Optimization for Heuristic Search

by

David Christopher Ferguson Rayner

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

University of Alberta

Abstract

Heuristic search is a central problem in artificial intelligence. Among its defining properties is the use of a *heuristic*, a scalar function mapping pairs of states to an estimate of the actual distance between them. Accurate heuristics are generally correlated with faster query resolution and higher-quality solutions in a variety of settings, including GPS road navigation and video game pathfinding. Effective methods for defining heuristics remain at the forefront of heuristic search research.

This research puts the task of constructing good heuristics under the lens of optimization: minimizing a loss between the true distances and the heuristic estimates, subject to admissibility and consistency constraints. Starting with first principles and well-motivated loss functions, we show several instances where performing this optimization is both feasible and tractable. This novel approach reveals previously unobserved connections to other computing subfields (e.g., graph embedding), gives new insights into previous approaches to heuristic construction (e.g., differential heuristics), and proves empirically competitive in a number of domains.

Acknowledgements

Certainly not an exhaustive list, but my heartfelt thanks goes to:

My supervisors, Michael Bowling and Nathan Sturtevant, for their guidance and inspiration; and for helping me transform a curious observation into a deep, exciting, and significant research project.

Rob Holte, Dale Schuurmans Martin Müller, and Kilian Weinberger – each went above and beyond his duties as a supervisory and/or committee member, and each provided me with valued guidance along the way.

Vadim Bulitko and Rich Sutton – both inside and outside of their past supervisory roles – for numerous fruitful and interesting discussions.

A sampling of the fellow researchers who inspired me along the way: Ariel Felner, James Neufeld, Rick Valenzano, Thomas Degris, Nolan Bard, Ramon Lawrence, David Thue, Marc Bellemare, Marc Lanctot, Joel Veness, Nelson Amaral, David Wingate, and Mike Barley.

My ever-supportive friends: Tom Williams, Senthil Ponnusamy, Ryan Zaari, Kenya Kondo, Suneth Leelananda, and Jason Treit.

And to my family – my parents, for their constant support, and my wife, Julia, for her incredible warmth, patience, and unwavering encouragement.

Table of Contents

Chapter 1

Introduction

On a break from work, the mathematician Timothy J. Pennings observed dozens of trials of his dog Elvis fetching a ball from a lake. Elvis could run quickly, but only swim slowly; when chasing the ball, Elvis had to make a difficult tradeoff between the two, and decide where to cross the beach line. But over the course of several of these trials, Pennings noticed that Elvis was choosing paths that "agreed remarkably closely with the optimal path" [46], and with minimal deliberation, suggesting Elvis had *good judgment* about how to efficiently fetch from the lake.

At the highest level, this thesis is about constructing computational analogs to "good judgment" called *heuristics*. These heuristics can help computer problem solvers, called *agents*, solve arbitrary *state-space search* problems in rapid succession. Throughout, the studies into how to build these data structures will be grounded in the methodology of optimization. This introductory chapter presents these concepts in summary detail, and concludes with an overview of the research contributions that will be described in subsequent chapters.

1.1 State-Space Search

In *state-space search*, a problem solver (i.e., agent) is tasked with generating low or minimal cost sequences of *actions*, through a space of connected and discrete *states*, that deliver the agent from its start state to a given goal state. In our earlier example, the actions could involve the movement of Elvis between adjacent locations along the beach and water, and the costs could simply be the time Elvis needs to per-

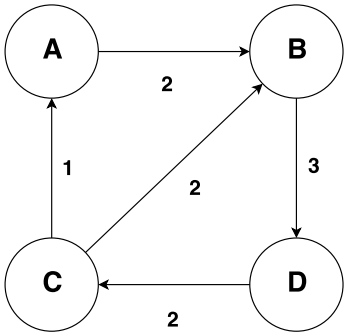


Figure 1.1: A simple example of a search graph, comprised of four states and five directed edges, labelled with various associated costs.

form each action (which might be greater when moving between water states than when moving between states on the beach). But more practical examples of the applications of state-space search include planning paths or other strategic action sequences in video games, GPS routing for road networks on portable, low-power devices with goals of minimizing time and/or fuel consumption, solving combinatorial puzzles (e.g., the Rubik's Cube or the Sliding Tile Puzzles), and designing carefully constrained international flight itineraries across multiple airlines.

1.1.1 Formalism

Formally, a state-space search problem is defined as $P = (G, \delta, s, g)$. The parameter $G = (V, E)$ is a directed or undirected finite graph with a set of vertices, V (states), and a set of edges, $E \subseteq V \times V$ (actions) with associated costs, δ (an illustration of an example search graph is shown in Figure 1.1).

Each *state* $i \in V$ in the graph can have significant representative power. A state might denote a geographic location such as a city or set of coordinates, a specific arrangement of pieces in a puzzle, or a robot's joint angles; it might also include vital side information like time (modulo some schedule), remaining fuel, or the positions of other entities that are beyond the agent's direct control. While states are discrete, fine-grained ranges of continuous values can be represented.

Each *action* $(i, j) \in E$ links one state to another with some associated *cost* (or weight or distance) given by $\delta(i, j)$. This cost might represent time, effort, fuel consumption, etc. In general, we can define $\delta : V \times V \rightarrow \mathbb{R}^+$ as a mapping

from *any* pair of states (not necessarily neighboring states) to the total cost of the shortest (i.e., lowest-cost) path between them.[1] Note (V, δ) therefore defines a finite *quasimetric space* - that is, a finite metric space in which the symmetry axiom is relaxed. It is worth noting that, in practice, δ is rarely given explicitly; rather, only the distances between action-linked states are known *a priori*.

