University of Alberta

RANDOM WALK PLANNING: THEORY, PRACTICE, AND APPLICATION

by

Hootan Nakhost

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

Fall 2013
Edmonton, Alberta

To my mother

Abstract

This thesis introduces random walk (RW) planning as a new search paradigm for satisficing planning by studying its theory, its practical relevance, and applications. We develop a theoretical framework that explains the strengths and weaknesses of random walks as a tool for heuristic search. Based on the theory, we propose a general framework for random walk search (RWS). We identify and experimentally study the key components of RWS and for each component, design and test practical and adaptive algorithms. We study resource-constrained planning as an application of RWS and show that the developed techniques implemented on top of RWS greatly outperform the state of the art in solving resource-constrained tasks. While RWS alone can lead to inefficient long plans, we introduce efficient postprocessing techniques that can significantly improve the results. We push the state of the art in planning by developing several RW planners that have strong performance in terms of both coverage and solution quality.

Acknowledgements

First of all, I would like to thank my supervisor, Martin Müller. I was truly blessed working with him. He is one of the finest people I have ever met. He is also a great researcher and supervisor. He knows well how to inspire and how to lead. While I had complete freedom to pursue my research interests, he was always available to help me out when I was confused or was not sure which direction to go next. His never-ceasing passion for knowing more taught me that satisfying curiosity is the main driving force of any good research.

People in planning community are very supportive and friendly. During ICAPS conferences I have always felt like being in a big family reunion. This friendly environment is very fruitful. I have had my most productive times during ICAPS conferences. It does not matter whether it is daytime during a talk or at night hanging out with other researchers, ideas keep flowing. I especially learned a lot collaborating with Jörg Hoffmann on the paper ”Resource-constrained planning: A Monte Carlo random walk approach”. It is also hardly imaginable that these days an AI planning thesis gets done without benefiting from Malte Helmert's great contributions to the community. I want to thank Malte Helmert not only for sharing Fast Downward codebase with us but also for his always insightful comments and encouragements.

We have a small but very active research group working on planning and heuristic search at University of Alberta. Our regular meetings have always been a great place to discuss ideas and learn new things. I would like to especially thank Xie Fan, Rick Valenzano, and Levi Lelis for being great friends and co-authors. I would like to also thank Robert Holte and Jonathan Schaeffer who, after my supervisor, had the greatest impact on determining the research direction in my thesis. I would like to also acknowledge University of Alberta and Alberta Innovates for financially supporting me.

I thank my friends Farzin Shemirani and Mohammad Reza Fayyaz whose friendship helped me to survive my loneliest and hardest times in Edmonton. I also thank my other friends, Masood Feyzbakhsh and Ormazd Bakhtiar, on the other side of the world, Iran, whose friendship had a big role in shaping who I am now.

I want to express my heartfelt thank you to my dear love, Sadaf. Her passion for life, for growth, and for happiness has changed my perspective to the world and has taught me a lot about how to live my life. Her love showed me how wonderful and enjoyable this life can be. Our relationship is the most precious thing that has ever happened to me.

Last, but not least, I thank my parents, Saeideh Zoroufchi and Alireza Nakhost, and my dear brother and friend, Hooman Nakhost, for their constant love and support during my whole life.

Table of Contents

1 Introduction **1**
1.1 Automated Planning . . . 1
1.1.1 Representation . . . 1
1.1.2 Planning Methods . . . 2
1.1.3 Planning Using Heuristic Search . . . 4
1.1.4 Search Algorithms . . . 5
1.1.5 Search Enhancements . . . 6
1.2 Contributions of this Thesis . . . 7

2 A Theoretical Framework to Study Random Walk Planning **11**
2.1 Introduction . . . 11
2.1.1 A First Example Motivating the Study of Random Walks . . . 12
2.1.2 Choice of Basic Search Algorithms . . . 12
2.1.3 Contributions . . . 13
2.2 Background and Notation . . . 14
2.2.1 Heuristic Functions, Plateaus, Exit Points and Exit Time . . . 16
2.3 Fair Homogenous Graphs . . . 17
2.3.1 Example domain: One-handed Gripper . . . 20
2.3.2 Biased Action Selection for Random Walks . . . 21
2.4 Extension to Bounds for Other Graphs . . . 22
2.5 Fair Strongly Homogeneous Graphs . . . 23
2.5.1 Analysis of the Transport Example . . . 24
2.6 Analysis of Restarting Random Walks . . . 25
2.6.1 A Grid Example . . . 29
2.7 Related Work . . . 30
2.8 Conclusion . . . 30

3 Random Walk Search: an Experimental Exploration **31**
3.1 Introduction . . . 31
3.1.1 Contributions . . . 32
3.2 Related Work . . . 33
3.2.1 RW planning . . . 33
3.2.2 Other Local Search Planners . . . 34
3.2.3 Stochastic Local Search . . . 34
3.2.4 Rapidly Exploring Random Trees . . . 34
3.3 The Experimental Framework . . . 35
3.3.1 An Artificial Domain to Control Key State Space Parameters . . . 35
3.4 Baseline: a Simple RW Planner . . . 36
3.4.1 Restarting: Parameter Study . . . 37
3.4.2 Adaptive Global Restarting . . . 38
3.4.3 Adaptive Local Restarting . . . 42
3.4.4 Comparison with Systematic Search . . . 43
3.5 The Rate of Heuristic Evaluation . . . 47
3.6 Testing Greedy vs. Delayed Jumping . . . 49
3.7 Biasing Action Selection . . . 51
3.7.1 Monte Carlo Helpful Action . . . 53
3.7.2 Monte Carlo Deadlock Avoidance (MDA) . . . 55
3.8 The Effect of the Heuristic Function on RW planning . . . 55
3.8.1 Cost Sensitivity . . . 57
3.8.2 The Effect of Changing Heuristic Function in RWS . . . 58
3.9 Arvand-2013 as a planning system . . . 60

3.9.1 Configuration Learner . . . 60
3.9.2 Experiments on All IPC Benchmarks . . . 61
3.10 Conclusions . . . 62

4 Resource-constrained Planning: a Random Walk Planning Approach 64
4.1 Introduction . . . 64
4.1.1 Contributions . . . 65
4.2 Planning with Resources . . . 66
4.2.1 RCP Formalism . . . 66
4.2.2 Previous Work . . . 66
4.2.3 IPC Benchmarks . . . 67
4.3 Resource Constrainedness . . . 67
4.3.1 Characterizing Resource Constrainedness . . . 67
4.3.2 RCP Benchmark Domains Controlling C . . . 68
4.3.3 Previous Findings when Controlling C . . . 69
4.4 Improving RW Search . . . 69
4.4.1 On-Path Search Continuation . . . 69
4.4.2 Smart Restarts . . . 70
4.5 Experiments . . . 71
4.6 Conclusion . . . 77

5 Plan Improvement Using Postprocessing 78
5.1 Introduction . . . 78
5.1.1 Contributions . . . 79
5.2 Related Work . . . 79
5.3 Two Approaches to Plan Improvement . . . 80
5.4 Action Elimination . . . 80
5.4.1 A greedy Algorithm for Action Elimination . . . 81
5.5 Plan Neighborhood Graph Search . . . 82
5.5.1 Local Search Methods for PNGS . . . 84
5.5.2 Comparison of PNGS with Related Work . . . 84
5.6 Experiments . . . 85
5.6.1 Experiment 1: Postprocessing for IPC-2008 Domains . . . 86
5.6.2 Action Elimination . . . 88
5.6.3 Experiment 2: IPC-2004 - ARAS vs LPG . . . 90
5.6.4 Integration with Random Walk Planning . . . 92
5.7 Conclusions . . . 93

6 Random Walk Planners 99
6.1 Arvand-2009: Establishing the foundation . . . 99
6.1.1 Default Configuration . . . 100
6.2 Arvand-RC: Using RW Search for RCP . . . 100
6.3 Arvand-2011: Learning the Best Configuration and Using Aras . . . 100
6.3.1 Default Configuration . . . 101
6.4 Arvand-LS: Random Walks with Memory . . . 101
6.4.1 Default Configuration . . . 102
6.5 ArvandHerd: Parallel portfolio . . . 103
6.5.1 Default Configuration . . . 103

7 Conclusions 105
7.1 Summary of Contributions . . . 105
7.2 Limitations and Open questions . . . 106
7.2.1 Theoretical Framework for RW Planning . . . 106
7.2.2 Random Walk Search Framework . . . 107
7.2.3 Resource-constrained Planning . . . 109
7.2.4 Plan Improvement . . . 109
7.2.5 Planning Systems . . . 110

Bibliography 111

List of Tables

2.1 Regress and branching factor in Gripper 20

3.1 Configurations used in Arvand-2013 61
3.2 Number of problems solved in all IPC. 63

4.1 Coverage and average runtime of Arvand-RC in IPC-2011 75

5.1 Definition of actions in a planning task example. 82
5.2 Comparing Aras with LPG 92
5.3 IPC quality score of Arvand-2013 compared with top planners 93
5.4 Problems where ARAS improved the *best known* plans. 95

6.1 Default configurations for Arvand-2009 100
6.2 The configurations used in Arvand-2011 101

List of Figures

2.1 A motivating example of superior performance of RWS 14
2.2 An illustration of the proof for Theorem 2 17
2.3 An illustration of the behaviour of random walks in Fair Homogenous graphs. . . . 18
2.4 Comparing the experimental and theoretical results in Gripper 21
2.5 An illustration of the behaviour of random walks in an IRH graph. 26
2.6 The effect of restarting rate in Grid domain 29

3.1 The effect of different settings for local and global restarting 39
3.2 h_{min} vs. the number of walks for (a) Elevators-03 and (b) Floortile-01 40
3.3 The performance of RWS using Adaptive Global Restarting (AGR) 41
3.4 h_{min} vs. the number of evaluated states for (a) Visitall-14 and (b) Elevators-05 . . 44
3.5 The performance of RWS using adaptive local restarting (ALR) 45
3.6 Coverage of RWS compared with GBFS, WA* and EHC in IPC-2011 46
3.7 The performance of RWS compared with GBFS in the artificial domain 48
3.8 The effect of different evaluation rates 50
3.9 Greedy vs. lazy jumping 52
3.10 The effect of w and T on MHA 54
3.11 The effect of T on MDA 55
3.12 Cost-sensitive vs. cost-ignorant heuristic 56
3.13 The effect of different heuristic functions 59

4.1 Coverage of planners over resource constrainedness C 73
4.2 Coverage of Arvand-RC as a function of different parameters 74

5.1 An example comparing PNGS with Related Work 85
5.2 IPC score of LAMA plus Aras using different schedules 86
5.3 IPC score of Arvand-2009 plus Aras using different schedules 87
5.4 The profile of plan cost and size of neighborhood graph for Aras 89
5.5 The profile of plan cost and expanded nodes for ITSA* 89
5.6 IPC scores of top planners using different configurations of Aras 91
5.7 IPC scores of the six tested planners combined with Aras 94

6.1 The search strategies of Arvand-2009 (left) and Arvand-LS (right). From (Xie, Nakhost, & Müller, 2012). 102

Chapter 1

Introduction

1.1 Automated Planning

Planning is one of the fundamental cognitive abilities that differentiate Homo Sapiens from other species. Anticipating the future, humans are able to plan their actions to avoid dangerous situations and achieve their goals efficiently. To plan successfully, an agent must have a basic understanding or model of the possible actions and their effects on the environment. The agent can get the model from other agents, e.g., a chess player who learns the rules from his coach, or learn it by interacting with his environment, e.g., a toddler who learns the constraints on his body movements by trial and error. Automated planning studies algorithms that, given a model of the world, generate a plan to achieve predefined goals. Success in automated planning not only facilitates the development of autonomous agents but also reduces the programming time and cost by serving as an alternative to developing problem specific solvers.

1.1.1 Representation

Regardless of how the domain model is obtained, a formal representation should be used to express the model. In a planning system a good representation facilitates the planning process and provides tools to describe the world concisely and accurately. Most of the classical planners, which are the focus of this thesis, use either STRIPS (Fikes & Nilsson, 1971) or SAS$^+$ (Bäckström & Nebel, 1995) formalisms to model a planning problem. Both formalisms are able to encode problems with state spaces which are exponentially larger than the size of the encoding itself. This compression power makes it easier to reason about problems with big state spaces. The following gives the formal definition of STRIPS representation.

Definition 1 (STRIPS)**.** *A STRIPS* planning task *is a tuple* $\Pi = (P, I, G, A, c)$ *where*

- P *is a set of propositions.*
- $I \subseteq P$ *is the* initial state*.*
- $G \subseteq P$ *is the* goal*.*

- *A is a set of* actions. *Each $a \in A$ is a triple (pre_a, add_a, del_a) of subsets of P.*
- *$c : A \to \mathbb{R}^{\geq 0}$ is a cost function assigning a cost to each action.*

A state *is identified with the set of propositions $s \subseteq P$ that are true in the state. Action a is* applicable *to s if $pre_a \subseteq s$; the result of executing a is $(s \setminus del_a) \cup add_a$. A* plan *$\pi$ is a sequence of actions that are applicable one after another starting from I, and result in a state $s \supseteq G$. The cost of the plan is the sum of the individual costs of all actions in the plan.*

While STRIPS is a *classical* representation that uses sets of propositions to model states, SAS$^+$ is a *state-variable* representation that represents a state by a tuple of values of a finite number of state variables: $[v_1, \ldots, v_n]$. For a formal definition of SAS$^+$ see (Helmert, 2006).

Any formalism used for classical planning including STRIPS and SAS$^+$ is based on a *restricted* model of the *world* including the agent and its environment (Ghallab, Nau, & Traverso, 2004). The restricted model assumes that:

- The environment is fully observable, finite and static (it only changes when the agent acts).
- Actions are discrete, instantaneous (have no duration), and deterministic (the state of the world after applying the action is known).
- States are discrete.

Although these assumptions limit the application of classical planners, they can provide useful abstractions for more general problems. For instance, (Yoon, Fern, & Givan, 2007) introduce the planner FF-replan, the top performer in the International Probabilistic Planning Competitions IPPC-2004 and IPPC-2006, that uses the classical planner FF (Hoffmann & Nebel, 2001) to solve problems with nondeterministic actions. FF-replan uses FF to find a plan for an abstract version of the problem, in which all the actions are deterministic. FF-replan executes the computed plan and replans whenever an action generates an unexpected outcome.

In spite of the simplifying assumptions in the restricted model, classical planning is hard: given a STRIPS planning problem Π and an integer $k > 0$, the decision problems corresponding to plan existence (Is Π solvable?) and plan length (does Π have a solution of length $\leq k$?) are both PSPACE-complete (Bylander, 1994). The same holds for SAS$^+$ (Helmert, 2006). Theoretically, STRIPS and SAS$^+$ representations have equivalent expressive power: one can be translated to the other with a constant factor overhead (Ghallab et al., 2004).

1.1.2 Planning Methods

Given a representation of a planning problem, what algorithms can be used to solve the problem? Most state-of-the-art planners use *state-space* search: the planner searches a state space in which each state corresponds to a state of the world and each state transition corresponds to an action of

the agent. The dominance of state-space search is mainly due to the successful development of domain-independent heuristics since (Bonet & Geffner, 2001). Alternative methods include *plan-space* planning (Penberthy & Weld, 1992), *SAT-based* planning (Kautz & Selman, 1996), and techniques based on *planning graphs* (Blum & Furst, 1997).

Plan-space search: plan-space planners search the space of partial plans: every state is a partial solution and every transition modifies the solution by adding a new action or a new ordering of actions to the partial plan. Searching the plan space can avoid wasting time permuting unnecessary orderings of actions. However, to the best of our knowledge, the heuristic functions available for plan space are not as effective as recent heuristics developed for state space.

SAT-based planners: SAT-based planners use strong SAT solvers (Eén & Sörensson, 2003; Selman, Kautz, & Cohen, 1993) to solve planning problems translated to CNF formulas. SAT-based planners operate in an iterative manner: in each iteration the SAT solver checks whether a solution of length k exists. SAT-based planners usually start with a small k, and increase it after each iteration until a solution is found. General SAT solvers are ignorant of the fact that the given problem is a planning problem.

Planning graphs: the idea of planning graphs was first used in Graphplan (Blum & Furst, 1997). A planning graph is a *layered* graph that alternates between *propositional* and *action* layers: the propositional layer 0 contains all the propositions in the initial state. The action layer $i \geq 0$ contains all the actions that have all their preconditions in propositional layer i. The propositional layer $i + 1$ contains all the propositions in layer i plus the add effects of actions in layer i. The planning graph also keeps track of *mutex* pairs at each level i: pairs of propositions that cannot both be true at level i. Mutexes are inferred from the negative interactions of actions achieving the propositions. Graphplan constructs the planning graph layer by layer until it reaches a layer containing all goal propositions. At this stage a backtracking search algorithm is used to chain the goals backward to the initial state using the information in the graph. If the process fails, when a goal proposition cannot reach the initial state without violating mutexes, then the planning graph is extended by one more layer. Graphplan plus memoization is a complete algorithm and is guaranteed to generate solutions with optimal *makespan*. For details, including the termination condition see (Blum & Furst, 1997).

Although the above techniques are not the mainstream, they are still relevant: ideas based on planning graphs are used to derive heuristics (Hoffmann & Nebel, 2001); the planner LPG (Gerevini, Saetti, & Serina, 2003), which uses local search to find a solution in the planning graph, has always been a strong alternative when state-space planners fail; and new developments in SAT-based planning are emerging that introduce planning-specific SAT solvers (Rintanen, 2012).

1.1.3 Planning Using Heuristic Search

Heuristic search is the most effective technique for state-space search. The seminal paper by Bonet and Geffner (2001) introduced two heuristics h_{add} and h_{max}, which are classified as *delete-relaxation* heuristics.

The additive and max heuristics: The delete relaxation Π^+ relaxes a STRIPS problem Π by removing the negative effects of the actions: once a proposition becomes true in a state, it remains true in all successor states. h^+, the cost of an optimal solution for Π^+, is a lower bound on the cost of any solution for Π. While the problem of computing h^+ is NP-complete, approximations such as h^{add} and h^{max} can be computed in polynomial time. Both h^{add} and h^{max} estimate the cost of reaching a set of propositions P (P can be the goal or preconditions of an action) by aggregating the cost estimates of each individual proposition $p \in P$. While h^{add} uses the sum of the estimates: $h^{add}(P) = \sum_{p \in P} h^{add}(p)$, h^{max} uses the maximum: $h^{max}(P) = \max_{p \in P} h^{max}(p)$. The cost estimate of a proposition p is computed recursively based on the cost of actions achieving p and the cost estimate of their preconditions; the cost estimate of propositions satisfied in the current state is 0. While h^{max} underestimates h^+ and can be used as an admissible heuristic, h^{add} is more informative. An implicit assumption in the h^{add} computation is that propositions are achieved independently. This leads to over-counting of actions that can be used to achieve more than one proposition.

The fast forward heuristic: Hoffmann and Nebel (2001) address the over-counting issue by explicitly building a solution, a *relaxed plan*, for Π^+. The emerging heuristic is the venerable h^{FF}. The computation of h^{FF} has two phases: in the first phase the *relaxed planning graph*, a planning graph without mutexes, is built. In the second phase a plan is extracted from the relaxed planning graph. Unlike Graphplan, no backtracking is needed: in absence of mutexes, every plan that satisfies the goal suffices. The explicit plan construction helps FF to avoid over-counting by reusing actions that are already in the plan. The cost or length of the plan is used as the heuristic function.

Causal graph and Context enhanced additive heuristics: Delete-relaxation heuristics by nature ignore the negative interactions between actions. The *causal graph* heuristic h^{CG} (Helmert, 2006) and its successor, the *context enhanced additive* heuristic h^{cea} (Helmert & Geffner, 2008) do not have this limitation. Instead of relaxing the problem by ignoring all negative effects, h^{CG} relaxes the problem by implicitly forming subtasks and overlooking some of the interactions between them. h^{CG} relies on two high-level representations, Causal Graphs (CG) and Domain Transition Graphs (DTG), that are built on top of SAS$^+$. While CG explicitly expresses the causal relations between state variables, DTG shows how a single variable transitions between different values. The heuristic function operates on these constructs to form effective subtasks. h^{CG} assumes that no circular causal relation exists between state variables: this holds, for example, in a transportation domain in which the actions that affect the location of goods have preconditions on the location of vehicles but not vice versa. If CG contains circular causal relations, then h^{CG} relaxes the problem

by breaking the circles. h^{cea} is a generalization of h^{CG} that works with circular causal relations without breaking them. While h^{CG} is explained procedurally, h^{cea} is defined mathematically using recurrence relations. This formal definition reveals that there is a relation between h^{cea} and h^{add}: if all the state variables are binary, then h^{cea} is the same as h^{add}.

The landmark heuristic: *landmarks* (Hoffmann, Porteous, & Sebastia, 2004) are propositional formulas that must be satisfied at least once in every valid plan. Richter and Westphal (2010) were the first who used landmarks to derive heuristic estimates. The heuristic function is called h^{LM}. Unlike its predecessors, h^{LM} is a path-sensitive heuristic: the value of the heuristic function depends both on the evaluated state s and the path reaching s. The heuristic estimates the number of landmarks that need to be achieved before reaching the goal. While h^{LM} alone is weaker than other heuristic functions such as h^{FF} and h^{cea}, using it in a *multi-heuristic* setting with other heuristic functions is the main reason that LAMA (Richter & Westphal, 2010) outperformed every other planner in IPC-2011 and IPC-2008.

1.1.4 Search Algorithms

Aside from the heuristic function, the search algorithm itself also plays a key role in the performance of a planner. Most of the algorithms that serve as the search engine of top planners are categorized as *best first search* (BFS).

Best first search: BFS uses an evaluation function $f(.)$ to rank the states. The underlying principle is to always expand the state s that has the lowest $f(s)$, hence the name. BFS uses a priority queue to keep states ordered. The evaluation function is the key factor differentiating BFS algorithms. In A*, for example, $f(s) = g(s) + h(s)$, where $g(s)$ is the cost of reaching s and $h(s)$ is the heuristic value of s. The most common BFS in planning is Greedy Best First Seach (GBFS) (Russell & Norvig, 2010): top planners such as LAMA, Fast Downward, and many others built on top of these planners use variations of enhanced GBFS to at least find the first solution. GBFS only uses the heuristic values to rank the states: $f(s) = h(s)$. The idea is to move towards a goal as quickly as possible with no concern about solution quality. After finding the first solution using GBFS, most of the current planners such as LAMA switch to a more conservative search algorithm such as WA*, another BFS algorithm with ranking function $f(s) = g(s) + w \times h(s)$ with $w \geq 1$. When $w = 1$, WA* is the same as A*. WA* achieves a trade-off between runtime and solution quality by adjusting the weight w. Recently, as the result of cross-fertilization of ideas between the heuristic search and planning communities, more elaborate search algorithms are being developed and tested on planning benchmarks (Thayer & Ruml, 2008; Thayer, Stern, Felner, & Ruml, 2012).

Local search: Local search has also been explored in the context of planning. A prominent example is Enforced Hill Climbing (EHC), used by the FF (Hoffmann & Nebel, 2001) planner. Like standard *hill climbing*, EHC iteratively expands a successor that decreases the current heuristic. Hill climbing becomes trapped in a local minimum or plateau if there is no such successor state. Once

trapped at the current state s, EHC runs a *breadth first search* to force the search out of the local minimum or plateau by finding a state s' with $h(s') < h(s)$. EHC fails if breadth first search exhaustively searches all the reachable states and finds no escape: this happens when the current state is a dead-end. FF does not use any restarting mechanism for EHC: when EHC fails, FF starts from scratch using GBFS. Another successful application of local search to planning is the planner LPG (Gerevini & Serina, 2002). The local search explores the space of partial plans inside the planning graph, which is fundamentally different from the state space used in planners such as FF and Arvand.

1.1.5 Search Enhancements

Planning systems rarely use the basic version of the search algorithms: they usually customize the algorithm for planning problems by using search enhancements. Common enhancements are *deferred evaluation*, *preferred operators* and *multi-heuristic* search: these are all the legacy of the successful planners FF and FD.

Deferred evaluation: Deferred evaluation delays the heuristic evaluation of a node until it is expanded: in absence of the heuristic value, the parent's heuristic value is used to rank the nodes in the priority queue. The idea is to save time by just evaluating the states that are expanded: this pays off when heuristic computation is very costly, the branching factor is large, and only a small portion of generated states gets expanded.

Preferred operators: As mentioned earlier, heuristic functions such as h^{FF} not only compute a cost estimation, they also find an actual solution plan for the relaxed problem. Hoffmann and Nebel (2001) showed that information other than just the length or cost of the relaxed plan can be used to improve the search: they limit EHC to only consider actions that at least achieve one proposition used in the relaxed plan. These actions are referred to as Preferred Operators (PO). Other heuristic functions such as h^{CG}, h^{cea}, and h^{LM} have adopted the same notion. Richter and Helmert (2009) refer to Preferred Operators as "actions that contribute to solving the relaxed version of the task". Helmert (2006) proposes a different and less aggressive way of using preferred operators: *multiple queues*. BFS is modified to use two queues: one only for *preferred successors*, states generated by preferred operators, and one for all the successors. While preferred successors go into both queues, non-preferred successors only go into the second queue. For expansion, BFS alternates between the two queues. Since the number of preferred operators is usually much lower than the total number of applicable actions, preferred successors have a higher chance to be expanded. For a detailed study of the performance of preferred operators and its interaction with deferred evaluation, see (Richter & Helmert, 2009). Beyond preferred operators, the planners YAHSP (Vidal, 2004) and Macro-FF (Botea, Müller, & Schaeffer, 2007) use the actions in the relaxed plan to build macro actions.

Multi-heuristic search: If multiple heuristics have different strengths and weaknesses, then they can complement each other. The common way to use multiple heuristics in BFS is to use

multiple queues (Helmert, 2006): one for each heuristic function. The search alternates between queues to select the next state. Each time a state is expanded, it is evaluated using all the heuristic functions and is inserted into all the queues. The ordering of the states in the queue is determined by the corresponding heuristic function. Therefore, each time that a state is expanded, it has the lowest heuristic value according to at least one of the heuristic functions. This approach can be seen as running multiple searches in parallel and sharing the information about visited states.

1.2 Contributions of this Thesis

This work introduces *Random Walk Search* (RWS) as an alternative to standard systematic search. Chapter 2 proposes a theoretical model (Nakhost & Müller, 2012) for comparing the performance of random walk (RW) and systematic search methods. This model gives well-founded insights into the relative strengths and weaknesses of these approaches. One main result is that in contrast to systematic search methods, for which the branching factor plays a decisive role, the performance of random walk methods is determined to a large degree by the *regress factor* (rf), the ratio between the probabilities of progressing towards and regressing away from a goal with an action. Besides rf, the other key variable affecting the average runtime of basic random walks on a graph is the *largest goal distance* (D) in the whole graph, which appears in the exponent of the expected runtime. For large values of D, *restarting random walks* (RRW) can offer a substantial performance advantage. At each search step, with probability r a RRW restarts from a fixed initial state s. It is shown that the expected runtime of RRW depends only on the goal distance of s, not on D. In addition to the work already presented in (Nakhost & Müller, 2012), the analysis in this chapter is extended to search spaces that contain dead ends.

Based on the theory developed in Chapter 2 and detailed experiments, Chapter 3 proposes a general framework for random walk planning (Nakhost & Müller, 2013). The approach is to build a random walk planner from scratch, identifying and exploring alternative designs and key questions regarding random walk search along the way. Six major insights are:

1. Adjusting the restarting parameter according to the progress speed in the search space performs better than any fixed setting.

2. A high state evaluation frequency is usually superior to the endpoint-only evaluation used in earlier systems.

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.

4. Random walks are beneficial using a wide range of heuristic functions.

5. Even simple forms of random walk planning can compete with systematic search.

6. Random walk search scales better than GBFS when the heuristic accuracy decreases.

The resulting planner, Arvand-2013, is a very competitive system which performs at the same level as top planners such as LAMA (Richter & Westphal, 2010). In 33 out of 45 *International Planning Competition* (IPC) domains, Arvand-2013 achieves the largest coverage and in 5 domains, it is the single winner of the domain.

Chapter 4 studies the application of random walk planning to resource constrained planning (Nakhost, Hoffmann, & Müller, 2012). The need to economize limited resources, such as fuel or money, is a ubiquitous feature of planning problems. If the resources cannot be replenished, the planner must make do with the initial supply. It is then of paramount importance how constrained the problem is, i.e., whether and to which extent the initial resource supply exceeds the minimum need. While there is a large body of literature on numeric planning and planning with resources, such resource constrainedness has only been scantily investigated. We start to address this in more detail. We generalize the previous notion of resource constrainedness, characterized through a numeric problem feature $C \geq 1$, to the case of multiple resources. We implement an extended benchmark suite controlling C. We conduct a large-scale study of the current state of the art as a function of C, highlighting which techniques contribute to success. We introduce two new techniques on top of the random walk planner Arvand-2009, resulting in a new planner Arvand-RC, that, in these benchmarks, outperforms previous planners when resources are scarce (C close to 1).

Compared with systematic search, random walk search can solve much harder problems but may produce overly costly and long plans. Chapter 5 proposes a simple but effective method for plan improvement: *Plan Neighborhood Graph Search* (PNGS) (Nakhost & Müller, 2010). This method finds a new, shorter plan by creating a plan neighborhood graph $PNG(\pi)$ of a given plan π, and then extracts a shortest path from $PNG(\pi)$. Experiments show that PNGS combined with *Action Elimination* (Nakhost & Müller, 2010; Fink & Yang, 1992) significantly improves not only random walk planners, but also top systematic search planners such as LAMA. The Aras postprocessor implementing these methods improves the *best known* results for 62 problems used in IPC-2008. Section 5.6.4 shows that Arvand-2013 integrated with an improved version of Aras (Nakhost, Müller, Valenzano, & Xie, 2011) performs as well as top systematic search planners such as FDSS2 (Helmert, Röger, & Karpas, 2011) and achieves the highest IPC score in 4 out of 14 IPC-2011 domains.

During this study of random walk search several random walk planners have been developed: Arvand-2009 (Nakhost & Müller, 2009), Arvand-LS (Xie et al., 2012), Arvand-RC (Nakhost et al., 2012), ArvandHerd (Valenzano, Nakhost, Müller, Schaeffer, & Sturtevant, 2012), Arvand-2011 (Nakhost et al., 2011) and Arvand-2013. Chapter 6 briefly introduces each planner and discusses how each fits in the general framework developed in Chapter 3.

This thesis contains material from, and extends, the following publications:

- Nakhost, H., & Müller, M. (2013). Towards a second generation random walk planner: an

experimental exploration. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI 2013*, to appear. [Chapter 3]

  - A journal version of this paper is prepared and will be submitted to JAIR.

- Nakhost, H., & Müller, M. (2012). A theoretical framework to study random walk planning. In *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS 2012*, 2012. [Chapter 2]

  - A journal version is submitted to AI Communications.

- Nakhost, H., Hoffmann, J., & Müller, M. (2012). Resource-constrained planning: A Monte Carlo random walk approach. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012*, pp. 181–189, 2012. [Chapter 4 and Section 6.2]

  - I designed the algorithms, implemented all the code for the search algorithms and the problem generators. I also ran all the experiments. Jörg Hoffmann helped on designing the experiments and preparing the papers.

- Xie, F., Nakhost, H., & Müller, M. (2012). Planning via random walk-driven local search. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012*, pp. 315–322, 2012. [Section 6.4]

  - I collaborated on developing the algorithm, designing the experiments, and preparing the paper. Fan Xie implemented all the code and ran all the experiments.

- Valenzano, R., Nakhost, H., Müller, M., Schaeffer, J., & Sturtevant, N. (2012). ArvandHerd: Parallel planning with a portfolio. In *Proceedings of the Twentieth European Conference on Artificial Intelligence, ECAI 2012*, pp. 113–116, 2012. [Section 6.5]

  - I contributed to the algorithm design. I also implemented the Arvand part of the system and provided support to run experiments: developed the system that runs the experiments. Rick Valenzano implemented the parallel code, ran the experiments and did the most of the writing.

- Valenzano, R., Nakhost, H., Müller, M., Schaeffer, J., & Sturtevant, N. (2011). ArvandHerd: Parallel planning with a portfolio. In *The 2011 International Planning Competition, IPC 2011*, Universidad Carlos III de Madrid, pp. 15–16, 2011. [Section 6.5]

- Nakhost, H., Müller, M., Valenzano, R., & Xie, F. (2011). Arvand: the art of random walks. In *The 2011 International Planning Competition, IPC 2011*, Universidad Carlos III de Madrid, pp. 15–16. [Section 6.3]

– I designed and implemented almost all the algorithms in Arvand-2011. Rick Valenzano helped me in designing the parameter learning system. He also implemented a new command processing system for Arvand. Fan Xie worked on ideas to use UCT for planning. I prepared the paper.

- Nakhost, H., Hoffmann, J., & Müller, M. (2010). Improving local search for resource-constrained planning. Extended abstract in *Proceedings of the Third Annual Symposium on Combinatorial Search, SOCS 2010*, pp. 81–82, 2010. [Chapter 4 and Section 6.2]

- Nakhost, H., & Müller, M. (2010). Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling, ICAPS 2010*, pp. 121–128, 2010. [Chapter 5]

- Nakhost, H., & Müller, M. (2009). Monte-Carlo exploration for deterministic planning. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence, IJCAI 2009*, pp. 1766–1771, 2009. [Section 6.1]

Chapter 2

A Theoretical Framework to Study Random Walk Planning

This chapter proposes a formal framework for comparing the performance of random walk and systematic search methods. *Fair homogenous* and *infinitely regressable homogenous* graphs are proposed as graph classes that represent characteristics of the state space of prototypical planning domains, while still allowing a theoretical analysis of the performance of both random walk and systematic search algorithms. This gives well-founded insights into the relative strength and weaknesses of the approaches. The close relation of the models to some well-known planning domains is shown.

2.1 Introduction

Random walks, which are paths through a search space that follow successive randomized state transitions, are a main building block of prominent search algorithms such as Stochastic Local Search techniques for SAT (Selman, Levesque, & Mitchell, 1992; Pham, Thornton, Gretton, & Sattar, 2008) and Monte Carlo Tree Search in game playing and puzzle solving (Gelly & Silver, 2008; Finnsson & Björnsson, 2008; Cazenave, 2009).

While the success of RW methods in related research areas such as SAT and Monte Carlo Tree Search serves as a good general motivation for trying them in planning, it does not provide an explanation for why RW planners perform well. Previous work has highlighted three main advantages of random walks for planning:

- Random walks are more effective than systematic search approaches for escaping from regions where heuristics provide no guidance (Coles, Fox, & Smith, 2007; Nakhost & Müller, 2009; Lu, Xu, Huang, & Chen, 2011).
- Increased sampling of the search space by random walks adds a beneficial *exploration* component to balance the *exploitation* of the heuristic in planners (Nakhost & Müller, 2009).

- Combined with proper *restarting* mechanisms, random walks can avoid most of the time wasted by systematic search in dead ends. Through restarts, random walks can rapidly back out of unpromising search regions (Coles et al., 2007; Nakhost et al., 2012).

These explanations are intuitively appealing, and give a qualitative explanation for the observed behavior on IPC benchmarks. Typically, planners are evaluated by measuring their coverage, runtime, or plan quality in such benchmarks. To give a deeper understanding of RW planning, this chapter provides a theoretical analysis of how RW and other search algorithms behave on idealized classes of planning problems which are amenable to such analysis. The main goal is a careful theoretical investigation of the first advantage claimed above - the question of how RW manage to escape from plateaus faster than other planning algorithms.

2.1.1 A First Example Motivating the Study of Random Walks

As an example, consider the following well-known plateau for the FF heuristic, h^{FF}, discussed in (Helmert, 2004). This heuristic estimates the goal distance by solving a relaxed planning problem in which all the negative effects of actions are ignored. Consider a transportation domain in which trucks are used to move packages between n locations connected in a single chain $c_1, \cdots, c_n$. The goal is to move one package from c_n to c_1. Figure 2.1 shows the results of a basic scaling experiment on this domain with $n = 10$ locations, varying the number of trucks T from 1 to 20. All trucks start at c_2. The results compare basic *Random Walk Search* (RWS) from Arvand-2009 (Chapter 6) and basic Greedy Best First Search (GBFS) from LAMA-2011. Figure 2.1 shows how the runtime of GBFS grows quickly with the number of trucks T until it exceeds the memory limit of 64 GB. This is expected since the effective branching factor grows with T. However, the increasing branching factor has only little effect on RWS: the runtime grows only linearly with T.

2.1.2 Choice of Basic Search Algorithms

All the examples in this chapter use state of the art implementations of basic, unenhanced search methods. GBFS as implemented in LAMA-2011 represents systematic search methods, and the RWS implementation of Arvand-2009 represents random walk methods. Both programs use h^{FF} for their evaluation. All other enhancements such as preferred operators in LAMA and Arvand, multi-heuristic search in LAMA, and MHA in Arvand are switched off.

The reasons for selecting this setup are:

1. A focus on theoretical models that can explain the substantially different behavior of random walk and systematic search methods. Using simple search methods allows a close alignment of experiments with theoretical results.

2. Enhancements may benefit both methods in different ways, or be only applicable to one method, so may confuse the picture.

3. A main goal here is to understand the behavior of these two search paradigms in regions where there is a lack of guiding information, such as plateaus. Therefore, in some examples even a *blind* heuristic, which only distinquishes between goal and non-goal states, is used. While enhancements can certainly have a great influence on search parameters such as branching factor, regress factor, and search depth, the fundamental differences in search behavior will likely persist across such variations.

2.1.3 Contributions

This chapter improves and extends the results reported at the SOCS 2012 conference (Nakhost & Müller, 2012). The main contributions are:

Regress factor and goal distance for random walks: The key property introduced to analyze random walks is the *regress factor* rf, the ratio of two probabilities: *progressing* towards a goal and *regressing* away from it. Besides rf, the other key variable affecting the average runtime of basic random walks on a graph is the *largest goal distance* D in the whole graph, which appears in the exponent of the expected runtime.

Fair Homogenous graph model: In the *homogenous graph* model, the regress factor of a node depends only on its goal distance and in a *fair* graph a random step changes the goal distance at most by one unit. Theorem 3 shows that the runtime of RW mainly depends on rf. As an example, the state space of Gripper (Long et al., 2000) is close to a fair homogenous graph.

Bounds for other graphs: Theorem 4 extends the theory to compute upper bounds on the expected runtime for graphs which are not homogeneous, but for which bounds on the progress and regress chances are known.

Strongly homogenous graph model: In *strongly homogenous graphs*, almost all nodes share the same rf. Theorem 5 explains how *rf* and D affect the hitting time. A transport example is used for illustration.

Model for Restarting Random Walks: For large values of D, *restarting random walks* (RRW) can offer a substantial performance advantage. At each search step, with probability r a RRW restarts from a fixed initial state s. Theorem 6 gives the expected runtime of RRW on Homogenous Graphs, relaxing the fairness condition. Furthermore, Theorem 7 proves that the expected runtime of RRW depends only on the goal distance of s, not on D.

Extension to infinitely regressable and non-fair graphs: In *infinitely regressable graphs* and *non-fair* graphs, a random step can arbitrarily increase the goal distance. The main contributions here are Lemma 2 and Theorem 6.

Compared to the conference version, the current chapter introduces the extension to infinitely regressable and non-fair graphs. It also contributes more elegant, simpler proofs of Lemma 1 and Theorem 4.

