results in a huge improvement in Philosophers, but is surprisingly ineffective in Optical Telegraph. This requires further study.

Satellite is an easy domain for FF. The local minima in this domain are not too extensive. Helpful actions very effectively prune irrelevant regions of the search. MHA, which uses helpful actions in a different way, performs better than pure random walk, and solves four more problems on average in this domain.

Figure 1 compares the runtimes of ARVAND and FF in Pipesworld Tankage. For ARVAND, both the median and the minimum time over ten runs are shown. Median time is shown only if a task was solved in all ten runs. Since FF does not do any exploration, it solves the easier tasks much faster than ARVAND. However, in harder problems the benefits of exploration become apparent and ARVAND outperforms FF. For better performance on easier cases, a n-way parallel system could start one copy of FF in addition to $n - 1$ copies of ARVAND.

Regarding solution length, the focus of the research on ARVAND so far has been to improve the number of tasks that are solved. Currently, a simple post-processing algorithm is used to remove useless loops in the plan, and to prune single ac-

tions that are apparently useless. More sophisticated solution length optimization algorithms are a topic for future work. Figure 2 shows the plan length of the obtained solutions by FF, ARVAND, and YAHSP for Tankage. For ARVAND, both the average and the minimum plan length obtained over ten runs are shown. Among the evaluated planners, YAHSP and ARVAND solve the most problems in this domain. In terms of plan quality, the results are worse than FF and close to YAHSP. This pattern can also be seen in the No Tankage domain. However, in other domains YAHSP does better than ARVAND in terms of plan quality.

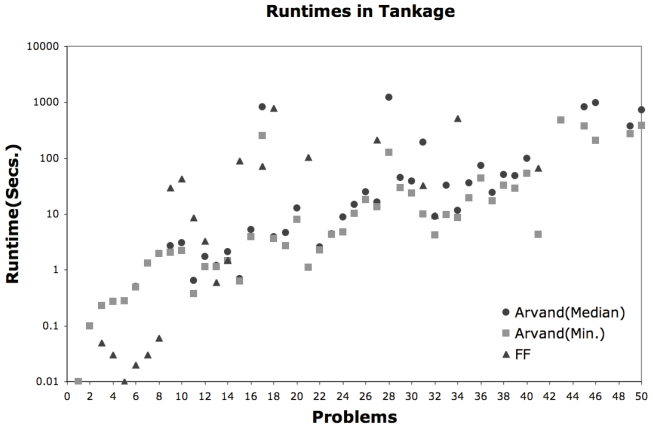

Figure 1: Runtimes of ARVAND (median and minimum over ten runs) and FF in Tankage.

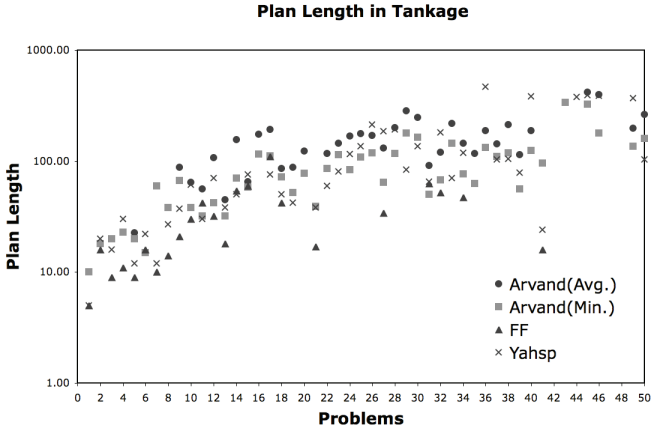

Figure 2: Plan Lengths of ARVAND (average and minimum over ten runs), YAHSP and FF in Tankage domain.

The parameter values for all configurations are shown in Table 3. All these values were determined experimentally and were fixed in all the tested domains. The parameters $EXTENDING\_PERIOD$ and $EXTENDING\_RATE$ control the length extending schedule. For example, the values of 300 and 1.5 for pure RW mean that if the best seen h-value does not improve over 300 random walks, then $LENGTH\_WALK$ is increased by a factor of 1.5.

Parameters	Pure RW	MDA	MHA
α	0.9	0.9	0.9
NUM_WALK	2000	2000	2000
$LENGTH\_WALK$	10	1	10
$EXTENDING\_PERIOD$	300	300	300
$EXTENDING\_RATE$	1.5	2	1.5
MAX_STEPS	7	7	7
τ	-	0.5	10

Table 3: Parameters used in different configurations.

Most parameter settings in Table 3 are the same for all three methods, except for length extension in MDA. In this method, most of the early random walks in problems with a high density of dead-end states are aborted. Therefore, the initial value of $LENGTH\_WALK$ is lowered to one, in order to decrease the probability of hitting a dead-end in early walks. A larger $EXTENDING\_RATE$ enables MDA to explore more in the later walks. In all configurations, setting $\alpha = 0.9$ heavily biases the *acceptable progress* towards recent progress in the search. Settings for the temperature τ reflect the fact that the average Q-values are much larger for MHA than for MDA.