Finally, the states $s \in V$ and $g \in V$ are the agent's start and goal states respectively, and so a feasible (if not cost-minimal) solution to P is a state sequence:

$$\langle p_1, p_2, ..., p_{\ell-1}, p_\ell \rangle, \tag{1.1}$$

originating at the start state ($p_1 = s$), terminating at the goal state ($p_\ell = g$), and comprised of valid actions from E (i.e., for all $1 \leq i < \ell$, $(p_i, p_{i+1}) \in E$).

1.1.2 Solution Criteria

There are many automated methods for finding good solutions to state-space search problems. But what does it mean for a solution to be "good?" There are multiple criteria. *Cost minimization* is the simplest and most intuitive goal. In particular, we say that a solution is *optimal* if it minimizes the sum of costs in the action sequence;[2] and any *suboptimality* in a solution is usually measured as a multiplicative ratio of the optimal solution cost. Nevertheless, several alternative measures do exist.

Another criterion is whether a solution can even be found at all – this is a question of *tractability*. In traditional applications of heuristic search (in particular, in the case of combinatorial puzzles), as well as in planning, simply being able to *solve* a state-space search problem (either to optimality or not) can be considered a significant success. In fact, a traditional indicator of progress in state-space search has been to deliver a new, optimal solution to some previously intractable state-space.

Yet another criterion, related to but not identical to the previous, is that of *solution time*. Indeed, this is the one we are most interested in addressing in this thesis. Now, more than ever, it has become an important application of state-space search to very *quickly* find high-quality paths between arbitrary states in small graphs that

[1] Unless stated otherwise we assume G is connected (or strongly connected in the directed case) and, correspondingly, $\delta(i, j)$ is always defined and finite for all states i and j.

[2] A method guaranteed to find a solution if one exists, and fail otherwise, is called *complete*.

might otherwise be considered trivial. This can be linked to search's proliferation in end-user consumer products, such as computer games [4] (which drive a multi-billion dollar industry) and the road networks [23], stored on point-to-point GPS applications which are now ubiquitous. While the problems on these graphs are trivial to solve, the difference between 1ms and 0.1ms to solve is significant because it influences battery life and the end-user experience, with empirical evidence suggesting "that reading the data dominates the running time" [25].

This emerging context presents us with the unique opportunity to *train* an agent on its search graph, (i.e., to develop its "intuition") during what might be called an *offline* pre-deployment phase, a period during which there is access to far greater computational resources than there is during the *online* post-deployment phase. From a practical standpoint, the cost of this training is more or less unimportant, as long as it remains tractable, because end-users never experience it. More specifically, we will explore using this training time to *optimize* the heuristics.

1.2 Heuristics

All sound techniques for solving state-space search problems maintain a list of states generated but not yet explored, (i.e., an "open list") but the agent might also have access to side information about which states on this list are more promising than the others. For instance, A* [28] and IDA* [33], which are among the most popular state-space search algorithms, iteratively expand states from a start node, generating and pruning their candidate paths under the direct influence of the values given by a *heuristic*, a fundamental way of coding this side information.

1.2.1 The Heuristic Function

The heuristic function is a subject of significant study, and many novel variations on the concept have been developed. The traditional and most common definition of the heuristic is as a scalar function that estimates the cost-to-go from any state to the goal. A more general definition defines the heuristic as a function of *any two* states, $i \in V$ and $j \in V$. That is, $h : V \times V \rightarrow \mathbb{R}^+$ is a function that gives an estimate of the cost of traversal between arbitrary states $s \in V$ and $g \in V$.

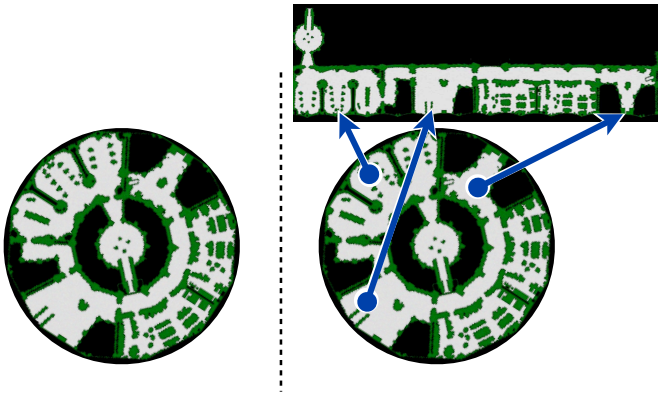
1.2.2 Defining Good Heuristics

Designing a heuristic function is just a proxy for a search algorithm's state selection (or generator) rule, but research has focused more on heuristic construction than on directly learning a better state selection rule. Why is this? The answer mostly lies in the heuristic function's solid theoretical backing: asserting modest properties on a heuristic give general guarantees before search even begins.

In particular, a heuristic is *admissible* when it underestimates true shortest path distances; when this property holds, an admissible search algorithm like A* will always return optimal (i.e., cost-minimal) solutions through the state space. A heuristic is *consistent* if it obeys a form of the triangle inequality; consistent heuristics can keep an A* agent from evaluating any state more than once [44], thereby evading A*'s worst-case exponential time cost [39].

Generally, the more accurate the heuristic is, the better. Accurate heuristics can significantly improve search efficiency by enabling a search algorithm to drastically reduce the number of nodes that need to be examined during search. This also has the effect of keeping the open list (i.e., the subset of the state space comprising the solution frontier) small, which can have a significant impact in practice – especially in performance-critical, real-time applications. Even algorithms that trade path optimality for speed, like weighted A* [48] and other real-time heuristic search variants [8, 9], tend to produce better solutions when a good heuristic is available. The construction of better heuristics remains an important field of ongoing research.

But in an effort to have the heuristic sustain the properties of admissibility and consistency, much of the research into building heuristics has involved what has been described as the human-driven "discovery" of a heuristic [45], which requires careful human analysis on a per-problem basis. Nevertheless, research spanning the past two decades shows a progression toward *automating* the process of constructing heuristics, and the work in this thesis continues this new line of inquiry.