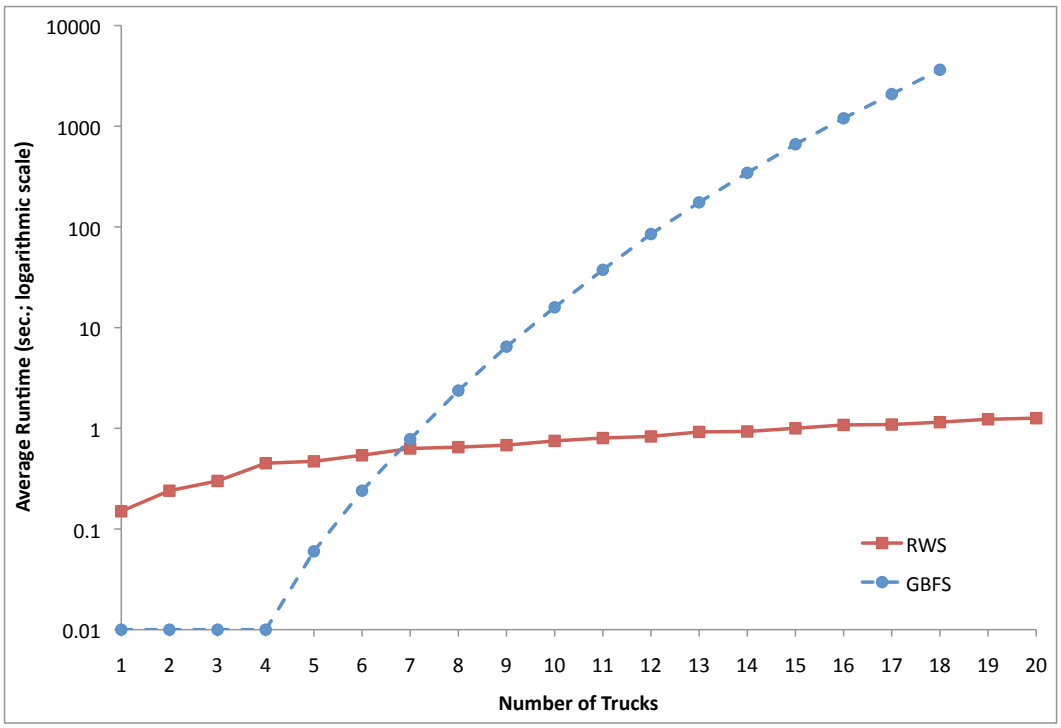


Figure 2.1: Average runtime of GBFS and MRW varying the number of trucks (x-axis) in Transport domain. Missing data means memory limit exceeded.

2.2 Background and Notation

Notation follows standard references such as (Norris, 1998). Throughout this chapter, the notation $P(e)$ denotes the probability of an event e occurring, $G = (V, E)$ is a directed graph, and $u, v \in V$ are vertices.

Definition 2 (Markov Chain). *The discrete-time random process $X_0, \ldots, X_N$ where each X_i, $0 \leq i \leq N$, is a random variable defined over a set of states S is $Markov(S, \mathbb{P})$ iff $P(X_n = j_n | X_{n-1} = j_{n-1}, \ldots, X_0 = j_0) = P(X_n = j_n | X_{n-1} = j_{n-1})$. In the matrix $\mathbb{P}(p_{ij})$, $p_{ij} = P(X_n = j_n | X_{n-1} = i_{n-1})$ are the transition probabilities of the chain. In time-homogenous Markov chains as used in this thesis, $\mathbb{P}$ does not depend on n.*

Definition 3 (Distance d_G). *$d_G(u, v)$ is the length of a shortest path from u to v in G. The distance $d_G(v)$ of a single vertex v is the length of a longest shortest path from a node in G to v: $d_G(v) = max_{x \in V} d_G(x, v)$.*

Definition 4 (Successors). *The successors of $u \in V$ is the set of all vertices in distance 1 of u: $S_G(u) = \{v | v \in V \wedge d_G(u, v) = 1\}$.*

Definition 5 (Random Walk). *A random walk on G is a Markov chain $Markov(V, \mathbb{P})$ where $p_{uv} = \frac{1}{|S_G(u)|}$ if $(u, v) \in E$, and $p_{uv} = 0$ if $(u, v) \notin E$.*

The *restarting random walk* model used here is a random walk which *restarts* from a fixed initial state s with probability r at each step, and uniformly randomly chooses among neighbour states with probability $1 - r$.

Definition 6 (Restarting Random Walk)**.** *Let $s \in V$ be the initial state, and $r \in [0, 1]$. A restarting random walk $RRW(G, s, r)$ is a Markov chain M_G with states V and transition probabilities p_{uv}:*

$$p_{uv} = \begin{cases} \frac{1-r}{|S_G(u)|} & \text{if } (u,v) \in E, v \neq s \\ r + \frac{1-r}{|S_G(u)|} & \text{if } (u,v) \in E, v = s \\ 0 & \text{if } (u,v) \notin E, v \neq s \\ r & \text{if } (u,v) \notin E, v = s \end{cases}$$

A RW is the special case of RRW with $r = 0$.

Definition 7 (Hitting Time)**.** *Let $M = X_0, X_1, \ldots, X_N$ be $Markov(S, \mathbb{P})$, and $u, v \in S$. Let $H_{uv} = min\{t \geq 0 : X_t = v \wedge X_0 = u\}$. Then the hitting time h_{uv} is the expected number of steps in a random walk on G starting from u which reaches v for the first time: $h_{uv} = E[H_{uv}]$. Therefore, $h_{vv} = 0$.*

To keep analyses simple, this study focuses on graphs with single goal vertex $v \in V$. However, the obtained results can be easily used for problems with a set of goals $X \subseteq V$. Let the hitting time $h_u(X)$ be the average number of steps until a vertex $x \in X$ is reached for the first time and G' be a modified version of G in which each goal node $x \in X$ has a single outgoing edge to a newly added dummy vertex g. Then $h_u(X) = h_{ug} - 1$.

Definition 8 (Unit Progress Time)**.** *The unit progress time u_{uv} is the expected number of steps in a random walk after reaching u for the first time until it first gets closer to v. Let $R = RRW(G, s, r)$. Let $U_{uv} = min\{t \geq H_{su} : d_G(X_t, v) = d_G(u, v) - 1\}$. Then $u_{uv} = E[U_{uv}]$.*

Definition 9 (Progress, Regress, Infinite Regress and Stalling Chance; Regress Factor)**.** *Let $X : V \rightarrow V$ be a random variable with the following probability mass function:*

$$P(X(u) = v) = \begin{cases} \frac{1}{|S_G(u)|} & \text{if } (u,v) \in E \\ 0 & \text{if } (u,v) \notin E \end{cases} \tag{2.1}$$

Let X_u be short for $X(u)$. The progress chance $pc(u, v)$, regress chance $rc(u, v)$, infinite regress chance $irc(u, v)$ and stalling chance $sc(u, v)$ of u regarding v, are respectively: the probabilities of getting closer, further away, infinitely further away or staying at the same distance to v after one

random step at u.

$$pc(u,v) = P(d_G(X_u, v) = d_G(u,v) - 1)$$
$$rc(u,v) = P(d_G(X_u, v) > d_G(u,v))$$
$$irc(u,v) = P(d_G(X_u, v) = \infty)$$
$$sc(u,v) = P(d_G(X_u, v) = d_G(u,v))$$

The regress factor of u regarding v is rf$(u,v) = \frac{rc(u,v)}{pc(u,v)}$ if $pc(u,v) \neq 0$, and undefined otherwise.

In a Markov Chain, the probability transitions play a key role in determining the hitting time. In all the models considered here, the movement in the chain corresponds to moving between different goal distances. Therefore it is natural to choose progress and regress chances as the main properties.

Theorem 1. *(Norris, 1998) Let M be $Markov(V, \mathbb{P}(p_{ij}))$. Then for all $u, v \in V$ with $u \neq v$,*

$$h_{uv} = 1 + \sum_{x \in V} p_{ux} h_{xv}, \tag{2.2}$$

Theorem 2. *Let $s, u, v \in V$, $R = RRW(G, s, r)$, $V_d = \{x : x \in V \wedge d_G(x, v) = d\}$, and $P_d(x)$ be the probability of x being the first node in V_d reached by R. Then the hitting time $h_{uv} = \sum_{d=1}^{d_G(u,v)} \sum_{x \in V_d} P_d(x) u_{xv}$.*

Proof. Let the random variable X_d denote the first vertex at goal distance d from v that R reaches after visiting u (Figure 2.2 shows a schematic representation of these variables). Let $1_{\{X_d\}}(x)$ be an indicator random variable which returns 1 if $X_d = x$ and 0 if $X_d \neq x$. Then

$$H_{uv} = \sum_{d=1}^{d_G(u,v)} \sum_{x \in V_d} 1_{\{X_d\}}(x) U_{xv} \tag{2.3}$$

Since $1_{\{X_d\}}$ and U_{xv} are independent,

$$E[H_{uv}] = \sum_{d=1}^{d_G(u,v)} \sum_{x \in V_d} E[1_{\{X_d\}}(x)] E[U_{xv}]$$
$$h_{uv} = \sum_{d=1}^{d_G(u,v)} \sum_{x \in V_d} P_d(x) u_{xv}$$

□

2.2.1 Heuristic Functions, Plateaus, Exit Points and Exit Time

What is the connection between the models introduced here and plateaus in planning? Using the notation of (Hoos & Stützle, 2004), let the heuristic value $h(u)$ of vertex u be the estimated length of a shortest path from u to a goal vertex v. A *plateau* $P \subseteq V$ is a connected subset of states which

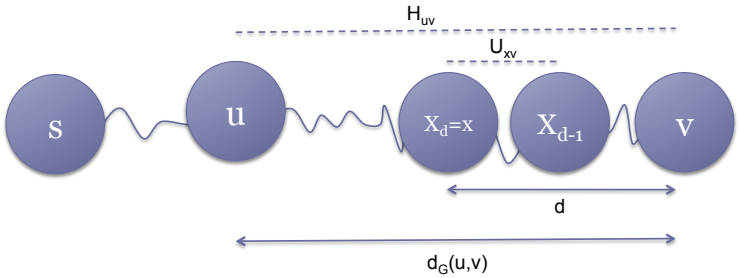


Figure 2.2: An illustration of the proof for Theorem 2. Circles represent nodes.

share the same heuristic value h_P. A state s is an *exit point* of P if $s \in S_G(p)$ for some $p \in P$, and $h(s) < h_P$. The *exit time* of a random walk on a plateau P is the expected number of steps in the random walk until it first reaches an exit point. The problem of finding an exit point in a plateau is equivalent to the problem of finding a goal in the graph consisting of P plus all its exit points, where the exit points are goal states. The expected exit time from the plateau equals the hitting time of this problem. In practice, the search time of planners is often dominated by periods spent in such attempted escapes from plateaus and local minima.

2.3 Fair Homogenous Graphs

A fair homogeneous (FH) graph G is the main state space model introduced here. *Homogenuity* means that both progress and regress chances are constant for all nodes at the same goal distance. *Fairness* means that an action can change the goal distance by at most one.

Definition 10 (Homogenous Graph)**.** *For $v \in V$, G is v-homogeneous iff there exist two real functions $pc_G(x,d)$ and $rc_G(x,d)$, mapping $V \times \{0,1,\ldots,d_G(v)\}$ to the range $[0,1]$, such that for any two vertices $u,x \in V$ with $d_G(u,v) = d_G(x,v)$ the following two conditions hold:*

1. *If $d_G(u,v) \neq 0$, then*
   $pc(u,v) = pc(x,v) = pc_G(v, d_G(u,v))$.
2. $rc(u,v) = rc(x,v) = rc_G(v, d_G(u,v))$.

G is homogeneous iff it is v-homogeneous for all $v \in V$. $pc_G(x,d)$ and $rc_G(x,d)$ are called progress chance and regress chance of G regarding x. The regress factor of G regarding x is defined by $rf_G(x,d) = rc_G(x,d)/pc_G(x,d)$.

Definition 11 (Fair Graph)**.** *G is fair for $v \in V$ iff for all $u \in V$, for all $x \in S_G(u)$, $|d_G(u,v) - d_G(x,v)| \leq 1$. G is fair if it is fair for all $v \in V$.*

Lemma 1. *Let $G = (V,E)$ be FH and $v \in V$. Then for all $x \in V$ with $d = d_G(x,v)$, h_{xv} has the same value which we call h_d.*

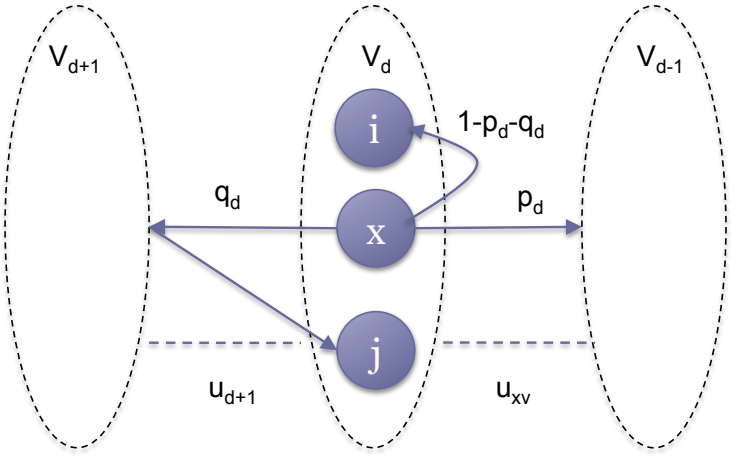


Figure 2.3: An illustration of the behaviour of random walks after visiting a node x at the goal distance d.

Proof. Let $p_d = pc_G(v,d)$, $q_d = rc_G(v,d)$, $c_d = 1 - pc_G(v,d) - rc_G(v,d)$, $D = d_G(v)$, and $V_d = \{x : x \in V \wedge d_G(x,v) = d\}$. If $d > 0$, then each $x \in V_d$ is connected to at least one node at goal distance $d-1$. Thus, $p_d > 0$. The main proof step uses induction from $d+1$ to d to show that for all $x \in V_d$, u_{xv} has the same value which we call u_d. To prove the induction step, assume for all $x' \in V_{d+1}$, $u_{x'v} = u_{d+1}$. The base case for $d = D$ will be shown at the end of the proof since it uses a similar setup as the induction step. After visiting $x \in V_d$ one of the following three cases happens for the random walk (Figure 2.3):

- with probability p_d it performs a $(d-1)$-visit.
- with probability $1 - p_d - q_d$ it stalls at the same goal distance d hitting some node $i \in V_d$.
- with probability q_d it regresses to the goal distance $d+1$ and after on average u_{d+1} step it hits some $j \in V_d$.

Therefore for $d < D$,

$$u_{xv} = q_d(u_{d+1} + u_{jv}) + (1 - p_d - q_d)u_{iv} + 1$$

The following shows for all $i, j \in V_d$, $u_{xv} = u_d$. Let $\alpha = \arg\max_{k \in V_d}(u_{kv})$ and $\beta = \arg\min_{k \in V_d}(u_{kv})$. Then,

$$\begin{aligned} u_{\alpha v} &= q_d(u_{d+1} + u_{jv}) + (1 - p_d - q_d)u_{iv} + 1 \\ &\leq q_d(u_{d+1} + u_{\alpha v}) + (1 - p_d - q_d)u_{\alpha v} + 1 \\ &\leq \frac{q_d}{p_d}u_{d+1} + \frac{1}{p_d} \end{aligned}$$

Furthermore,

$$\begin{aligned} u_{\beta v} &= q_d(u_{d+1} + u_{jv}) + (1 - p_d - q_d)u_{iv} + 1 \\ &\geq q_d(u_{d+1} + u_{\beta v}) + (1 - p_d - q_d)u_{\beta v} + 1 \\ &\geq \frac{q_d}{p_d}u_{d+1} + \frac{1}{p_d} \end{aligned}$$

Therefore,

$$\frac{q_d}{p_d}u_{d+1} + \frac{1}{p_d} \leq u_{\beta v} \leq u_{xv} \leq u_{\alpha v} \leq \frac{q_d}{p_d}u_{d+1} + \frac{1}{p_d}$$
$$u_{xv} = \frac{q_d}{p_d}u_{d+1} + \frac{1}{p_d} = u_d \tag{2.4}$$

For the base case $d = D$, for all $x \in V_D$

$$\begin{aligned} u_{xv} &= (1 - p_D)u_{iv} + 1 \\ u_{\alpha v} &\leq \frac{1}{p_D} \\ u_{\beta v} &\geq \frac{1}{p_D} \\ u_{xv} &= \frac{1}{p_D} = u_D \end{aligned} \tag{2.5}$$

The lemma now follows from Theorem 2:

$$h_{xv} = \sum_{d=1}^{d_G(x,v)} \sum_{k \in V_d} P_d(k)u_{kv} = \sum_{d=1}^{d_G(x,v)} u_d = h_d$$

□

Theorem 3. *Let $G = (V, E)$ be FH, $v \in V$, $p_i = pc_G(v, i)$, $q_i = rc_G(v, i)$, and $d_G(v) < D$. Then for all $x \in V$,*

$$h_{xv} = \sum_{d=1}^{d_G(x,v)} \left(\beta_D \prod_{i=d}^{D-1} \lambda_i + \sum_{j=d}^{D-1} \left(\beta_j \prod_{i=d}^{j-1} \lambda_i \right) \right)$$

where for all $1 \leq d \leq D$, $\lambda_d = \frac{q_d}{p_d}$, and $\beta_d = \frac{1}{p_d}$.

Proof. According to Equations 2.4 and 2.5,

$$\begin{aligned} u_d &= \lambda_d u_{d+1} + \beta_d \quad (0 < d < D) \\ u_D &= \beta_D \end{aligned}$$

By induction on d, for $d < D$

$$u_d = \beta_D \prod_{i=d}^{D-1} \lambda_i + \sum_{j=d}^{D-1} \left(\beta_j \prod_{i=d}^{j-1} \lambda_i \right) \tag{2.6}$$

Robot	Gripper	pc	rc	rf	b	d
A	full	$\frac{1}{2}$	$\frac{1}{2}$	1	1	$4\|A\|+2$
A	empty	$\frac{\|A\|}{\|A\|+1}$	$\frac{1}{\|A\|+1}$	$\frac{\mathbf{1}}{\|\boldsymbol{A}\|}$	$\|\boldsymbol{A}\|$	$4\|A\|-1$
B	full	$\frac{1}{2}$	$\frac{1}{2}$	1	1	$4\|A\|+1$
B	empty	$\frac{1}{\|B\|+1}$	$\frac{\|B\|}{\|B\|+1}$	$\|B\|$	$\|B\|$	$4\|A\|$

Table 2.1: Random walks in One-handed Gripper. $|A|$ and $|B|$ denote the number of balls in A and B.

This is trivial for $d = D - 1$. Assume that Equation 2.6 holds for $d + 1$.

$$\begin{aligned}
u_d &= \lambda_d \left(\beta_D \prod_{i=d+1}^{D-1} \lambda_i + \sum_{j=d+1}^{D-1} \left(\beta_j \prod_{i=d+1}^{j-1} \lambda_i \right) \right) + \beta_d \\
&= \beta_D \prod_{i=d}^{D-1} \lambda_i + \lambda_d \sum_{j=d+1}^{D-1} \left(\beta_j \prod_{i=d+1}^{j-1} \lambda_i \right) + \beta_d \\
&= \beta_D \prod_{i=d}^{D-1} \lambda_i + \sum_{j=d+1}^{D-1} \left(\beta_j \prod_{i=d}^{j-1} \lambda_i \right) + \beta_d \prod_{i=d}^{d-1} \lambda_i \\
&= \beta_D \prod_{i=d}^{D-1} \lambda_i + \sum_{j=d}^{D-1} \left(\beta_j \prod_{i=d}^{j-1} \lambda_i \right)
\end{aligned}$$

Then by Theorem 2 for h_{xv},

$$h_{xv} = \sum_{d=1}^{d_G(x,v)} \left(\beta_D \prod_{i=d}^{D-1} \lambda_i + \sum_{j=d}^{D-1} \left(\beta_j \prod_{i=d}^{j-1} \lambda_i \right) \right)$$

□

The largest goal distance D and the regress factors $\lambda_i = q_i/p_i$ are the main determining factors for the expected runtime of random walks in homogenous graphs.

2.3.1 Example domain: One-handed Gripper

Consider a one-handed gripper domain, where a robot must move n balls from room A to B by using the actions of picking up a ball, dropping its single ball, or moving to the other room. The states of the search space fall into four categories shown in Table 2.1. For each category the branching factor b, the progress chance pc, the regress chance rc and the regress factor rf are shown. The search space is fair homogenous: any two states with the same goal distance d have the same distribution of balls in the rooms and also belong to the same category. The graph is fair since no action changes the goal distance by more than one. The hitting time is given by Theorem 3.

Figure 2.4 plots the predictions of Theorem 3 together with the results of a scaling experiment, varying n for both random walks and greedy best first search. To simulate the behaviour of both algorithms in plateaus with a lack of heuristic guidance, a blind heuristic, where all the states except

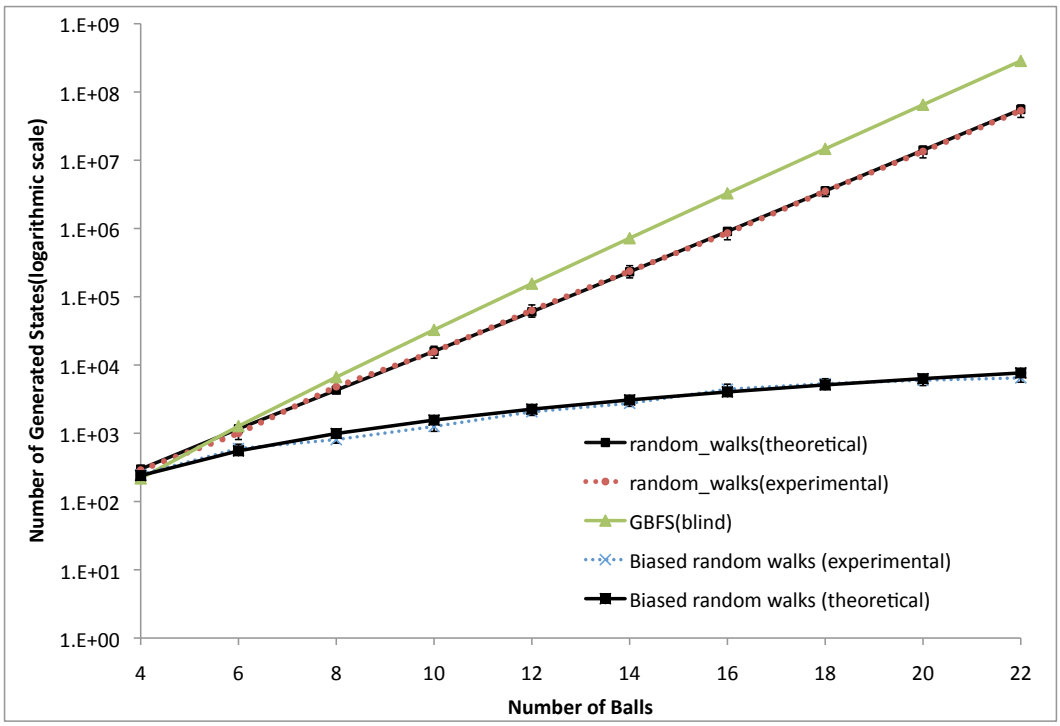


Figure 2.4: The average number of generated states varying the number of balls (x-axis) in Gripper domain.

the goals have the same heuristic value, is used which returns 0 for the goal and 1 otherwise. Search stops at a state with a heuristic value lower than that of the initial state. Because of the blind heuristic, the only such state is the goal state. The prediction matches the experimental results extremely well. Random walks outperform greedy best first search. The regress factor rf never exceeds b, and is significantly smaller in states with the robot at A and an empty gripper - almost one quarter of all states.

2.3.2 Biased Action Selection for Random Walks

Regress factors can be changed by biasing the action selection in the random walk. It seems natural to first select an action type uniformly randomly, then ground the chosen action. In gripper, this means choosing among the balls in the same room in case of the pick up action.

With this biased selection, the search space becomes fair homogenous with $q = p = \frac{1}{2}$. The experimental results and theoretical prediction for such walks are included in Figure 2.4. The hitting time grows only linearly with n. It is interesting that this natural way of biasing random walks is able to exploit the symmetry inherent in the gripper domain.

2.4 Extension to Bounds for Other Graphs

While many planning problems cannot be exactly modelled as FH graphs, these models can still be used to obtain upper bounds on the hitting time in any fair graph G which models a plateau. Consider a corresponding FH graph G' with progress and regress chances at each goal distance d respectively set to the minimum and maximum progress and regress chances over all nodes at goal distance d in G. Then the hitting times for G' will be an upper bound for the hitting times in G. In G', progressing towards the goal is at most as probable as in G.

Theorem 4. *Let $G = (V, E)$ be a fair directed graph, $s, v \in V$, and $D = d_G(v)$. Let $p_{min}(d)$ and $q_{max}(d)$ be the minimum progress and maximum regress chance among all nodes at distance d of v. Let $G' = (V', E')$ be an FH graph, $v', s' \in V'$, $d_{G'}(v') = D$, $pc_{G'}(v', d) = p_{min}(d)$, $rc_{G'}(d) = q_{max}(d)$, and $sc_{G'}(d) = 1 - p_{min}(d) - q_{max}(d)$. Then starting at the same goal distance the hitting time in G' is an upper bound for the hitting time in G, i.e., $h_{sv} \leq h'_{s'v'}$ if $d_G(s, v) = d_{G'}(s', v')$.*

Proof. The first step is to show for all $0 \leq d \leq D$, $sc_{G'}(d) \geq 0$. Let q_x and p_x be the regress and progress chance of node $x \in V$, and $V_d = \{x | x \in V \wedge d_G(x, v) = d\}$, and $j = \arg\max_{x \in V_d}(q_x)$. Then,

$$\begin{aligned} q_{max}(d) = q_j &\leq 1 - p_j \leq 1 - p_{min}(d) \\ q_{max}(d) + p_{min}(d) &\leq 1 \\ sc_{G'}(d) &\geq 0. \end{aligned}$$

Assume for all $x \in V_d$, $u_{xv} \leq u'_d$, where u'_d is the unit progress time at distance d of v'. According to Theorem 2,

$$\begin{aligned} h_{sv} &= \sum_{d=1}^{d_G(s,v)} \sum_{x \in V_d} P_d(x) u_{xv} \\ &\leq \sum_{d=1}^{d_G(s,v)} \sum_{k \in V_d} P_d(x) u'_d \\ &\leq \sum_{d=1}^{d_G(s,v)} u'_d \sum_{k \in V_d} P_d(x) \\ &\leq \sum_{d=1}^{d_{G'}(s',v')} u'_d \\ &\leq h'_d \end{aligned}$$

To prove $u_{xv} \leq u'_d$ by induction, assume for all $x' \in V_{d+1}$, $u_{x'v} \leq u'_{d+1}$ (the induction step; again the base case is shown later). After visiting $x \in V_d$ one of the following three cases happens for the random walk:

- with probability p_x it performs a $(d-1)$-visit.
- with probability q_x it regresses to the goal distance $d+1$ and, on average, after at least u_{d+1} steps it hits $i \in V_d$.
- with probability $1-p_x-q_x$ it stalls at the same goal distance d hitting $j \in V_d$.

Then for $d < D$,

$$u_{xv} \leq q_x(u_{d+1} + u_{iv}) + (1 - p_x - q_x)u_{jv} + 1.$$

The following shows that for all $i, j \in V_d$, $u_{xv} = u_d$. Let $\alpha = \arg\max_{i \in V_d}(u_{iv})$. Then for $d < D$,

$$\begin{aligned} u_{\alpha v} &\leq q_\alpha(u'_{d+1} + u_{iv}) + (1 - p_\alpha - q_\alpha)u_{jv} + 1 \\ &\leq q_\alpha(u'_{d+1} + u_{\alpha v}) + (1 - p_\alpha - q_\alpha)u_{\alpha v} + 1 \\ &\leq \frac{q_\alpha}{p_\alpha}u'_{d+1} + \frac{1}{p_\alpha} \\ &\leq \frac{q_{max}(d)}{p_{min}(d)}u'_{d+1} + \frac{1}{p_{min}(d)} \end{aligned}$$

Furthermore, according to Equation 2.4,

$$\frac{q_{max}(d)}{p_{min}(d)}u'_{d+1} + \frac{1}{p_{min}(d)} = u_d \tag{2.7}$$

Therefore, $u_{xv} \leq u_{\alpha v} \leq u_d$. Analogously, for the base case $d = D$, for all $x \in V_D$

$$u_{\alpha v} \leq \frac{1}{p_\alpha} \leq \frac{1}{p_{min}(d)} \leq u'_d.$$

□

2.5 Fair Strongly Homogeneous Graphs

A fair strongly homogenous (FSH) graph G is a FH graph in which pc and rc are constant for all nodes except goal nodes and nodes at maximum distance. FSH graphs are simpler to study and suffice to explain the main properties of FH graphs. Therefore, this model is used to discuss key issues such as dependency of the hitting time on largest goal distance D and the regress factors.

Definition 12 (Strongly Homogeneous Graph)**.** *Given $v \in V$, G is strongly v-homogeneous iff there exist two real functions $pc_G(x)$ and $rc_G(x)$ with domain V and range $[0,1]$ such that for any vertex $u \in V$ the following two conditions hold:*

1. *If $u \neq v$ then $pc(u,v) = pc_G(v)$.*
2. *If $d(u,v) < d_G(v)$ then $rc(u,v) = rc_G(v)$.*

G is strongly homogeneous iff it is strongly v-homogeneous for all $v \in V$. The functions $pc_G(x)$ and $rc_G(x)$ are respectively called the progress and the regress chance of G regarding x. The regress factor of G regarding x is defined by $rf_G(x) = rc_G(x)/pc_G(x)$.

Theorem 5. *For $u, v \in V$, let $p = pc_G(v) \neq 0$, $q = rc_G(v)$, $c = 1 - p - q$, $D = d_G(v)$, and $d = d_G(u, v)$. Then the hitting time h_{uv} is:*

$$h_{uv} = \begin{cases} \beta_0 \left(\lambda^D - \lambda^{D-d}\right) + \beta_1 d & \text{if } q \neq p \\ \alpha_0(d - d^2) + \alpha_1 Dd & \text{if } q = p \end{cases} \tag{2.8}$$

where $\lambda = \frac{q}{p}$, $\beta_0 = \frac{q}{(p-q)^2}$, $\beta_1 = \frac{1}{p-q}$, $\alpha_0 = \frac{1}{2p}$, $\alpha_1 = \frac{1}{p}$.

The proof follows directly from Theorem 3 above. When $q > p$, the main determining factors in the hitting time are the regress factors $\lambda = q/p$ and D; the hitting time grows exponentially with D and polynomially, with degree D, with λ. As long as λ and D are fixed, changing other structural parameters such as the branching factor b can only increase the hitting time linearly. Note that also for $q > p$, it does not matter how close the start state is to the goal. The hitting time mainly depends on D, the largest goal distance in the graph.

2.5.1 Analysis of the Transport Example

Theorem 5 helps explain the experimental results in Figure 2.1. In this example, the plateau consists of all the states encountered before loading the package onto one of the trucks. Once the package is loaded, h_{FF} can guide the search directly towards the goal. Therefore, the exit points of the plateau are the states in which the package is loaded onto a truck. Let $m < n$ be the location of a most advanced truck in the chain. For all non-exit states of the search space, $q \leq p$ holds: there is always at least one action which progresses towards a closest exit point - move a truck from c_m to c_{m+1}. There is at most one action that regresses, in case $m > 1$ and there is only a single truck at c_m which moves to c_{m-1}, thereby reducing m.

According to Theorem 4, setting $q = p$ for all states yields an upper bound on the hitting time, since increasing the regress factor can only increase the hitting time. By Theorem 5, $-\frac{x^2}{2p} + (\frac{2D+1}{2p})x$ is an upper bound for the hitting time. If the number of trucks is multiplied by a factor M, then p will be divided by at most M, therefore the upper bound is also multiplied by at most M. The worst case runtime bound grows only linearly with the number of trucks. In contrast, systematic search methods suffer greatly from increasing the number of vehicles, since this increases the effective branching factor b. The runtime of systematic search methods such as greedy best first search, A* and IDA* typically grows as b^d when the heuristic is ineffective. Regarding the memory usage, since RW in its simplest form only store the *current state* of the walk, increasing the number of trucks does not increase the number of states stored, however, the size of the state grows linearly.

This effect can be observed in all planning problems where increasing the number of objects of a specific type does not change the regress factor. Examples are the vehicles in transportation domains

such as Rovers, Logistics, Transport, and Zeno Travel, or agents which share similar functionality but do not appear in the goal, such as the satellites in the satellite domain. All of these domains contain symmetries similar to the example above, where any one of several vehicles or agents can be chosen to achieve the goal. Other examples are "decoy" objects which cannot be used to reach the goal. Actions that affect only the state of such objects do not change the goal distance, so increasing the number of such objects has no effect on *rf* but can increase b. Techniques such as plan space planning, backward chaining planning, preferred operators, or explicitly detecting and dealing with symmetries can often prune such actions.

Theorem 5 suggests that if $q > p$ and the current state is close to an exit point in the plateau, then systematic search is more effective, since random walks move away from the exit with high probability. This problematic behavior of RW can be fixed to some degree by using restarting random walks.

2.6 Analysis of Restarting Random Walks

While FH graphs provide bounds for RW on any fair graph, *Infinitely Regressable Homogenous* (IRH) graphs provide bounds for RRW on any strongly homogenous graph. A random step on an IRH graph either gets closer to the goal, stalls at the same goal distance or hits a dead end state – a state with no path to the goal. To keep the analysis simple, it is assumed that all nodes have at least one outgoing edge. Therefore, a RW can never reach a state where no action is available.

Definition 13 (Infinitely Regressable Homogenous Graph)**.** *Given $v \in V$, G is infinitely regressable (IR) v-homogeneous iff for any vertex $u \in V$ there exists at least one vertex x such that $(u, x) \in E$ and there exist three real functions $pc_G(.)$, $sc_G(.)$, and $irc_G(.)$ with domain V and range $[0, 1]$ such that for any vertex $u \in V$ the following three conditions hold:*

1. *If $u \neq v$ then $pc(u, v) = pc_G(v)$.*
2. *$irc(u, v) = irc_G(v)$.*
3. *$sc(u, v) = 1 - irc_G(v) - pc_G(v)$.*

G is IRH iff for any $v \in V$ it is IR v-homogeneous. The functions $pc_G(x)$, $sc_G(x)$ and $irc_G(x)$ are respectively called the progress chance, the stall chance and the infinite regress chance of G regarding x.

Lemma 2. *Let $G = (V, E)$ be an IRH graph. Let $RRW(G, s, r)$ be a restarting random walk. Then, for all $v, x, x' \in V$ with $d_G(x, v) = d_G(x', v) = d$ and $d \leq d_G(s, v)$, $h_{xv} = h_{x'v}$.*

Proof. Let $p = pc_G(v)$, $c = sc_G(v)$, and $i = irc_G(v)$. Similar to Lemma 1 by induction on the goal distance d, we show that for $d \leq d_G(s, v)$, $u_{xv} = u_d$. Let $V_d = \{x : x \in V \wedge d_G(x, v) = d\}$.

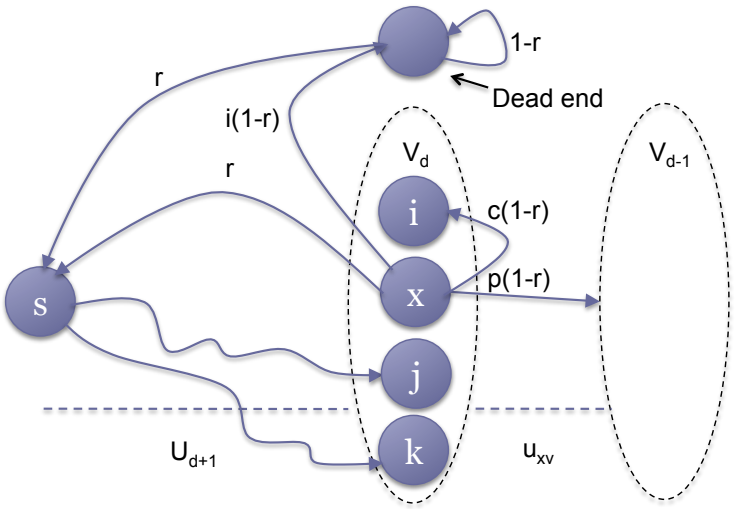


Figure 2.5: An illustration of the behaviour of random walks in an IRH graph.

Assume for the induction step that for all $x' \in V_{d+1}$, $u_{x'v} = u_{d+1}$. Once more, the proof for the base case follows later. Whenever the random walk transitions to a deadend, it restarts after on average $\frac{1}{r}$ steps (the expected value of a geometric distribution with the success probability r). After each restart a random walk needs on average $U_{d+1} = \sum_{i=d+1}^{d_G(s,v)} u_i$ steps to visit a state with the goal distance d (d-visit). Therefore, after visiting $x \in V_d$ one of the following four cases happens for the random walk (Figure 2.5):

- with probability r it restarts from s and after on average U_{d+1} steps performs the next d-visit hitting some node $n \in V_d$.
- with probability $c(1-r)$ it stalls at the same goal distance d hitting some $j \in V_d$.
- with probability $i(1-r)$ it transitions to a deadend and after on average $\frac{1}{r} + U_{d+1}$ steps it performs the next d-visit hitting some $k \in V_d$.
- with probability $p(1-r)$ it performs a $(d-1)$-visit.

Therefore, for $d < d_G(s,v)$,

$$u_{xv} = r(U_{d+1} + u_{nv}) + c(1-r)u_{jv} + i(1-r)(\frac{1}{r} + U_{d+1} + u_{kv}) + (1-r)$$

Note that restarting itself is not counted as a random walk step. The following shows that the identity of nodes n, j and k does not matter. Let $\alpha = \arg\max_{x \in V_d}(u_{xv})$ and $\beta = \arg\min_{x \in V_d}(u_{xv})$. Then,

$$\begin{aligned} u_{\alpha v} &\leq r(U_{d+1} + u_{\alpha v}) + c(1-r)u_{\alpha v} + i(1-r)(\frac{1}{r} + U_{d+1} + u_{\alpha v}) + (1-r) \\ &\leq \frac{(r + i(1-r))U_{d+1} + (1-r)(1+i/r)}{(1-r)(1-i-c)} \end{aligned}$$

Furthermore,

$$\begin{aligned} u_{\beta v} &\geq r(U_{d+1} + u_{\beta v}) + c(1-r)u_{\beta v} + i(1-r)(\frac{1}{r} + U_{d+1} + u_{\beta v}) + (1-r) \\ &\geq \frac{(r + i(1-r))U_{d+1} + (1-r)(1+i/r)}{(1-r)(1-i-c)} \end{aligned}$$

Therefore,

$$u_{xv} = u_{\alpha v} = u_{\beta v} = u_d = \frac{(r + i(1-r))U_{d+1} + (1-r)(1+i/r)}{(1-r)(1-i-c)}$$

The base case $d = d_G(s, v)$ has the same four cases, except that after restarting, the random walk immediately performs the d-visit at s:

$$\begin{aligned} u_{xv} &= ru_{sv} + c(1-r)u_{jv} + i(1-r)(\frac{1}{r} + u_{kv}) + (1-r) \\ u_{xv} &= u_{\alpha v} = u_{\beta v} = u_d = \frac{(1+i/r)}{(1-i-c)} \end{aligned}$$

The lemma now follows directly from Theorem 2:

$$h_{xv} = \sum_{d=1}^{d_G(x,v)} \sum_{k \in V_d} P_d(k) u_{kv} = \sum_{d=1}^{d_G(x,v)} u_d = h_d$$

□

Theorem 6. *Let $G = (V, E)$ be an IRH graph, $v \in V$, $p = pc_G(v) > 0$, $c = sc_G(v)$, and $i = irc_G(v)$. Let $R = RRW(G, s, r)$ with $0 < r < 1$. Then the hitting time $h_{sv} \in \Theta\left(\beta\lambda^{d_s - 1}\right)$, where $\beta = \frac{i+r}{rp}$, $\lambda = \frac{i}{p} + \frac{r}{(1-r)p} + 1$ and $d_s = d_G(s, v)$.*

Proof. According to Theorem 1 and Lemma 2,

$$\begin{aligned} h_0 &= 0 \\ h_x &= rh_{d_s} + c(1-r)h_x + i(1-r)(\frac{1}{r} + h_{d_s}) + (1-r) + ph_{x-1} \end{aligned}$$

Let $u_x = h_x - h_{x-1}$ then

$$\begin{aligned} u_x &= (1-r)\left(ph_{x-1} - ph_{x-2} + ch_x - ch_{x-1}\right) \\ &= (1-r)(pu_{x-1} + cu_x) \\ &= \frac{(1-r)p}{1 - c + cr} u_{x-1} \end{aligned}$$

Since $c = 1 - p - i$

$$\begin{aligned} u_x &= \frac{(1-r)p}{i(1-r) + p(1-r) + r} u_{x-1} \\ &= \lambda^{-1} u_{x-1} \end{aligned}$$

For $x < d_s$,

$$\begin{aligned} u_x &= \lambda^{d_s - x} u_{d_s} \\ h_x &= \sum_{i=1}^{x} u_i \\ &= u_{d_s} \sum_{i=1}^{x} \lambda^{d_s - i} \\ &= \lambda^{d_s - x} (\frac{\lambda^x - 1}{\lambda - 1}) u_{d_s} \end{aligned}$$

The value u_{d_s} is the progress time from the goal distance d_s. Therefore,

$$\begin{aligned} u_{d_s} &= r u_{d_s} + c(1-r) u_{d_s} + i(1-r)(\frac{1}{r} + u_{d_s}) + (1-r) \\ &= (r + (1-r)(1-p))\, u_{d_s} + (i/r + 1)(1-r) \\ &= \frac{i + r}{pr} \\ &= \beta \end{aligned}$$

Therefore,

$$\begin{aligned} h_{d_s} &= u_{d_s} + h_{d_s - 1} \\ h_{d_s} &= \beta + \beta\lambda(\frac{\lambda^{d_s - 1} - 1}{\lambda - 1}) \\ h_{d_s} &\in \Theta\left(\beta \lambda^{d_s - 1}\right) \end{aligned} \tag{2.9}$$

□

The next theorem shows how the results for IRH graphs can be used to derive bounds for any strongly homogeneous graph, even if it is not fair.