6 Conclusion

This research shows how Monte-Carlo search can improve the solving power of classical deterministic planners. To the best of our knowledge, this is the first successful use of Monte-Carlo search as the main search method of a classical planner. Contrary to greedy local search methods that try to immediately exploit their local knowledge, MRW planning explores the local search space by using random walks. Many random walks are tested from a search state until either a state with *acceptable progress* is found, or the number of random walks exceeds a pre-set limit. The algorithmic benefits of the provided exploration are twofold: First, the method is more robust in presence of misleading heuristic estimates, since it obtains more information from the local neighborhood. Second, when the search is trapped in local minima, the iteratively deepening random walks can rapidly find an exit. A practical benefit of Monte-Carlo explorations is that they can be easily parallelized.

Two different techniques, MDA and MHA, use the outcome of earlier random walks to provide better guidance for exploration in the search space. Experiments showed that both techniques can be very effective in problems that are hard for pure random walks.

In the ARVAND planning system, MRW is used in conjunction with the FF heuristic. The presented results show that ARVAND outperforms FF for hard problems in most of the tested domains, and is competitive with other state of the art planning systems that are based on the FF heuristic.

7 Future Work

For future work, there are many research avenues to explore to improve MRW planning. First, ARVAND and *Identidem* should be compared in detail since they both use stochastic approaches.

We have started to investigate using the UCT algorithm in deterministic planning. UCT uses Monte-Carlo tree search and tries to balance exploration and exploitation by treating each random action selection as a bandit problem. One advantage of UCT is that it focuses on regions in the search space that look more promising.

Another idea, which recently has been used to provide fast and better exploration in motion planning, is Rapidly-exploring Random Trees (RRTs) [LaValle, 2006]. This technique gradually builds a tree that expands effectively in the search space. In each phase, first either the goal state or a randomly selected state is chosen as a target. Then, the nearest node in the current tree is extended towards this target. This algorithm has been very effective in path finding. The main challenge in using RRTs in deterministic planning is to find good enough heuristic functions that are also very fast. Each time a randomly selected state is chosen as the target, the distances of all the nodes in the tree to this new target must be estimated.

8 Acknowledgments

The authors wish to thank Malte Helmert for providing the source code for FD, and the anonymous referees for their valuable advice. This research is supported by a Provost Doctoral Entrance Award funded by the University of Alberta, and by grants from iCORE, the province of Alberta's Informatics Circle of Research Excellence, and NSERC, the Natural Sciences and Engineering Research Council of Canada.

References

[Auer *et al.*, 1995] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. Gambling in a rigged casino: The adversarial multi-arm bandit problem. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science, FOCS 1998*, pages 322–331, 1995.

[Bäckström and Nebel, 1995] Christer Bäckström and Bernhard Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11:625–656, 1995.

[Bryce *et al.*, 2006] Daniel Bryce, Subbarao Kambhampati, and David E. Smith. Sequential Monte Carlo in probabilistic planning reachability heuristics. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006*, pages 233–242, 2006.

[Chen *et al.*, 2006] Yixin Chen, Benjamin W. Wah, and Chih-Wei Hsu. Temporal planning using subgoal partitioning and resolution in SGPlan. *Journal of Artificial Intelligence Research (JAIR)*, 26:323–369, 2006.

[Coles and Smith, 2007] Andrew Coles and Amanda Smith. Marvin: A heuristic search planner with online macro-action learning. *Journal of Artificial Intelligence Research (JAIR)*, 28:119–156, 2007.

[Coles *et al.*, 2007] Andrew Coles, Maria Fox, and Amanda Smith. A new local-search algorithm for forward-chaining planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007*, pages 89–96, 2007.

[Edelkamp, 2004] Stefan Edelkamp. International planning competition: The 2004 competition, 2004. Available at `http://ls5-www.cs.tu-dortmund.de/~edelkamp/ipc-4/`, retrieved January 5, 2009.

[Fern *et al.*, 2004] Alan Fern, Sung Wook Yoon, and Robert Givan. Learning domain-specific control knowledge from random walks. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pages 191–199, 2004.

[Finnsson and Björnsson, 2008] Hilmar Finnsson and Yngvi Björnsson. Simulation-based approach to general game playing. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008*, pages 259–264, 2008.

[Gelly and Silver, 2008] Sylvain Gelly and David Silver. Achieving master level play in 9 x 9 computer Go. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008*, pages 1537–1540, 2008.

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

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

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

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *Proceedings of 17th European Conference on Machine Learning, ECML 2006*, pages 282–293, 2006.

[LaValle, 2006] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Also available at `http://ls5-www.cs.tu-dortmund.de/~edelkamp/ipc-4/`.

[Schaeffer, 1989] J. Schaeffer. The history heuristic and alpha-beta search enhancements in practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(11):1203–1212, 1989.

[Vidal, 2004] Vincent Vidal. A lookahead strategy for heuristic search planning. In Shlomo Zilberstein, Jana Koehler, and Sven Koenig, editors, *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling, ICAPS 2004*, pages 150–160. AAAI, 2004.