(a) (b)

Figure 1.2: A video game map from BioWare's *Dragon Age: Origins* (a) is shown to be topologically closer to a long hallway, or open plane (b).

1.3 Research Theme

This thesis studies new, optimization-driven representations of the *heuristic function* with the goal of improving search performance. The specific representation used to store the heuristic in this work is a cloud of points in (quasi-metric) spaces of varying dimension. One can imagine moving these points around in different ways, producing a different heuristic function with each manipulation. But what is the best way to arrange these points? Should they be free to move around in high-dimensional spaces, or constrained to a lower-dimensional space? This problem can be interpreted in a principled fashion as a constrained optimization problem.

1.3.1 An Intuitive Guide

An intuitive guide to this approach follows. Many search domains, such as video games and road networks, specify an underlying geometry (or *embedding*) that can be used to define a default heuristic. An example is the straight-line Euclidean distance between the geographic coordinates in a road network or the cells in a grid. Unfortunately, this underlying geometry – while typically being of small enough size to keep in memory – can actually poorly represent the true distances that must be traversed to move between two points. This point can be illustrated by looking at a map of one floor of a tower from the video game *Dragon Age: Origins* in

Figure 1.2.

The original configuration shown in Figure 1.2 (a) has a wall at the top that prevents the walkable (white) portions of the map from forming a complete loop. The default heuristic given by the underlying or "default" geometry is therefore very poor, as two points at the top of the map that look close are actually far apart. Figure 1.2 (b) demonstrates what would happen if the map were *somehow* "unrolled", while still occupying two dimensions. The unrolled configuration contains the same rooms and obstacles as the original, but rearranged in such a way that the heuristic given by taking straight-line distances is far more accurate. Given this new layout, search queries to this domain could be made faster and more efficient.

One way to interpret this work is as an investigation into how a search graph can be (re-)arranged geometrically, in one or many dimensions, so that the underlying distances are more accurate – while still maintaining admissibility and consistency.

1.3.2 Thesis Outline

A high-level outline of the chapters to come is as follows: First, **Chapter 2** will launch our study by presenting a summary overview of some different *existing* methods for heuristic construction. Highlighting the benefits and drawbacks of these methods will clarify some of the distinguishing and novel properties of the optimization-driven approaches we consider in the chapters that follow.

Chapter 3 will break ground on the use of optimization to generate geometric heuristics, and will detail a method for constructing good heuristics out of points in a multidimensional metric (Euclidean) space. A careful choice of weighted loss will enable us to rewrite this optimization as a semidefinite program, which will facilitate a tractable optimization. It will also reveal an exact correspondence to a recently proposed method for manifold learning in the field of dimensionality reduction [60]. This chapter is based on work published in 2011 [52].

Chapter 4 analyzes a special form of the optimization described in Chapter 3 by carefully parametrizing the weights on the loss. This will reveal that the semidefinite program *generalizes* an existing method for automatically constructing search heuristics [56], and moreover can be used to improve that approach. This chapter is

based on a key result in the aforementioned 2011 publication [52].

Motivated to address the computational expense associated with solving the semidefinite programs of the preceding chapters, and inspired by the successes achieved when building *low* dimensional heuristics, **Chapter 5** turns to building heuristics directly on the line using an iterative, convergent sequence of linear programs. **Chapter 6** follows up on this effort by using the a similar optimization approach to drive the construction of a new kind of *directed* heuristic for domains in which the costs between states are asymmetric. This heuristic can be flexibly parametrized to address different kinds of directed domains – the first of its kind.

Having introduced a number of novel techniques for building search heuristics, **Chapter 7** will then turn to the problem of selecting a best subset among them. This study relies on the theory of submodular function maximization, which will provide strong approximation guarantees on the resulting heuristic subsets. This chapter is based on work published in 2013 [53].

Chapter 2

Background

Before presenting several new ways to perform heuristic construction, we summarize some prominent existing methods – both fundamental and the state of the art. The primarily focus is on point-to-point pathfinding settings in which multiple problem instances need to be solved in rapid succession *online*, but where substantial flexibility is given in the time required to precompute data structures *offline*. But several relevant methods that are not specifically designed for this purpose will be included to paint a more complete picture.

This survey is divided into two distinct categories, which are called the *memory-based* and the *statistical* approaches respectively. Beyond giving a background summary, this survey will conclude by drawing the reader's attention to a gap existing between these two categories, one which this thesis aims to bridge.

2.1 Memory-Based Heuristic Construction

Some of the most popular approaches to heuristic construction rely on the careful and complete analysis of an explicitly given, or at least tractably enumerable, search graph (or its abstraction).[1] The information resulting from such an analysis is stored and possibly compressed into some kind of fixed data structure that can be efficiently queried from memory at runtime, hence "memory-based".

In what follows, key instances of memory-based approaches from the literature are examined. These are tabulated in Table 2.1 for comparison: the rows of

[1] The work in this thesis is no exception!

Table 2.1: Instances of memory-based heuristics (here, n is the number of nodes in a given search graph, m is the number of edges, and k a flexible parameter).

Name	Offline	Space	Online	Admiss.	Any goal	References
APSP	$O(n^3)$	$O(n^2)$	$O(1)$	yes	yes	[19, 59]
PDBs	$O(kn)$	$O(k)$	$O(1)$	yes	no	[13]
CH-k	$O(k^2+kn)$	$O(k^2+n)$	$O(1)$	yes	yes	[56]
DH-k	$O(kn)$	$O(kn)$	$O(k)$	yes	yes	[24, 41, 56]
PH-k	$O(k^3)$	$O(k^2)$	$O(k)$	yes	yes	[26]
ADO	$O(kmn^{\frac{1}{k}})$	$O(kn^{1+\frac{1}{k}})$	$O(k)$	no*	yes	[57]

* Inadmissibility is bounded by a multiple of $2k+1$.

this table index different techniques, and the columns enumerate different algorithmic properties: *offline* indicates the time complexity of running the algorithm (e.g., computing the data structure) before deployment, *space* indicates the space complexity of the resulting data structure, and *online* indicates the per-query time complexity of using the method during deployment. Meanwhile, *admiss.* refers to whether or not the algorithm produces admissible heuristics, and *any goal* indicates whether the method can supply heuristics for any goal.