Theorem 7. *Let $G = (V, E)$ be a strongly homogeneous graph, $v \in V$, $p = pc_G(v) > 0$ and $q = rc_G(v)$. Let $R = RRW(G, s, r)$. The hitting time $h_{sv} \in O\left(\beta \lambda^{d-1}\right)$, where $\lambda = \left(\frac{q}{p} + \frac{r}{p(1-r)} + 1\right)$, $\beta = \frac{q+r}{pr}$ and $d = d_G(s, v)$.*

Proof. For any goal distance x, $h_x \leq \frac{1}{r} + h_d$. This is because the random walk on average restarts from s after $\frac{1}{r}$ steps. The right hand side of this inequality is the hitting time of a random walk stuck in an infinitely large dead end. Therefore, with the pessimistic assumption that each time the random walk regresses from the goal the walk is in a deadend, we can obtain an upper bound for a homogenous graph using the theorem for IRH graphs. It is enough to simply replace i with q in Equation 2.9.

□

Therefore, by decreasing r while λ decreases, β increases. Since the upper bound increases polynomially (the degree depends on $d(s, v)$) by λ and only linearly by β, to keep the upper bound

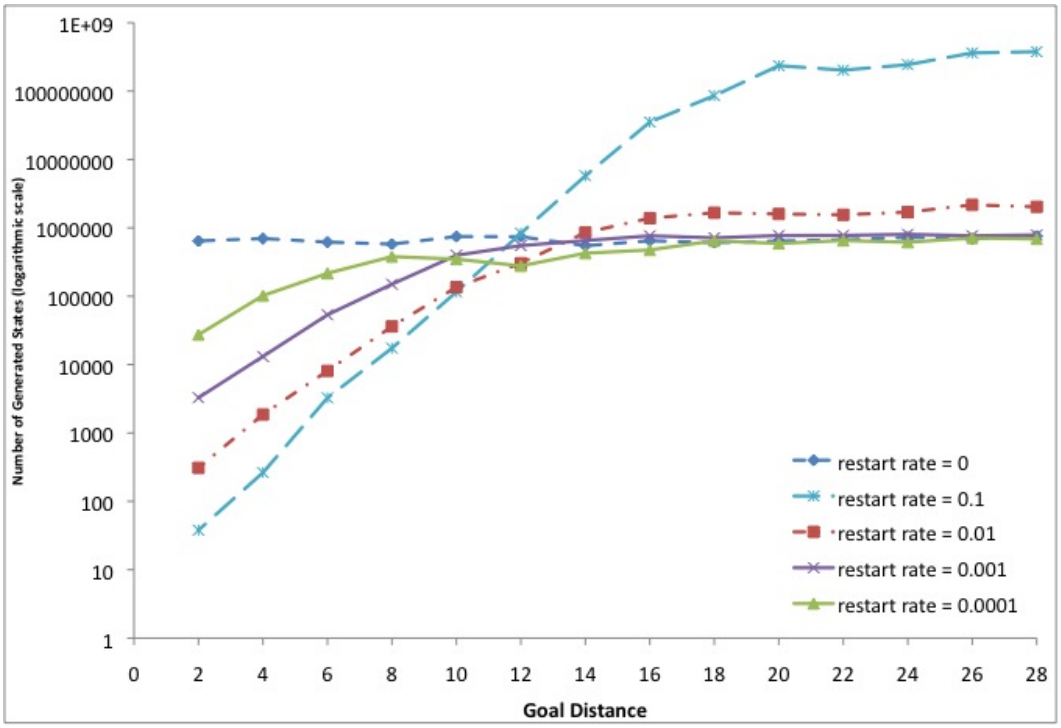


Figure 2.6: The Average number of generated states varying the goal distance of the starting state (x-axis) and the restart rate in the Grid domain.

low a small value should be chosen for r, especially when $d(s, v)$ is large. The r-value which minimizes the upper bound can be computed from Equation 2.9.

Comparing the values of λ in the hitting time of RW and RRW, Equations 2.9 and 2.8, the base of the exponential term for RRW exceeds the regress factor, the base of the exponential term for RW, by $\frac{r}{p(1-r)} + 1$. For small r, this is close to 1.

The main advantage of RRW over simple random walks is for small $d(s, v)$, since the exponent of the exponential term is reduced from D to $d(s, v) - 1$. Restarting is a bit wasteful when $d(s, v)$ is close to D.

2.6.1 A Grid Example

Figure 2.6 shows the results of RRW with restart rate $r \in \{0, 0.1, 0.01, 0.001\}$ in a variant of the Grid domain with an $n \times n$ grid and a robot that needs to first pick up a key at location (n, n), then unlock a door at $(0, 0)$. The robot can only move left, up or down, except for the top row, where it is also allowed to move right, but not up.

In this domain, all states before the robot picks up the key share the same h_{FF} value. Figure 2.6 shows the average number of states generated until this subgoal is reached, with the robot starting from different goal distances plotted on the x-axis. Since the regress factors are not uniform in this domain, Theorem 7 does not apply directly. Still, comparing the results of RRW for different

$r > 0$ with simple random walks where $r = 0$, the experiment confirms the high-level predictions of Theorem 7: RRW generates slightly more states than simple random walks when the initial goal distance is large, $d \geq 14$, and r is small enough. RRW is much more efficient when d is small; for example it generates three orders of magnitude fewer states for $d = 2$, $r = 0.01$.

2.7 Related Work

Random walks have been extensively studied in many different scientific fields including physics, finance and computer networking (Gkantsidis, Mihail, & Saberi, 2006; Fama, 1965; Qian, Nassif, & Sapatnekar, 2003). Linear algebra approaches to discrete and continuous random walks are well studied (Norris, 1998; Aldous & Fill, 2002; Yin & Zhang, 2005; Pardoux, 2009). The current chapter mainly uses methods for finding the hitting time of simple chains such as birth–death, and gambler chains (Norris, 1998). Such solutions can be expressed easily as functions of chain features.

Properties of random walks on finite graphs have been studied extensively (Lovász, 1993). One of the most relevant results is the $O(n^3)$ hitting time of a random walk in an undirected graph with n nodes (Brightwell & Winkler, 1990). However, this result does not explain the strong performance of random walks in planning search spaces which grow exponentially with the number of objects. Despite the rich existing literature on random walks, the application to the analysis of random walk planning seems to be novel.

2.8 Conclusion

The study of FH graphs provides insights regarding the key features, regress factor rf, the initial goal distance, and in case of non-restarting walks the largest goal distance D, that affect the performance of RW. Analysis of RRW shows a big advantage for restarting: the hitting time decreases with the initial goal distance. IRH graphs help analyzing the behaviour of RRW in non-fair graphs including those that have dead ends. The results also suggest that future work on adjusting the restarting rate r, and biasing techniques has a great potential.

Chapter 3

Random Walk Search: an Experimental Exploration

The current chapter proposes a general framework for random walk search and begins a systematic study of the design space and alternative choices for building a random walk planner inside the framework. What are effective ways of restarting? How should RWS explore the local neighborhood and control the distribution of samples? How to balance between computation time and the information gained in each sample? Where in the search space should the algorithm focus to sample from and how long should it continue exploring?

3.1 Introduction

The most common current technique for building satisficing planning systems is heuristic search (Bonet & 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 GBFS, WA* and Enforced Hill Climbing. These algorithms mainly exploit the evaluation function and do not explore the search space much. This lack of exploration hurts performance in case of inaccurate evaluations, which are very common with 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.

This chapter proposes random walk search as an alternative. The assumption is that random walk explorations in many cases, especially when the heuristic values are inaccurate, can reach a lower heuristic value with fewer state generations or evaluations than the standard greedy approaches. While Chapter 2 confirms this assumption for plateaus, Section 3.4.4 provides more evidence by studying the performance of RWS as a function of *heuristic accuracy*, *regress factor*, and *solution depth*, showing that RWS scales better than GBFS when heuristic accuracy decreases. The RWS framework is built based on the insights gained from the RW theory (Chapter 2) and detailed experiments exploring the design space. To keep the scope of the experiments feasible, this research only

studies algorithms that use state-space local search.

Local search has its own strengths and weaknesses that are inherited by the developed algorithms: while the search is not guaranteed to find a solution, if it finds a solution, it finds it fast; it also tends to use much less memory than alternative systematic approaches. Despite these legacies, the behaviour of the search can still vary a lot depending on the *neighborhood relation* and the *step function* (Hoos & Stützle, 2004). Local search is a repetitive process of performing *search steps* that jump from the *current state* s_i to the next state s_{i+1}, which is selected from the local neighborhood. While the neighborhood relation determines the set of neighbors, and hence, the candidates for s_{i+1}, the step function selects one of them as s_{i+1}.

To determine the neighbors, RWS samples the search space using random walks: each sample is considered as a candidate for s_{i+1}. Therefore, due to the stochastic nature of random walks, the neighborhood relation is not static. However, the distribution of the sampled states is a function of the parameters controlling the *length* of random walks, the *evaluation rate*, and the action selection *bias*: fixed settings for these parameters lead to a fixed distribution. Likewise, the step function in RW search is determined by the parameters that control *restarting* and *jumping* (restarting here is considered as a search step). While the parameters affecting the neighborhood relation determine local exploration, restarting and jumping strategies control the global exploration.

3.1.1 Contributions

The current chapter takes the reader on a journey of building a RW planner, answering key research questions regarding RW search by running careful experiments. The questions can be categorized into three categories:

1. Local exploration. What is an effective way to control the length of walks (Sections 3.4.1 and 3.4.3)? Does it pay off to not evaluate all the generated states (Section 3.5)? How to use extra information such as preferred operators to bias action selection (Section 3.7)?

2. Global exploration. Which state should be selected as the next state (Section 3.6)? How to control global restarting (Sections 3.4.1 and 3.4.2)?

3. The effect of the heuristic function. How does RW search perform in presence of different heuristic functions (Section 3.8)? How does RW search behave as a function of heuristic accuracy change (Section 3.4.4)?

Experiments are designed to answer these and other questions stemming from the unproven assumptions and design alternatives left unexplored in previous research on RW planning (Nakhost & Müller, 2009; Nakhost et al., 2012; Xie et al., 2012; Valenzano et al., 2012). Key findings are as follows:

- Adjusting the restarting parameter according to the progress speed in the search space performs better than any fixed setting.

- A high state evaluation frequency is usually superior to the endpoint-only evaluation used in earlier systems (Nakhost & Müller, 2009).
- Biasing action selection towards preferred operators of only the current state is better than Monte Carlo Helpful Actions (Nakhost & Müller, 2009), which depend on the number of times an action has been a preferred operator in previous walks.
- Random walks are beneficial using a wide range of heuristic functions.
- RW search scales better than GBFS when the heuristic accuracy decreases.
- Even simple forms of random walk planning can compete with systematic search.

3.2 Related Work

3.2.1 RW planning

Random Walk planning for classical satisficing planning started with Arvand-2009 (Nakhost & Müller, 2009). Arvand-2009 uses forward chaining local search that starts from the initial state. The steps in the local search form a sequence of actions connecting the current state to the next state. At each step, Arvand-2009 uses bounded random walks to sample the local neighborhood in order to determine the next state. After sampling n states, the search transitions (jumps) to the sampled state with lowest heuristic value. Arvand-LS (Xie et al., 2012) is a successor of Arvand-2009 that uses a combination of local greedy best first search and random walks. The use of memory and the more systematic search allows Arvand-LS to generate better quality solutions. Nakhost et al. (2012) developed Arvand-RC, a RW planner that significantly improved the state of the art in *resource constrained planning*. Arvand-RC augments Arvand-2009 with a more elaborate restarting mechanism called *smart restarts* and a new jumping strategy called *on-path search continuation*. Arvand-2011 (Nakhost et al., 2011), the RW planner that competed in IPC-2011, is an anytime version of Arvand-2009 that uses the postprocessing system Aras (Nakhost & Müller, 2010) to generate high quality solutions. Roamer (Lu et al., 2011), which is strongly influenced by Arvand-2009, uses GBFS enhanced by RW to escape from local minima or plateaus. For more details see Chapter 6.

The Arvand-2013 planner built here is not an exact copy or superset of previous Arvand versions containing all their features. There are two reasons for this: first, the building process is guided by the lessons learned from these previous systems and therefore naturally the components and ideas considered in this version are different. Second, in order to keep the scale of the experiments feasible and focused on the major open problems, alternative designs that have fewer parameters to tune are preferred. For example, instead of the complicated RW length scheduling mechanism used in all previous versions of Arvand, which contained three different parameters, the simpler idea of restarting random walks (Nakhost & Müller, 2012) is used, which has only a single parameter.

3.2.2 Other Local Search Planners

The Identidem planner (Coles et al., 2007) performs a two-level nested local search to find the goal. The first level guides the global search in a hill-climbing manner and uses the second level of local search to escape from local minima. Each run of the second level local search is called a *probe*. A probe starts from the current state, the entrance to the local minimum, and at each step selects one of the successors according to their heuristic value: states with lower heuristic value have a higher chance to be selected. A probe terminates when it reaches a depth threshold or an exit-point with improved heuristic value. Compared with RW, probes are much more costly: they generate and evaluate all the successor states along the path. Probes are also heavily biased towards low heuristic values.

3.2.3 Stochastic Local Search

Stochastic Local Search (SLS) (Hoos & Stützle, 2004) techniques have long been used for the kind of combinatorial optimization problems where every search state is a solution, and the goal is to maximize a given evaluation function. The closest SLS algorithm to RW search is Iterated Local Search (ILS) (Lourenco, Martin, & Stützle, 2003). This method starts from the initial state and at each step, first performs a perturbation on the current state s, and then performs a local search on the perturbed state. Based on an acceptance criterion, ILS either selects the state emerged from the local search as the next state or reverts to s. A variation of RW search that jumps to the first endpoint with lower heuristic can be seen as a special case of ILS: a random walk performs the perturbation and the local search phase is omitted. However, the more general setting of RW, where the jumping can be delayed until a number of walks are run, goes beyond the ILS framework: while ILS does not remember states that are rejected, RW search keeps the best sampled state with the lowest heuristic.

3.2.4 Rapidly Exploring Random Trees

A related technique to RW search is Rapidly Exploring Random Trees (RRT) (LaValle, 2006), a popular algorithm in motion planning and pathfinding. The core idea is to build a tree inside the search space that quickly covers different regions of the search space. This allows the algorithm to efficiently explore regions that would not be explored by only using heuristic guidance. RRT builds its tree in an iterative manner: in each iteration, it either explores or exploits. For exploration, the tree is extended towards a state uniformly randomly chosen from the whole search space. For exploitation, the tree is extended towards the goal. For both operations RRT first uses a heuristic function to select the state s in the tree that is closest to the target (a goal or a random state), then uses a search algorithm to extend a branch from s towards the target.

The main challenge of using RRT in planning is to randomly sample from the state space: the full state space in planning is usually exponentially larger than the actual reachable state space, and determining whether a state is reachable or not can be as hard as solving the planning problem

itself. Alcázar, Veloso, and Borrajo (2011) partially addressed this issue by using mutexes, pairs of propositions that cannot both be true in any reachable state. Their planner uses efficient techniques to find subsets of all possible mutexes and discards sampled states that violate the obtained mutexes. (Alcázar & Veloso, 2011) improved the sampling process using landmarks information: states that satisfy a landmark or a possible landmark predecessor, propositions that can be used to achieve a landmark, are sampled more often. The resulting planner BRT scored 49 units lower than Arvand-2011 and ranked 13$^{\text{th}}$ out of 27 participants in IPC-2011 (Coles et al., 2012).

3.3 The Experimental Framework

Arvand-2013 is built on top of the FD (Helmert, 2006) code base. This facilitates fair comparisons with other search algorithms implemented in FD; it ensures that all algorithms use the same successor generator, the same implementation of the heuristic function, and the same data structures to represent states and actions.

All experiments throughout this chapter are run on a 2.5 GHz machine with 4 GB memory limit, and unless otherwise mentioned, the runtime cut-off was set to 30 minutes. The 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 throughout the current chapter is on the number of problems solved and the runtime.

Like most experimental studies on AI planning, experiments are run on IPC benchmarks. The benchmarks provide a diverse set of domains and problems. Despite several studies (Helmert, 2008; Hoffmann, 2005, 2011) on understanding these domains, most of the search space features such as branching factor and solution depth are uncontrolled, and unknown for all but the smallest instances in most of these domains. It is therefore hard to precisely quantify the behaviour of algorithms solely based on experiments on IPC domains. Extra experiments are run on an artificial domain (AD) introduced by (Richter & Helmert, 2009), where input parameters can precisely control many features of the search space and the heuristic function. By using this artificial domain, the interplay between the parameters of the algorithm and problem features such as branching factor and heuristic accuracy is studied carefully.

3.3.1 An Artificial Domain to Control Key State Space Parameters

AD enables users to control the solution depth s_d, the length of a shortest path from the initial state to the closest goal, the branching factor bf, and the regress factor rf. The state space underlying an AD problem forms an undirected graph where all states have the same bf and rf. The only exception are the goal states, in which rf is undefined.

To be able to test the latest versions of RW and systematic search, an AD simulator is reimplemented in a recent FD codebase (August, 2012). The simulator provides a successor generator that enables the search algorithms to navigate the state space. The simulator also includes an artificial

heuristic, proposed by (Richter & Helmert, 2009), with an input $0 \leq h_{ac} \leq 1$ that controls the heuristic accuracy. Let d_s be the actual goal distance of a state s. The heuristic value of the state s is set to a random value in range $[0, d_s h_{ac}]$. Therefore, regardless of h_{ac}, a state closer to the goal has a higher chance of getting a smaller heuristic value. Thus, the heuristic values are positively correlated with the goal distances. Section 3.4.4 discusses how this positive correlation affects the behaviour of systematic and RW search.

3.4 Baseline: a Simple RW Planner

This section introduces the general structure of RWS and studies two of its fundamental features: the length of random walks and the restarting strategy. The key questions regarding these features are explored experimentally. 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?

To answer these questions, the idea is to start building Arvand-2013 with a simple algorithm, which allows to isolate the effect of restarting and the walk length distribution. Algorithm 1 shows the pseudocode, a forward chaining local search. Each run of Arvand-2013 consists of one or more search episodes, and terminates as soon as the most recent episode meets a termination condition. Each episode starts from the initial state $state_0$ and performs a series of search steps until it meets a restarting or termination condition. Let h_{min} be the minimum heuristic value reached in the current episode. Each search step $step_i$ starts from $state_{i-1}$ and ends in $state_i$ with $h(state_i) < h_{min}$. At each $step_i$, the planner runs random walks to select $state_i$. As soon as random walks reach a state $sampledState$ with $h(sampledState) < h_{min}$, the algorithm jumps to $sampledState$ by setting $state_i = sampledState$. The algorithm records the sequence of actions reaching $state_i$ from $state_0$. The termination condition holds in two cases:

- A goal state is reached. In this case, the sequence of actions that starts from $state_0$ and reaches the goal is returned as the solution.
- A time or a memory limit is exceeded. In this case, the planner terminates without returning any solution.

Whenever a restarting condition holds, the algorithm starts a new search episode by resetting the current state to s_0. Arvand-2013 can use two different mechanisms for restarting: restarting threshold and restarting rate. When using a restarting threshold t_g, local search restarts when the most recent t_g walks have not reached a heuristic lower than h_{min}: this is similar to all previous versions of Arvand. An alternative is to use a restarting rate r_g (g stands for global): if the last walk leaves h_{min} unchanged, then the algorithm restarts from the initial state with probability r_g. Restarting rates are common in combinatorial optimization (Hoos & Stützle, 2004) and are easier to analyze than restarting thresholds as in Section 2.6.

Algorithm 2 gives the pseudocode for random walks. Unlike other versions of Arvand (Chapter 6), the base algorithm evaluates all visited states. A walk stops early if it reaches a heuristic value lower than h_{min}, a goal state, or a dead end; otherwise it runs until a termination condition holds. Like restarting random walks (Chapter 2), the algorithm uses a local restarting rate r_l (l stands for local): at each step, the walk terminates with probability r_l. In the absence of early stops, the length of walks is geometrically distributed with mean $\frac{1}{r_l}$. The restarting rates r_l and r_g are used for different purposes. While the local restarting rate r_l determines the length of the walks, the global restarting rate r_g determines when to stop an episode. As the heuristic function, Arvand-2013 uses the cost-sensitive version of h^{FF} (Hoffmann & Nebel, 2001) from the FD code base.

The remainder of this section studies the effects of these mechanisms for global and local restarting. It also presents parameter studies for these mechanisms.

Algorithm 1 Random Walk Search

Input Initial State $state_0$, goal condition G and
Output A solution plan
Parameters t_g and r_l

$currentState \leftarrow state_0$
$h_{min} \leftarrow h(state_0)$
loop
  $sampledState \leftarrow RandomWalk(currentState, G, h_{min}, r_l)$
  if $sampledState \supseteq G$ **then**
    return the plan reaching the $sampledState$
  else if $sampledState \neq Deadend$ **and** $h(sampledState) < h_{min}$ **then**
    $currentState \leftarrow sampledState$
    $h_{min} \leftarrow h(currentState)$
  else if $Restart()$ **then**
    $currentState \leftarrow state_0$ {restart from initial state}
    $h_{min} \leftarrow h(state_0)$
  end if
end loop

3.4.1 Restarting: Parameter Study

Figure 3.1 shows the effect of t_g, r_g and r_l on the coverage of Arvand-2013 in ten of the fourteen IPC-2011 (Coles et al., 2012) domains. To improve readability, the results for Transport, Barman and Openstacks are omitted: none of the configurations solves more than 10% in Barman and Transport, and all configurations solve more than 90% in Openstacks and Pegsol. The key observations are as follows:

- Even this very basic RW planner can solve a large percentage of IPC problems. For a detailed comparison with other search algorithms, see Section 3.4.4.
- The results for t_g and r_g follow the same pattern: large t_g values perform well in the same domains where small r_g perform well, and similar for small t_g and large r_g.

Algorithm 2 Random Walks

Input $currentState$, goal condition G, h_{min}, and r_l
Output $sampledState$

```
loop
  sampledState ← currentState
  A ← ApplicableActions(currentState)
  if A = φ or h(sampledState) = ∞ then
    return Deadend
  end if
  a ← UniformlyRandomSelectFrom(A)
  sampledState ← apply(sampledState, a)
  if h(sampledState) < h_min or sampledState ⊇ G then
    return sampledState
  end if
  with probability r_l: return sampledState
end loop
```

- None of the settings for global or local restarting performs well across all the domains. Different configurations work for different domains. For example, while shorter walks, i.e., larger r_l, perform better in Nomystery and Woodworking, they are worse than longer walks in Tidybot and Visitall. Similarly, while in Elevators restarting less often, $t_g = 10000$ or $r_g = 0.0001$, increases the coverage compared to frequent restarting, $t_g = 100$ or $r_g = 0.01$, in the domains of Nomystery, Floortile, Parcprinter and Tidybot, restarting more often is better.

- The best value for each parameter rarely depends on the value for the other one. For example, larger r_l, independently from the frequency of global restarting, always wins in Nomystery and Woodworking; and always loses in Tidybot and Visitall. Also while higher t_g is never worse in Woodworking or Elevators, it is detrimental in the other domains.

In light of these results, finding a robust setting for t_g, r_g and r_l that works well across all domains seems infeasible. This is not surprising in domain-independent planning, where the characteristics of target problems vary widely. Incorporating a parameter learning system seems essential in order to fully realize the potential of RW.

3.4.2 Adaptive Global Restarting

Why does Arvand-2013, using a small restarting threshold, perform well in domains such as Elevators, but fail in domains such as Floortile and NoMystery? Is it possible to learn an effective restarting strategy online? Figures 3.2.a and 3.2.b show more detailed data for two typical examples from Elevators and Floortile. The figures plot h_{min} as a function of the number of RW for two different t_g values. In Floortile, the heuristic decreases very quickly at first, then stops when either a dead end or a very big heuristic plateau is reached. Therefore, in this domain, using large t_g wastes lots of time in dead ends and plateaus. In contrast, restarting more often with a small t_g increases the exploration and the chance of reaching the goal. The data in Elevators shows the opposite behaviour:

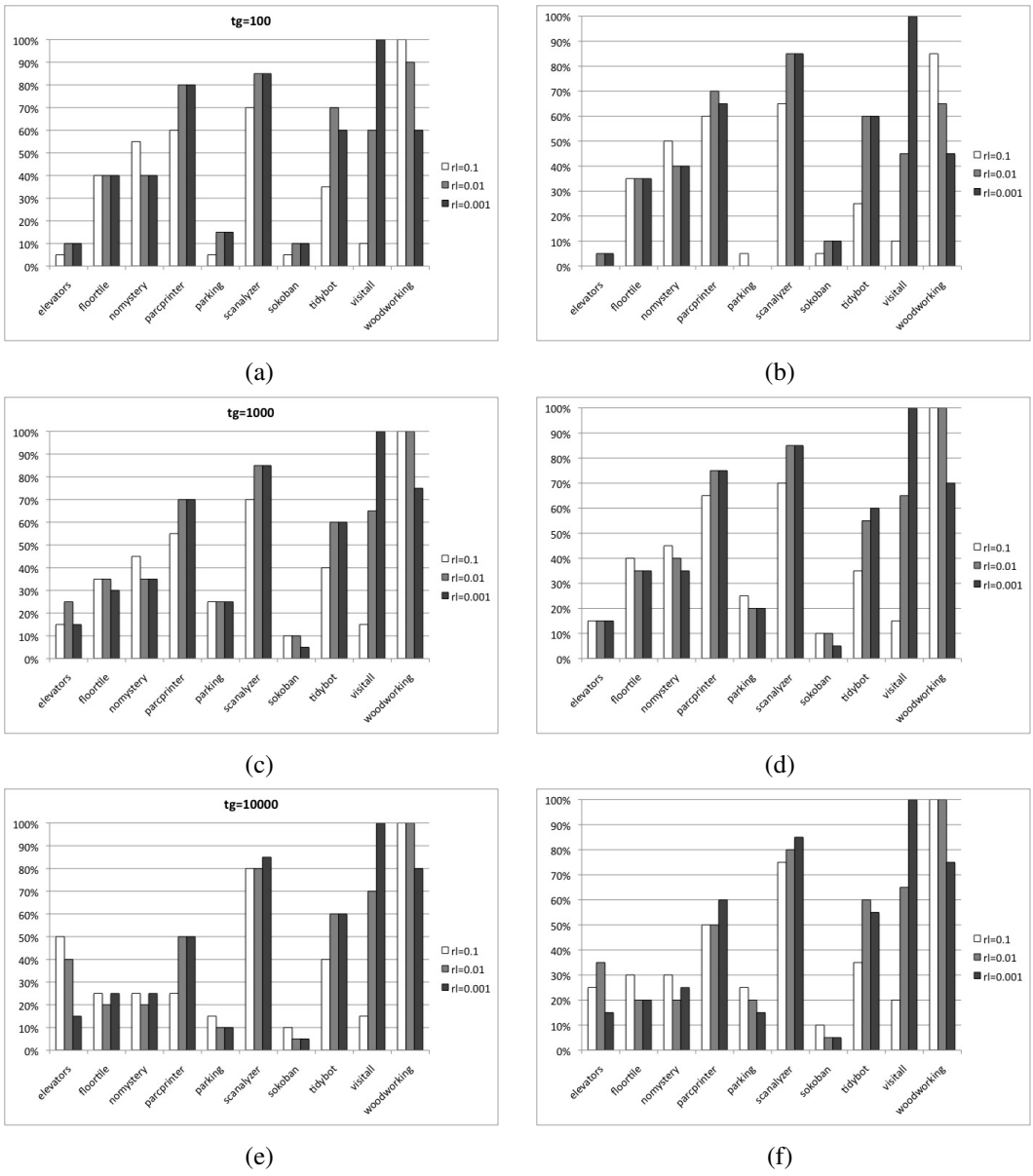


Figure 3.1: Coverage of RWS varying r_l [0.1 0.01 0.001 in all (a) – (f)], t_g [100 1000 10000 in (a), (b) and (c)], and r_g [0.01 0.001 0.0001 in (d), (e), and (f)].

h_{min} makes steady, slow progress towards 0. Fast restarts stop this steady progress of the search before it hits a goal.

Let V_w (V and w stand for velocity and walks, respectively) be the average heuristic improvement per walk. Then, on average $\frac{h(state_0)}{V_w}$ walks are needed to reach $h = 0$. Adaptive global restarting (AGR) is a learning algorithm that adjusts t_g by continually estimating V_w and setting $t_g = \frac{h(state_0)}{V_w}$. Algorithm 3 gives the pseudocode. The algorithm starts with the initial $t_g = 1000$ and after each episode, updates both t_g and the estimate for V_w. Before the i^{th} episode, AGR measures V_w^i, the average number of random walks to reach h_{min} and sets $V_w = Avg_{j \leq i}\{V_w^j\}$.

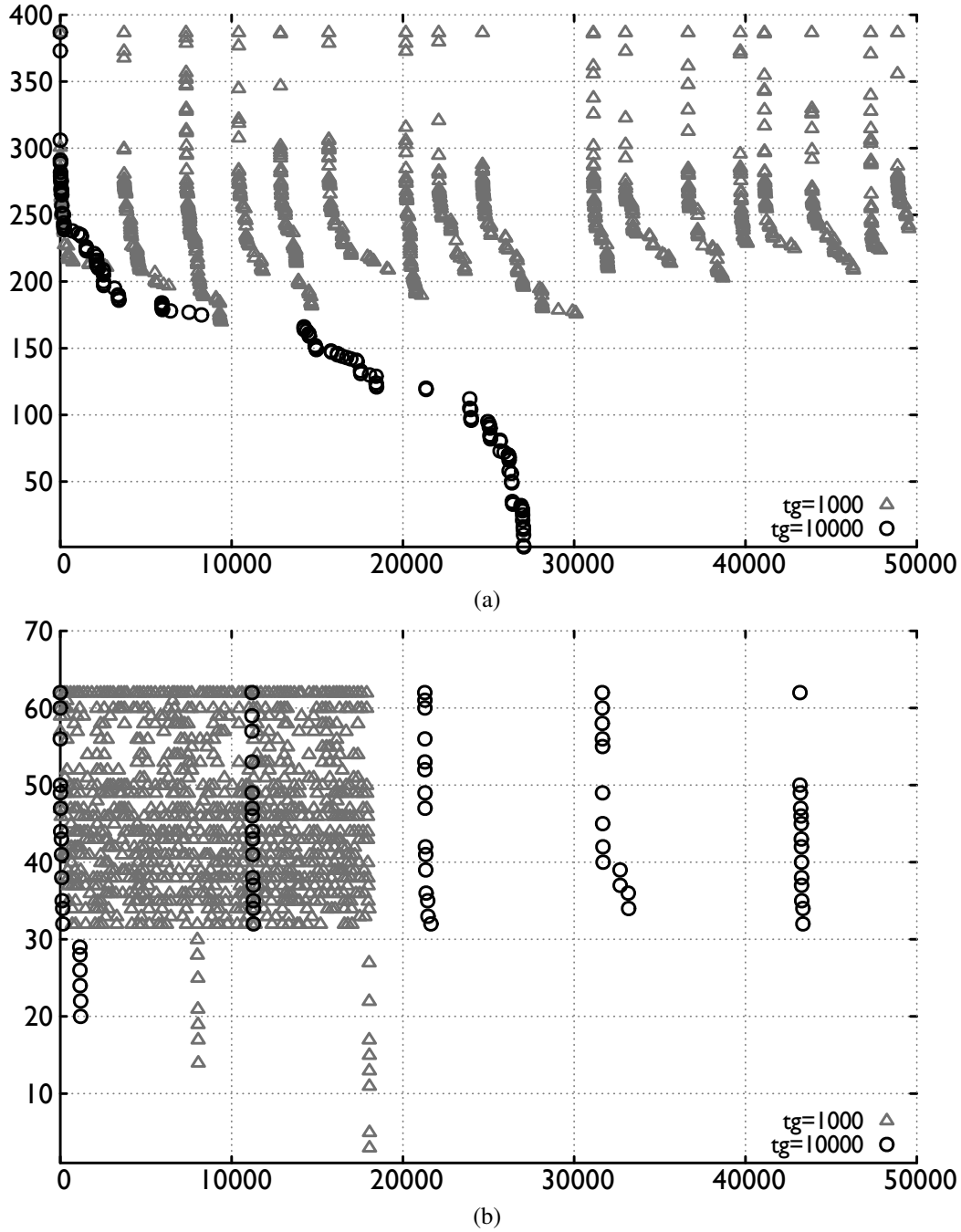
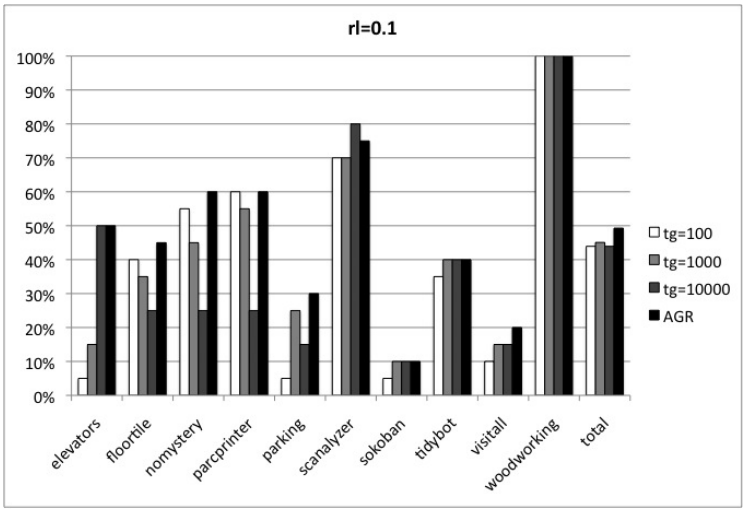
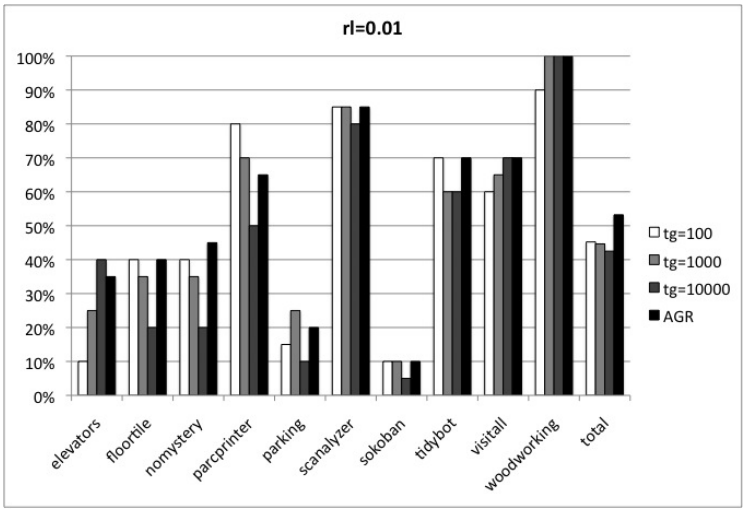


Figure 3.2: h_{min} vs. the number of walks for (a) Elevators-03 and (b) Floortile-01

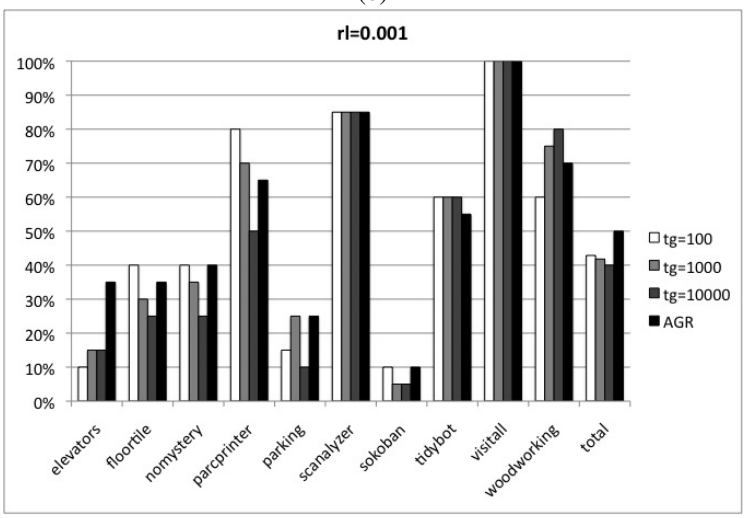
Figure 3.3 compares adaptive restarting to restarting with fixed t_g. Although AGR is not always the best setting in all the domains, it is a robust setting resulting in either best or close to best results. Arvand-2013 using AGR and $r_l = 0.01$ solves in total 149 problems, 22 more problems than the best setting for a fixed threshold: $t_g = 100$ and $r_l = 0.01$. In the tests, the initial value of t_g was set to 1000.



(a)



(b)



(c)

Figure 3.3: Coverage of AGR versus fixed threshold restarting varying tg [100 1000 10000 in all (a) – (c)] and r_l [0.1 in (a), 0.01 in (b), 0.001 in (c)]

Algorithm 3 Random Walk Search using AGR

Input Initial State $state_0$ and goal condition G
Output A solution plan
Parameters t_g and r_l

$currentState \leftarrow state_0$; $h_{min} \leftarrow h(state_0)$
$numRestarts \leftarrow 0$; $numWalks \leftarrow 0$; $lastImprovingWalk \leftarrow 0$
$V_w \leftarrow 0$
loop
  $sampledState \leftarrow RandomWalk(currentState, G, h_{min}, r_l)$
  $numWalks$ ++
  if $sampledState \supseteq G$ **then**
    return the plan reaching the $sampledState$
  else if $sampledState \neq Deadend$ **and** $h(sampledState) < h_{min}$ **then**
    $currentState \leftarrow sampledState$
    $h_{min} \leftarrow h(sampledState)$
    $lastImprovingWalk \leftarrow numWalks$
  else if $numWalks - lastImprovingWalk > t_g$ **then**
    $V_w^i \leftarrow (h(state_0) - h_{min})/lastImprovingWalk$
    $V_w \leftarrow (V_w^i - V_w)/numRestarts + V_w$ {update the estimate}
    $t_g \leftarrow h(state_0)/V_w$ {update t_g}
    $currentState \leftarrow state_0$ {restart from initial state}
    $h_{min} \leftarrow h(state_0)$
    $numWalks \leftarrow 0$; $numRestarts++$
  end if
end loop

3.4.3 Adaptive Local Restarting

Analogous to the analysis for global restarting, Figure 3.4 shows h_{min} as a function of 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 stands for evaluations) be the average heuristic improvement per evaluation when $r_l = r$. The larger V_e, the faster the search progress towards the goal.

The value V_e can explain why the best setting for Visitall is the worst for Elevators and vice versa. While smaller r_l leads to faster progress (larger V_e) in Visitall, it is slower (smaller V_e) in Elevators. Therefore, the larger is $V_e(r)$, the more effective is $r_l = r$.

Adaptive local restarting (ALR) is a multi-armed bandit solver (Gittins, Glazebrook, & Weber, 2011) that estimates $V_e(.)$ to learn the best r_l setting. Before running each random walk, ALR selects r_l from a candidate set $C = \{r_1, \ldots, r_n\}$. ALR treats each r_i, $1 \leq i \leq n$, as a bandit arm. Selecting a value for r_l is analogous to playing the arm. For each r_i, ALR keeps track of the average number of evaluations $avg_e(r_i)$ and the average heuristic improvement $avg_h(r_i)$ (for negative heuristic improvements, 0 is inserted into the computation of $avg_h(r_i)$). The algorithm uses $\frac{avg_h(r_i)}{avg_e(r_i)}$ as an estimation for $V_e(r_i)$. ALR uses ϵ-greedy (Sutton & Barto, 1998) to sample the arms based on the estimated $V_e(r_i)$: with probability ϵ, $0 \leq \epsilon \leq 1$, it selects one of the arms uniformly randomly and with probability $1 - \epsilon$ it selects one of the arms with largest $V_e(r_i)$.

Figure 3.5 compares ALR using $\epsilon \in \{0.1, 1\}$ with three fixed settings $r_l = 0.1, 0.01, 0.001$. To

ensure the results are comparable, the candidate set $C = \{0.1, 0.01, 0.001\}$. For all the configurations, AGR is used to control global restarting. When $\epsilon = 1$ ALR always selects r_l uniformly randomly. The key observations are as follows:

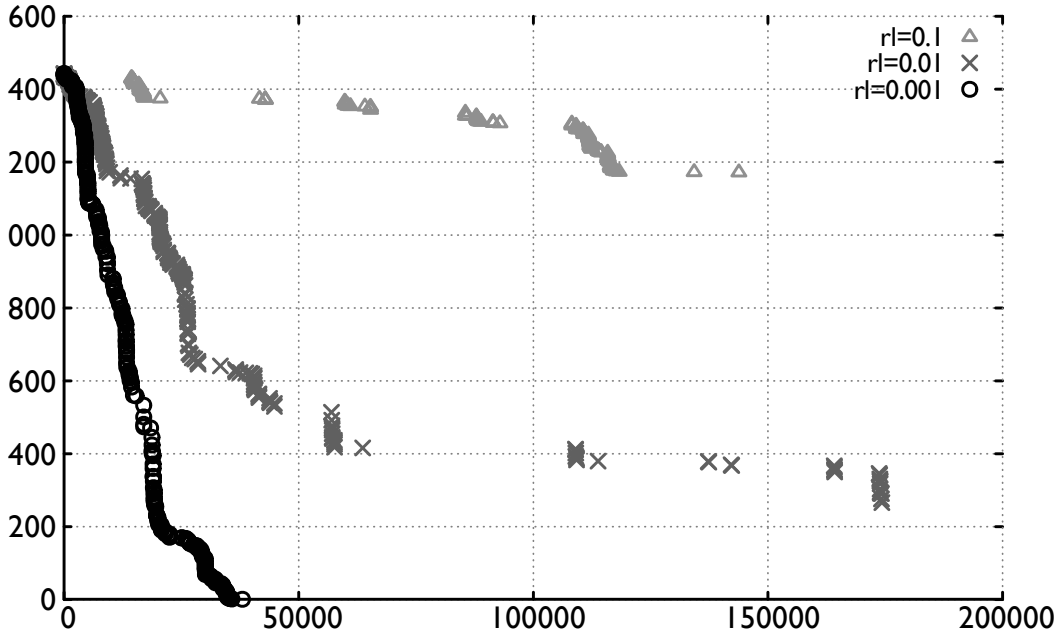
- ALR performs robustly across all domains: the gap between ALR and the best setting for a domain is never more than 10%, except in Elevators where $r_l = 0.1$ solves 15% more problems.
- Sampling based on $V_e(.)$ shows a small advantage over uniform sampling: in total, $\epsilon = 0.1$ solves 6 (2%)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

3.4.4 Comparison with Systematic Search

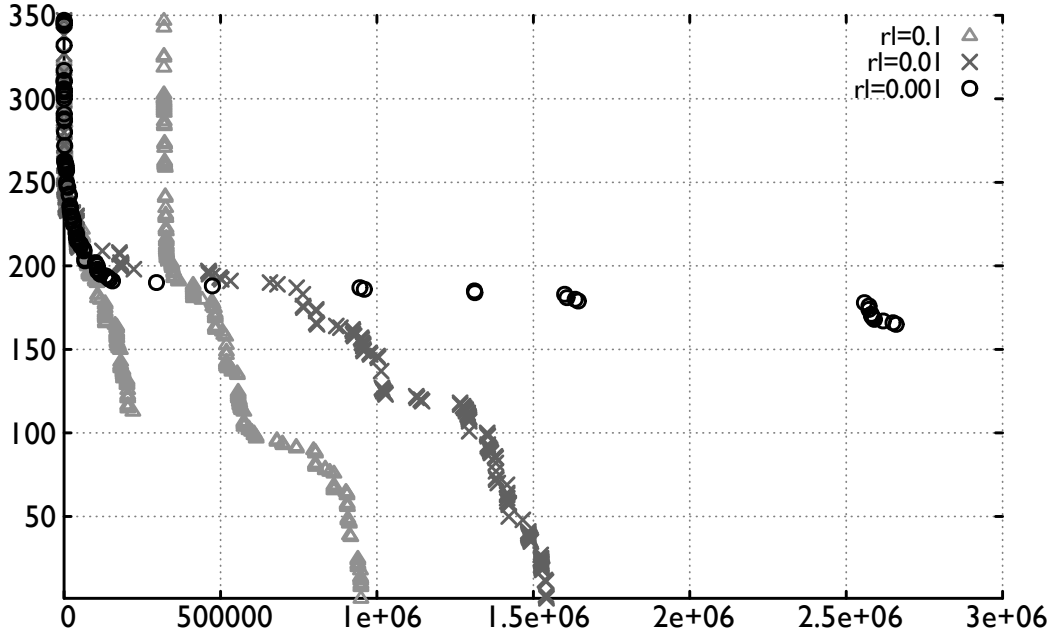
While the above parameter studies give valuable insights regarding random walks, they do not explain how they perform compared with common systematic algorithms such as GBFS and WA*. How does Arvand-2013, at this stage of development, compare with systematic search using the same amount of information? Which characteristics of a problem make random walks a better choice? Is it true that random walks perform better when the heuristic values are more inaccurate? Does the regress factor have any effect in non-plateau search regions? Does the regress factor also affect the systematic search? To answer these questions, experiments were run on all IPC domains and the artificial domain introduced in Section 3.3. While the IPC experiments give the big picture of how RW search compares with systematic search, the tests on AD show how the characteristics of search space and heuristic function such as solution depth, regress factor, and heuristic accuracy affect RW and systematic search.

The tested algorithms include GBFS, WA* varying $w \in \{1, 2, 3, 5\}$ as in LAMA-2011 and EHC. Although EHC is a local search and is not categorized as a systematic search, unlike RWS, it uses a systematic exploration of the local neighborhood. To keep the results comparable to the basic RWS in Arvand-2013, all tested algorithms use a single heuristic function h^{FF}, the same heuristic function as in Arvand-2013, and no preferred operators. Figure 3.6 shows the coverage on IPC domains. The results confirm that:

- RW search and GBFS have very different strengths and weaknesses: while RW search solves 25% to 80% more problems than GBFS in Elevators, Parcprinter, and Visitall, GBFS solves 35% to 75% more problems in Barman, Parking, and Sokoban.
- RW search is quite competitive with standard systematic search using the same information: Arvand-2013 in total solves 2% more problems than GBFS.
- Generally, WA* performs weaker than both RWS and GBFS. In total RWS solves 22 (8%) more problems than the best w setting. WA* has the same strengths and weaknesses of GBFS



(a)



(b)

Figure 3.4: h_{min} vs. the number of evaluated states for (a) Visitall-14 and (b) Elevators-05

with the exceptions of much weaker performance in woodworking and stronger performance in Floortile and Nomystery.

- RWS significantly outperforms EHC across all domains. In total RWS solves 97 (35%) more problems.

Experiments on AD test the effect of three parameters: solution depth, regress factor, and heuris-

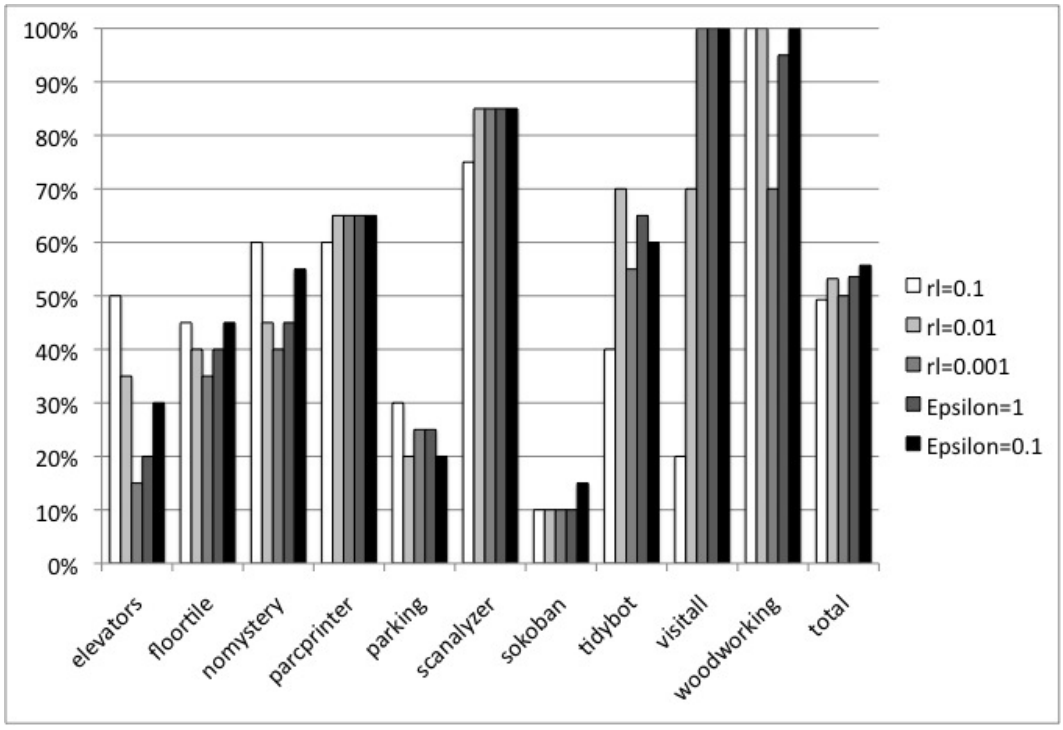


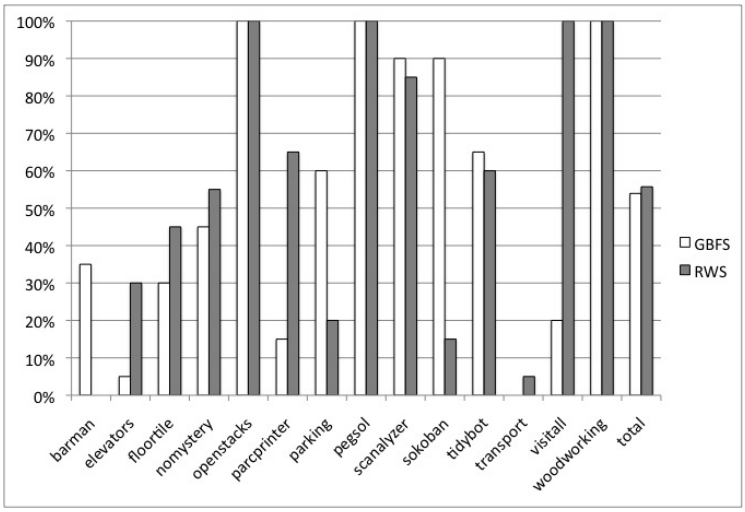
Figure 3.5: Coverage of ALR and local restarting with fixed rate varying ϵ [1 0.1] and r_l [0.1 0.01 0.001]

tic accuracy. Like the experiments in (Richter & Helmert, 2009), the branching factor was fixed to 25. After some initial experiments, the settings $r_l = 0.1$ and $t_g = \infty$ were used for Arvand-2013. Therefore, the algorithm never restarts: since there are no dead-ends or extensive plateaus in AD, after several search steps, the local search is almost always at a state closer to the goal than the initial state. Thus, restarting often hurts.

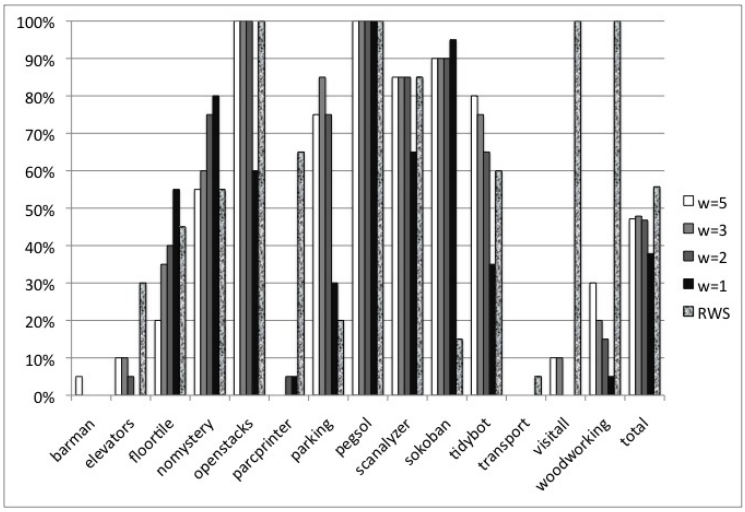
Figure 3.7 summarizes the results. The plots show the average number of generated nodes for both GBFS and RW search as a function of the regress factor rf, the heuristic accuracy h_{ac}, and the solution depth s_d. The key observations are as follows:

- Unlike in plateaus, the regress factor significantly affects the performance of GBFS: for example, GBFS on a problem with $rf = 24$, depending on the solution depth, evaluates 2 to 5 orders of magnitude more nodes than when $rf = 1$. It is unclear if this is a general phenomenon or an artifact of the well-behaved heuristic model of the artificial domain: no matter what the heuristic accuracy is, the heuristic values are positively correlated with the goal distance. Therefore, rf, which is a function of goal distance, is correlated with the distribution of heuristic values. This is not the case in a plateau where there is no correlation between the heuristic values and the goal distances.

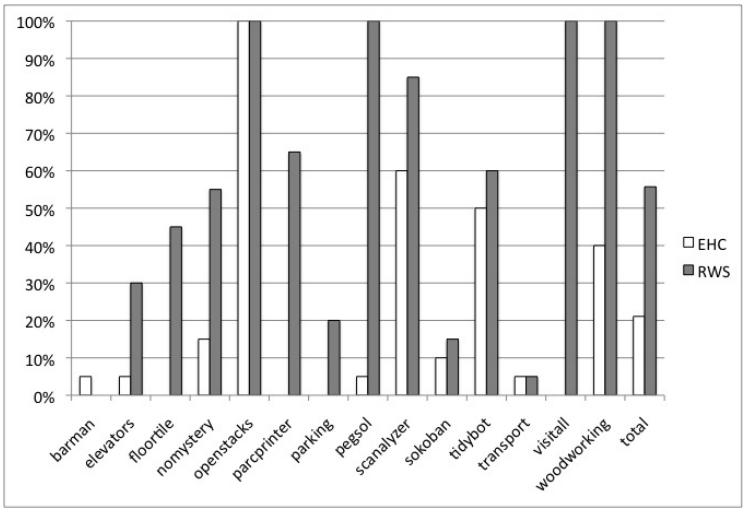
- Random walks scale better than GBFS when the solution depth s_d increases: up to the limits of our tests, in all cases GBFS starts out as more efficient for small s_d but is eventually



(a)



(b)



(c)

Figure 3.6: Coverage of RWS compared with (a) GBFS (b) WA* ($w \in \{1, 2, 3, 5\}$) and (c) EHC in IPC-2011

overtaken by RW as the solution depth increases. Let the crossing point $C(rf, h_{ac})$ be the s_d value where both algorithms search the same number of nodes. For example, when $rf = 24$ and $h_{ac} = 0.4$, $C(24, 0.4) = 80$. The data suggest that first, $C(rf, h_{ac})$ is unique. Second, the number of nodes generated by RW divided by the number of nodes generated by GBFS decreases monotonically with s_d for all fixed rf and h_{ac}.

- Compared with GBFS, random walks perform better as the heuristic becomes less accurate: when keeping rf fixed, $C(rf, h_{ac})$ decreases when h_{ac} decreases. For example,

$$C(24, 0.6) = 110 > C(24, 0.4) = 80.$$

- Compared with GBFS, random walks perform better when rf is larger: when keeping h_{ac} fixed, $C(rf, h_{ac})$ decreases when rf increases. For example,

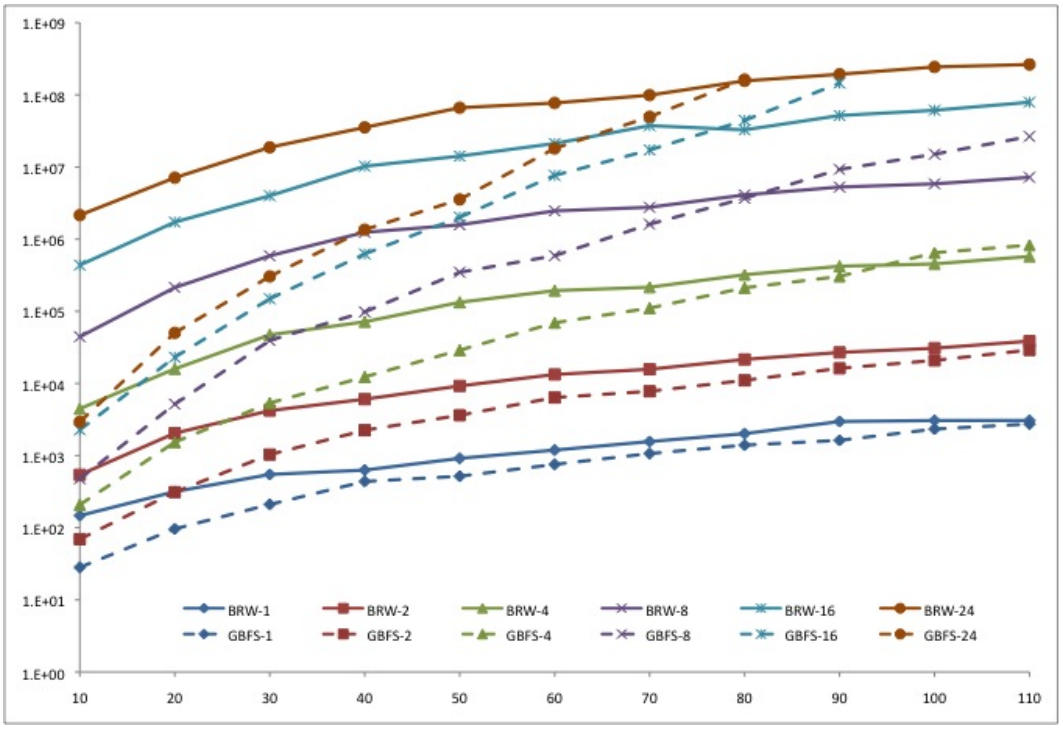
$$C(8, 0.6) = 130 > C(24, 0.6) = 110.$$

3.5 The Rate of Heuristic Evaluation

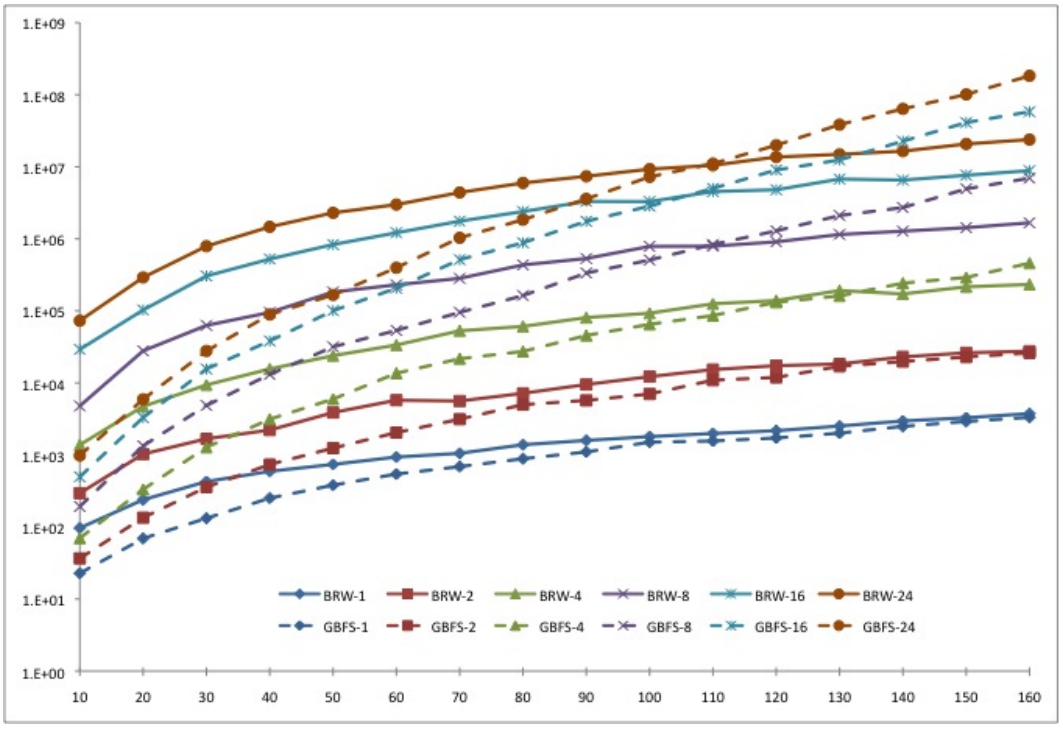
Although heuristic values provide key information to guide search, they do not come for free: heuristic computation consumes resources such as time and memory. Algorithms such as deferred evaluation, implemented in Fast Downward (Helmert, 2006), and RW search, implemented in Arvand-2009, use evaluation policies to guide the search using fewer heuristic evaluations. Deferred evaluation in GBFS delays the heuristic evaluation of a node until it is expanded: in absence of the heuristic value, the parent's heuristic value is used to rank the nodes in the open list. Arvand-2009, in a more radical way, avoids evaluating all intermediate generated states before the endpoint of a random walk. The following questions are addressed by tests focused on the evaluation policy: How effective is it to just evaluate endpoints? How often should a RW planner evaluate states?

Experiments were run to test different evaluation scenarios. Instead of evaluating all states, Arvand-2013 is modified to evaluate the endpoint of all random walks and evaluate other states with probability p_{eval}. When $p_{eval} = 1$, the algorithm is the same as in previous sections, and when $p_{eval} = 0$, it is the same as MRW in Arvand-2009. For all configurations, ALR ($\epsilon = 0.1$) and AGR are used to control local and global restarting. Figure 3.8 shows the coverage and average runtime in IPC-2011 when varying p_{eval}. The results divide the domains into four categories:

- Domains where more evaluation always hurts: Arvand-2013 can solve all instances of Openstack, Visitall and Woodworking even when $p_{eval} = 0$; RW search is so effective in these domains that higher evaluation rates only increase the runtime. Tidybot is similar but harder: Arvand-2013 solves the most problems when $p_{eval} = 0$. In Tidybot, the heuristic function is very costly and misleading: a blind random walk search that only goal checks the states,



(a)



(b)

Figure 3.7: The number of evaluated states by RW search and GBFS when varying rf [1 2 4 8 16 24 in (a) and (b)], s_d [shown on the x-axis in (a) and (b)] and h_{ac} [0.4 in (a), 0.6 in (b)]

and uses no evaluations, solves 90% of Tidybot instances. Only one planner, BRT(Alcázar & Veloso, 2011), could solve more in IPC-2011.

- Domains where always evaluating more often pays off: Parcprinter and Nomystery. The running time decreases monotonically with increasing evaluation rate. In these domains, the information gained from evaluation is always worth the cost.

- Domains where $p_{eval} = 0$ provides too little information and $p_{eval} = 1$ is too costly: an evaluation rate in between works the best. In the tests, Elevators, Floortile, and Parking were of this type.

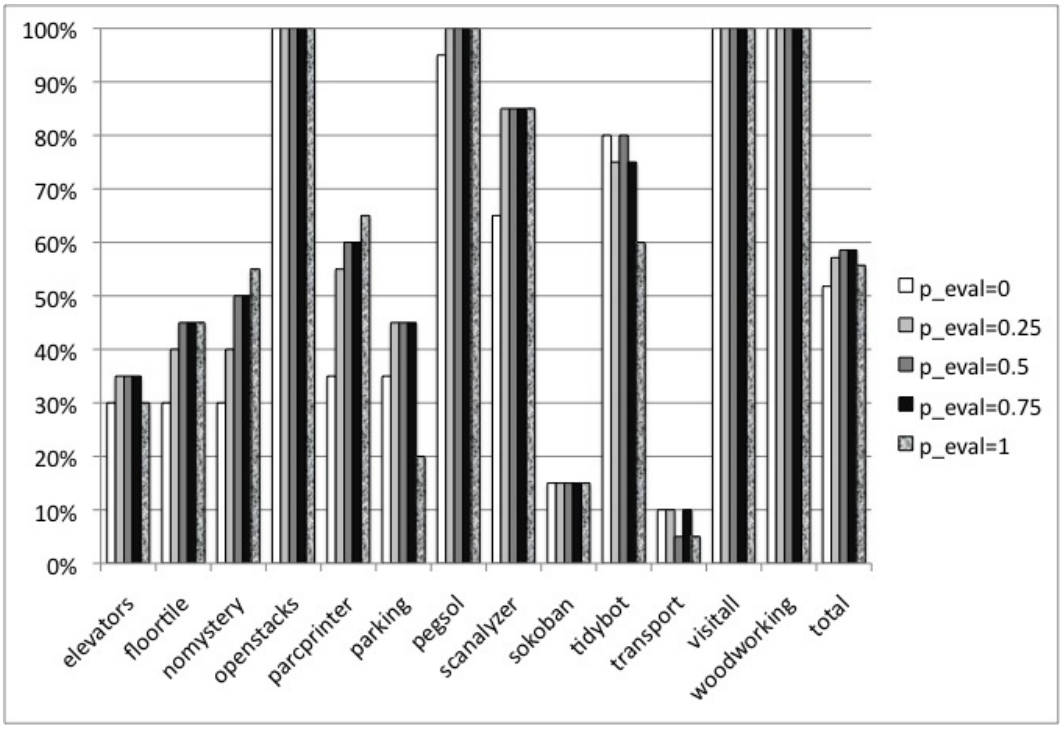
- Domains where the results are the same for all tested $p_{eval} > 0$: Scanalyzer, Pegsol, Sokoban, and Transport.

In these experiments, RW search uses the heuristic computation only for the heuristic values. However, preferred operators can be obtained from most of the common heuristic functions in planning at no additional cost. Using preferred operators to guide random walks might change the results in favour of higher evaluation rates. The baseline results above confirm that for at least some domains and heuristic functions it pays off to evaluate less and search more.

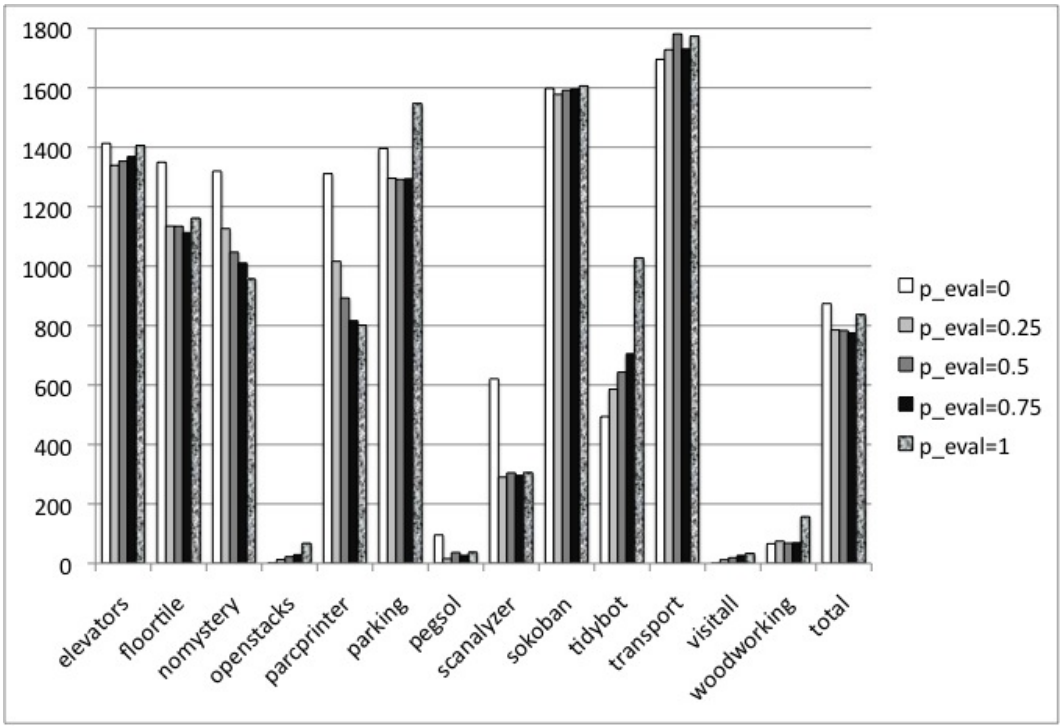
3.6 Testing Greedy vs. Delayed Jumping

Nakhost and Müller (2009) suggested a delayed commitment strategy to move through search space: instead of greedily jumping to the first state with a lower heuristic value, run n walks and then jump to the state with the lowest heuristic value. The assumption is that jumping to a state yielding a larger heuristic improvement increases the chance of getting closer to the goal. However, large n can lead to wasting lots of time without leading to larger heuristic improvements. The question is whether the additional exploration is worth the cost. Based on past experience, we believe that the answer depends on search space characteristics such as the correlation between the heuristic values and the goal distances, the distribution of heuristic values, and the distribution of the states sampled by random walks. Deriving a model to determine an effective n based on key search space characteristics is a topic for future research.

Nakhost and Müller (2009) also proposed a simple intuitive algorithm based on *acceptable progress* (AP) to dynamically control n. The algorithm stops exploration if a state with small enough h-value is reached. The *progress* of candidate state s measures the decrease (if any) in h_{min} in this state: $P(s) = \max(0, h_{min} - h(s))$. As soon as progress exceeds an acceptable progress threshold, exploration is stopped and the candidate state is immediately selected. Let $step_i$ be the search step i starting from $state_{i-1}$ and ending in $state_i$. The progress made at $step_i$, $P_i = P(state_i)$. Acceptable progress, AP_i, is defined as a weighted combination of progress made in earlier search steps, $AP_1 = P_1$, $AP_{i+1} = (1-\alpha)AP_i + \alpha P_i$, with a parameter α, $0 \leq \alpha \leq 1$. Higher α values put more



(a)



(b)

Figure 3.8: Coverage (a) and average runtime (b) in the IPC-2011 benchmark domains varying p_{eval} [0 0.25 0.5 0.75 1]. For unsolved instances, the time limit of 1800 seconds is inserted into the computation of average runtime.

emphasis on recent progress. The algorithm uses an initial value of n to limit the exploration in the first search step. AP is used in all previous versions of Arvand, but it has never been thoroughly tested in isolation.

The following experiments evaluate the potential of the delayed commitment strategy and determine how effectively AP can control it. To run the tests, Arvand-2013 is modified as follows: for each search step, first run n walks. If a state s with $h(s) < h_{min}$ is found during these n walks, then jump to the best such s, otherwise keep running walks until such a state is found or a restarting/termination condition holds. AGR and ALR ($\epsilon = 0.1$) are used to control global and local restarting. The evaluation rate p_{eval} is set to 0.5. When $n = 1$, the search behaves exactly as in the previous section.

Figure 3.9 shows the coverage and average runtime of Arvand-2013 using four fixed settings $n \in \{1, 10, 100, 1000\}$. Using larger n means more exploration. The results of AP using the initial $n = 1000$ and $\alpha = 0.9$ are also included. The key observations are as follows:

- Parcprinter is the only domain where more exploration can lead to a significant improvement: Arvand-2013 with $n = 100$ solves 25% more problems than when $n = 1$. In other domains, more exploration either has no significant effect or hurts.

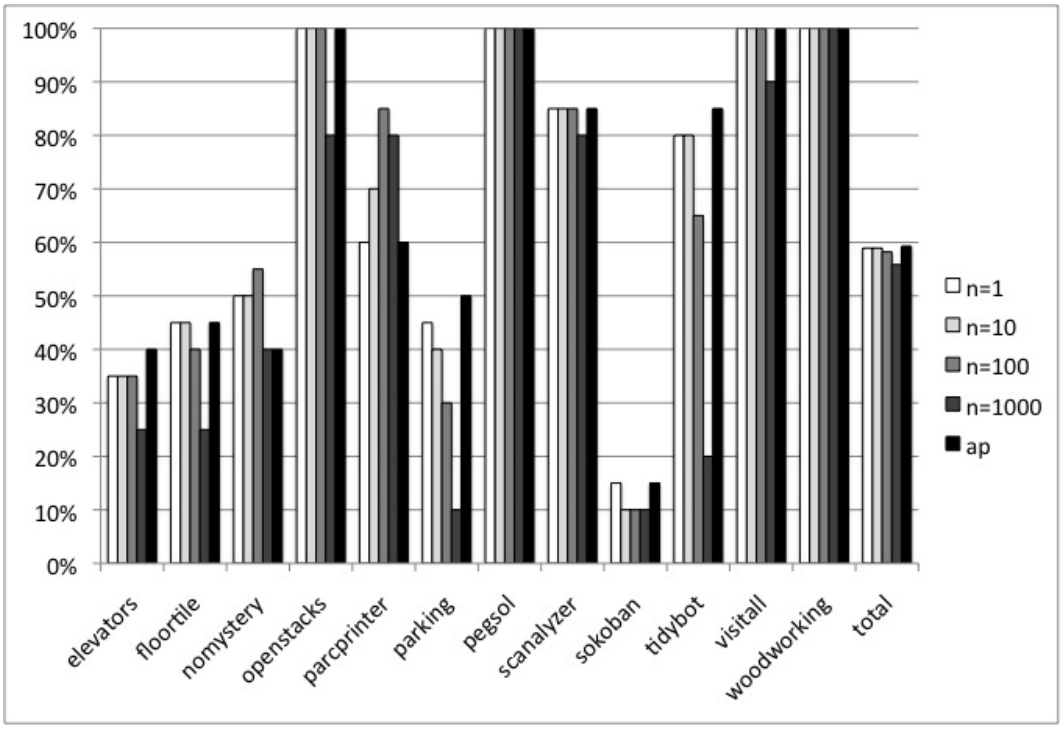
- While AP performs slightly better than the best fixed n in Elevators, Floortile, Parking and Tidybot, it does not perform well in Parcprinter where more exploration is beneficial: AP solves 25% less Parcprinter problems than setting $n = 100$.

3.7 Biasing Action Selection

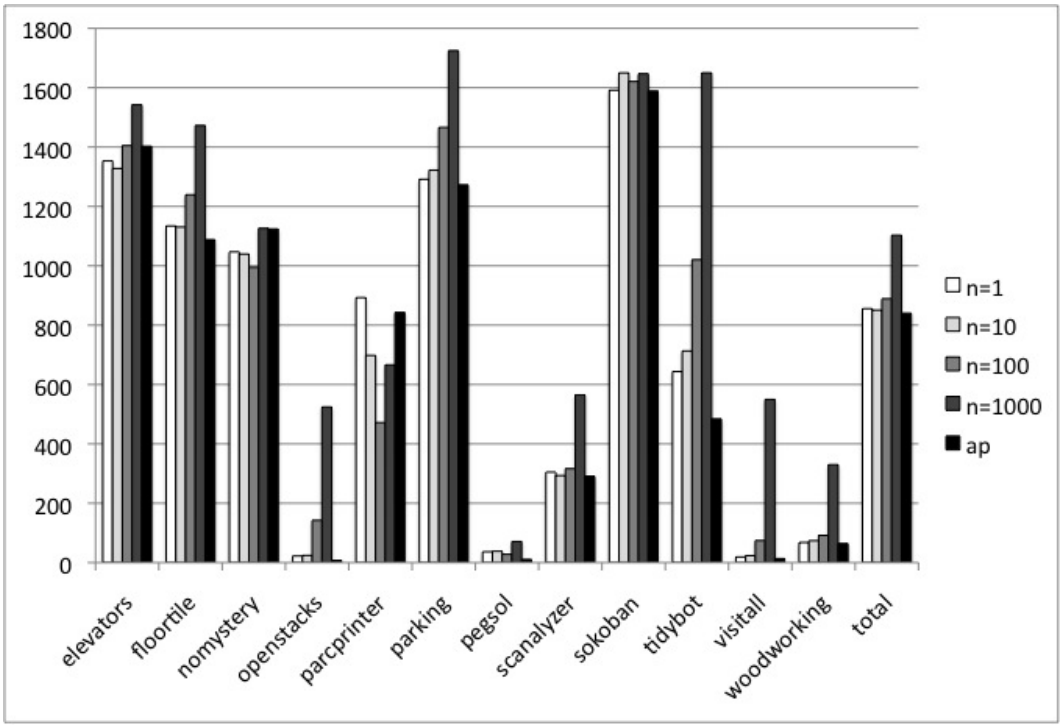
Using Homogenous graphs, Chapter 2 showed that biased action selection can significantly decrease the runtime of random walks by decreasing the regress factor. This section studies Monte Carlo Helpful Action (MHA) and Monte Carlo Deadlock Avoidance (MDA), two techniques to bias random action selection according to the information gathered from the earlier random walks (Nakhost & Müller, 2009). Both techniques update scores $Q(a)$ for any possible action a, and sample from a Gibbs distribution, which is a function of the scores, to select the actions. Let s be the current state. The probability $p(a, s)$ that the action a is chosen among all the applicable actions $A(s)$ is set to

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

The temperature T stretches or flattens the probability distribution, and therefore, determines how much the action selection prefers actions with larger scores. High temperatures yield (nearly) uniform action selection. To compute $Q(a)$ scores, Arvand only considers the statistics gathered from the current state and reinitializes the scores as soon as it jumps to another state. Specific choices of the $Q(a)$ function for MHA and MDA will be discussed in the following subsections.



(a)



(b)

Figure 3.9: Coverage (a) and average runtime (b) of Arvand-2013 in the IPC-2011 benchmark domains using fixed n [1 10 100 1000] and AP. For unsolved instances, the time limit of 1800 seconds is inserted into the computation of average runtime.

3.7.1 Monte Carlo Helpful Action

In the previous implementation of MHA (Nakhost & Müller, 2009), $Q(a)$ counts the number of times an action is preferred in the current search step. Therefore, the preferred operators of the current state are treated in the same way as any other preferred operators. This seems counterintuitive: one would expect that the current preferred operators get higher priority. This is not addressed in Arvand-2009 because that planner only evaluates the endpoint of walks, and the preferred operators for intermediate states are unknown. Preferred operators are only available for the states that are evaluated. Section 3.5 shows that evaluating some intermediate states usually pays off. Therefore, it makes sense to use a modification of MHA that gives higher priority to current preferred operators, if they are known. The following scoring method gives such a system. Let $n(a)$ be the number of times that action a has been a preferred operator, and $maxN$ be the maximum $n(a)$ among all the possible actions. The score $Q(a)$ is set to:

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

The parameter W, $0 \leq W \leq 1$, controls the weight of the current preferred operators: the larger W, the higher the chance that a current preferred operator is selected. If the current preferred operators are unknown, the scoring function preforms the same as in the original MHA.