2.1.1 All Pairs Shortest Paths

The facile memory-based approach is to precompute the $n \times n$ table of shortest paths between all pairs of n states. Such a table can be obtained by dynamic programming (e.g., by using what has come to be referred to as the Floyd-Warshall algorithm [19, 59]) in $O(n^3)$ time. This approach is commonly called *all-pairs shortest paths* (APSP), and the resulting distance information defines a perfect heuristic.

If we are given a perfect heuristic, an admissible search algorithm (e.g., A* [28], IDA* [33], Fringe Search [5]) can be used to give perfect search performance, in the sense of having minimal node expansions (in particular, only states lying on an optimal path will be expanded) and, potentially, much lower execution time.

APSP's obvious drawback is the amount of memory it requires, which in general is quadratic in the number of points. And while storing *all* of the APSP matrix is not necessary for undirected graphs due to symmetry, but this does nothing to alleviate its large asymptotic memory requirements. This is problematic even when dealing

with relatively small search graphs, since many of the real-world applications of heuristic search (such as video games or GPS route planners) contain other high priority subsystems that compete for main memory. Moreover, empirical evidence suggests reading from memory can dominate search time on mobile devices [25], suggesting large structures like an APSP table may not be ideal, even in those rare cases that it is feasible. Essentially, every byte of memory that can be saved is important, and APSP leaves substantial room for improvement in this regard.

2.1.2 Pattern Databases

A common approach for reducing the memory required to store a memory-based heuristic is abstraction. A popular and effective method following this design, which is also considered fundamental to the heuristic search literature, is that of *pattern databases* (PDBs) [13]. Rather than store distances between individual states, PDBs store distances measured over k abstracted states, and use these k values as the basis for admissible heuristics to a single goal state. This approach is powerful enough to render intractable search problems tractable, as demonstrated on combinatorial problems [13, 21], and can serve as features in some of the statistical models [1, 47] that will be described later in Section 2.2.

As noted, pattern databases are typically designed with only a single goal state in mind. This restriction is natural when looking at the kinds of problems to which pattern databases typically apply (combinatorial puzzles like the Rubik's Cube or the Sliding Tile Puzzle – often thought of as having a single "solved" state). While it is feasible that a pattern database could be generated for every possible goal state, this would lead to asymptotic memory requirements of the same order as APSP. Although it has been observed that many small PDBs can be more effective together than a single monolithic PDB [31], how to best choose among them appears to lack a clear optimization interpretation in the literature.

2.1.3 Canonical Heuristics

When memory limits are coupled with the possibility of having to determine a path to an *arbitrary* goal state (as with online trip planners, road networks, mobile GPS

on low-power devices, and video games), heuristic functions tend to be examples of *true-distance heuristics*, which work by precomputing and storing a small fraction of the entries in the APSP matrix. By appealing to simple geometric arguments, heuristics between any pair of states can then be computed, on-the-fly, using just these stored values. Precisely which pairs of states and which geometric arguments are used differ with the technique, but all "true-distance" methods share a common core of somehow storing a subset of the "true" distance information.

One true-distance technique that resembles the aforementioned PDBs is that of *canonical heuristics* (CHs) [56], since its selection of canonical states resembles a sort of abstraction. In particular, every search state i in the graph is assigned to a nearby representative (or "canonical") state, c_i, and all that is stored is (*i*) the APSP distances between the $k \ll n$ canonical states and (*ii*) the n distances between each state and its nearest canonical neighbor. Admissible and consistent heuristics can then be determined between two states i and j by subtracting the sum of the distances to their respective canonical states, $\delta(i, c_i)$ and $\delta(j, c_j)$, from the true distance between their canonical states, $\delta(c_i, c_j)$, i.e.:

$$h(i, j) = |\delta(c_i, c_j) - \delta(i, c_i) - \delta(j, c_j)| \leq \delta(i, j) \tag{2.1}$$

Online, these heuristic lookups can be done in constant time. Offline, the placement of the canonical states is typically done to maximize some semblance of "coverage", though once again appears to lack an optimization interpretation in the literature. Note that multiple, complementary sets of CHs can be generated and then maximized over; naturally this would come with greater memory requirements.

2.1.4 Differential Heuristics

Sturtevant *et al.* (2009) observed that canonical heuristics are in fact a general case of differential heuristics (DHs) [56], the likes of which have made numerous independently-driven appearances in the research literature [24, 37, 41].[2] Empirically, differential heuristics have been observed performing better than their canonical cousins under identical memory constraints [56].

[2]These are extremely closely related to ALT heuristics [24] (*A* with Landmarks and the Triangle inequality*) and are a special case of metric embedding theory's *Lipschitz embeddings*. [37].

A single DH is built by designating a pivot state p, from which the true path lengths to all other states are cached. In an undirected graph, these path lengths necessarily come from a finite metric space, and so the triangle inequality can be used to give a lower bound on the distances between any two states i and j:

$$h(i,j) = |\,\delta(i,p) - \delta(j,p)| \leq \delta(i,j) \tag{2.2}$$

Another way to view the construction of a DH is to think of the search graph being "folded" over the pivot. The resulting distances between points gives the same heuristic values. (For more detail on this interpretation, see Chapter 4.)

DHs are cheaply computable, high performing in practice, and can be deployed in sets of k, implying a heuristic lookup that maximizes over all k available heuristics (in $O(k)$ time). Each DH corresonds to a row (or column) in the APSP distance matrix; if $k = n$, the DH heuristics together comprise the full APSP matrix, facing similar drawbacks to that approach for building heuristics. The best way to arrange $k < n$ pivots is an open optimization problem, as noted by Sturtevant et al. [56]:

> "More insights are also needed into the nature of these heuristics to give guidance to an application developer as to which heuristic (and its parameters) to choose for a given problem".

A number of practically effective approaches have been proposed [25] to provide such insight, but these approaches tend to lack a solid optimization interpretation. This is a problem that we explore later in this thesis.[3]

2.1.5 Portal Heuristics

Another memory-based approach is that of portal heuristics (PHs). These resemble canonical heuristics, but involve a domain-specific partitioning of the state space into disjoint regions separated by transition bottlenecks [26]. The k states on the borders between these regions are designated "portals", and the true distances between all pairs of portal states are stored in memory. The idea is that if an agent must pass between regions, it will necessarily cross a portal state, so an admissible heuristic can be made from the precomputed distances between portals.

[3] For more detail on this, the reader is directed to Chapter 7.

Portal Heuristics can do exceptionally well in domains that have significant bottlenecks, but can do poorly, e.g., compared to DHs, when this is not the case. In borderline cases, it might be important to know whether the competing set of DHs is at or near-optimal, but without an optimization interpretation this is not possible.

2.1.6 Approximate Distance Oracles

The effective use of a mere subset of the true distances is also present in Thorup & Zwick's Approximate Distance Oracle (ADO) framework [57]. The first step in building an ADO is to define a random hierarchical nesting of depth k over the nodes in the graph. Then the distance between each node and the nearest node on each nested level is stored in a hash table (accessible in time linear in k). Distance queries between states are then answered by searching down the nested levels for a node to which both states are closest. Once this node is found, its distances to the query states are summed to give a bounded distance estimate.

It is worth emphasizing that Thorup & Zwick do not consider these values to be search heuristics per se, and they cannot be guaranteed admissible. In practice, ADOs tend to perform poorly when used as a search heuristic as compared to alternatives (for instance, differential heuristics). But the technique is included here, before we move on to statistical approaches to heuristic construction, to illustrate the asymptotic possibilities of memory-based heuristics, and point to the potential for formalzed approaches to be useful in generating and analyzing heuristics.

2.2 Statistical Approaches to Heuristic Construction

A radically different approach to heuristic construction is what we term the *statistical* approach.[4] Rather than treat the entire search graph as a cohesive unit of analysis (note this may not be possible if the graph is very large), these approaches involve using training data from solved problem instances to infer some kind of a function with which to describe the stochastic link between the states' description variables ("features") and their heuristic values to a fixed goal state.

[4]Note: what we are calling the "statistical approach" in this chapter has also been referred to as the "subsymbolic" approach to heuristics elsewhere in the literature [18].

Table 2.2: Statistical models of the heuristic function.

Name	Model	Admiss.	Training Method	Reference
ALP	Linear	yes[a]	Linear Programming	[47]
Neural Heuristics	ANN	no	Neural Backprop.	[18]
Bootstrap + Neural	ANN	no	Bootstrap Learning	[1]
MMP	Linear[b]	no[c]	Quadratic Programming	[51]

[a] This approach is only admissible when it has been exposed to all states.
[b] It is possible to use the kernel trick to build nonlinear models for MMP.
[c] Although not a heuristic per se but a bias term (see text).

To provide a summary overview, the properties of each method described in this section are shown in Table 2.2: the rows of this table index techniques, and the columns list algorithm properties. The *model* heading indicates whether a linear or nonlinear relationship between features and the heuristic is modelled, *admiss.* refers to whether or not the algorithm produces admissible heuristics, and *training method* indicates the specific procedure used to train the underlying model.

2.2.1 Approximate Linear Programming

Even simple linear models can give effective heuristics given an adequate set of features to describe each state. One such method is Petrik & Zilberstein's Approximate Linear Programming (ALP) [47]. ALP capitalizes on the basic connection between learning a value function for an MDP and learning a heuristic to a pre-specified goal state. Heuristics are determined by linearly combining the vector of a given state i's features, x_i, with a learned weight vector w:

$$h(i, \text{goal}) = w^\top x_i \approx \delta(i, \text{goal}) \tag{2.3}$$

Each state's feature summary x can include domain-specific properties, including traditional heuristics values as well as abstract state indicators. To determine the vector w, ALP extends earlier work [49] on formulating the solution to a Markov decision problem (MDP) as a convex and efficiently[5] optimizable linear program.

Clearly, the engineering of good features plays an important role and, in this case, both optimal and suboptimal solutions can be given as training data (with the

[5]To be solved "efficiently" in this case means solvable in asymptotically polynomial-time.

former being used to help bias the model toward admissibility). Petrik & Zilberstein provide approximation bounds on the largest possible heuristic error and, unusually for a statistical method, show that it can also produce admissible and consistent search heuristics, albeit under the restrictive condition that it has seen every state's cost to the goal state before learning the model.

2.2.2 Neural Backpropagation

More flexible, nonlinear models of the heuristic have also proven effective in rendering heuristic search more efficient. However, to be useful in reducing the computational effort required to complete a search, determining the model output needs to be inexpensive. This can preclude some nonlinear approaches, like kernel methods, due to the cost of recombining the training data.[6] Perhaps as a result, research so far has mostly focused on neural networks, which can be difficult to train but quick to answer queries. One such approach by Ernandes & Gori is to use a multilayer articial neural network (ANN), equipped with a loss function skewed to bias the learner toward admissibility [18]. This yields so-called "Neural Heuristics".

The typical optimization approach is neural backpropagation, where, for a given training example, the network's output is propagated backward through the network to determine gradients along which to modify the network's weights. Once again, good feature engineering plays an important role, as does the user's ability to have enough data to train the network with so that it can generate good, general search heuristics. However, the objective space of most complex neural networks are typically and infamously nonconvex; moreover, determining a good topology for the network can also be extremely challenging. Nevertheless, careful training methods (of which there are many) can often help to reduce the effects of local minima.