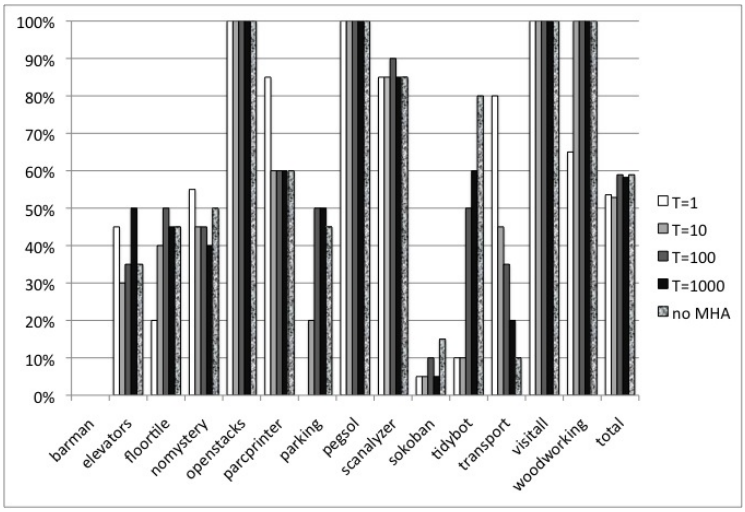
Figure 3.10 compares the coverage of Arvand-2013 using uniform selection (no MHA) and MHA when varying temperature and W. In all the configurations, $p_{eval} = 0.5$, $n = 1$, ALR($\epsilon = 0.1$) and AGR are used.

Let $MHA(w, t)$ denote MHA using $W = w$ and $T = t$. The key observations are as follows:

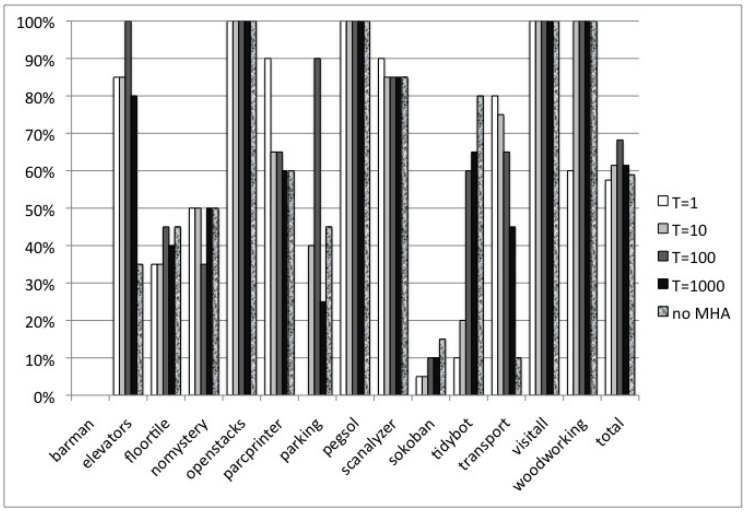
- MHA is very effective: significant improvements are achieved in Barman, Transport, Elevators, and Parking. While RW with no biasing solves no problem in Barman and only 2 (10%) problems in Transport, $MHA(1, 10)$ solves 18 (90%) problems in Barman and 20 (100%) problems in Transport. The same configuration $MHA(1, 10)$ also increases the coverage in Elevators and Parking by 65% and 40%, respectively. In total, Arvand-2013 solves 54 more problems (19%) with MHA than without.

- Using only the current preferred operators ($W = 1$) works the best: Consistently across all domains, $MHA(1, t)$ performs better than $MHA(0.5, t)$, and $MHA(0.5, t)$ performs better than $MHA(0, t)$.

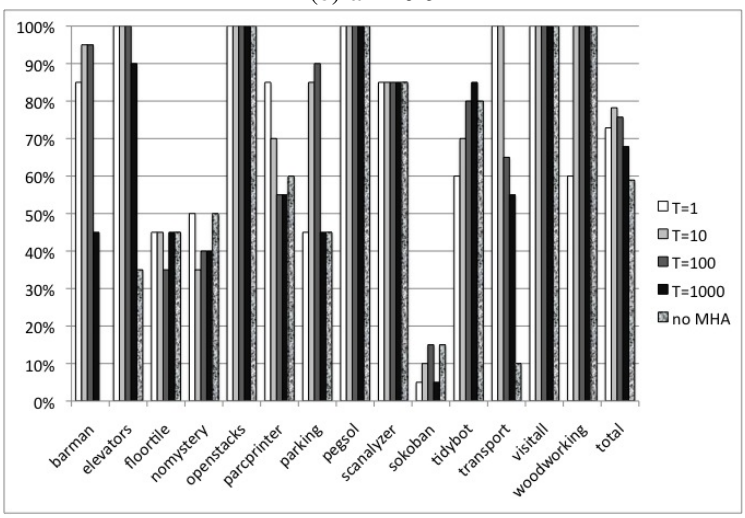
- Using $W = 1$, in most domains the performance peaks at relatively low temperatures of $T = 10$ and $T = 100$. The exceptions are Parcprinter and Tidybot, where the performance peaks at the lowest tested temperature of $T = 1$, and at the highest temperature of $T = 1000$, respectively.



(a) $w = 0$



(b) $w = 0.5$



(c) $w = 1$

Figure 3.10: Coverage of MHA versus uniform action selection (no MHA) varying T [1 10 100 1000 in all (a) – (c)] and w [0 in (a), 0.5 in (b), 1 in (c)]

3.7.2 Monte Carlo Deadlock Avoidance (MDA)

MDA (Nakhost & Müller, 2009) tries to avoid dead-end states by penalizing actions that appear in failed walks. Let $S(a)$ and $F(a)$ be the number of 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.

Figure 3.11 shows the coverage of MDA varying temperature. In this experiment, Arvand-2013 uses $n = 1$, $p_{eval} = 1$, ALR($\epsilon = 0.1$) and AGR. The data shows that:

- MDA clearly increases the coverage in Parking and Tidybot. Some settings of MDA also outperform uniform action selection (no MDA) in Parcprinter, Nomystery, Elevators and Transport. However, the results seem noisy for these domains.
- In IPC-2011, MDA is not as effective as MHA. While MHA can increase the coverage by as much as 90% (Figure 3.10), the largest improvement achieved by MDA is 25% in Parking.

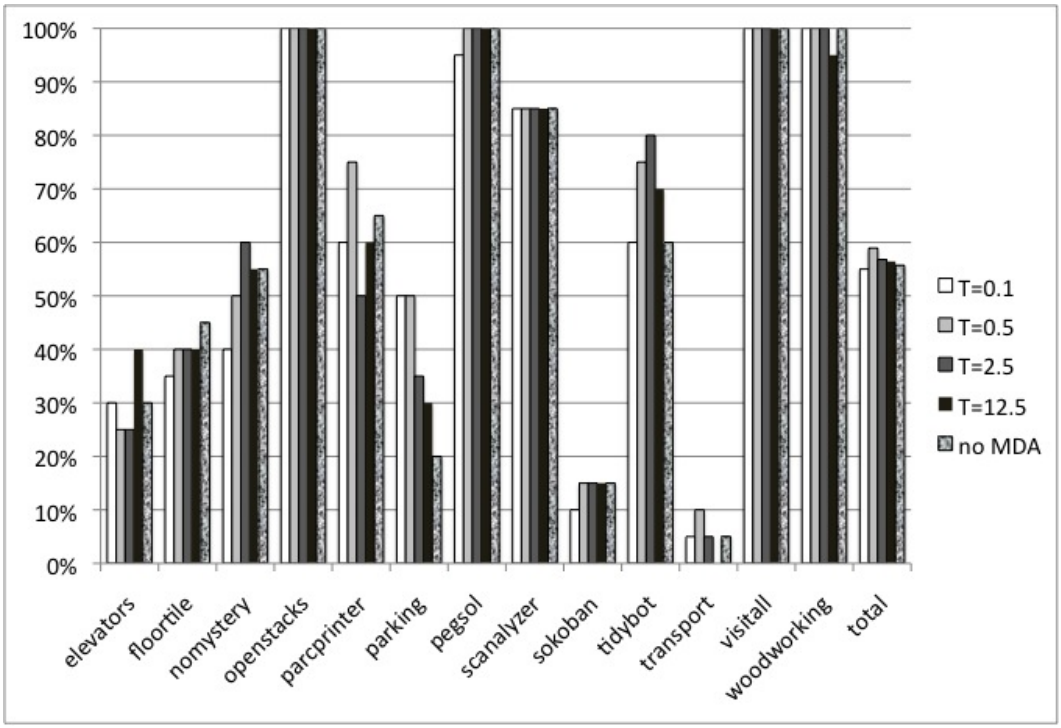
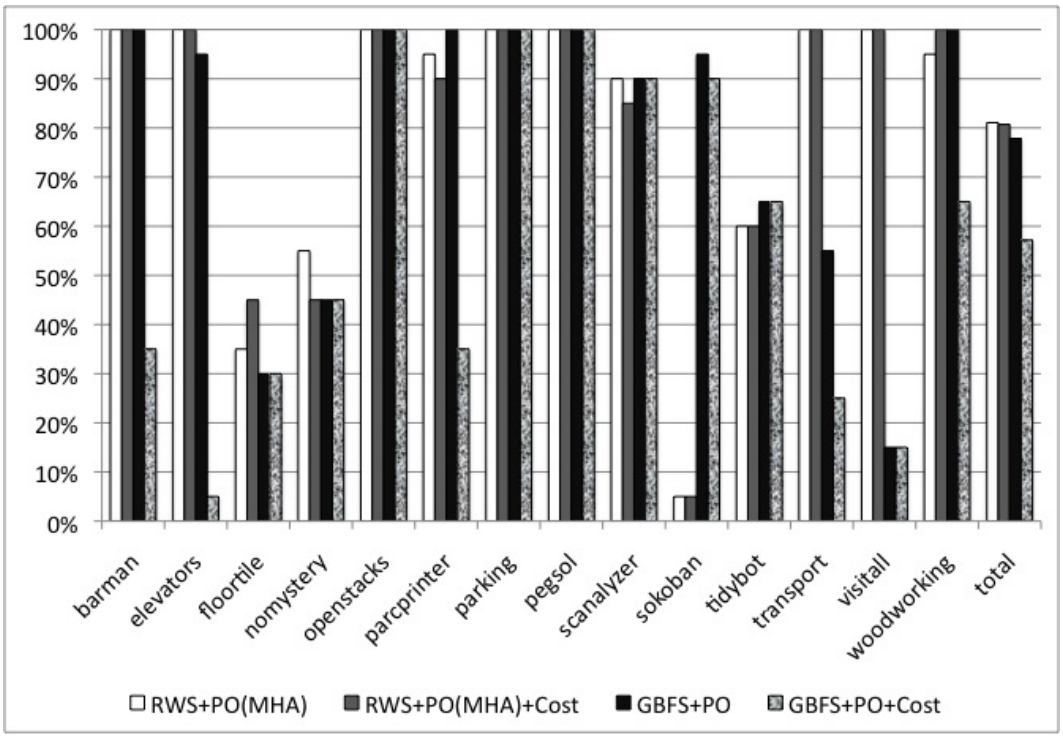


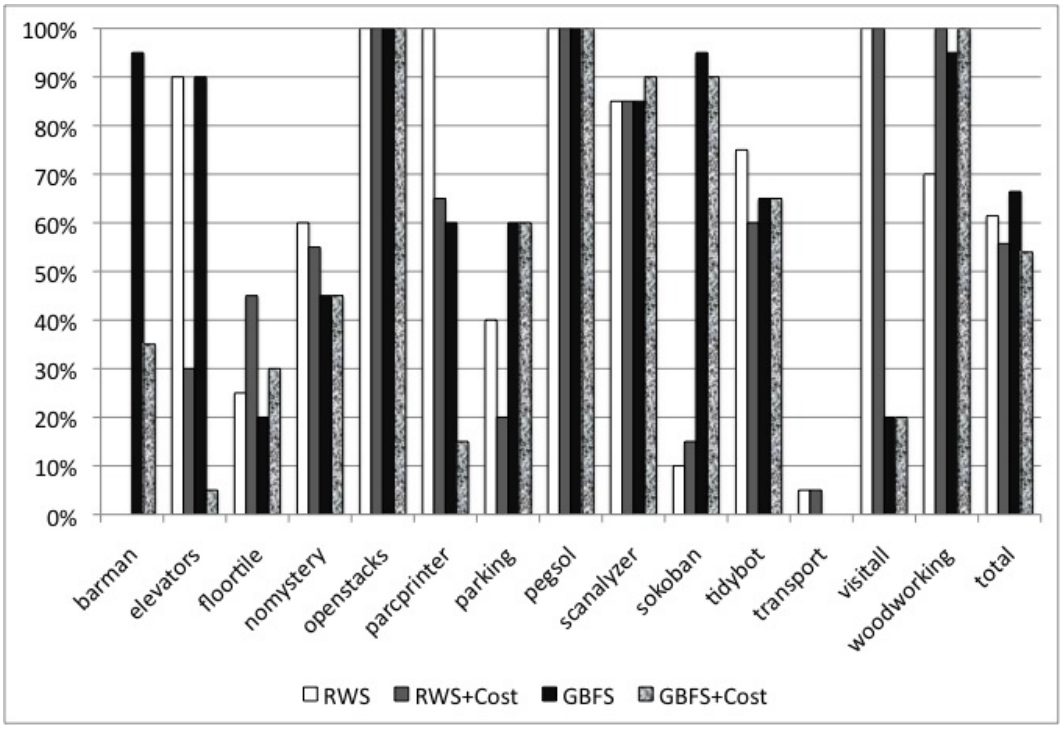
Figure 3.11: Coverage of MDA versus uniform action selection (no MDA) varying T [0.1 0.5 2.5 12.5]

3.8 The Effect of the Heuristic Function on RW planning

All the previous research on RW planning has been focused on cost-ignorant h^{FF}, the FF heuristic (Hoffmann & Nebel, 2001) that assumes that all the actions have unit cost. Therefore, it is unclear



(a)



(b)

Figure 3.12: Coverage of Arvand-2013 (RWS) and GBFS with (a) and without (b) preferred operators (PO) using cost-sensitive (+cost) and cost-ignorant h^{FF}.

whether random walks are only effective for h^{FF} or whether they are also useful with other heuristic functions. The interaction of RW search with cost-sensitive heuristics is also unknown. Nakhost and Müller (2012) partially address these questions by developing a theoretical model in which the effectiveness of random walks in escaping a plateau is a function of search space characteristics that are independent of the heuristic function used, namely regress factor and the shortest escape path. However, the use of random walks is not limited to plateaus, and even if it was, a heuristic function can affect the performance if it only forms plateaus in parts of search space where regress factors tend to be either high or low. More fundamentally, the shape and the size of the plateaus change with different heuristic functions. Also, heuristic functions may not form any plateau or local minima in some cases.

This section takes an experimental approach to evaluating the effect of the heuristic function: it tests Arvand-2013 with different heuristic functions and compares the results with common systematic search algorithms used in planning.

3.8.1 Cost Sensitivity

Most of the common heuristic functions provide both cost-sensitive and cost-ignorant estimations. While the former estimates the cost of the cheapest path to the goal, the latter estimates the length of the shortest path to the goal. Once a heuristic function finds a solution for an abstract problem, the cost and the length of the solution can be used respectively for cost-sensitive and cost-ignorant estimations. It is known that cost estimators usually lead to better quality plans at the expense of larger runtime (Wilt & Ruml, 2011; Cushing, Benton, & Kambhampati, 2011). That is why most of the anytime planners such as LAMA and recent versions of FD first use a cost-ignorant heuristic to find the first solution and then switch to cost-sensitive heuristics to improve the quality. Figure 3.12 compares Arvand-2013 with GBFS using both cost-sensitive and cost-ignorant implementations of h^{FF}. RWS in Figure 3.12 denotes a setting of Arvand-2013 that uses greedy jumping ($n = 1$), the evaluation rate $p_{eval} = 1$, adaptive local restarting ALR($\epsilon = 0.1$), and adaptive global restarting. RWS+PO(MHA) uses the same parameters as well as MHA(1,10). All tested algorithms use the same cost-sensitive h^{FF} introduced by LAMA, which uses the actual action costs plus one in order to avoid problems with zero-cost actions (Richter & Westphal, 2010). The key observations are as follows:

- Although cost-sensitive h^{FF} is usually detrimental, it increases the coverage of both RWS and GBFS in two domains: Floortile and Woodworking. The reason is that the action costs in Floortile and Woodworking provide useful information to solve the task. In Floortile, multiple robots should paint a grid of tiles without stepping on already painted tiles. The only way to solve the tasks is to start painting from the upmost row and proceed downward. The cost of moving up is larger than moving in any other direction. Therefore, the states in which the robot is already in an upper location have a lower cost estimation: the relaxed plan includes fewer

"move up" actions. That is why, unlike cost-ignorant heuristics, which have no direction preference, a cost-sensitive heuristic tends to move the robot up, if possible. Nakhost et al. (2012) report a similar behaviour for cost-augmented encodings of resource constrained problems, where an action cost reflects the amount of the resource consumed by the action. Using this encoding, a cost-sensitive heuristic decreases the chance of running out of resources (hitting deadends) by choosing cheaper paths.

- Arvand-2013 is less susceptible to cost-sensitive estimations when it uses preferred operators: while the cost-sensitive heuristic decreases the coverage of RWS by 6% (16 problems), it decreases the coverage of MHA only by 0.3% (1 problem). This is not the case for GBFS: once preferred operators are used, the total coverage gap between cost-ignorant and cost-sensitive versions increases from 35 (12%) to 58 (21%) problems.

3.8.2 The Effect of Changing Heuristic Function in RWS

The next experiment studies the effect of different heuristic functions on RWS. The tested heuristic functions are h^{cea}(Helmert & Geffner, 2008), h^{CG}(Helmert, 2006), $h^{M\&S}$ (Helmert, Haslum, & Hoffmann, 2007a), and $h^{goalcount}$, the goal count heuristic. The first two are among the most common heuristic functions used in planning. $h^{M\&S}$ is also tested as a representative of abstraction based heuristics such as pattern databases (PDB), which are very common in other heuristic search domains. Finally, $h^{goalcount}$ is tested for being one of the simplest heuristic functions available for planning: it counts the number of goal propositions satisfied at the evaluated state. In this experiment, all the tested heuristics ignore the action costs. Figure 3.13 shows the coverage for RWS and GBFS with and without preferred operators, varying the heuristic function.

The results confirm that the effectiveness of RW planning is not limited to h^{FF}, and also show that the advantage of RWS over GBFS increases when h^{FF} is replaced by h^{cea}: the overall 3% (9 problems; Figure 3.12) coverage lead of RWS over GBFS increases to 13% (37 problems) with h^{cea}. The following summarizes the results for each tested heuristic. $PO(h)$ denotes the preferred operators obtained from heuristic function h.

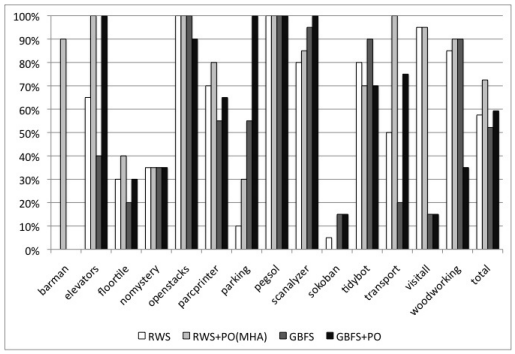
- h^{cea}: RWS outperforms GBFS, both with and without preferred operators, solving from 2 (10%) to 18 (90%) more problems in Barman, Elevators, Visitall, Transport, Floortile, Parcprinter and Woodworking. Parking and Sokoban are the only domains where GBFS clearly wins by solving 2 (20%) to 9 (45%) more problems. In total, RWS solves 14 (5%) more problems than GBFS and RWS+PO solves 37 (13%) more problems than GBFS+PO, a version of GBFS that uses two queues: one for preferred successors and one for all successors (see Chapter 1).

- h^{CG}: the results are mixed: while RWS solves 4 (20%) to 17 (85%) more problems than GBFS in Elevators, Parcprinter, Visitall, and Woodworking, GBFS solves 2 (10%) to 17

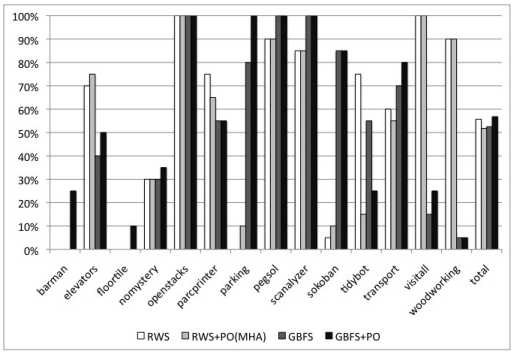
(75%) more problems in Parking, Pegsols, Scanalyzer, Sokoban, and Transport. In contrast to $PO(h_{Cea})$ and $PO(h_{\mathit{ff}})$, which significantly improve RWS in most domains, $PO(h_{cg})$ has a negative effect in Tidybot, Parcprinter and Transport and either no or little positive effect in other domains. For GBFS, although $PO(h_{cg})$ is only detrimental in Tidybot, it is still less effective than $PO(h_{Cea})$ and $PO(h_{\mathit{ff}})$ overall.

- $h^{goalcount}$: While GBFS solves 2 (10%) to 6 (30%) more problems in most domains, RWS solves 14 (70%) more problems in Openstacks. Overall, RWS is about level with GBFS, solving 5 (2%) fewer problems.

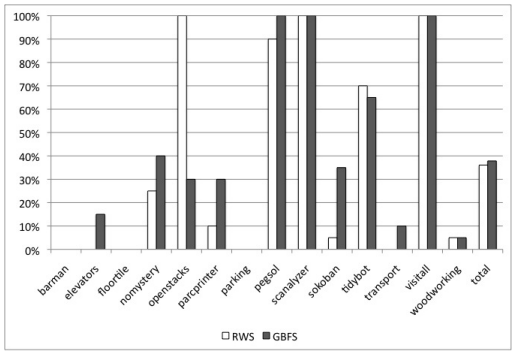
- $h^{M\&S}$: while both RWS and GBFS are much worse than with the other tested heuristics, RWS never performs better than GBFS. In Parking and Transport, GBFS solves respectively 17 (85%) and 11 (55%) more problems than RWS. In total, GBFS solves 32 (11%) more problems. It is still unclear if this performance gap is due to a fundamental shortcoming of RW dealing with admissible and consistent heuristics such as $h^{M\&S}$, or the effect of the settings used in Arvand-2013, which are tuned with common heuristic functions in satisficing planning.



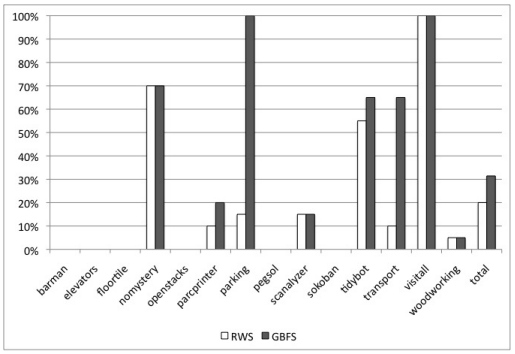
(a) h_{cea}



(b) h^{CG}



(a) $h^{goalcount}$



(b) $h^{M\&S}$

Figure 3.13: Coverage of RWS and GBFS with and without PO (if possible) varying the heuristic function [h_{cea} in (a), h^{CG} in (b), $h^{goalcount}$ in (c), $h^{M\&S}$ in (d)]

While the experimental analyses of the individual components of RW search provides many

practical insights regarding effective algorithms for each component, two observations stand out:

1. The importance of adaptive systems: since RWS is not designed for a specific domain, it is crucial to be able to adapt its parameters to the given tasks. This becomes more important for search algorithms like RWS that instead of systematically exploring all the 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. The key idea is to use an online learning algorithm that *rewards* the *progress* in the search space.

2. The big effect of action selection biasing: as the theory developed in Chapter 2 predicts and experiments in Section 3.7 confirm, action selection biasing can significantly improve the performance of RW search. MHA serves as a successful example of a biasing technique. Our intuition is that action selection biasing towards preferred operators reduces the effective regress factor, i.e., MHA increases the chance of sampling a state with a smaller goal distance.

3.9 Arvand-2013 as a planning system

The focus so far has been on isolated studies of different components of RW search. Planning, however, is not only about designing heuristic search algorithms: building powerful planning systems is also important. The experiments in this section compare Arvand-2013 as a complete planning system with other state-of-the-art planners. To achieve its full potential, Arvand-2013 is augmented with a learning subsystem, first used in Arvand-2011 (Nakhost et al., 2011), that finds the best configuration for a given problem.

3.9.1 Configuration Learner

Similar to *adaptive local restarting* introduced in Section 3.4.3, the problem of selecting a configuration for RW search can be viewed as an instance of a *multi-armed bandit* problem. Instead of using a single configuration, Arvand-2013 starts with an input set of configurations C. Before running a search episode, the planner first uses a bandit algorithm to select the configuration $c \in C$. When the episode ends, the performance of c is evaluated using h_{min}, the minimum heuristic value reached. If h_i is the heuristic value of the initial state, then the *reward* assigned to c is $max(0, 1 - (h_i/h_{min}))$. In the current implementation, the *upper confidence bounds* (UCB) algorithm (Auer, Cesa-Bianchi, & Fischer, 2002) is used for configuration selection. UCB uses the following formula to select the next configuration:

$$\arg\max_{c \in C} \left(Q(c) + \beta \sqrt{\frac{n(c)}{N}} \right)$$

Here, $Q(c)$ is the average reward assigned to c, $n(c)$ is the number of episodes configured with c, N is the total number of episodes run so far, and $\beta \geq 0$ is the *exploration constant*, which balances exploration with exploitation: larger β results in more exploration. After some initial experiments, the default value of β is set to 0.5. Table 3.1 lists the three configurations used by Arvand-2013. All three configurations use h^{FF}. For extensive tests evaluating a similar configuration learner for the planning system ArvandHerd see (Valenzano et al., 2012).

Configuration	Bias	Eval. Rate	Jumping	Glob. Restarts	Loc. Restarts
Config 1	$MHA(w = 1, T = 10)$	p_{eval} = 1	$n = 1$	AGR	ALR($\epsilon = 0.1$)
Config 2	$MHA(w = 1, T = 10)$	p_{eval} = 0.5	$n = 100$	AGR	ALR($\epsilon = 0.1$)
Config 3	$MDA(T = 0.5)$	p_{eval} = 0	$n = 1$	AGR	ALR($\epsilon = 0.1$)

Table 3.1: Configurations used in Arvand-2013

3.9.2 Experiments on All IPC Benchmarks

Experiments compare Arvand-2013 with the top three planners which solved the most problems in IPC-2011: LAMA-2011, FDSS2, and Probe, as well as a new version of the RW planner Roamer, which ranked 6$^{\text{th}}$ by this measure. The IPC-2011 version of Roamer were affected by a bug in the PDDL-to-SAS+ translator used. This issue has been fixed in the version of the planner used in the current experiments. Table 3.2 shows the coverage for all tested planners in all IPC domains. For domains that were used in more than one competition, the instances from the most recent competition are used, since they tend to be larger and harder. Table 3.2 shows the number of tasks in each domain in parentheses. Since Probe does not support derived predicates, the four domains that use these are scored separately. The key observations are as follows:

- Arvand-2013 is a strong and competitive system, solving 1547 problems without derived predicates plus 114 problems with derived predicates. Arvand-2013 is about level with the state-of-the-art planners LAMA-2011 ($1540 + 119$) and FDSS2 ($1530 + 135$). The fact that FDSS2 is a sequential portfolio system running different BFS algorithms using different heuristic functions while Arvand-2013 uses h^{FF} only, makes these results even more impressive. Arvand-2013 clearly outperforms Probe ($1422 + 0$) solving 118 more problems with no derived predicates, and beats Roamer ($1507 + 128$) by 26 problems.

- In 33 out of 45 domains, Arvand-2013 achieves the largest coverage and in 5 domains: Airport, Notankage, Tankage, Storage, and Optical telegraphs, more than any other tested planner, Arvand-2013 is the single winner of the domain. Arvand-2013 solves between 1 to 6 problems more than the second best planner in these domains: this result and the fact that Arvand-2013 uses only little memory make Arvand-2013 a good candidate for a sequential or parallel portfolio.

- Arvand-2013 performs much weaker than the other tested planners in Sokoban and PSR. In these domains, in order to achieve the goal or *escape* from a plateau, very often a specific

sequence of actions with a fixed ordering should be executed. Random walks have a very low chance to find such a sequence. An interesting future work is to use memory to avoid re-exploring irrelevant parts of the search space, and hence, increase the escape chance.

3.10 Conclusions

The general structure of RWS is based on local search. The key components of the search determining the neighborhood relation and step function are identified and experimentally explored. The experiments provide practical insight regarding the effect of the parameters and adaptive techniques to adjust them. Tests on AD and IPC domains show that RWS is a strong alternative to systematic search.

Domain	Arvand-2013	LAMA-2011	FDSS2	Probe	Roamer
Airport (50)	**44**	31	43	38	31
Assembly (30)	30	30	30	30	30
Barman (20)	20	20	15	20	18
Blocks (35)	35	35	35	35	35
Cyber Security (30)	30	30	30	27	30
Depot (22)	19	20	19	**22**	17
Driverlog (20)	20	20	20	20	20
Elevators (20)	20	20	18	20	16
Floortile (20)	4	5	**6**	5	2
Freecell (80)	80	79	80	79	79
Grid (5)	5	5	5	5	5
Gripper (20)	20	20	20	20	20
Logistics (28)	28	28	28	28	28
Miconic (150)	150	150	150	150	150
Miconic Full ADL (150)	139	136	139	45	137
Miconic Simple ADL (150)	150	150	150	150	150
Movie (30)	30	30	30	30	30
Mprime (35)	35	35	35	35	35
Mystery (30)	19	19	19	14	19
Nomystery (20)	15	13	13	7	10
Notankage (50)	**50**	44	43	45	44
Openstacks (20)	20	20	15	13	20
Parcprinter (20)	20	20	20	13	7
Parking (20)	18	**20**	18	16	15
Pathways (30)	30	30	30	29	28
Pegsol (20)	20	20	20	20	19
PSR Small (50)	50	50	50	50	50
Rovers (40)	40	40	40	40	40
Satellite (36)	32	35	36	31	36
Scanalyzer (20)	19	20	20	18	20
Schedule (150)	150	150	150	150	150
Sokoban (20)	1	**19**	18	17	15
Storage (30)	**30**	19	21	22	27
Tankage (50)	**44**	41	41	43	39
Tidybot (20)	15	16	15	**18**	15
TPP (30)	30	30	30	30	30
Transport (20)	19	17	16	19	18
Trucks (30)	16	13	**19**	8	12
Visitall (20)	20	20	6	20	20
Woodworking (20)	15	20	20	20	20
Zenotravel (20)	20	20	20	20	20
Total (no derived predicate) (1661)	**1552**	1540	1533	1422	1507
Optical Telegraphs (48)	**8**	4	6	-	2
Philosophers (48)	44	34	48	-	48
PSR Large (50)	19	31	31	-	28
PSR Middle (50)	43	50	50	-	50
Total (1857)	1666	1659	**1668**	1422	1635

Table 3.2: Number of problems solved in all IPC.

Chapter 4

Resource-constrained Planning: a Random Walk Planning Approach

While heuristic search, mostly based on standard greedy algorithms such as GBFS, is currently the most common method for most varieties of planning, its ability to solve critically resource-constrained problems is limited: current planning heuristics are bad at dealing with this kind of structure. To address this, one can try to devise better heuristics. An alternative approach is to change the nature of the search instead. The exploration power of random walks and their ability to handle uninformative and misleading heuristic values makes RW search a strong alternative. The current chapter introduces new techniques *smart restarts* and *on-path search continuation* implemented in the RW planner Arvand-RC, also referred to as A2, which significantly improves the state of the art in resource-constrained planning (RCP). Other contributions include a new benchmark suite controlling C, a numeric problem feature that characterizes the resource constrainedness, an extended notion of C for tasks with multiple resources, and a large-scale study of the performance of a diverse set of planning methods as a function of C.

4.1 Introduction

Planning is the art of acting intelligently, thus a key aspect of it is the prudent consumption of resources. Indeed, planning with resources, and more generally numeric planning, is one of the most prominent topics in the planning literature (Koehler, 1998; Haslum & Geffner, 2001; Fox & Long, 2003; Hoffmann, 2003; Gerevini et al., 2003; Edelkamp, 2003, 2004; Coles, Fox, Long, & Smith, 2008; Dvorak & Barták, 2010). Many applications of planning involve controlling autonomous agents with limited resources such as energy, fuel, money, and/or time.

Here, we investigate *consumable* resources (Haslum & Geffner, 2001). These cannot be replenished, i.e., the planner must make do with the initial supply. That situation occurs quite frequently. Consider, for example, the energy supply in underwater robotics, the fuel supply in space travel, fixed project budgets, and fixed delivery deadlines.

We consider the special case where all resources are consumable. We will refer to this as *resource-constrained planning (RCP)*. This specific case is still relevant, but has been given scant attention in the literature (all existing approaches deal with much more general settings). In particular, only two previous studies (Hoffmann, Kautz, Gomes, & Selman, 2007; Gerevini, Saetti, & Serina, 2008) consider *resource constrainedness*: the amount by which the initial resource supply exceeds the minimum need. This can be measured in terms of a constant $C \geq 1$, namely the maximum number by which we can divide the resource supply without rendering the task unsolvable. The closer C is to 1, the more *constrained* is the problem; $C = 1$ enforces minimal resource consumption. Obviously, C links to the complexity of (approximate) optimization of resource consumption in the underlying domain. In practice, one would expect planning to become harder as C approaches 1. But what is the state of the art in this situation? Which techniques tend to work well, and which do not? Can we design tailored techniques without losing performance elsewhere?

The literature hardly answers these questions. In the IPC benchmarks, C is not a controlled quantity. The single exception is IPC-2011 NoMystery contributed as part of (an earlier stage of) this work. Only two previous works (Hoffmann et al., 2007) and (Gerevini et al., 2008) run experiments controlling C. Each considers only one domain, with a single resource. Each runs a small selection of planners, drawing conclusions about which of these is most effective, but not about how we could design algorithms that work better.

4.1.1 Contributions

We herein begin to address RCP in more detail. We generalize its investigation to domains with multiple resources. We extend the existing suite of RCP benchmarks with controlled C, introducing one new domain, and generalizing NoMystery to include more than one resource. The benchmarks and generators are publicly available. Controlling C requires domain-specific consumption-optimal solvers, to determine the minimum amount of resources needed. Hence these new benchmarks contribute considerable implementation work.

We conduct a large-scale study of the current state of the art in RCP as a function of C. Amongst other observations, we show that, despite all the new developments in satisficing planning, Hoffmann et al.'s and Gerevini et al.'s conclusion about the performance of planners using delete-relaxation heuristics, which declines dramatically as C approaches 1, still holds. Together with the previously observed *resource persistence* in delete-relaxed plans (Coles et al., 2008) – which act as if what was once true will remain true forever – this motivates our attempt to emphasize RW search. The working hypothesis is that adding more *exploration* of the search space, as opposed to *exploitation* of the heuristic, will make planners less susceptible to errors in that heuristic.

We design two improvements to RW search introduced in Chapter 3. These improvements, called *smart restarts (SR)* and *on-path search continuation (OPSC)*, aim to improve the balance between exploitation and exploration, by trading off previous progress against the risk of repeating

previous mistakes. We call the resulting planner Arvand-RC. In our RCP benchmark suite, Arvand-RC almost universally outperforms all other planners when C is close to 1. This goes to show that algorithmic improvements *are* possible in this specific situation.

To verify whether this improvement comes at the price of performance losses in other planning domains, we run tests on the IPC-2011 benchmarks. On-path search continuation can be detrimental, while smart restarts improve performance there as well. We study the parameters influencing Arvand-RC's performance, and the effect of interactions between multiple resources.

We next define RCP and summarize the relevant literature. We then discuss resource constrainedness C, and describe our RCP benchmark suite. We explain our enhancements to Arvand, report our experiments, and conclude.

4.2 Planning with Resources

We define our RCP formalism, and summarize the most relevant prior work in the area. We outline the role of planning with resources in the IPC benchmarks.

4.2.1 RCP Formalism

In *resource-constrained planning (RCP)*, STRIPS planning tasks are extended with a set R of resource identifiers as well as functions $i : R \mapsto \mathbb{Q}_{\geq 0}$ and $u : A \times R \mapsto \mathbb{Q}_{\geq 0}$. $i(r)$ denotes the initial amount of resource $r \in R$, and $u(a, r)$ is the amount of r consumed when executing action a. Sufficient resource availability is requested by additional *resource preconditions* taking the form $s(r) \geq u(a, r)$.

Clearly, RCP is related to optimization of resource consumption, as a decision problem. But it is not, in general, the same thing in practice. In many applications, the primary objective is to optimize some other criterion, such as timespan or amount of data collected. The resources then only serve to encode a fixed budget that the plan has to make do with. This is the situation we aim at addressing here.

4.2.2 Previous Work

RCP is a special case of planning with resources, which in general allows resource production in addition to consumption. In turn, planning with resources is a special case of numeric planning. The latter was emphasized in the IPC-2002 (Fox & Long, 2003). A prominent line of planners (Hoffmann, 2003; Gerevini et al., 2003; Edelkamp, 2004; Gerevini et al., 2008) handling this uses direct extensions of delete-relexation heuristics, via "numeric relaxed planning graphs". As observed by Coles et al. (Coles et al., 2008), this kind of heuristic suffers from *resource persistence*. Relaxed plans act as if resource values persist forever, and are therefore fundamentally unsuited for reasoning about resource consumption.

Considering not general numeric planning, but planning with resources more specifically, Coles et al. address resource persistence in their LP-RPG planner. They extend relaxed planning with more informed numeric reasoning via linear programming (LP). In a nutshell, the numeric relaxed planning graph is encoded into *mixed integer linear program* (MILP), and the LP relaxation is used to provide informed upper and lower bounds for each resource. As Coles et al. point out, this is useful to avoid detrimental phenomena arising from the interaction between resource producers and consumers. However, as our experiments will show, for tightly constrained RCP this more informed heuristic is unfortunately still not good enough.

Other work on planning with resources makes use of very different heuristics, based on Graphplan (Koehler, 1998) and the h^m family (Haslum & Geffner, 2001). The Filuta system (Dvorak & Barták, 2010) uses dedicated reasoners to resolve resource conflicts during a plan-space search; there is no heuristic guidance estimating resource consumption.

4.2.3 IPC Benchmarks

Many IPC benchmarks incorporate planning with resources in some form. However, only few of these are RCP domains as considered here, namely: IPC-1998 *Mystery* and *Mprime* (fuel); IPC-2002 *Satellite* (fuel); IPC-2006 *TPP* (money) and *Trucks* (time, i.e., strict delivery deadlines); IPC-2011 *NoMystery* (fuel). IPC-2002 *Rovers* has energy consumption, but includes also a "recharge" operator.

Several other IPC domains feature resource consumption, yet impose it as an optimization criterion, not as a hard constraint. Since this does not force satisficing planners to make do with a given budget, it is quite different from the situation we are interested in here. That said, anytime planners may incrementally reduce resource consumption, and eventually bring it below the thresholds required. We evaluate this option, using LAMA-2011 in our experiments.

4.3 Resource Constrainedness

We formalize RCP constrainedness in terms of a parameter C. We describe our benchmark suite controlling C, and summarize previous findings from doing such control.

4.3.1 Characterizing Resource Constrainedness

For the case of a single resource, $R = \{r\}$, considered in previous work (Hoffmann et al., 2007; Gerevini et al., 2008), defining *resource constrainedness* C is straightforward. C should measure the factor by which the initial resource supply, $i(r)$, exceeds the minimum need. This can be derived from the equation $\frac{i(r)}{C} = M$, where M is the minimal resource consumption of any plan for the task.

In case there are several resources, matters are not that simple since there is no unique "minimum need". A small amount of resource r might be compensated for by a big amount of resource r'. In

other words, to define a unique value M, we would need to aggregate resource values (e.g., by their sum or maximum). However, there is no one aggregation method that is adequate across all possible domains. The solution we propose is to define C based on the notion of pareto-minimality. Reformulating $\frac{i(r)}{C} = M$ to $\max\{C \mid \frac{i(r)}{C} \geq M\}$, we observe that the above definition corresponds to downscaling $i(r)$ until it hits the pareto frontier given by the single pareto-minimal solution M. We generalize this simply by downscaling the whole vector i until it hits the more general pareto frontier.