2.2.3 Bootstrap Learning

One obstacle to training the preceding statistical heuristics, at least on large problems, is the data acquisition problem. These models require training data containing numerous examples of (possibly large) solved problem instances. But such

[6] See, for example, De Bie et al. [14] for an accessible overview detailing kernel methods.

instances may not be available *a priori*: a powerful heuristic may in fact be needed to generate such examples. This leads to some circular argumentation: if one can obtain these examples, why is there a need to learn such a heuristic in the first place?

Toward addressing this data acquisition problem, recent work by Arfaee et al. shows that even a weak starting heuristic can be parlayed into a more formidable one through a novel training procedure [1]. This "bootstrap" learning of heuristic functions was successfully used to train Ernandes & Gori's aforementioned Neural Heuristics into being able to solve nontrivial problem instances. But this method is general enough to be applied to other statistical models of the heuristic function, and it is a distinct possibility that this approach will see wider adoption in the future.

2.2.4 Large Margin Prediction

Another linear method applicable to efficient heuristic search is Ratliff et al.'s Maximum Margin Planning (MMP) [51]. Unlike the preceding approaches, MMP is uniquely motivated by the problem of *imitative* pathfinding: the search algorithm's goal is to create a path that is most similar to a path or set of paths given by a domain expert, in terms of the features of the states visited (for instance, risk-avoidance behavior). A naive, ad hoc approach to solving such problems would be to generate many (optimal and suboptimal) paths to the goal state, and to select among them the one that most resembles the expert path(s). The more efficient solution used by MMP is to learn bias terms that can be added to the heuristic to coax the solver into reproducing the expert-demonstrated behavior. Note that while the bias term is not a heuristic heuristic per se, and can lead to a loss of admissibility and consistency, we include this example as another instance in which the optimization of a linear model can help make search more efficient.

The specific optimization approach MMP uses falls into a category of (structured) large-margin prediction methods. It uses the same loss framework as support vector machines [12], a popular and highly effective approach to classification. The objective space is correspondingly convex, and therefore globally optimal solutions are efficiently obtainable; moreover, its quadratic programming solution is rendered unnecessary by an efficient subgradient descent.

2.3 Summary

Common among the "memory-based" approaches is their use of memory to store distance information between a subset (or abstraction) of the state-space, their treatment of the entire search graph as the fundamental unit of analysis, and their use of geometric arguments to derive distance estimates between *any* pair of states. Unfortunately, these methods tend to lack a true optimization interpretation. Asymptotic bounds on what might be achieved by using memory [57] and ad-hoc selection strategies [25] do exist in certain cases, but in general this means that there is no quantifiable way (in terms of a formalized objective) to describe what the resulting heuristics are striving to accomplish. The lack of an optimization interpretation can also make it difficult to describe their failure modes. That is, if a heuristic is failing, it is beneficial to know whether this is because the approach is flawed, or because there is room for improvement with respect to some objective function.

Meanwhile, the "statistical" approaches share a common core of optimization: each has a clearly defined scalar objective. Unfortunately, this optimization is *not* performed on any explicit representation of the search graph, and relies heavily on the existence of good training data. Because the models can be built from just a sample of solved problem instances, all of these approaches have the advantage that they require that only a "relevant" subgraph be generated before a heuristic can be learned.[7] But this is also a weakness: it is unclear what defines such a relevant subgraph, and without good training data, the resulting model cannot be expected to be very good (although this is somewhat ameliorated by the bootstrapping approach described in Section 2.2.3). Moreover, with only *partial* knowledge of the search graph, there is a trend among these methods to sacrifice the key properties of *admissibility and consistency*. Indeed, admissibility and consistency are infrequently discussed in the supervised learning literature, where these approaches draw from. Some modest guarantees do exist, for example by appealing to "overestimation frequency" in restricted domains [18]; and, despite the absence of such bounds, heuristics of this type have been successful in driving search to affordable solutions

[7] Interestingly, this also happens to also be one of the oft-cited benefits of tree search [45]: that is, that only a relevant subgraph needs to be generated to determine a solution.

on combinatorial puzzles. But they come nowhere near the solid admissibility and consistency guarantees common to the memory-based approaches of Section 2.1.

But there are clear benefits to *both* the memory-based methods and the statistical methods. One question we will explore in this thesis is whether we can find success in building approaches that combine elements of both.

Chapter 3

Euclidean Heuristic Optimization
via Semidefinite Programming

This chapter introduces *Euclidean Heuristics* alongside a tractable, globally solvable optimization problem for finding the best such heuristics via semidefinite programming, and is is based on work published in 2011 [52]. Our approach turns out to correspond to a recently proposed method for dimensionality reduction and, as a result, the ideas described in this chapter have impacted the literature by exposing a key connection between the AI subfields of search and machine learning.

3.1 Motivation

The preceding chapter divided existing heuristic building methods into two categories: the *memory-based* approaches easily achieved admissibility and consistency but lacked optimality criteria, while the *statistical* approaches achieved optimality according to some explicit objective but used implicit search graph representations that rendered admissibility and consistency elusive, to say the least. Motivated by the apparent gap between the two, this chapter investigates a new approach that

Representation/data structure	Points in $\mathbb{R}^d$ ($\mathbf{Y} \in \mathbb{R}^{n \times d}$)
Heuristic lookup	$h(i,j) = \|\mathbf{y}_i - \mathbf{y}_j\|_2$
Optimization method(s)	Semidefinite programming, the SVD
Solution optimality	Global[+]

[+] Followed by dimensionality reduction using PCA.

Table 3.1: Key characteristics of the Euclidean Heuristic framework.

capitalizes on the strengths of both memory-based and statistical approaches.

First, the search graph will be represented as a malleable configuration of points in a Euclidean space. Like many prior memory-based approaches, the distance between points in this space will be used to represent the heuristic values. Unlike prior approaches, this space will also be specifically multidimensional. This explicit representation of the search graph enables the approach to inherit many of the benefits of the memory-based approaches, including easy proofs of admissibility and consistency under simplified assumptions.

Next, we borrow from the statistical approaches and pose the best way to situate these points relative to one-another as an optimization problem. This formalism will reveal an exact correspondence to a recently proposed method for manifold learning in the field of dimensionality reduction [60], yielding empirical results with competitive or even significantly-improved search performance compared to the popular and effective differential heuristics [56]. Later in Chapter 4, we will also see that the proposed technique actually generalizes and can be used to *improve* differential heuristics. construction.

3.2 Euclidean Heuristics

We begin with a formal description of Euclidean Heuristics.

Definition 3.2.1 (Euclidean Heuristic)**.** A *Euclidean Heuristic* is a heuristic function whose values are given as distances between points in a Euclidean space of d dimensions. In particular, each state i in a search graph of size n is represented as a vector $\mathbf{y}_i \in \mathbb{R}^d$, and the heuristic value between two states is

$$h(i,j) = \|\mathbf{y}_i - \mathbf{y}_j\|_2 = \sqrt{\sum_k \left(y_i^{(k)} - y_j^{(k)}\right)^2}, \tag{3.1}$$

where $y_i^{(k)}$ denotes the kth entry in the vector $\mathbf{y}_i$.

For convenience, define $\mathbf{Y}$ as an n by d matrix whose rows are these vectors:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^\top \\ \mathbf{y}_2^\top \\ \vdots \\ \mathbf{y}_n^\top \end{bmatrix} = \begin{bmatrix} y_1^{(1)} & y_1^{(2)} & \cdots & y_1^{(d)} \\ y_2^{(1)} & y_2^{(2)} & \cdots & y_2^{(d)} \\ \vdots & \vdots & \ddots & \vdots \\ y_n^{(1)} & y_n^{(2)} & \cdots & y_n^{(d)} \end{bmatrix} \tag{3.2}$$

$\mathbf{Y}$ therefore determines Euclidean distances between all pairs of states, and so encodes heuristic values between all pairs of states as defined on line 3.1.

The problem then is to determine a (preferably low-dimensional – i.e., where d is small) configuration $\mathbf{Y}$ that best expresses the shortest path distances between each state represented. We will consider an optimization formulation for achieving this, and through a series of algebraic manipulations will show how to write this objective in a way that reveals it to be efficiently solvable.

3.3 Optimization Problem

By manipulating the location of the points in $\mathbf{Y}$, we propose to construct a consistent and admissible Euclidean heuristic as the solution to an optimization problem:

$$\begin{aligned} \underset{\mathbf{Y}}{\textbf{minimize}} \quad & \mathcal{L}(\mathbf{Y}) \\ \textbf{subject to} \quad & \mathbf{Y} \text{ is admissible and consistent.} \end{aligned} \tag{3.3}$$

where $\mathcal{L}$ is a loss function, measuring the error in the heuristic values in $\mathbf{Y}$ compared to the true shortest path distances. This problem can be thought of as looking among *all* admissible and consistent Euclidean heuristics, as encoded by $\mathbf{Y}$, for the one with lowest error under $\mathcal{L}$. Such a $\mathbf{Y}$ is called an *optimal Euclidean heuristic*, and the constraints and objective defining (3.3) will be examined in turn.

3.3.1 Constraints

The constraints on the admissibility and the consistency of the heuristic encoded by $\mathbf{Y}$ are respectively formalized with the following:

$$\forall\, i, j \quad \|\mathbf{y}_i - \mathbf{y}_j\| \leq \delta(i, j) \qquad \textbf{admissibility} \tag{3.4}$$

$$\forall\, i, j, k \quad \|\mathbf{y}_i - \mathbf{y}_j\| \leq \delta(i, k) + \|\mathbf{y}_j - \mathbf{y}_k\| \qquad \textbf{consistency} \tag{3.5}$$

Here, $\delta(i, j)$ is the true shortest path distance between i and j. These constraints apply to all combinations of states in the search space, however we can drastically simplify them by observing that satisfying a subset of them implies satisfying all of them. On the way to showing this, an important supporting lemma is given.[1]

Lemma 3.1 (consistency under the triangle inequality)**.** *Let $G = (V, E)$ be a directed or undirected search graph and let $h : V \times V \to \mathbb{R}$ be any scalar function over pairs of G's vertices that obeys the triangle inequality:*

$$\forall\, i, j, k \quad h(i, j) + h(j, k) \geq h(i, k) \tag{3.6}$$

h gives admissible and consistent heuristics if and only if it is locally admissible:

$$\forall (i, j) \in E\ h(i, j) \leq \delta(i, j) \tag{3.7}$$

Proof. The reverse implication is trivially true as local admissibility conditions are a subset of global admissibility conditions. The forward implication involves repeat application of the triangle inequality. For states i, j, k, let $p_1, p_2, ..., p_\ell$ be states on a shortest path from i to j with $(p_q, p_{q+1}) \in E$:

$$h(i, j) = h(p_1, p_\ell) \leq \sum_{q=1}^{\ell-1} h(p_q, p_{q+1}) \tag{3.8}$$

$$\leq \sum_{q=1}^{\ell-1} \delta(p_q, p_{q+1}) = \delta(i, j) \tag{3.9}$$

Line 3.8 uses (repeated) application of the triangle inequality on h, and line 3.9 applies the local admissibility condition. Therefore local admissibility on h implies global admissibility. Global consistency follows as a consequence of admissibility:

$$h(i, j) \leq h(i, k) + h(j, k) \tag{3.10}$$

$$\leq \delta(i, k) + h(j, k) \tag{3.11}$$

Line 3.10 is again the triangle inequality, and line 3.11 uses the global admissibility condition proved on lines 3.8–3.9, proving the admissibility and consistency of h.