For assignments $M, M' : R \mapsto \mathbb{Q}_{\geq 0}$, we write $M \geq M'$ to denote pointwise domination. M is *pareto-minimal* if: (a) the task is solvable when setting $i := M$; and (b) for any M' where $M \geq M'$, and $M(r) > M'(r)$ for at least one $r \in R$, setting $i := M'$ renders the task unsolvable. Denoting by $\mathcal{M}$ the set of all pareto-minimal assignments, we define C as $\max\{C \mid \exists M \in \mathcal{M} : \frac{i(r)}{C} \geq M\}$. In other words, C is the largest factor by which we can downscale the initial resource supply without rendering the task unsolvable.

Computing C for a given planning task is, obviously, hard. Such computation may be useful (e.g., to configure planners), but is not our focus here. Instead, we wish to design benchmarks controlling C, in order to investigate how planning algorithms scale in that parameter. Our methodology for doing so is to implement domain-specific solvers computing $\mathcal{M}$, i.e., all pareto-minimal resource supplies. Benchmark instances with resource constrainedness C are obtained by selecting some $M \in \mathcal{M}$, and setting $i := C \times M$.

4.3.2 RCP Benchmark Domains Controlling C

Given the need to develop domain-specific consumption-optimal solvers, it is not easy to implement benchmarks controlling C. Our RCP benchmark suite currently consists of three domains, namely *NoMystery*, *TPP*, and *Rovers*. The generators as well as all test instances used here are available at `http://code.google.com/p/rcp-benchmark-suite/`.

TPP is the same domain as used in IPC-2006. An agent with a given budget needs to buy a set of products from different markets, selling at different prices. The resource is money. Gerevini et al. (2008) implemented a generator allowing to control C, however that generator is not available anymore. Our test suite contains the original instances.

NoMystery is the same domain as used in IPC-2011. A set of packages must be transported between nodes in a graph; actions move trucks along edges, or load/unload packages. Each truck has its own fuel supply, which it consumes when moving. In the original generator we provided for IPC-2011, instances were restricted to have only a single truck (and thus only a single resource). Our extended generator removes this restriction. Our test instances contain two trucks, which is already quite challenging to solve for the domain-specific optimal solver as well as for state of the art domain-independent planners.

Rovers is the domain used in IPC-2002, except that we removed the "recharge" operator to fit

the domain into the RCP framework. The goal is to take a number of rock samples and images, and transfer them to a lander. Each rover has an energy supply, and all actions consume energy. We implemented a generator from scratch, allowing an arbitrary number of rovers. For the same reasons as in NoMystery, our test instances contain two rovers.

In all three domains, the resource amounts in our test instances (initial supply and per-action consumption) are integer, since Arvand-RC does not handle numeric variables. The integer values are encoded into STRIPS in the straightforward fashion, using one proposition per possible value. Arvand-RC is not even explicitly aware of the resources. As we will see, its performance is very good despite this. Every other planner in our experiments is supplied with the encoding leading to best performance.

4.3.3 Previous Findings when Controlling C

In previous experiments controlling C, Hoffmann et al. (2007) use an older version of the NoMystery domain, whose generator contained bugs, while Gerevini et al. (2008) use the TPP benchmark described above. Both experiments run satisficing heuristic search planners using relaxation heuristics: different versions of FF (Hoffmann, 2003), LPG (Gerevini et al., 2003), and MIPS (Edelkamp, 2003). Both observe that the performance of all these planners degrades dramatically as C approaches 1, yet less so for LPG than for the other planners. The latter observation is part of the motivation for our work. The similarity of the heuristics employed – which as observed by Coles et al. (2008) are not informative here – suggests that the advantage of LPG is mainly due to its local search paradigm. We show that one can push the performance margins far higher still, based on highly explorative RW search.

4.4 Improving RW Search

The Arvand-RC planner is built on top of Arvand-2009, which is a RW planner developed as part of this thesis. To understand the techniques introduced here it is enough to know that Arvand-2009 uses the general RWS framework: local search empowered by RW sampling, and $p_{eval} = 0$, evaluating only the endpoints of random walks. For more details, see Chapter 6. Arvand-RC introduces two improvements on Arvand-2009 which work especially well for RCP: On-Path Search Continuation (OPSC) and Smart Restarts (SR).

4.4.1 On-Path Search Continuation

Like all algorithms explored in Chapter 3, Arvand-2009 uses what we call *End-Point Search Continuation* (EPSC): after jumping to a sampled endpoint e, Arvand-2009 commits to all the actions on the path to e. The drawback for RCP is that, if some of the random actions leading to e consume too many resources and the problem becomes unsolvable, then all search effort from this point until

the next restart is wasted. This can cause severe search inefficiencies when resources are tightly constrained.

The new technique of *On-Path Search Continuation* (OPSC) avoids commitment to all actions leading to e, while still benefiting from the guidance of the selected path to e. Let $S = \{s_0, s_1, \ldots, s_k\}$ be the states visited along the path from the initial state $s_0 = I$ to the current endpoint $s_k = e$. EPSC chooses s_k as the starting point of all random walks within the current search step. OPSC generalizes this by choosing, for each individual random walk, the starting point according to some fixed probability distribution over S. In all the experiments reported here, OPSC uses a uniform probability distribution over S. The method for updating S is analogous to EPSC. In each search step, after running the random walks, OPSC selects an endpoint e with minimal h^{FF} value. S is then changed to the path from I to e.

4.4.2 Smart Restarts

The second innovation in Arvand-RC is a modified restarting strategy. Arvand-2009 restarts from scratch in every new search episode, discarding all previous information. However, previous episodes may contain valuable partial paths, which ultimately ended in failure only because of bad actions later on. *Smart Restarts (SR)* in Arvand-RC try to preserve information by maintaining a fixed-capacity *pool* of the *most promising episodes* so far.[1] Smart restarts begin at a state along the trajectory of such an earlier episode, instead of returning to the initial state every time. The method selects a random episode from the pool, then selects a random state along that episode as the next restarting point. If the pool is empty, it selects the initial state.

Let p denote the fixed *pool capacity*, i.e., the maximum number of episodes stored in the pool. The "worst" episode in the pool is replaced whenever the pool is full and a new "better" episode is discovered. The quality of an episode is defined as the smallest h^{FF} value along its trajectory $\langle s_0, \ldots, s_n \rangle$. Let s_{min} be the earliest state among those states with minimum heuristic value. Then the partial trace $\langle s_0, \ldots, s_{min} \rangle$ is the candidate for inclusion into the pool. The motivation for defining s_{min} as the *earliest* such state is that the amount of resources consumed grows monotonically along an episode.

Restarting from a pool of episodes balances exploration and exploitation, and therefore can increase the chance to quickly reach promising regions of the search space. If the pool capacity p is small, then the search is more greedy and concentrates on the hitherto best traces. Large pools lead to more exploration, which intuitively might improve performance on hard instances given sufficient time. In contrast, with limited time, more focus on exploitation should be beneficial.

Arvand-RC does not use smart restarts until an initial number N of search episodes has been completed. This avoids a heavy bias towards these early episodes.

[1] Solution-guided search (Heckman & Beck, 2011)) is a successful related algorithm in Scheduling. It differs significantly in context and technical details.

4.5 Experiments

Based on initial tests, we set $N = 50$ and $p = 50$ in Arvand-RC. For all the experiments *MHA*$(T = 10, W = 0)$ is used to run RW. All the other parameters inherited from Arvand are set to their default values. We run tests on the three RCP domains NoMystery, TPP and Rovers, as well as on IPC-2011 domains, on a 2.5 GHz machine with 2 GB memory limit. The runtime cut-off was set to 30 minutes for small instances, and to 40 minutes for large instances (see below). Results for the randomized planners LPG, Arvand-2009, Arvand-RC and their variants are averaged over 10 runs per instance.

For the RCP domains, we used several different encodings. The first uses propositions to represent the integer-valued resource levels; the second uses numeric variables. For tasks with a single resource r, we created cost-augmented variants of both, setting the cost of each action a to its consumption $u(a, r)$. For LAMA, we also supplied an anytime cost-augmented variant, requiring to minimize consumption of r, removing the resource preconditions $s(r) \geq u(a, r)$ but counting a task as solved only if the returned plan satisfied these. To exploit the ability of LPRPGP of handling preferences, we also used an encoding where the resource preconditions are changed to preferences and a unit cost is assigned to each preference violation; therefore, reducing the cost of violations translates to decreasing the number of actions that over-consume the resources. For each (planner, domain) pair, we show data for the most effective encoding for that pair.

A wide range of planners was tested, including Arvand-2009 and Arvand-RC as well as two variants Arvand-RC(SR) and Arvand-RC(OPSC) which use only one of the two new techniques; FF (Hoffmann & Nebel, 2001), the top performer at IPC-2000; LPG (Gerevini et al., 2003), the top performer at IPC-2002; Fast Downward Autotune 1 (FD-AT1) and Fast Downward Autotune 2 (FD-AT2) (Helmert, 2006), improved versions of the top performer at IPC-2004, whose parameters were optimized on IPC benchmarks; LAMA2011 (Richter & Westphal, 2010), the latest version of the top performer at IPC-2008 and IPC-2011; the recent SAT-based planners M and Mp (Rintanen, 2012) which are very competitive across many domains, and do not suffer from the weakness of relaxation heuristics in planning with resources; and LPRPGP (Coles et al., 2008; Coles & Coles, 2011), the most recent version of the LPRPG planner whose improved heuristic addresses that weakness.

We also ran several optimal planners, to check whether these are more effective for small C, as was previously observed by Hoffmann et al. (2007) for their step-optimal SAT-based numeric planner, num2sat, in a NoMystery test suite. Other than this, a comparison to the satisficing planners is not intended and should not be made. We include num2sat as well as: Merge-and-Shrink (M&S) (Helmert, Haslum, & Hoffmann, 2007b; Nissim, Hoffmann, & Helmert, 2011), a state abstraction heuristic; LM-cut (Helmert & Domshlak, 2009), the best known admissible relaxation heuristic; Selmax (Domshlak, Karpas, & Markovitch, 2010), which uses machine learning to selectively choose a heuristic per-state; and the optimal version of Fast Downward Autotune (FD-AT-OPT) (Fawcett, Helmert, Hoos, Karpas, Röger, & Seipp, 2011).

Our RCP benchmark set consists of 450 instances. These are obtained from a smaller set of

"base" instances, where $C = 1.0$. The benchmarks with larger values of C are generated by increasing the initial resource supply in these base instances. This setup ensures that the results across different C values are exclusively due to the level of resource constrainedness. The instances in TPP are the ones originally designed by Gerevini et al. (2008). The 5 base instances each have 1 agent, 8 markets and 8 products, and each is modified to obtain instances with $C = 1.1, \ldots, 1.5$. In each of NoMystery and Rovers, to demonstrate scaling, we designed a "small" and a "large" group. The small groups have 2 resources and 25 base instances, the large groups have a single resource and 5 base instances. In the large groups, having several resources was not feasible for the optimal solvers implemented inside the generators. Each base instance is modified to obtain instances with $C = 1.1, \ldots, 1.5, 2.0$. Small NoMystery instances contain 2 trucks, 9 locations, and 9 packages; large ones use 1 truck, 12 locations, and 15 packages. Small Rovers instances feature 2 rovers, 11 locations, and 16 objectives; large ones include 1 rover, 15 locations, and 20 objectives.

The base instances were generated randomly, except that in the small groups – where there are several resources – we explored a second interesting problem feature, namely the *distribution* of the resource budget. We first generated 5 instances randomly. Then, we selected 5 pareto-optimal resource allocations for each of these, yielding 25 base instances. The selection was made so that we included: an allocation in which the total amount of resources is assigned to one of the trucks/rovers (this is analogous to a problem with one resource); an allocation that is closest to where the resources are evenly distributed between trucks/rovers; and 3 others in between these two extremes.

Figure 4.1 summarizes coverage results in the RCP benchmarks. Missing data indicates that a planner did not solve any instance for that domain; the single exception is Arvand-RC(OPSC), not shown in Figure 4.1 (a,b,c) because there it almost coincides with Arvand-RC. The most effective encoding for each planner is: numerical encoding - FF, M, Mp, LPG, LPRPGP, and num2sat; propositional encoding - LM-cut, M&S, Selmax, and FD-AT-OPT. propositional for small domains, propositional + costs for large domains and TPP - LAMA, FD-AT1, FD-AT2, Arvand, and all Arvand-RC variations. Exceptions: For LAMA in large-rovers and TPP, propositional + costs with no hard constraints was best. The main observations are:

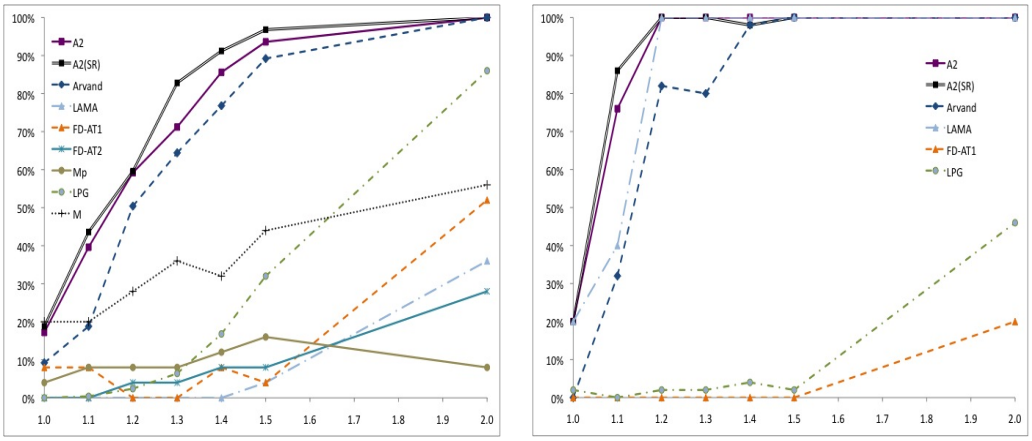
(i) Optimal planners can be effective for scarce resources, but only in small instances.

(ii) The improved heuristic of LPRPGP is not sufficient to obtain competitive performance here.

(iii) Current satisficing planners excel when resources are plentiful, but fail quickly as they get scarce. The same holds for M and Mp.

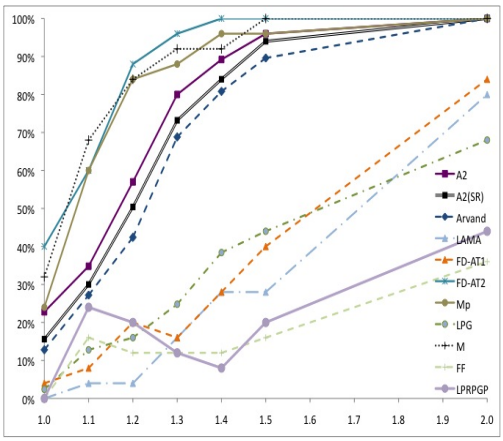
(iv) RWS can be a powerful tool to attack RCP. In particular, Arvand-RC almost universally outperforms other planners when C is close to 1.

(v) Arvand-RC has a consistent and significant advantage over Arvand-2009, showing the effectiveness of smart restarts and on-path search continuation.

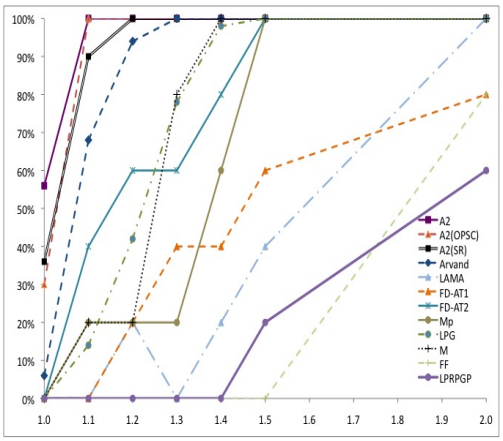


(a) Rovers, small

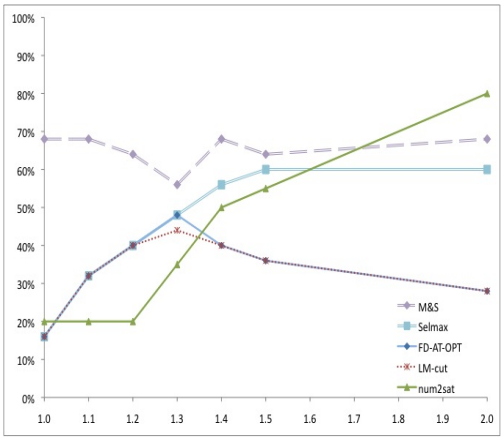
(b) Rovers, large



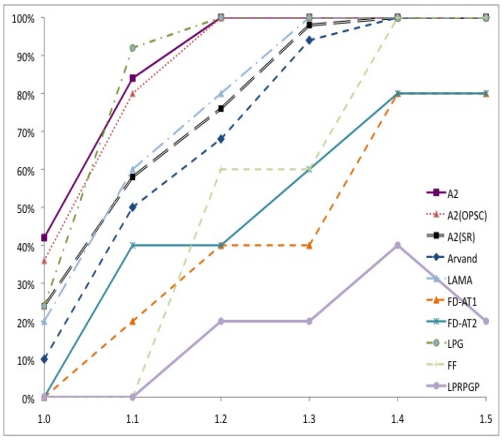
(c) NoMystery, small



(d) NoMystery, large



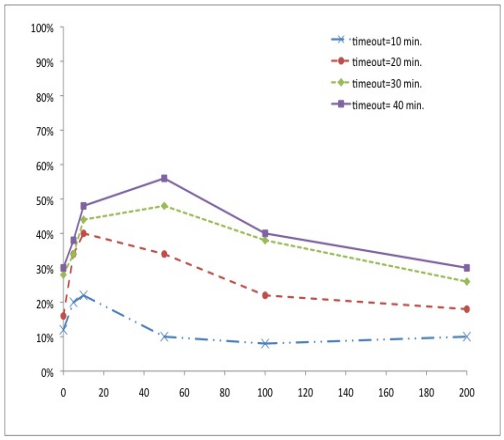
(e) NoMystery, small (optimal planners)



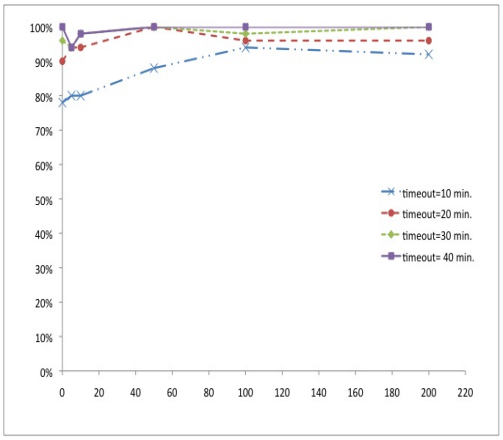
(f) TPP

Figure 4.1: Coverage of planners over resource constrainedness C, in (a,b) Rovers, (c,d,e) NoMystery, and (f) TPP. Randomized planners are run 10 times per instance, where each run counts separately towards coverage.

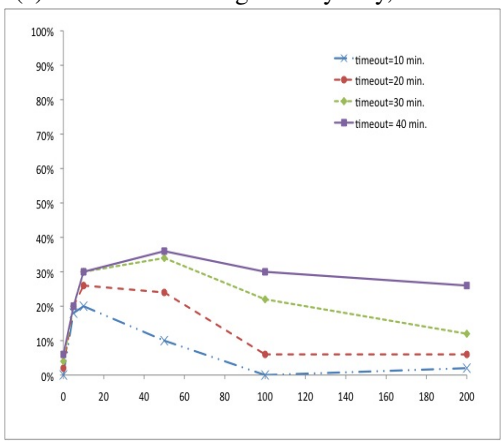
To see (i), note that none of the optimal planners solved any instance, in any domain other than small NoMystery. In the latter domain – compare Figures 4.1 (e) and (c) – optimal planners are more effective than satisficing planners, when C is close to 1. For M&S, the value of C has little



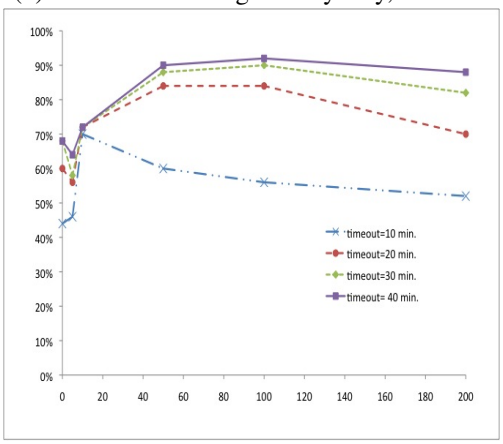
(a) Arvand-RC in large NoMystery, $C = 1.0$



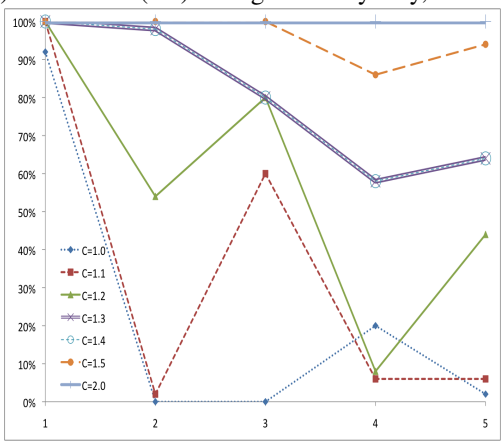
(b) Arvand-RC in large NoMystery, $C = 1.1$



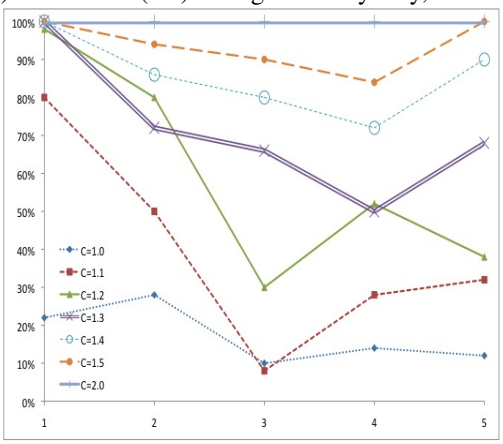
(c) Arvand-RC(SR) in large NNoMystery, $C = 1.0$



(d) Arvand-RC(SR) in large NNoMystery, $C = 1.1$



(e) Arvand-RC in small NoMystery



(f) Arvand-RC in small Rovers

Figure 4.2: Coverage of Arvand-RC as a function of different parameters. (a,b,c,d) vary the runtime cut-off, and the pool size p for smart restarts (x-axis; for $p = 0$, smart restarts are turned off). (e) and (f) vary the distribution of budget across resources (see text).

effect. All optimal planners fail as instance size increases (Figure 4.1 (d)).

Regarding point (ii), LPRPG does not solve any instance in Rovers, and is quite weak also in NoMystery and TPP. Point (iii) is obvious in all the plots showing data for satisficing planners

Domain	Arvand	A2 (SR)	A2 (OPSC)	A2	LAMA	FD-AT1	FD-AT2	M	Mp	LPRPGP
Barman	0%	0%	0%	0%	100%	100%	0%	0%	0%	10%
Elevators	100%	100%	25%	20%	100%	95%	90%	5%	65%	80%
Floortile	5%	10%	10%	5%	25%	20%	45%	0%	0%	10%
Nomystery	95%	95%	95%	100%	50%	45%	80%	80%	75%	35%
Openstacks	100%	100%	100%	100%	100%	95%	50%	0%	0%	85%
Parcprinter	100%	100%	100%	100%	100%	100%	85%	100%	100%	35%
Parking	15%	15%	0%	0%	95%	65%	75%	0%	0%	35%
Pegsol	100%	100%	100%	100%	100%	100%	100%	85%	100%	100%
Scanalyzer	85%	90%	85%	85%	100%	100%	90%	50%	80%	90%
Sokoban	10%	10%	5%	10%	90%	95%	85%	0%	10%	45%
Tidybot	85%	85%	80%	85%	80%	70%	75%	0%	30%	95%
Transport	65%	70%	35%	35%	85%	75%	70%	0%	5%	0%
Visitall	65%	70%	65%	65%	100%	10%	40%	0%	0%	20%
Woodworking	100%	100%	5%	15%	100%	100%	65%	100%	100%	0%
Total	66%	68%	50%	51%	88%	76%	68%	30%	40%	46%

(a) Coverage

Domain	Arvand	A2 (SR)	A2 (OPSC)	A2	LAMA	FD-AT1	FD-AT2	M	Mp	LPRPGP
Barman	1800	1800	1800	1800	5	256	1800	1800	1800	1669
Elevators	12	12	1414	1478	47	173	340	1735	634	637
Floortile	1783	1687	1626	1749	1372	1452	996	1800	1800	1666
Nomystery	164	142	194	54	900	1023	381	561	457	1192
Openstacks	26	24	92	60	44	289	1128	1800	1800	546
Parcprinter	8	8	21	20	0	0	273	0	0	1171
Parking	1778	1648	1800	1800	408	1097	926	1800	1800	1561
Pegsol	21	63	5	7	2	0	14	330	4	82
Scanalyzer	271	207	276	286	21	102	202	937	362	192
Sokoban	1635	1665	1711	1705	322	129	464	1800	1622	1180
Tidybot	385	335	681	606	392	584	753	1800	1268	221
Transport	651	569	1219	1243	523	634	705	1800	1711	1800
Visitall	28	40	662	633	42	1621	1128	1800	1800	1448
Woodworking	102	78	1710	1602	12	16	647	1	1	1800
Total	619	591	944	932	292	527	697	1283	1076	1083

(b) Runtime

Table 4.1: Coverage (a) and average runtime (b; in seconds) in the IPC-2011 benchmark domains. For unsolved instances, the time limit of 30 minutes is inserted into the computation of runtime.

(Figure 4.1 (a,b,c,d,f)). For the heuristic planners, this can be expected given the pitfalls of relaxed planning and was previously observed in much smaller experiments by Hoffmann et al. (Hoffmann et al., 2007) and Gerevini et al. (Gerevini et al., 2008). For M and Mp, it is quite interesting that their behavior over C is similar to that of the heuristic planners. It is not clear to us what causes this behavior; a plausible explanation could be that these planners, too, act in a rather greedy way.

Point (iv) is evident from the results of Arvand-RC and Arvand-2009, in all the domains. With small C, Arvand-RC vastly outperforms all other planners, by factors of 6 and more in coverage. The only exceptions are LAMA in large Rovers (Figure 4.1 (b)); FD-AT2, M, and Mp in small NoMystery (Figure 4.1 (c)); and LPG in TPP (Figure 4.1 (f)). Of these, LAMA's prowess in large Rovers is due to the anytime cost-augmented encoding, and does not extend to small Rovers (Figure 4.1 (a)) with several resources. FD-AT2, M, and Mp fall behind significantly in large NoMystery (Figure 4.1 (d)). LPG is competitive with Arvand-RC only in TPP.

Point (v) is also evident. In fact, in these benchmarks, neither smart restarts nor on-path search continuation ever hurt performance when added to Arvand-2009. Their effectiveness does depend on the domain, though. In Rovers, it is slightly better to use smart restarts only (as pointed out, Arvand-RC(OPSC) almost coincides with Arvand-RC there). In NoMystery, both techniques contribute to the improvement over Arvand. In TPP, smart restarts have a beneficial effect but on-path search continuation is more important by far.

For smart restarts, an influential parameter is the pool capacity p. Two observations are clear from the data in Figure 4.2 (a,b,c,d):

(vi) Pool size induces a sweet-spot behavior in both Arvand-RC and Arvand-RC(SR), especially for small values of C.

(vii) As the time-out increases, the sweet-spot behavior tends to become more pronounced, and the best value for p tends to become larger.

An intuitive explanation for both is that, as previously discussed, p controls the exploitation-exploration trade-off in smart restarts. Larger pools yield a more explorative search, which may pay off by solving more instances – but only if there is enough time. With limited runtime, a more greedy search may succeed more often.

Figure 4.2 (e,f) examines the performance of Arvand-RC as a function of the distribution of the resource budget. On the x-axis, we distinguish the 5 qualitatively different pareto-optimal resource allocations described above: $x = 1$ stands for an allocation assigning the whole budget to just one of the two resources, whereas $x = 5$ stands for an allocation assigning the budget as evenly as possible, i.e., minimizing the difference between the initial resource supplies. In between, we interpolate such that this difference decreases monotonically with growing x. Each value of x corresponds to 5 base instances, and the data shown are averages. The data is noisy, but still allows to observe:

(viii) Distributing the resource budget more evenly results in worse performance, especially with small C.

An intuitive explanation is that a more even distribution of the budget implies that the planner needs to reason more about which resource to use.

Finally, we ran our new planners on the standard IPC-2011 benchmarks, to cross-check whether their superior performance in RCP is bought at the price of deteriorated performance in other settings. All the planners were run under IPC-2011 conditions: 2GB memory and 30 minutes runtime; Table 4.1 confirms that smart restarts work well on IPC benchmarks – Arvand-RC(SR) has better coverage than Arvand. On-path search continuation, by contrast, can be detrimental. The reduction in coverage stems mainly from 4 of the 14 domains.

4.6 Conclusion

We all must consume our resources prudently, and so must planners in a multitude of applications. While this general issue has long been researched, not much has been done to specifically address the RCP situation where resources are scarce and cannot be replenished. Starting to investigate this more carefully, we have shown that state of the art planners typically behave very badly. We have demonstrated the potential of random walk search to improve this, and contributed an extended test base for future research.

Chapter 5

Plan Improvement Using Postprocessing

This thesis, so far, has been focused on studying random walks as a tool that helps planners to solve more problems. The quality of solutions, however, is also of great importance. Chapter 2 demonstrates that long random walks containing many randomly selected actions are useful to escape plateaus but not all these random actions are necessary to achieve the goals. This chapter studies effective postprocessing techniques implemented in the system Aras that decrease the cost of solutions by removing *irrelevant* actions and finding new *shortcuts*. A great advantage of the developed methods is that they are general: they can be used for any planning method and are not limited to RW planning.

5.1 Introduction

Satisficing planners can solve much harder instances than optimal planners but may generate plans that are far from optimal. Earlier planning competitions have emphasized coverage in terms of total number of problems solved, as well as raw speed. The focus of IPC-2008 and IPC-2011 was on quality: finding the best plan with a given finite amount of resources. Much work in satisficing planning has gone into generating a high quality plan directly. Such systems output a single plan and then stop. In contrast, anytime planners such as LAMA (Richter & Westphal, 2010) and LPG (Gerevini et al., 2008) aim to quickly find a lower-quality plan, then improve it over time. While the framework introduced in LAMA and LPG can serve as a general guideline to improve solutions, the specific techniques such as using WA* with smaller weights in LAMA and restarting from a partial plan in LPG cannot be directly used in other planing systems such as RW planners that rely on inherently different search algorithms. A useful alternative is to improve plans in a postprocessing phase.

Fink and Yang (1992) analyze plan improvement as a problem of removing *unjustified* parts of the original plan. They also propose a greedy algorithm, which is here referred to as *Action*

Elimination (AE), to remove actions that are unnecessary to achieve the goals (AE is also developed as a part of this research, independently of (Fink & Yang, 1992). However, since (Fink & Yang, 1992) significantly predates this work, the algorithm itself is not presented as a contribution of this research). AE can only remove actions from the original plan and is incapable of finding new paths by considering actions outside the plan. This issue is resolved here by combining AE with the much stronger postprocessing method of *Plan Neighborhood Graph Search (PNGS)*, which cannot only remove unnecessary actions but also find new paths in the state space.

5.1.1 Contributions

This chapter contributes to the plan improvement research in three ways:

- The development of the effective postprocessing algorithm *Plan Neighborhood Graph Search (PNGS)*.
- A postprocessing system Aras that implements and combines AE and PNGS.
- A detailed experimental analysis of AE and PNGS both as standalone methods and in combination.

Both AE and PNGS can take any valid plan as input and attempt to improve it. AE is a fast algorithm, while PNGS works in anytime fashion. Both AE and PNGS improve the performance of all the planners tested as measured by the IPC metric. In contrast to LAMA, AE and PNGS search for local improvements "near" an existing plan. In contrast to LPG, they search in state space not plan space.

There are many ways to measure plan quality. Two popular metrics for unit cost actions are sequential plan length measured in total number of actions, and makespan, the shortest execution time of a plan if actions can be executed in parallel. The IPC metric for non-uniform action costs (including zero) is additive cost, with the total cost of a plan defined as the sum of all action costs.

5.2 Related Work

Weighted A*, or WA* (Ratner & Pohl, 1986), produces plans that are within a constant factor W of optimal. The LAMA planner, winner of the two last competitions IPC-2008 and IPC-2011, uses GBFS to quickly produce an initial plan, then switches to WA* with and gradually reduces the weight W, while using the best found plan for additional pruning. Anytime A* (Hansen & Zhou, 2007) also uses successive runs of WA*. While LAMA restarts the search from the initial state each time a solution is found, Anytime A* continues the current search with new parameters. Anytime Window A* (Aine, Chakrabarti, & Kumar, 2007) uses A* within a window in the search space that moves in a depth first manner. The size of the window is increased when a solution is found.

The path improvement methods Joint and LPA* (Ratner & Pohl, 1986) and ITSA* (Furcy, 2006) are closely related to Plan Neighborhood Graph Search and a detailed comparison will follow later.

The LPG planner (Gerevini et al., 2008) uses heuristic local search in plan space. It optimizes an objective function that measures the difficulty of resolving the inconsistencies and the estimated cost of the solution. When LPG is used as an anytime system for plan improvement, it restarts from a partial plan, obtained from the current best plan by removing some actions randomly, preferring the most expensive ones. An added numerical constraint on the cost forces the next solution to be cheaper.

For the makespan metric, the post-processing approaches of (Do & Kambhampati, 2003; Veloso, Pérez, & Carbonell, 1990; Bäckström, 1998) aim to reduce the make-span of a given totally ordered plan by converting it to a partially ordered plan. Since these approaches do not change the set of actions in the plan, they do not improve the cost according to the other metrics above. In planning by rewriting (Ambite & Knoblock, 2001), domain-specific rules rewrite a given plan into a better quality one. Rewriting rules are given by an expert or learned from training examples (Upal, 1999).

5.3 Two Approaches to Plan Improvement

While there is a number of current algorithms for plan improvement in the weighted A* family, there has been no recent work on the general case when the quality of the initial plan is unknown. This is surprising since such plans are arguably most in need of improvement! The two methods AE and PNGS studied here take any plan produced by a satisficing planner and try to improve it. The methods produce no global guarantees on the solution quality. However, any known quality bound for the input plan, such as W in a plan produced by WA* with an admissible heuristic, implies a corresponding tighter bound on the improved plan.

Both AE and PNGS search for the best possible plan within a *neighborhood* of similar plans, but use different concepts of neighborhood. AE only removes actions from a given plan. PNGS exactly solves a shortest path problem in a neighborhood of a plan consisting of states close to the plan's trajectory in state space.

5.4 Action Elimination

Given a plan π, the goal of Action Elimination is to find a shorter plan by removing actions from π.

Definition 14 (Reduction)**.** *Let* Π *be a planning task,* π *a plan for* Π*, and* π' *a subsequence of* π*.* π' *is a* reduction *of* π*, denoted by* $reduct(\pi, \pi')$*, iff* π' *is also a plan for* Π*.*

Definition 15 (Minimal Reduction)**.** *Let* π *be a plan and* π' *be a reduction of* π*.* π' *is a* minimal reduction *of* π *if for every* π'' *such that* $reduct(\pi, \pi'')$*,* $cost(\pi') \leq cost(\pi'')$*.*

A minimal reduction is a lowest-cost plan that can be achieved by removing actions. Finding a minimal reduction can be difficult: the corresponding decision problem is NP-complete (Fink & Yang, 1992).

Algorithm 4 Action Elimination

Input Initial State s_0, plan $\pi = (a_1, \ldots, a_n)$, and
goal condition G
Output A plan reduction

```
s ← s_0
i ← 1
repeat
  mark a_i {try to remove a_i}
  s' ← s
  for j ← i + 1 to length(π) do
    if a_j is not applicable to s' then
      mark a_j
    else
      s' ← apply(s', a_j)
    end if
  end for
  if s' satisfies G then
    remove marked actions from π {commit}
  else
    unmark all actions
    s ← apply(s, a_i)
  end if
  i ← i + 1
until i > length(π)
return π
```

5.4.1 A greedy Algorithm for Action Elimination

Action Elimination iteratively improves a given plan $\pi = (a_1, \ldots, a_n)$ by computing a plan reduction in each iteration. The details are given in Algorithm 4. Starting from a_1, the algorithm tentatively tries to remove one action a. After removing a, all other actions that lose their support - at least one of their preconditions becomes unsatisfied - are removed from the plan. If the reduced sequence remains a solution, the algorithm commits to this new plan. Otherwise, the plan is restored to the state before a was removed. The process continues until all actions in the remaining plan have been tried. Validating a single reduction takes $O(n \times p)$ time, where p is the maximum number of preconditions of an action. The time complexity of the whole algorithm is $O(n^2 \times p)$.

Algorithm 4 is just one specific, simple implementation of the idea of using successive plan reductions and cannot identify all the removable actions. In general, different reduction sequences do not necessarily lead to a unique irreducible plan. For example, if the original plan contains two redundant but different ways of achieving the same goal, a sequence of reductions could remove either one (but not both).

Name	pre	add	del
OP_k	ϕ	$\{k\}$	$\{p\}$
OP_r	ϕ	$\{r\}$	ϕ
OP_p	ϕ	$\{p\}$	$\{q\}$
OP_q	$\{k\}$	$\{q\}$	ϕ

Table 5.1: Definition of actions in a planning task example.

Algorithm 4 does not identify all the removable actions. Consider a planning task with initial state $\{p, q\}$, goal state $\{p, q, r\}$ and the four actions defined in Table 5.1. In the plan $\pi = (OP_k, OP_p, OP_r, OP_q)$, only OP_r is necessary and OP_k, OP_p and OP_q can be removed. The algorithm first marks OP_k, causing OP_q to lose its support and be marked as well. However, since the remaining sequence (OP_p, OP_r) is not a plan, this step fails and nothing is removed. Next, the algorithm tries OP_p, but again the remaining sequence is not a plan. Therefore, no action is removed from the plan by Algorithm 4. The main reason of this failure is that the algorithm focuses on positive effects, and does not properly capture the negative interactions between interleaving subsequences.

5.5 Plan Neighborhood Graph Search

Most state of the art planners behave in a "greedy" way in terms of a heuristic function. They only examine a tiny subset of the state space, following narrow paths guided by their heuristic. In contrast, the search of optimal planners is much broader since A* with admissible heuristics needs to expand every state with f-value below the minimum solution cost. Plan Neighborhood Graph Search (PNGS) takes a middle ground between these two approaches. The plan neighborhood graph represents a subset of the state space "near" the existing plan that is wider than the path searched by greedy planners. Like optimal planners, it finds the best possible solution in a search space. However, like satisficing planners, this search is limited to a small part of the whole state space. PNGS uses local search around the plan trajectory to build the neighborhood graph, then extracts a shortest path from this graph.

Let M be a graph search method, such as breadth-first or best-first search. M must be able to expand the graph of a finite search space one node at a time from a given start state s_0 to generate a sequence of states $(s_0, s_1, s_2, \ldots, s_n)$. To be useable in PNGS, M must provide a method $edgeto(s)$ that returns the edge along which state s was most recently reached in the search.

For a given expansion limit L, let $L' \leq L$ be the number of states actually expanded by M starting from s_0. Let $v(s_0, M, L) = \{s_i | 0 \leq i \leq L'\}$ be the set of all these states and $e(s_0, M, L) = \bigcup_{i=1 \cdots L'} edgeto(s_i)$ be the set of directed edges generated in this search.

Neighborhood graph search expands a given *seed graph* $SG = (V, E)$ by running M from each start state in V with exploration limit L. The *neighborhood graph* of SG is defined as $NG(SG, M, L) = (\bigcup_{x \in V} v(x, M, L), E \cup \bigcup_{x \in V} e(x, M, L))$. Algorithm 5 gives pseudocode.

Algorithm 5 Computation of Neighborhood Graph

Input A subgraph (V, E) of a state space with
$V = \{v_0, \ldots, v_n\}$, $E \subseteq V \times V$, nonnegative integer
L, and search method M
Output The graph $NG(V, M, L)$

```
V' ← V
for i ← 0 to n do
  M.initialize(v_i) {search neigborhood of v_i}
  for j ← 1 to L do
    s ← M.get_next_state()
    if is_null_state(s) then
      return (V', E)
    end if
    V' ← V' ∪ s
    E ← E ∪ M.edgeto(s)
  end for
end for
return (V', E)
```

Let $\pi = (a_1, \ldots, a_n)$ be a plan, $S_\pi = \{s_0, \ldots, s_n\}$ the set of all states visited when executing π, with $s_i = apply(s_0, (a_1, ..., a_i))$ for $0 < i \leq n$, and $E_\pi = \{(s_0, s_1), \ldots, (s_{n-1}, s_n)\}$ the edges linking successive states in the plan. With M and L defined as above, the *L-plan neighborhood graph* of π is defined as $PNG(\pi, M, L) = NG((S_\pi, E_\pi), M, L)$. Informally, $PNG(\pi, M, L)$ contains the original seed plan augmented by the union of the neighborhoods constructed using M around each state along the plan π.

The number of vertices in $PNG(\pi, M, L)$ is bounded by $(L + 1) \times (n + 1)$. While building a neighborhood graph, all goal states are identified. A lowest-cost path from s_0 to a goal state in the graph is built by a standard Dijkstra-type shortest path algorithm. If the search method M uses forward search, backward chaining from the goal states works well since the branching factor in regression is often much smaller. For backwards plan extraction, the priority queue in Dijkstra's algorithm is initialized with all goal states in $PNG(\pi, M, L)$.

A simple anytime version of PNGS can be implemented by iteratively doubling the exploration limit L up to a resource limit. Each iteration starts with the best plan from the previous iteration as seed plan. One benefit of the exploration limit L is that it corresponds directly to the amount of resources used by the search method M. Methods such as breadth-first and best-first search need time and memory at most linear in the number of states.

The notion of plan neighborhood graph can be extended to multiple input plans as well as multiple local search methods. For multiple input plans, compute the neighborhood graph of the union of all input plans. If S_{π_0}, E_{π_0}, S_{π_1} and E_{π_1} are the states and action edges of plans π_0 and π_1, then $PNG(\pi_0 \cup \pi_1, M, L) = NG((S_{\pi_0} \cup S_{\pi_1}, E_{\pi_0} \cup E_{\pi_1}), M, L)$. Different search methods M_0 and M_1 can be used to construct a merged neighborhood combining the expansion strategies of each

method as $PNG(\pi, \{M_0, M_1\}, L) = PNG(\pi, M_0, L) \cup PNG(\pi, M_1, L)$.

Extended neighborhood graphs utilize several input plans and/or search methods in order to find a better plan. Using multiple input plans allows PNGS to search near good-quality fragments of several different plans. Multiple search methods may allow better exploration of the state space.

5.5.1 Local Search Methods for PNGS

The experiments reported here use either a single search method, M_{A*}, or a combination of two search methods $M_{A*} + M_{bbfs}$: M_{A*} is derived from the baseline uniform cost search algorithm from the optimal track of IPC-2008 (Helmert, Do, & Refanidis, 2008). It performs a "blind" A* search with the heuristic h set to 0 for goal states and to the minimum action cost in the problem for other states. However, as in LAMA (Richter & Westphal, 2010), M_{A*} is modified to better deal with the zero cost actions present in several competition domains. Since blind A* never expands any other action as long as zero cost actions are available, all action costs are increased by 1 while building the neighborhood graph. For extracting the shortest path, they are reset to the true action cost to guarantee that the returned plan's cost never exceeds the input plan's cost.

The combined search $M_{A*} + M_{bbfs}$ uses forward M_{A*} search as well as backward breadth first search (bbfs). Bbfs generates predecessor states and actions that lead to a given state, ignoring action costs.

5.5.2 Comparison of PNGS with Related Work

Joint and LPA* (Ratner & Pohl, 1986) improve a given plan by using an optimal solver. The optimal solver searches for shortcuts between any pair of states that are a fixed distance d apart in the input plan. In contrast to these approaches that redefine the goal state for each search, PNGS always uses the original goal states of the planning problem for its search. Another key difference is that instead of searching for each shortcut in isolation, PNGS builds the complete neighborhood graph before extracting a shortest path. In the example in Figure 5.1, building a neighborhood graph improves on separate searches. Here, M is the A* algorithm with the blind heuristic, $L = 4$, and the input plan has three states. When A* is run from each point separately, it fails to improve the input plan, as in Figures 5.1.b and 5.1.c. However, PNGS improves the cost by 5 units.

ITSA* (Furcy, 2006) improves a given path in a graph using an A* search restricted to a tunnel near the given path π. The tunnel contains all states s with $dist(s, \pi) \leq d$, where $dist(s, \pi)$ is defined as the minimum cost path from any state in π to s. ITSA* successively increases d in each iteration and terminates when a memory limit is exceeded. In (Furcy, 2006), ITSA* was tested on problems with unit-cost actions, setting $d = 0, 1, 2, \cdots$. For the experiments in this chapter on domains with non-uniform costs, d was set to the minimum distance among all unexplored states, as is standard practice in iterative deepening A* with non-unit costs. Each iteration runs until the first goal state is expanded.

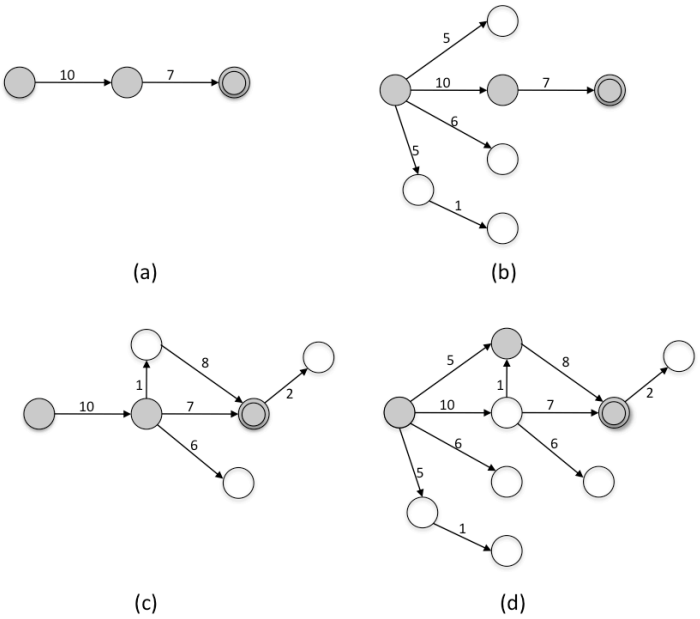


Figure 5.1: (a) The input plan. (b) and (c) Separate local searches fail. (d) The neighborhood graph contains an improved plan.

Compared to ITSA*, PNGS uses a different search control and separates neighborhood creation from search. In contrast to the L parameter in PNGS, the search effort of ITSA* iterations can not be easily predicted from the d parameter in domains with nonuniform branching factor. ITSA*'s distance function can also lead to an unbalanced expansion at different points along the input plan, since its number of states expanded corresponds to the number of low-cost paths available. ITSA* expands many more nodes in regions where many cheap actions are available.

Building and searching the neighborhood simultaneously as in ITSA* allows some more pruning. One advantage of the two phase computation in PNGS is that different action costs can be used in each phase, which works better for domains with zero-cost actions. The option to merge neighborhoods generated by different search methods with complementary strengths is also useful.

5.6 Experiments

The ARAS plan postprocessor implements Action Elimination and PNGS on the basis of the Fast Downward (FD) (Helmert, 2006) framework. M_{A*} and M_{bbfs} are implemented as local search methods. For direct comparison, ITSA* was implemented in the same environment. Increased action costs are also used in ITSA* to avoid problems with zero cost actions. This improves the performance but might lead ITSA* to a plan of higher cost. ARAS supports propositional PDDL2.2, excluding derived predicates, as well as action costs in PDDL3.1. ARAS and LPG were used to improve the results of the Arvand-2009 (Nakhost & Müller, 2009) and FF (Hoffmann & Nebel, 2001) planners in the IPC-2004 domains Pipesworld Tankage, Pipesworld NoTankage, Airport and

Satellite. Further, ARAS is compared to ITSA* in all IPC-2008 domains on plans produced by Arvand-2009, FF, and the top four planners from the competition: LAMA, FF_{sa}, FF_{ha}, and C3. Currently, LPG does not support IPC-2008 domains. The input plans for ARAS were generated by a single run of the latest available version of each planner. Tests used a 2.7 GHz AMD processor with 4GB memory and 30 minutes time limit per problem.

5.6.1 Experiment 1: Postprocessing for IPC-2008 Domains

Tests used the IPC scoring function, the cost of the best plan produced by any planner divided by the cost of the generated plan, with the cost of the best plan produced by any satisficing planner at the IPC-2008 competition divided by the cost of the generated plan. Unlike the competition itself, and in order to measure progress since then, a plan that is better than the best IPC-2008 plan achieves a score higher than one.

For the planners returning a single plan, FF, FF_{sa}, FF_{ha} and C3, the planner is run until it finds a solution. The remaining time up to 30 minutes total is used to improve the plan with ARAS or ITSA*.

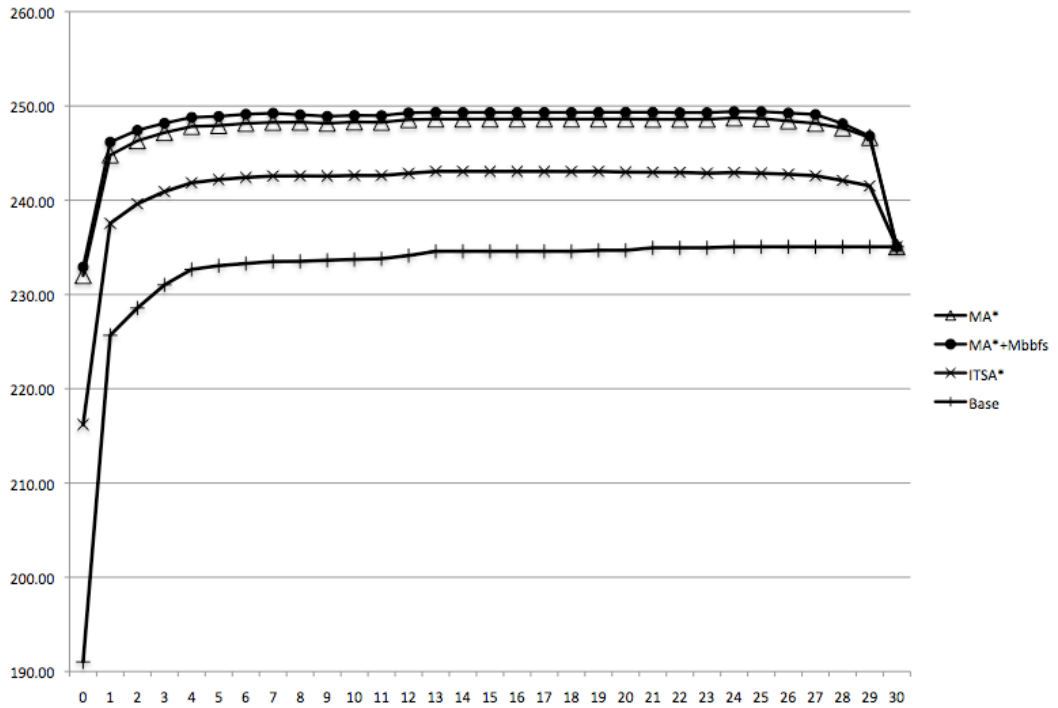


Figure 5.2: Total IPC score for varying cutoff times combining LAMA with M_{A*}, $M_{A*} + M_{bbfs}$ and ITSA*

Both LAMA and Arvand-2009 can run in an anytime setting. Given both an anytime planner and an anytime postprocessor, an experiment was run to determine a reasonable allocation of time between them as follows: first, the planner is run until a fixed cutoff time is reached. If no solution

is found yet, it is kept running until the first solution is found. Next, the postprocessor is used to improve the planner's best generated plan until the 30 minute timeout. The cutoff time is varied from 0 to 30 minutes in 1 minute intervals. Figures 5.2 and 5.3 show the total scores of LAMA and Arvand-2009 over all IPC-2008 domains, when combined with the postprocessors M_{A*}, $M_{A*} + M_{bbfs}$, and ITSA*. For comparison, the baseline shows the anytime planner stopped at the cutoff time without any postprocessing. For both LAMA and Arvand-2009, the PNGS methods outperform ITSA*. $M_{A*} + M_{bbfs}$ and M_{A*} are very close for Arvand-2009. $M_{A*} + M_{bbfs}$ is slightly superior for LAMA. The best schedule for LAMA is 24 minutes (or until the first plan is found) for the planner followed by 6 minutes for ARAS, while for Arvand-2009 the optimum is at 18 + 12 minutes. For both planners, the performance curve is almost flat for cutoff times ranging from about 7 to 26 minutes.

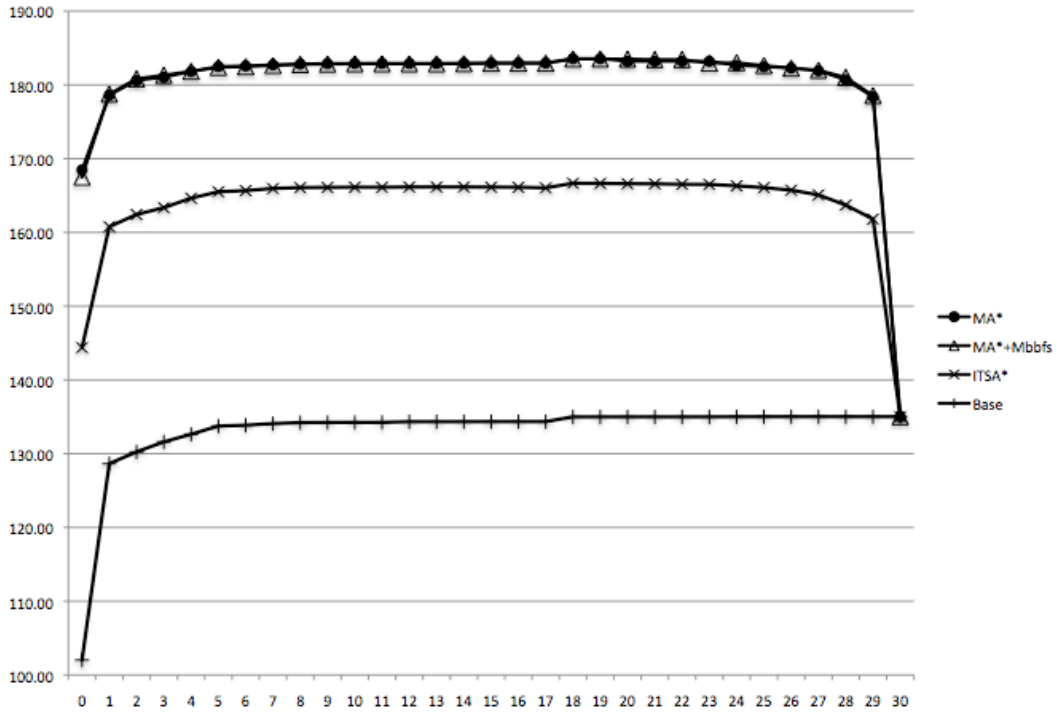


Figure 5.3: Total IPC score for varying cutoff times combining Arvand-2009 with M_{A*}, $M_{A*} + M_{bbfs}$ and ITSA*

The results for all tested planners on IPC-2008 are summarized in Figure 5.7. For each planner/postprocessor pair the total score, and the score obtained in each domain is shown. LAMA and Arvand-2009 use the cutoff times determined above. Cybersecurity is included in the totals, but no detail graph is shown; in this domain, postprocessors did not improve any plan except some generated by Arvand-2009.

The total scores shown in the bottom right of Figure 5.7 on page 94 illustrate that postprocessing would have provided an advantage in the IPC-2008 competition: any of the planners that took

places 3-5, FF_{sa}, FF_{ha}, and C3, would have improved to second place. Both ARAS and ITSA* find substantial improvements for many LAMA and FF plans as well, advancing the state of the art.

ARAS seems to be most effective on problems consisting of several loosely coupled subtasks. In these domains, due to low interaction between different parts of a plan, effective local improvements are possible. For example, all postprocessors perform very well in the transportation domains Transport and Elevator. In other domains, results vary greatly by planner. Postprocessing in Pegsol gains more than 10 points for FF variants and C3, and 4 points for Arvand-2009. However, there is very limited scope for improvement for LAMA, since it already solves 27 out of 30 tasks optimally in this domain.

In PNGS, $M_{A*} + M_{bbfs}$ outperforms M_{A*}, especially in domains where local improvements are effective. Although the size of the largest neighborhood graph is equal for both search methods in these experiments, their structure is totally different. In $M_{A*} + M_{bbfs}$, the expanded states are closer to the plan, which contributes to finding better shortcuts.

In contrast, in Openstacks such local improvements are hard. This domain models a combinatorial optimization problem, with the goal of minimizing the maximum number of stacks used in manufacturing. It seems very unlikely to find a shortcut in solutions for this planning domain. Actions that affect the total cost - adding a stack - completely change the search neighborhood; propositions showing the availability of the added stacks will be present in all successive states. This makes it difficult to locally improve plans. Most of the improvements in this domain are obtained on smaller tasks where the largest neighborhood graph is large enough to contain a new goal state. In this domain, using all memory for M_{A*} works better than splitting it between M_{A*} and M_{bbfs}.

Both M_{A*} and $M_{A*} + M_{bbfs}$ usually outperform ITSA*, which has trouble when there are large cost differences between actions. For example, in Transport, *pick up* and *drop* have unit cost, while the distance-dependent cost of *drive* is usually much larger. ITSA* tends to explore sequences of many cheap actions, but largely ignores crucial *drive* actions. For example in Transport-14 the cheapest driving action has cost 12, and ITSA* reached a maximum $d = 71$, while the neighborhood graph of PNGS with M_{A*} contained some nodes up to a cost of 253 from the input graph. It found a solution of overall cost 2217 compared to ITSA*'s 2617.

Figures 5.4 and 5.5 show the effect of varying expansion and distance limit using Elevators-22 as an example. The input plan was generated by LAMA and has a cost of 663. The size of the neighborhood graph in PNGS grows linearly with the expansion limit. The growth rate of ITSA* varies depending on the average branching factor in the explored regions at each iteration.

5.6.2 Action Elimination

Figure 5.6 reports results for two configurations of ARAS that use Action Elimination: AE represents a single run of Action Elimination. PNGS + AE* runs PNGS and Action Elimination alternately:

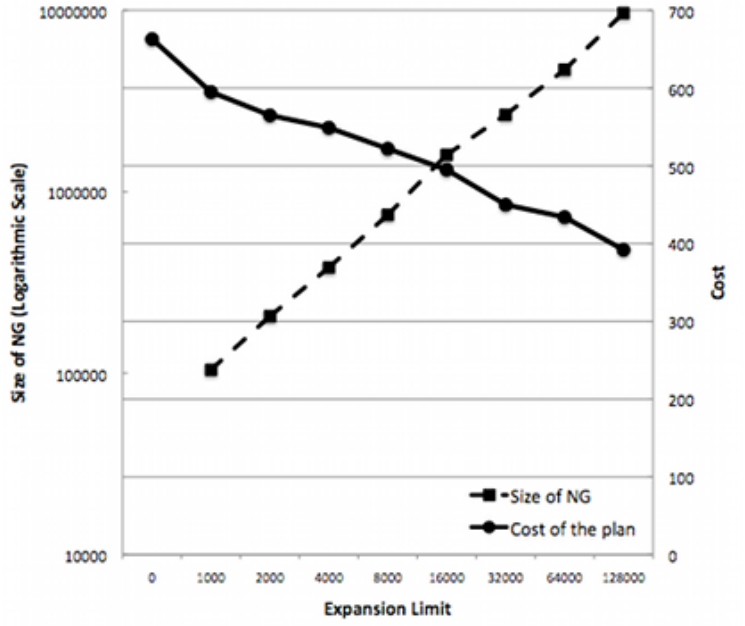


Figure 5.4: Plan cost and size of neighborhood graph for M_{A*} when varying the expansion limit in Elevators-22.

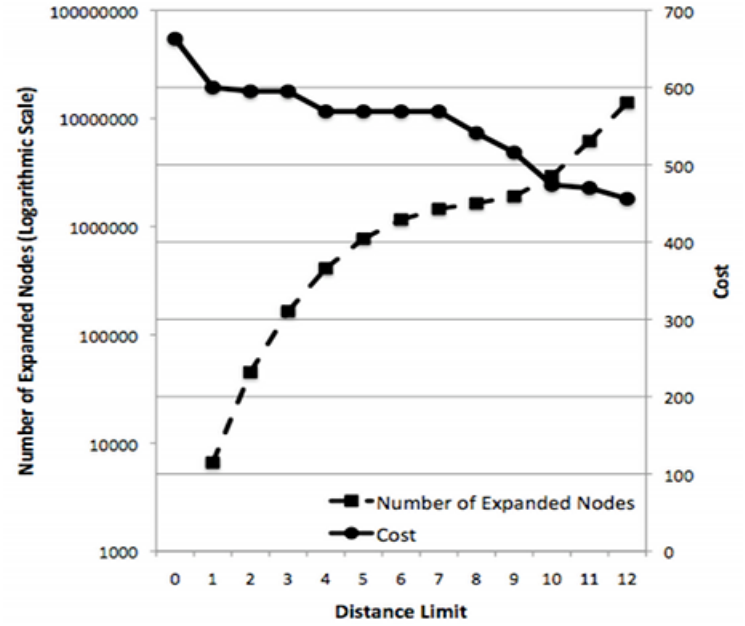


Figure 5.5: Plan cost and nodes expanded by ITSA* with varying d in Elevators-22.

AE is used before each iteration of PNGS. $M_{A*} + M_{bbfs}$ are used as search methods in PNGS. This combination works better than either AE or PNGS alone. Running AE alternately helps to remove actions that are no longer necessary due to reductions made by the previous iteration of PNGS. For example, if PNGS replaces a sequence of actions s with a less expensive alternative, then previous actions that were supporting propositions used by s may become redundant. PNGS cannot easily identify such redundancies since paths excluding these actions do not often hit another state in the plan; usually all nearby states in the plan already contain the effects generated by earlier, now redundant actions.

Table 5.4 lists the problem instances in which ARAS improved on the best previously known results. For each problem, the cost of the best previously known plan, the cost of ARAS' improved plan, and the base planner whose plan was used as the input are shown. An interesting entry in this table is Woodworking problem 13. While the original plans generated by the five planners had very different costs and lengths, after running ARAS with PNGS + AE*, all of them converged to the same cost 445, which improved on the previously best result of 555. Out of the total of 270 instances tested, ARAS with PNGS + AE* improved the best previously published results for 60 instances.

On average, a PNGS run consists of 10 to 12 iterations and each iteration takes 30 seconds. AE is much faster: the average time for a single run is less than a second.

5.6.3 Experiment 2: IPC-2004 - ARAS vs LPG

Table 5.2 summarizes the results for IPC-2004 with an IPC metric: cost of the best plan computed in all experiments divided by cost of the generated plan. Arvand-2009 plans were generated by a single run of the planner. LPG results are averaged over five runs. The timeout for planning and then postprocessing was set to 30 minutes total.

ARAS performs much better than LPG in improving the longer plans generated by Arvand-2009. The results are close for FF-generated plans, with a slight overall edge for ARAS. LPG and ARAS have different strengths since they search different spaces. Long plans with a large branching factor in plan space affect LPG much more than ARAS, while a large branching factor in state space does not necessarily slow down LPG's search in plan space. Apart from the search space, the heuristic search in LPG is better suited to find global alternatives for good quality plans generated by planners such as FF, than to finding local improvements in a long Arvand-2009 plan.

The results for Arvand-2009 in Satellite are interesting. In this domain, Arvand-2009 generates solutions with many unnecessary actions. LPG, focusing more on causal relations, is much better than PNGS in removing irrelevant actions. However, the combination of action elimination and PNGS can beat LPG: action elimination identifies irrelevant actions while PNGS searches for shortcuts.

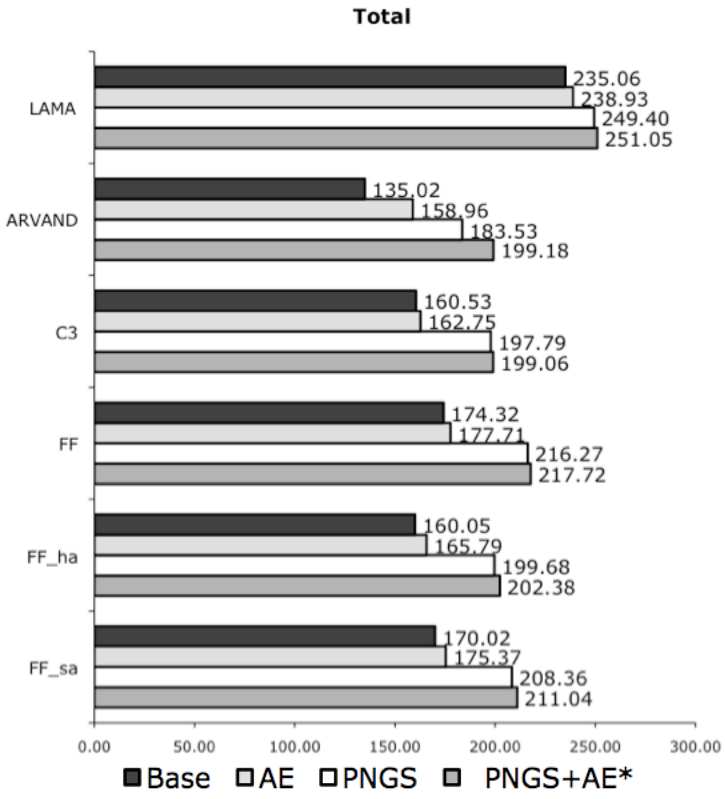


Figure 5.6: IPC scores of planners LAMA, Arvand-2009, FF, FF_{sa}, FF_{ha}, and C3 with ARAS versions AE, PNGS, PNGS + AE*.

Arvand-2009

Postprocessor	No Tankage	Tankage	Airport	Satellite	Total
None	15.26	16.20	44.6	5.01	81.07
LPG	41.43	27.22	**46.53**	32.43	147.62
AE	27.73	21.81	44.6	16.11	110.25
PNGS	44.26	39.98	45	23.68	152.92
PNGS+AE*	**45.6**	**42.47**	45	**33.62**	**166.69**

FF

Postprocessor	No Tankage	Tankage	Airport	Satellite	Total
None	25.57	16.88	35.2	33.9	111.55
LPG	33.01	18.37	**36.47**	**35.45**	123.30
AE	26.39	16.93	35.2	34.58	113.1
PNGS	34.66	21.45	35.6	34.67	126.38
PNGS+AE*	**34.81**	**21.45**	35.59	34.98	**126.83**

Table 5.2: Combining Arvand-2009 and FF with ARAS (AE, PNGS, PNGS+AE*) and LPG in four IPC-2004 domains.

5.6.4 Integration with Random Walk Planning

An anytime system integrating Arvand-2013 and an improved version of Aras is developed and tested. In this version, Aras runs PNGS+AE* but instead of using a fixed expansion limit L, it starts with initial $L = 1000$ and doubles L after each iteration. Therefore, the size of the neighborhood graph gradually increases over the time. The process continues until a memory or time limit is reached.

The whole integrated system works as follows: first a solution is found by Arvand-2013. This solution is saved and fed into anytime Aras to be improved. After Aras reaches its memory limit, which is set to 2 GB (no time limit is used; depending on the task and the input plan Aras hits the memory limit in 5 to 80 seconds), a new round of search plus postprocessing starts. This process of alternating Arvand-2013 and Aras continues until the time limit is reached. Any time a plan with better quality is found, it is saved.

Arvand-2013 also uses a bounding mechanism to stop episodes or random walks that already exceed the cost of a previously found solution. The solution bound, however, is only updated by plans generated by Arvand-2013 itself: the solution costs achieved by Aras are too tight for Arvand and significantly lower the probability of reaching any solution. Bounding using postprocessed plans, therefore, greatly decreases the number and diversity of the consecutive input plans for Aras: this has a detrimental effect on the best-quality plan that can be achieved by the system. (Xie, Valenzano, & Müller, 2013) studies the effect of the diversity and the number of input solutions on

the outcome of Aras. A similar system integrating Aras and RW planning was first used and tested in Arvand-2011.

Domain	Arvand-2013	LAMA-2011	FDSS2	FDSS1	Roamer
Scanalyzer	16.17	15.63	16.91	**17.70**	15.46
Pegsol	**19.88**	**19.88**	16.02	14.70	18.11
Floortile	5.00	4.46	**6.35**	5.44	1.63
Tidybot	11.22	14.53	11.23	**14.82**	13.03
Nomystery	**13.39**	11.33	10.80	13.33	9.51
Transport	12.10	12.39	9.14	9.46	**14.39**
Parcprinter	**19.00**	18.87	18.95	16.65	5.83
Elevators	8.64	10.62	8.70	**12.41**	11.74
Visitall	11.89	15.84	3.08	2.77	**16.89**
Parking	10.11	**16.96**	12.40	8.72	8.34
Woodworking	12.75	14.23	18.42	**18.56**	11.78
Barman	**19.93**	17.15	10.86	14.31	15.30
Sokoban	1.00	**16.28**	13.90	15.88	13.22
Openstacks	11.83	**18.36**	11.11	12.68	17.57
Total	172.88	**206.52**	167.88	177.43	172.80

Table 5.3: IPC score of the top three planners in IPC-2011, and of the two RW planners Arvand-2013 and Roamer.

To see how Arvand-2013 performs regarding the solution quality, experiments are run to compare Arvand-2013 with the top three planners achieving the highest IPC scores, which reflects the quality of the generated solutions, in IPC-2011: LAMA-2011, FDSS2, and FDSS1. Comparisons are also made with the RW planner Roamer, which achieved the 5th best IPC score in IPC-2011. Table 5.3 summarizes the results. The cost of the *best known* plans generated in IPC-2011 is used to compute the scores. While the search component in Arvand-2013 is not tuned for quality, overall Arvand-2013 scores lower than LAMA but is very competitive with other top planners, scoring 5 points higher than FDSS2, the third planner in IPC-2011. The improved Roamer performs better than its competition version, scoring at the same level of Arvand-2013. Arvand-2013 achieves the highest score in 4 domains: Pegsol, Nomystery, Parcprinter and Barman. Arvand-2013 also improved the best known plans for 15 tasks, 13 of which are from the Barman domain. This level of performance from RW planners is very promising and make them a strong alternative not only to increase the coverage but also to achieve high-quality solutions.

5.7 Conclusions

Experiments with the two plan improvement methods implemented in ARAS, Action Elimination and Plan Neighborhood Graph Search, show substantial improvements of a large variety of plans and for all tested planners. It is also shown that RW planning integrated with Aras achieves very competitive quality scores.

The limitation of the postprocessing techniques is that they can only find local improvements near the previous plan. This approach is ineffective in domains such as Cybersecurity or Openstacks.

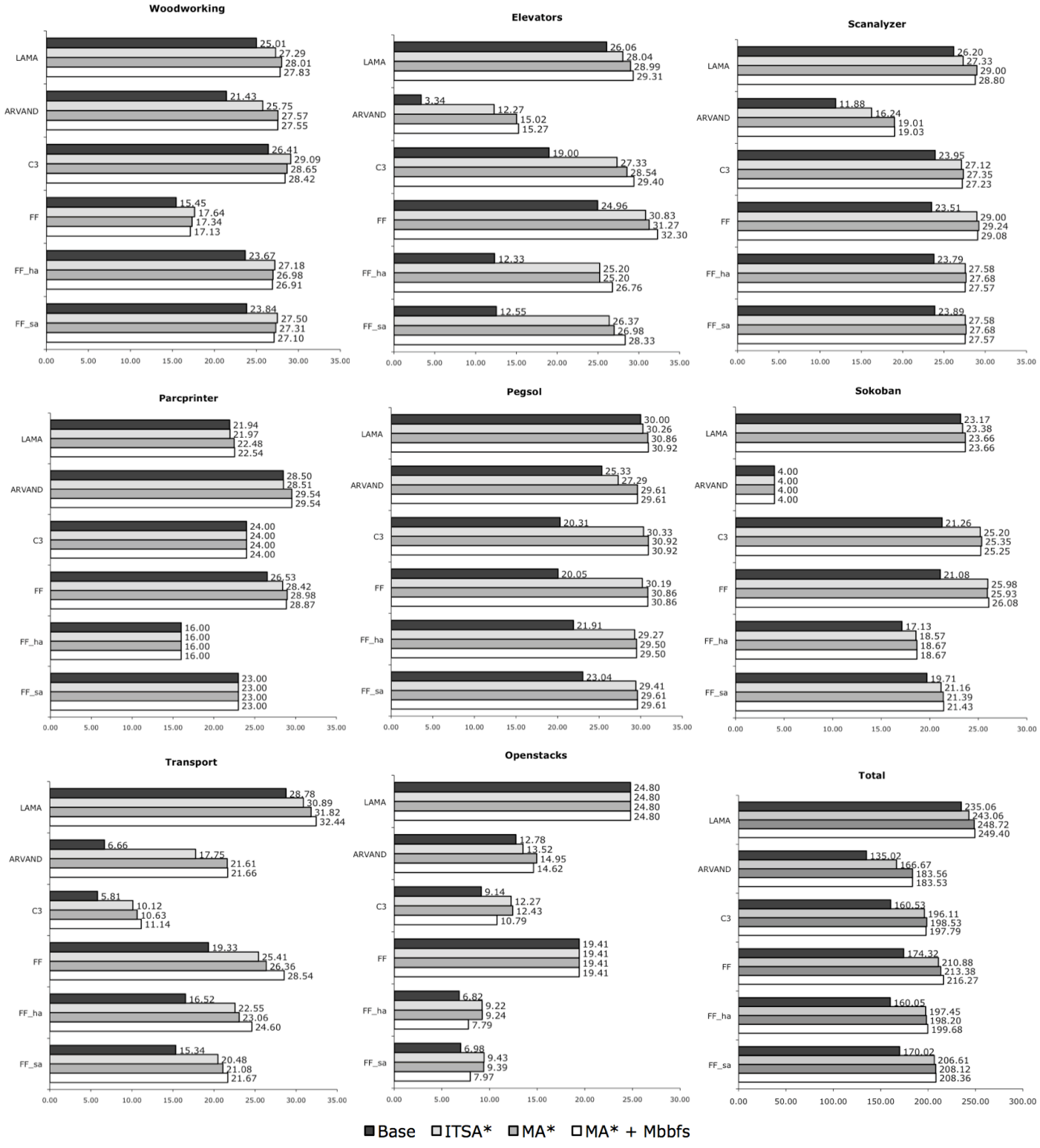


Figure 5.7: IPC scores of planners (LAMA, Arvand-2009, FF, FF$_{sa}$, FF$_{ha}$, and C3) combined with no postprocessors (base), ITSA*, ARAS M_{A*} and ARAS $M_{A*} + M_{bbfs}$. Total scores include Cybersecurity.

Table 5.4: Problems where ARAS improved the best known plans. "Best Known" is the cost of the best previously known plan generated by a planner. "ARAS" is the cost of the improved plan. For each pair of task and configuration the base planner is shown. Multiple entries indicate that the same result was achieved for input plans from more than one planner. "Total" (last row) show the number of tasks in which the configuration could improve on best known results.

Problem	**ITSA***	M_{A*}	$M_{A*} + M_{bbfs}$	**AE**	**PNGS + AE***	**Best Known**	**ARAS**
Elevators 03	-	LAMA , Arvand-2009 FF_{ha} , FF_{sa}, C3	LAMA , FF_{sa}	-	LAMA , Arvand-2009 FF_{ha} , FF_{sa}, C3	70	66
Elevators 04	-	C3	-	-	-	93	88
Elevators 06	-	FF_{ha}	FF	-	Arvand-2009	121	86
Elevators 07	-	LAMA	-	-	-	100	92
Elevators 08	-	LAMA , FF	LAMA	-	FF	94	88
Elevators 09	-	LAMA	LAMA	-	LAMA	101	99
Elevators 10	-	-	-	-	LAMA	187	146
Elevators 11	FF , FF_{sa}	FF , FF_{sa}	FF , FF_{sa}	-	Arvand-2009, FF, FF_{sa}	108	91
Elevators 12	-	-	FF	-	FF_{sa}	157	130
Elevators 13	-	-	FF_{ha}	-	-	186	142
Elevators 14	-	LAMA	-	-	-	208	183
Elevators 15	-	-	-	-	FF_{ha}	261	200
Elevators 16	-	-	-	-	LAMA	287	213
Elevators 17	-	-	-	-	FF , FF_{sa}	356	238
Elevators 18	-	-	-	-	FF_{sa}	387	293
Elevators 19	FF	-	-	-	-	311	297
Elevators 20	-	LAMA	LAMA	-	LAMA	429	247
Elevators 21	-	FF	-	-	FF	195	163

Continued on next page...

Table 5.4 – Continued

Problem	ITSA*	M_{A*}	$M_{A*} + M_{bbfs}$	AE	PNGS + AE*	Best Known	ARAS
Elevators 22	-	-	-	-	FF	362	312
Elevators 23	-	-	LAMA	-	LAMA	321	274
Elevators 24	-	-	-	-	FF_{ha}	541	439
Elevators 25	-	-	-	-	FF_{ha}	571	507
Elevators 26	-	-	-	-	FF_{ha}	809	547
Elevators 27	-	-	-	-	LAMA	773	489
Elevators 28	-	-	FF	-	FF	724	681
Elevators 29	-	-	-	-	LAMA	1045	605
Elevators 30	-	-	-	-	FF	769	694
Parcprinter 15	FF	LAMA , Arvand-2009 FF , FF_{ha} FF_{sa} , C3	LAMA , Arvand-2009 FF , FF_{ha} FF_{sa} , C3	LAMA , FF_{ha} FF_{sa} , C3	LAMA , Arvand-2009 FF , FF_{ha} FF_{sa} , C3	1695510	1695507
Parcprinter 16	FF	LAMA , FF FF_{ha} , FF_{sa} C3	LAMA , FF FF_{ha} , FF_{sa} C3	LAMA , FF_{ha} FF_{sa} , C3	LAMA , FF FF_{ha} , FF_{sa} C3	1675410	1675408
Parcprinter 17	FF	LAMA , FF FF_{ha}, FF_{sa}, C3	LAMA , FF FF_{ha} , FF_{sa}, C3	LAMA, FF_{ha} FF_{sa} , C3	LAMA , FF FF_{ha} , FF_{sa}, C3	1713580	1713576
Parcprinter 19	-	Arvand-2009 , FF FF_{ha} , FF_{sa}, C3	Arvand-2009 , FF FF_{ha} , FF_{sa}, C3	LAMA , FF_{ha} FF_{sa} , C3	LAMA , FF FF_{ha} , FF_{sa}, C3	3353260	3353256
Parcprinter 20	FF	FF , FF_{ha} FF_{sa} , C3	FF , FF_{ha} FF_{sa} , C3	FF_{ha} , FF_{sa} C3	FF , FF_{ha} FF_{sa} , C3	2754190	2754187
Pegsol 28	FF_{sa}	LAMA , Arvand-2009 FF , FF_{ha} FF_{sa} , C3	LAMA , Arvand-2009 FF , FF_{ha} FF_{sa} , C3	-	LAMA , Arvand-2009 FF , FF_{ha} FF_{sa} , C3	16	12

Continued on next page...

Table 5.4 – Continued

Problem	ITSA*	M_{A*}	$M_{A*} + M_{bbfs}$	AE	PNGS + AE*	Best Known	ARAS
Pegsol 29	-	LAMA , Arvand-2009 FF , FF_{sa} C3	LAMA , Arvand-2009 FF , FF_{sa} C3	-	LAMA , Arvand-2009 FF , FF_{sa} C3	14	11
Pegsol 30	-	C3	LAMA , C3	-	LAMA , C3	25	19
Transport 03	-	-	-	-	FF_{sa}	369	357
Transport 05	-	-	-	-	FF_{ha}	597	588
Transport 07	-	-	LAMA	-	LAMA	1260	861
Transport 08	-	-	-	-	FF	1216	947
Transport 09	-	-	LAMA	-	LAMA	1001	805
Transport 10	-	LAMA	LAMA	-	LAMA	1285	1083
Transport 13	FF_{ha} , FF_{sa}	-	LAMA , Arvand-2009 FF , FF_{ha}, FF_{sa}	-	LAMA , Arvand-2009 FF_{ha} , FF_{sa}	1125	959
Transport 14	-	-	-	-	FF	2157	1513
Transport 15	-	-	-	-	Arvand-2009	2954	2306
Transport 16	-	-	-	-	LAMA	4928	3692
Transport 17	-	-	-	-	FF_{ha}	4193	3826
Transport 18	-	-	LAMA	-	LAMA	4151	3707
Transport 19	-	-	-	-	LAMA	7648	5533
Transport 20	-	-	-	-	LAMA	6773	5761
Transport 23	-	-	-	-	FF	837	825
Transport 24	-	-	FF_{ha}	-	-	1301	1034
Transport 25	-	-	-	-	Arvand-2009	1833	1496
Transport 26	-	-	LAMA	-	LAMA	2502	2260

Continued on next page...

Table 5.4 – Continued

Problem	ITSA*	M_{A*}	$M_{A*} + M_{bbfs}$	AE	PNGS + AE*	Best Known	ARAS
Transport 27	-	-	-	-	FF	3317	2917
Transport 28	-	-	-	-	LAMA	3027	2867
Transport 29	-	-	-	-	FF_{ha}	3294	2505
Transport 30	-	-	-	-	FF	5513	5102
Woodworking 03	C3	FF_{ha} , FF_{sa} C3	FF_{ha} , FF_{sa} C3	-	FF_{ha} , FF_{sa} C3	620	445
Woodworking 04	-	C3	-	-	-	835	755
Woodworking 05	FF , C3	-	-	-	FF	685	545
Woodworking 07	-	-	-	-	Arvand-2009	1230	1070
Woodworking 08	C3	C3	C3	-	C3	1465	1460
Woodworking 10	-	-	-	-	Arvand-2009	1525	1470
Woodworking 13	-	LAMA , FF FF_{ha} , FF_{sa}	LAMA , FF FF_{ha} , FF_{sa}	-	LAMA , FF FF_{ha} , FF_{sa}, C3	555	445
Woodworking 14	C3	-	-	-	-	585	485
Woodworking 15	-	FF_{sa}	-	-	-	885	815
Woodworking 18	-	-	-	-	FF_{ha} , FF_{sa}	1310	1260
Woodworking 25	C3	-	-	-	-	650	640
Woodworking 26	FF	-	-	-	-	1000	985
Woodworking 27	C3	C3	C3	C3	C3	900	870
Woodworking 30	-	-	-	-	C3	1605	1515
Total	15	25	29	6	**60**	-	-

Chapter 6

Random Walk Planners

The current chapter gives an overview of the planning systems developed as an outcome of the research for this thesis. The algorithmic features that deviate from what is explored in Chapter 3 are briefly explained and the default configurations are given.

6.1 Arvand-2009: Establishing the foundation

Arvand-2009 (Nakhost & Müller, 2009) is the first RW planner developed. The general structure of the algorithm follows the same principles introduced in Chapter 3: random walks are used inside a local search framework to form the *neighborhood* by sampling from the search space. The key features different from what is discussed so far are the *length scaling* of RW, the use of biasing as a *fallback strategy*, *non-progressive* (NP) jumps and global restarting.

Walk length scaling: Instead of using a local restarting rate r_l, Arvand-2009 uses a length bound l_b. At each search step, the algorithm starts with an initial l_b. If the best seen h-value, h_{min}, does not change quickly enough, l_b is increased and the sample space iteratively expands. If the algorithm encounters better states frequently enough, l_b remains unchanged.

Biasing Activation: Arvand-2009 uses both MDA and MHA as fall-back strategies. The planner always starts with *pure* random walks using uniform action selection, and falls back to one of these biasing techniques only when one of the following thresholds is exceeded: MHA is switched on if the average branching factor exceeds 1000. Otherwise, MDA is activated whenever more than 50% of random walks hit a dead-end.

Non-progressive Jumps: Arvand-2013 keeps running random walks until either it reaches a state s with $h(s) < h_{min}$ or a restarting condition holds. In contrast, if Arvand-2009 does not reach a lower heuristic value after running *numW* walks, then it jumps to the state s that has the lowest h-value among the sampled states, even if $h(s) > h_{min}$. The heuristic value of the *current state* can increase.

Global Restarting: In contrast to Arvand-2013, which uses either t_g or r_g to control global restarting, Arvand-2009 uses a threshold on non-progressive jumps. The planner restarts after *maxNPJ*

Parameters	Pure RW	$MDA(T = 0.5)$	$MHA(w = 0, T = 10)$
α	0.9	0.9	0.9
$numW$	2000	2000	2000
$initialLengthW$	10	1	10
$extendingPeriod$	0.1	0.1	0.1
$extendingRate$	1.5	2	1.5
$maxNPJ$	7	7	7

Table 6.1: Default configurations for Arvand-2009.

non-progressive jumps.

6.1.1 Default Configuration

Table 6.1 shows the default parameter values. These values were determined based on initial experiments on a subset of IPC-2004 domains. The parameters $extendingPeriod$ and $extendingRate$ control the length scaling. For example, the values of 0.1 and 1.5 for pure RW mean that if h_{min} does not decrease over $0.1 \times numW$ random walks, then $initialLengthW$ is increased by a factor of 1.5.

Most parameter settings in Table 6.1 are the same for all three biasing schemes, except for length extension in MDA. The intuition was that most of the early random walks in problems with a high density of dead-end states are aborted. Therefore, the initial value of $initialLengthW$ is lowered to one, in order to decrease the probability of hitting a dead-end in early walks. A larger $extendingRate$ enables MDA to explore more in the later walks.

Arvand-2009 uses *acceptable progress* to control the jumping mechanism. Setting $\alpha = 0.9$ heavily biases the acceptable progress towards recent progress in the search. Settings for the temperature T reflect the fact that the average Q-values are much larger for MHA than for MDA.

6.2 Arvand-RC: Using RW Search for RCP

(Nakhost et al., 2012) show that RW search significantly outperforms systematic search in problems with scarce resources. Arvand-RC is an outcome of this research that uses the RW search principles. Compared to its predecessor Arvand-2009, Arvand-RC uses a new restarting technique *smart restarts* and a new jumping strategy called *on-path search continuation*. Both techniques were described in detail in Chapter 4.

6.3 Arvand-2011: Learning the Best Configuration and Using Aras

Arvand-2011 (Nakhost et al., 2011) is a successor of Arvand-2009 that instead of using MHA and MDA only as fallback strategies, includes them in a set of candidate configurations and uses the

UCB algorithm (Section 3.9) to select one. Arvand-2011 also benefits from a full integration with the postprocessor Aras (Chapter 5) to achieve high-quality solutions.

6.3.1 Default Configuration

Configurations	Config 1	Config 2	Config 3
$bias$	$MHA(w = 0, T = 10)$	$MHA(w = 0, T = 10)$	$MDA(T = 0.5)$
$initialLengthWalk$	1	10	1
$extendingRate$	2	1.5	2

Table 6.2: The configurations used in Arvand-2011

The default configuration learner of Arvand-2011 uses the three configurations shown in Table 6.2: two MHA versions with initial length of RW 1 and 10; and one MDA with initial length 1. Since in some problems running a search episode might be quite slow, and in the initial phase of UCB all the configurations are tried once, the best configuration might not be selected enough times to be able to solve the task. To remedy this problem, for the initial episodes a smaller number of random walks per search step is used to speed up the learning process. Specifically, for the first three episodes $numW$ is set to 100 and for the following episodes $numW$ is doubled up until it reaches the maximum 2000.

Like Arvand-2009, Arvand-2011 uses acceptable progress with $\alpha = 0.9$ for jumping and *maxNPJ* is set to 7. Arvand-2011 uses *smart restarts* and shares the pool between different configurations. This can have a positive effect of using different configurations for different parts of the search space: a configuration c_1 can start searching from a state s along a path traversed by another configuration c_2.

6.4 Arvand-LS: Random Walks with Memory

RW search as explored in Chapter 3 and implemented in Arvand-2009, Arvand-RC and Arvand-2011 does not use memory for duplicate state detection. This design choice has both advantages and disadvantages: the emerging planners can perform under very tight memory constraints, which proved to be a valuable feature for a solver in a portfolio (Valenzano et al., 2012). Re-expanding states, however, can waste computational time. Arvand-LS (Xie et al., 2012) uses an effective way to benefit from memory by combining systematic and RW explorations.

Figure 1 compares the search strategies of Arvand-LS and Arvand-2009. Both planners 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 Arvand-2009, 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

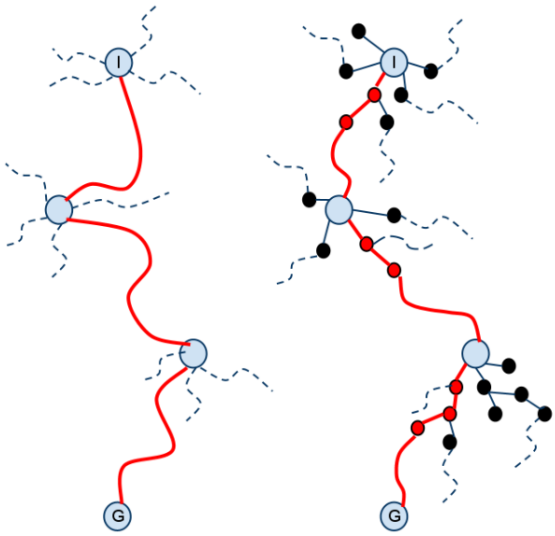


Figure 6.1: The search strategies of Arvand-2009 (left) and Arvand-LS (right). From (Xie et al., 2012).

from s ending in a state r. States in the open list are ordered by a linear combination of the heuristic values $h(s)$ and $h(r)$, that is, $w \times h(s) + h(r)$. The parameter w controls the trade-off between exploration and exploitation.

6.4.1 Default Configuration

Arvand-LS initializes $numWalks$ to 100. After each restart from the initial state, 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. This strategy of doubling $numWalks$ is inspired by the way UCB adjusts $numWalks$ in Arvand-2011. 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. Like Arvand-2009 $maxNPJ = 7$: in tests, the algorithm was robust against changes to $maxNPJ$ 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. Parameters are expressed as a tuple $(initialLenW, extensionRate, extensionPeriod, bias)$. The three configurations used are:

- *config-1*: $(10, 2, 0.1, MHA(T = 10, W = 0))$
- *config-2*: $(1, 2, 0.1, MDA(T = 0.5))$

- *config-3*: $(1, 2, 0.1, MHA(T = 10, W = 0))$

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

6.5 ArvandHerd: Parallel portfolio

ArvandHerd (Valenzano et al., 2012), the winner of the IPC-2011 multi-core track (Coles et al., 2012), is a portfolio planner that combines LAMA-2008 (Richter, Westphal, & Helmert, 2011) with several instances of Arvand-2011. Given a machine with n cores, ArvandHerd runs one instance of a modified LAMA-2008 on 1 core and $n - 1$ instances of Arvand-2011 on the remaining cores. LAMA-2008 is modified by using randomized operator ordering and restarting whenever the memory limit is reached. Furthermore, for tasks with non-uniform action costs, ArvandHerd runs LAMA-2008 with three heuristics: cost-sensitive and cost-ignorant versions of h^{FF} as well as h^{LM}. All these modifications are shown to improve the coverage of LAMA-2008 (Valenzano et al., 2012).

All instances of Arvand-2011 use the same restarting pool and share the UCB configuration learner. LAMA-2008 also has access to the pool, and whenever it finds a new solution it adds the solution trajectory. ArvandHerd uses both bounding and postprocessing to improve the quality of the solutions. Any time a solution is found, an instance of Aras is run on the same core to improve the quality. Both LAMA-2008 and ArvandHerd run in anytime mode. While LAMA-2008 uses the cost of the best solution found by any method as a bound, such pruning is ineffective for Arvand-2011, which instead only uses the best cost of a solution found strictly with Arvand-2011 as a bound. As Chapter 5 also discusses, creating a diverse set of plans with Arvand-2011 and improving them with Aras is more effective than forcing Arvand-2011 to create low cost plans directly by using the global bound.

6.5.1 Default Configuration

Since ArvandHerd is a portfolio planner, it contains parameters that control how the resources are shared between portfolio members. The effective settings for these parameters depend heavily on the available memory and time resources. The following explains the settings used in IPC-2011. The wall-clock time limit in the competition was 30 minutes and the memory limit was 6 GB. The planner could use up to 4 cores on a shared memory system.

ArvandHerd ran LAMA on one core and three copies of Arvand-2011 on the remaining three cores. Any time a solution is found, an instance of Aras is created and run on the current solution. If the initial solution was found by Arvand-2011, Aras is given a 60 second time-limit. If the initial solution was found by LAMA, Aras is given a 40 second time-limit. This limit is lower for LAMA since that planner already has a fairly effective plan improvement scheme.

As the memory requirements of Arvand-2011 are limited to space for the current trajectory, the

best random walk seen thus far, the walk pool, and the UCB configuration selection, it uses much less than the 6 GB memory limit. This is not the case for Aras and LAMA. As such, these processes need to be prevented from exhausting all the memory given to the planner, thereby crashing the whole system, and preventing further search by the processors running Arvand-2011. To address this problem, the PNGS phase of each Aras instance is limited to using only 500 MB, and the total memory of the open and closed lists in LAMA is set to 2.7 GB. If the Aras limit is hit, Aras quits and returns the best solution found so far. If the LAMA limit is hit, the current search iteration is ended and the open lists are emptied. The next iteration of LAMA then begins with the possibility that the diversity introduced by changing the weight and tie-breaking may avoid the mistakes made on the previous iterations. If the final 0-weight iteration also runs out of memory, this processor starts running another copy of Arvand-2011 instead.

ArvandHerd uses the following four configurations to run random walks:

- *config-1*: $(10, 1.5, 0.1, MHA(T = 10, w = 0))$
- *config-2*: $(1, 1.5, 0.1, MHA(T = 10, w = 0))$
- *config-3*: $(3, 1.5, 0.1, MDA(T = 0.5))$
- *config-4*: $(1, 2, 0.1, MDA(T = 0.5))$

The activation level and size of the walk pool is set to 100. Other parameters controlling RW search are the same as in Arvand-2011.

Chapter 7

Conclusions

The current chapter summarizes the contributions of this thesis and proposes several directions to follow for future work.

7.1 Summary of Contributions

This thesis proposed RW search as an effective framework for satisficing planning. It started with a theoretical study that revealed the potential of using RW exploration. Based on the insights gained from the theoretical models and detailed experiments, several planning systems were developed that improved the state of the art in resource-constrained planning, parallel planning, and satisficing planning in general. Beyond the design and study of new search algorithms, this thesis has made significant contributions to research areas including RCP and plan improvement by providing new benchmarks and effective postprocessing techniques.

Chapter 2 introduced homogenous graphs as a framework to study the performance of restarting and non-restarting random walks in plateaus. The key characteristics that affect the performance of RW are: regress factor, the largest goal distance D and the initial goal distance. The conclusion was that in domains with a regress factor smaller than the effective branching factor and the initial goal distance close to D, RW exploration is much faster than systematic exploration. Another result is that the runtime of restarting random walks does not depend on D. This leads to a big advantage when the initial goal distance is small. Connections between the theoretical models and the planning benchmarks were also studied, and it was shown how the results in these models provide upper bounds for more general graphs.

Chapter 3 used an experimental approach to build and study effective search algorithms that use random walks. Key parameters affecting the local and global exploration were identified and tested. In terms of algorithm design this study had several significant outcomes:

1. The general RWS framework, which uses RW exploration in a local search.

2. The adaptive systems ALR and AGR that dynamically adjust two key parameters: the global restarting rate and the length of walks.

3. The biasing techniques MHA and MDA that use information such as preferred operators and the frequency of dead-end states to significantly improve the performance.

4. A configuration learning system that uses bandit algorithms to find the most effective configuration of the planner.

Chapter 3 also presented a detailed study of the design space of RWS, providing valuable insights regarding the effect of other parameters such as the heuristic function, the jumping strategies, and the rate at which the states are evaluated.

Chapter 4 studied RWS as a tool to solve RCP problems. This study made significant contributions to RCP by extending the notion of resource constrainedness C to the case of multiple resources, and building two new benchmarks, NoMystery and Rovers, with problem generators that allow to control C. The development of these generators involved designing and implementing efficient domain-specific optimal solvers. Extensive experiments were also run to measure the performance of a wide range of satisficing and optimal planners as a function of C. RWS improved by smart restarts and on-path search continuation significantly outperformed other planners when C is close to 1. The problem generators and the benchmark problems created for this research are publicly available.

Chapter 5 introduced novel postprocessing techniques, PNGS and PNGS+AE, to improve solution quality. The developed algorithms are fast and are not dependent on any planning system. Even top planning systems such as LAMA, which are designed to generate high-quality solutions, can greatly benefit from the developed techniques. Chapter 5 also proposed an effective method that integrates RW planning with postprocessing. This anytime system performs at the level of top systematic planners and achieves the highest score in four IPC-2011 domains.

Chapter 6 showed how principles of RWS can be used to build effective planning systems including the winner of the IPC-2011 multicore track, ArvandHerd. Compared with common satisficing planners, RW planners inherently have very different strengths and weaknesses. This makes them an appealing choice for portfolio planners such as ArvandHerd. This work on ArvandHerd explored a simple effective way of exploiting the strengths of RW. A related contribution is the planning system Arvand-LS, which uses the RWS framework combined with local GBFS.

7.2 Limitations and Open questions

The following discusses some of the limitations of the current research and proposes interesting directions for future work. The topics follow the same order as in the thesis.

7.2.1 Theoretical Framework for RW Planning

A limitation is that the relation between the theoretical models and the results of the experiments on RW planning is not clear. The main reason is that the distribution of the values for the key parameters

such as regress factors are unknown for full planning benchmarks. Another key reason is the lack of a model for studying the effect of search enhancements. Answering the following open questions can establish the relation between theory and practice more clearly.

Relation to full planning benchmarks: Can they be described within these models in terms of bounds on their regress factor? Can the models be extended to represent the core difficulties involved in solving more planning domains? What is the structure of plateaus within their state spaces, and how do plateaus relate to the overall difficulty of solving those instances? Instances with small state spaces could be completely enumerated and such properties measured. For larger state spaces, can measurements of true goal distances be approximated by heuristic evaluation, by heuristics combined with local search, or by sampling?

Effect of search enhancements: To move from abstract, idealized algorithms towards more realistic planning algorithms, it would be interesting to study the whole spectrum starting with the basic methods studied in this thesis up to state of the art planners, switching on improvements one by one and studying their effects under both RW and systematic search scenarios. For example, the RW enhancements MHA and MDA (Chapter 3) should be studied.

Hybrid methods: Develop theoretical models for methods that combine random walks with using memory and systematic search such as (Lu et al., 2011; Xie et al., 2012).

Tighter bounds: FH graphs can be used to obtain upper bounds for any fair graph and IRH graphs provide upper bounds for any homogenous graph. Finding tighter bounds for arbitrary graphs is an interesting future work.

Backtracking random walks: A fundamental problem with non-restarting random walks is that they cannot recover from a dead end. This can be fixed either with restarting or backtracking. While Chapter 2 analyzes the former in detail, the latter is left as future work. Backtracking RW remembers the path traversed from the start state to the current state s and randomly selects the next state from the augmented successor set $(S_G(s) \cup p(s))$, where $S_G(s)$ contains all successors of s and $p(s)$ is the predecessor of s. It might be possible to extend the results for non-restarting RW on FH graphs to backtracking walks in any homogenous graph. The idea is that if we measure the goal distance of a state s based on the paths that the random walk can traverse from s, then backtracking walks can never change this distance by more than one unit. An advantage is that the graph will be fair using this measure. A complication, however, is that this new measure is not static and changes depending on the path that the walk uses to reach s.

7.2.2 Random Walk Search Framework

The RWS framework introduced here is limited to local search. Therefore, all limitations of local search are inherited: the search algorithm is incomplete and it might keep getting stuck in promising local minima or dead-ends and never recover from them even with restarting. Future work includes investigating alternative designs beyond the local search framework and also several unexplored

possibilities inside the framework.

Exploration in path-commitment: In all the planners developed here, except Arvand-RC, the implicit strategy is to randomly explore the states regardless of their heuristic value, but *commit* (jump) to the one that has the lowest h-value. The underlying assumption is that local exploration is much cheaper than path-commitment: if a RW expands a *bad* state, e.g., a dead end, the only thing that is wasted is the time used to run the rest of the walk. However, if the algorithm jumps to a bad state, the whole time spent to run the rest of the episode is wasted. However, exploration in path-commitment can be effective if it is balanced with the cost of a possible mistake. The *on-path search continuation* used in Arvand-RC partially addresses this issue by decreasing the cost of commitment. Other search strategies such as simulated annealing (Hoos & Stützle, 2004) are also interesting to explore: In this method, a biased random choice that favours lower heuristic values is used to decide whether to commit or not.

The parameter n in the RWS framework, the number of walks before jumping, also affects the path-commitment strategy. A larger n increases the number of samples and the chance of committing to a state with lower heuristic value. Deriving a model to determine an effective n based on key search space characteristics is also an interesting topic for future work.

Multiple Heuristics: As Chapter 1 explains, state of the art planners such as LAMA and Fast Downward can combine multiple heuristics by using multiple queues. Since RWS does not use open lists, the idea of using multiple queues is not directly applicable. However, the same general principles of running parallel searches and sharing information can be used. Algorithm 6 gives the pseudocode of such a technique for RW with multiple heuristics. Compare this algorithm with Algorithm 1 (Chapter 3), which uses only one heuristic: Instead of having just one $currentState$ and one variable h_{min}, the algorithm keeps a separate current state $currentState[i]$ and $h_{min}[i]$ for each input heuristic function h_i. To run a random walk, the algorithm selects one of the current states uniformly randomly as the starting point. For all heuristic functions h_i the algorithm updates both $currentState[i]$ and $h_{min}[i]$ as soon as it samples a state s with $h_i(s) < h_{min}[i]$. The algorithm restarts from the initial state if the last T_G walks did not decrease any of the h_{min} values. The random walks are the same except that evaluations of states uses all heuristics, not just one. The effectiveness of such an algorithm remains to be investigated.

Beyond local search: Random walks are not fundamentally limited to local search. Roamer (Lu et al., 2011) is an example of successful application of RW exploration to global search. A key question here is whether the main advantages of RW and systematic exploration can be combined. A systematic way of running random walks combined with duplicate detection might be the answer. An algorithm using similar concepts is *Monte Carlo Tree Search* (Browne et al., 2012), which uses a combination of systematic in-tree exploration and random playouts. Although the research on Arvand-LS focuses on local search, it explores some of these concepts.

Algorithm 6 Random Walks with Multiple Heuristics

Input Initial State s_0, goal condition G
available actions A and heuristic functions $(h_1, \dots h_n)$
Output A solution plan
Parameters r_l

Initialize $currentState$ to a list containing n copies of s_0
$h_{min} \leftarrow [h_1(s_0), h_2(s_0), \dots, h_n(s_0)]$
loop
  $startState \leftarrow selectRandomly(currentState)$
  $sampledState \leftarrow RandomWalk(startState, G, h_{min}, r_l)$
  if $sampledState \supseteq G$ **then**
    return the plan reaching the $sampledState$
  else if $sampledState \neq Deadend$ **then**
    for $i = 1 \rightarrow n$ **do**
      if $h_i(sampledState) < h_{min}[i]$ **then**
        $(currentState[i], h_{min}[i]) \leftarrow (sampledState, h_i(sampledState))$
      end if
    end for
  end if
  if $Restart()$ **then**
    Initialize $currentState$ to a list consisting of n copies of s_0 {restart from initial state}
    $h_{min} \leftarrow [h_1(s_0), h_2(s_0), \dots, h_n(s_0)]$
  end if
end loop

7.2.3 Resource-constrained Planning

A limitation of RWS developed for RCP is that they do not exploit an explicit encoding of resources. The danger here probably would be to not over-fit the algorithms and lose too much performance elsewhere. Apart from this, important topics for future research include: developing more problem generators that allow to control C, understanding the behaviour of various algorithms, such as M and Mp, better; and investigating to what extent we can devise automatic configuration methods for deciding whether or not to switch these tailored techniques on.

7.2.4 Plan Improvement

While compared with anytime systems such as LAMA, Aras is much faster, it is limited to local improvements. That is why Aras does not improve solutions in domains such as Openstacks and Cybersecurity. There are many promising directions for future work on plan improvement:

- Improve Action Elimination to be more efficient and find more reductions. Action Elimination can be formulated as an optimization problem. Search algorithms such as hill-climbing could be used to find better solutions.

- Aras is capable of processing multiple input plans at the same time. Experimental results show a rather large variation in the improvability of different plans for the same problem instance. Therefore, using a large number of input plans should increase the chances of the system to

find good improved plans. Inputs for ARAS could include many diverse plans as in (Srivastava et al., 2007), or multiple randomized solutions generated by a RW planner.

- Instead of expanding nodes uniformly, adapt the search effort per node in PNGS.
- Focus more on avoiding expensive actions in PNGS.
- Use PNGS during the search to improve partial plans.
- Investigate the effect of macros on plan improvability: for example, compare ARAS on plans produced by standard FF (Hoffmann & Nebel, 2001) and Macro-FF (Botea, Enzenberger, Müller, & Schaeffer, 2005).

7.2.5 Planning Systems

Arvand-2013 has been built with many and easily exposed parameters according to the *Programming by Optimization (PbO)* paradigm (Hoos, 2012). Work with Chris Fawcett has started on tuning the system both for overall performance and on a per-domain basis using the ParamILS configurator (Hutter, Hoos, Leyton-Brown, & Stützle, 2009). Other interesting work includes building a new version of ArvandHerd using Arvand-2013 and LAMA-2011, and to make the planning systems developed here open source.

Bibliography

Aine, S., Chakrabarti, P. P., & Kumar, R. (2007). AWA* - a window constrained anytime heuristic search algorithm. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007, Hyderabad, India, January 6-12, 2007*, pp. 2250–2255.

Alcázar, V., & Veloso, M. M. (2011). BRT: Biased rapidly-exploring tree. In *The 2011 International Planning Competition, IPC 2011, Universidad Carlos III de Madrid*, pp. 17–20.

Alcázar, V., Veloso, M. M., & Borrajo, D. (2011). 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*.

Aldous, D., & Fill, J. (2002). *Reversible Markov Chains and Random Walks on Graphs*. University of California, Berkeley, Department of Statistics.

Ambite, J. L., & Knoblock, C. A. (2001). Planning by rewriting. *Journal of Artificial Intelligence Research*, *15*, 207–261.

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, *47*, 235–256.

Bäckström, C. (1998). Computational aspects of reordering plans. *Journal of Artificial Intelligence Research*, *9*, 99–137.

Bäckström, C., & Nebel, B. (1995). Complexity results for SAS+ planning. *Computational Intelligence*, *11*, 625–656.

Blum, A., & Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, *90*(1-2), 281–300.

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

Botea, A., Enzenberger, M., Müller, M., & Schaeffer, J. (2005). Macro-FF: Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research*, *24*, 581–621.

Botea, A., Müller, M., & Schaeffer, J. (2007). Fast planning with iterative macros. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007, Hyderabad, India, January 6-12, 2007*, pp. 1828–1833.

Brightwell, G., & Winkler, P. (1990). Maximum hitting time for random walks on graphs. *Random Struct. Algorithms*, *1*, 263–276.

Browne, C., Powley, E. J., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012). A survey of Monte Carlo Tree Search methods. *IEEE Trans. Comput. Intellig. and AI in Games*, *4*(1), 1–43.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, *69*(1-2), 165–204.

Cazenave, T. (2009). Nested Monte-Carlo search. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009, Pasadena, California, USA, July 11-17, 2009*, pp. 456–461.

Coles, A., & Coles, A. (2011). LPRPG-P: Relaxed plan heuristics for planning with preferences. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011*.

Coles, A. J., Coles, A., Olaya, A. G., Jiménez, S., López, C. L., Sanner, S., & Yoon, S. (2012). A survey of the seventh international planning competition. *AI Magazine*, *33*(1).

Coles, A., Fox, M., Long, D., & Smith, A. (2008). A hybrid relaxed planning graph - LP heuristic for numeric planning domains. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*, pp. 52–59.

Coles, A., Fox, M., & Smith, A. (2007). 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*, pp. 89–96.

Cushing, W., Benton, J., & Kambhampati, S. (2011). Cost based satisficing search considered harmful. In *ICAPS 2011 Workshop on Heuristics for Domain-independent Planning*, pp. 43–52.

Do, M. B., & Kambhampati, S. (2003). Improving temporal flexibility of position constrained metric temporal plans. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling, ICAPS 2003, June 9-13, 2003, Trento, Italy*, pp. 42–51.

Domshlak, C., Karpas, E., & Markovitch, S. (2010). To max or not to max: Online learning for speeding up optimal planning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, pp. 1071–1076.

Dvorak, F., & Barták, R. (2010). Integrating time and resources into planning. In *Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2010, Arras, France, 27-29 October 2010*, Vol. 2, pp. 71–78.

Edelkamp, S. (2003). Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Intelligence Research*, *20*, 195–238.

Edelkamp, S. (2004). Generalizing the relaxed planning heuristic to non-linear tasks. In *Proceedings of the 27th Annual German Conference on AI, KI 2004, Ulm, Germany, September 20-24, 2004*, pp. 198–212.

Eén, N., & Sörensson, N. (2003). An extensible SAT-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003*, pp. 502–518.

Fama, E. F. (1965). Random walks in stock-market prices. *Financial Analysts Journal*, *21*, 55–59.

Fawcett, C., Helmert, M., Hoos, H., Karpas, E., Röger, G., & Seipp, J. (2011). FD-Autotune: Automated configuration of Fast Downward. In *The 2011 International Planning Competition, IPC 2011, Universidad Carlos III de Madrid*, pp. 31–37.

Fikes, R., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence. London, UK, September 1971*, pp. 608–620.

Fink, E., & Yang, Q. (1992). Formalizing plan justifications. In *Proceedings of the Ninth Conference of the Canadian Society for Computational Studies of Intelligence, CAIAC 1992, 1992*, pp. 9–14.

Finnsson, H., & Björnsson, Y. (2008). Simulation-based approach to general game playing. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008*, pp. 259–264.

Fox, M., & Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, *20*, 61–124.

Furcy, D. (2006). ITSA*: Iterative tunneling search with A*. In *AAAI Workshop on Heuristic Search, Memory-Based Heuristics and Their Applications*, pp. 21–26.

Gelly, S., & Silver, D. (2008). Achieving master level play in 9 x 9 computer Go. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008*, pp. 1537–1540.

Gerevini, A., Saetti, A., & Serina, I. (2003). Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research*, *20*, 239–290.

Gerevini, A., Saetti, A., & Serina, I. (2008). An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artificial Intelligence*, *172*(8-9), 899–944.

Gerevini, A., & Serina, I. (2002). LPG: A planner based on local search for planning graphs with action costs. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, April 23-27, 2002, Toulouse, France*, pp. 13–22.

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated Planning: Theory & Practice*. Morgan Kaufmann.

Gittins, J., Glazebrook, K., & Weber, R. (2011). *Multi-armed bandit allocation indices*. Wiley.

Gkantsidis, C., Mihail, M., & Saberi, A. (2006). Random walks in peer-to-peer networks: algorithms and evaluation. *Perform. Eval.*, *63*, 241–263.

Hansen, E. A., & Zhou, R. (2007). Anytime heuristic search. *Journal of Artificial Intelligence Research*, *28*, 267–297.

Haslum, P., & Geffner, H. (2001). Heuristic planning with time and resources. In *Proceedings of the 6th European Conference on Planning, ECP 2001, Toledo, Spain, September 12-14, 2001*, pp. 121–132.

Heckman, I., & Beck, J. C. (2011). Understanding the behavior of solution-guided search for job-shop scheduling. *Journal of Scheduling*, *14*(2), 121–140.

Helmert, M. (2004). A planning heuristic based on causal graph analysis. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling, ICAPS 2004, June 3-7, 2004, Whistler, British Columbia, Canada*, pp. 161–170.

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

Helmert, M. (2008). *Understanding planning tasks: domain complexity and heuristic decomposition*, Vol. 4929 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Helmert, M., Do, M., & Refanidis, I. (2008). International Planning Competition-2008, Deterministic Part.. Available at `http://ipc.informatik.uni-freiburg.de/`.

Helmert, M., & Domshlak, C. (2009). Landmarks, critical paths and abstractions: What's the difference anyway. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*, pp. 162–169.

Helmert, M., & Geffner, H. (2008). Unifying the causal graph and additive heuristics. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*, pp. 140–147.

Helmert, M., Haslum, P., & Hoffmann, J. (2007a). Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26*, pp. 176–183.

Helmert, M., Haslum, P., & Hoffmann, J. (2007b). Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, pp. 176–183.

Helmert, M., Röger, G., & Karpas, E. (2011). Fast Downward Stone Soup: A baseline for building planner portfolios. In *ICAPS 2011 Workshop on Planning and Learning*, pp. 13–20.

Hoffmann, J. (2003). The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*, *20*, 291–341.

Hoffmann, J. (2005). Where 'Ignoring Delete Lists' works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research*, *24*, 685–758.

Hoffmann, J. (2011). Analyzing search topology without running any search: On the connection between causal graphs and h+. *Journal of Artificial Intelligence Research*, *41*(2), 155–229.

Hoffmann, J., Kautz, H., Gomes, C., & Selman, B. (2007). SAT encodings of state-space reachability problems in numeric domains. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007, Hyderabad, India, January 6-12, 2007*, pp. 1918–1923.

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

Hoffmann, J., Porteous, J., & Sebastia, L. (2004). Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, *22*, 215–278.

Hoos, H. (2012). Programming by optimization. *Communications of the ACM*, *55*(2), 70–80.

Hoos, H., & Stützle, T. (2004). *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann.

Hutter, F., Hoos, H., Leyton-Brown, K., & Stützle, T. (2009). ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, *36*, 267–306.

Kautz, H. A., & Selman, B. (1996). Pushing the envelope: Planning, propositional logic and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence, AAAI 96, Portland, Oregon, August 4-8, 1996*, Vol. 2, pp. 1194–1201.

Koehler, J. (1998). Planning under resource constraints. In *Proceedings of the 13th European Conference on Artificial Intelligence, ECAI 1998, Brighton, UK, August 23-28, 1998*, pp. 489–493.

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press, Cambridge, U.K.

Long, D., Kautz, H. A., Selman, B., Bonet, B., Geffner, H., Koehler, J., Brenner, M., Hoffmann, J., Rittinger, F., Anderson, C. R., Weld, D. S., Smith, D. E., & Fox, M. (2000). The AIPS-98 planning competition. *Artificial Intelligence*, *21*(2), 13–33.

Lourenco, R. H., Martin, O., & Stützle, T. (2003). Iterated local search. *Handbook of metaheuristics*, 320–353.

Lovász, L. (1993). Random walks on graphs: A survey. *Combinatorics, Paul Erdös is Eighty*, *2*(1), 1–46.

Lu, Q., Xu, Y., Huang, R., & Chen, Y. (2011). The Roamer planner random-walk assisted best-first search. In *The 2011 International Planning Competition, IPC 2011, Universidad Carlos III de Madrid*, pp. 73–76.

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

Nakhost, H., & Müller, M. (2009). 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*, pp. 1766–1771.

Nakhost, H., & Müller, M. (2010). Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, pp. 121–128.

Nakhost, H., & Müller, M. (2012). A theoretical framework to study random walk planning. In *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS 2012, Niagara Falls, Canada, July 19-21, 2012*.

Nakhost, H., & Müller, M. (2013). Towards a second generation random walk planner: an experimental exploration. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI 2013, to appear*.

Nakhost, H., Müller, M., Valenzano, R., & Xie, F. (2011). Arvand: the art of random walks. In *The 2011 International Planning Competition, IPC 2011, Universidad Carlos III de Madrid*, pp. 15–16.

Nissim, R., Hoffmann, J., & Helmert, M. (2011). Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011, Barcelona, Catalonia, Spain, July 16-22, 2011*, pp. 1983–1990.

Norris, J. R. (1998). *Markov chains*. Cambridge series in statistical and probabilistic mathematics. Cambridge University Press.

Pardoux, É. (2009). *Markov processes and applications: algorithms, networks, genome and finance*. Wiley series in probability and statistics. Wiley/Dunod.

Penberthy, J. S., & Weld, D. S. (1992). UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning, KR 1992, Cambridge, MA, October 25-29, 1992*, pp. 103–114.

Pham, D. N., Thornton, J., Gretton, C., & Sattar, A. (2008). Combining adaptive and dynamic local search for satisfiability. *Journal on Satisfiability, Boolean Modeling and Computation*, *4*(2-4), 149–172.

Qian, H., Nassif, S. R., & Sapatnekar, S. S. (2003). Random walks in a supply network. In *Proceedings of the 40th Design Automation Conference, DAC 2003, Anaheim, CA, USA, June 2-6, 2003*, pp. 93–98.

Ratner, D., & Pohl, I. (1986). Joint and LPA*: combination of approximation and search. In *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, August 11-15, 1986.*, pp. 173–177.

Richter, S., & Helmert, M. (2009). Preferred operators and deferred evaluation in satisficing planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*, pp. 273–280.

Richter, S., & 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., & Helmert, M. (2011). LAMA 2008 and 2011. In *The 2011 International Planning Competition, IPC 2011, Universidad Carlos III de Madrid*, pp. 17–20.

Rintanen, J. (2012). Planning as satisfiability: Heuristics. *Artificial Intelligence*, *193*, 45–86.

Russell, S. J., & Norvig, P. (2010). *Artificial intelligence: a modern approach*. Prentice Hall Upper Saddle River, NJ.

Selman, B., Kautz, H., & Cohen, B. (1993). Local search strategies for satisfiability testing. *Cliques, coloring, and satisfiability: Second DIMACS implementation challenge*, *26*, 521–532.

Selman, B., Levesque, H. J., & Mitchell, D. (1992). A new method for solving hard satisfiability problems. In *Proceedings of the 10th National Conference on Artificial Intelligence, AAAI 1992, San Jose, CA, July 12-16, 1992*, pp. 440–446.

Srivastava, B., Nguyen, T. A., Gerevini, A., Kambhampati, S., Do, M. B., & Serina, I. (2007). Domain independent approaches for finding diverse plans. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007, Hyderabad, India, January 6-12, 2007*, pp. 2016–2022.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Thayer, J. T., & Ruml, W. (2008). Faster than weighted A*: An optimistic approach to bounded suboptimal search. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*, pp. 355–362.

Thayer, J. T., Stern, R., Felner, A., & Ruml, W. (2012). Faster bounded-cost search using inadmissible estimates. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*.

Upal, M. A. (1999). Learning rewrite rules to improve plan quality. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence, AAAI 1999, July 18-22, 1999, Orlando, Florida, USA*, p. 984.

Valenzano, R., Nakhost, H., Müller, M., Schaeffer, J., & Sturtevant, N. (2012). ArvandHerd: Parallel planning with a portfolio. In *Proceedings of the 20th European Conference on Artificial Intelligence, ECAI 2012, Montpellier, France, August 27-31, 2012*, pp. 113–116.

Veloso, M. M., Pérez, M. A., & Carbonell, J. G. (1990). Nonlinear planning with parallel resource allocation. In *DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pp. 207–212. Morgan Kaufmann.

Vidal, V. (2004). A lookahead strategy for heuristic search planning. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling, ICAPS 2004, June 3-7 2004, Whistler, British Columbia, Canada*, pp. 150–160.

Wilt, C. M., & Ruml, W. (2011). Cost-based heuristic search is sensitive to the ratio of operator costs. In *Proceedings of the Forth Annual Symposium on Combinatorial Search, SOCS 2012, Barcelona, Spain, July 15-16, 2011*.

Xie, F., Nakhost, H., & Müller, M. (2012). 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*, pp. 315–322.

Xie, F., Valenzano, R., & Müller, M. (2013). Better quality search via randomization and postprocessing under time constraint. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling ICAPS 2013, to appear*.

Yin, G., & Zhang, Q. (2005). *Discrete-time Markov chains: two-time-scale methods and applications*. Applications of mathematics. Springer.

Yoon, S., Fern, A., & Givan, R. (2007). FF-Replan: A baseline for probabilistic planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, Vol. 7, pp. 352–359.