□

[1] A similar result of Passino and Antsaklis [43] applies specifically to metric space heuristics.

Lemma 3.1 is very broadly applicable due to the ubiquity of the triangle inequality, which arises in metric spaces as well as a variety of other spaces that might relax the axiomatic foundations of metrics (i.e., discernibility, symmetry, and non-negativity). Furthermore, with this result in place, we can now return our attention to simplifying the constraints on lines 5.4 and 5.5.

Theorem 3.1. *The Euclidean heuristic* $\mathbf{Y}$ *is admissible and consistent if and only if it is locally admissible (i.e., distances between neighbors are non-overestimating).*

Proof. A Euclidean space is a trivial example of a space satisfying the triangle inequality. It follows immediately from Lemma 3.1 that a Euclidean heuristic $\mathbf{Y}$ is admissible and consistent if and only if it is locally admissible. □

Consequently, when solving the optimization problem on line 3.3, it is only necessary to require constraints on the local admissibility of $\mathbf{Y}$. This requires just one constraint per edge in the search graph, which greatly simplifies the optimization.

3.3.2 Objective

The loss function $\mathcal{L}$ combines the errors between the heuristic distances and the true distances into a single scalar value. It specifies the relative importance of the heuristic between each pair of states, and a trade-off between many small errors versus a single large error. This chapter studies the following loss on $\mathbf{Y}$:

$$\mathcal{L}(\mathbf{Y}) = \sum_{i,j} W_{ij} \left| \delta(i,j)^2 - \|\mathbf{y}_i - \mathbf{y}_j\|^2 \right| \tag{3.12}$$

Each entry W_{ij} in the weight matrix $\mathbf{W}$ specifies the importance of the heuristic between two states. For example, if it is important that the heuristic value between two states be accurate, the corresponding entries in $\mathbf{W}$ can be set to large values.[2] The squaring of terms emphasizes heuristic errors on longer paths over the same magnitude errors on shorter paths – which can often be desirable – and proves to be computationally convenient, since it induces a convex objective function.

[2]Note that any asymmetric entries in $\mathbf{W}$ will simply end up being symmetrized by averaging.

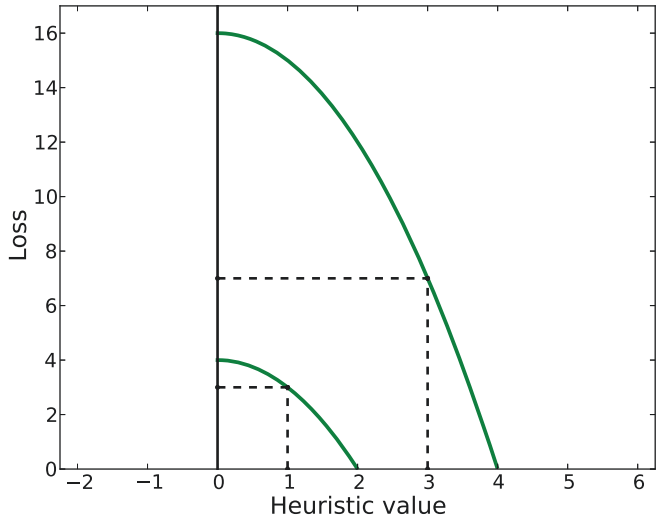


Figure 3.1: Curves showing how the error in a Euclidean heuristic between two states interacts with loss. The heuristic is a Euclidean distance and cannot be less than 0; the admissibility condition guarantees no negative loss can be incurred.

Examples. Some examples of the loss interacting with the errors in the heuristic are sketched in Figure 3.1. If the true distance between a pair of states is 2, then the loss follows the bottom curve; if the true distance is 4, the top curve. Note the differences in magnitude: a heuristic value of 3 when the true distance is 4 will be penalized by a significantly larger amount than a heuristic distance of 1 when the true distance is 2, even though the absolute magnitude of the errors is the same. This phenomenon is indicated by the dotted lines in Figure 3.1.

Simplification. In the context of the optimization problem (3.3), this loss can be greatly simplified. Since $\mathbf{Y}$ must be admissible, the squared true distance must be at least as great as the squared heuristic and it is therefore unnecessary to take the absolute value. This enables the loss to be broken up as follows:

$$\mathcal{L}(\mathbf{Y}) = \sum_{i,j} W_{ij} \left(\delta(i,j)^2 - \|\mathbf{y}_i - \mathbf{y}_j\|^2\right) \tag{3.13}$$

$$= \sum_{i,j} W_{ij}\, \delta(i,j)^2 - \sum_{i,j} W_{ij} \|\mathbf{y}_i - \mathbf{y}_j\|^2 \tag{3.14}$$

Since the first term of Equation 3.14 does not depend on $\mathbf{Y}$, minimizing $\mathcal{L}(\mathbf{Y})$ is equivalent to *maximizing* the weighted sum of squared distances between the points

in $\mathbf{Y}$. Altogether, we arrive at a specific optimization problem:

$$\begin{aligned} \underset{\mathbf{Y}}{\textbf{maximize}} \quad & \sum_{i,j} W_{ij} \left\|\mathbf{y}_i - \mathbf{y}_j\right\|^2 \\ \textbf{subject to} \quad & \forall (i,j) \in E \quad \left\|\mathbf{y}_i - \mathbf{y}_j\right\| \leq \delta(i,j). \end{aligned} \tag{3.15}$$

(Note that the dimensionality d of the Euclidean heuristic has not yet been specified. Such a dimensionality constraint will be imposed in Section 3.4.2.)

3.4 Optimization Approach

The objective function (3.15) is not convex and so, unfortunately, it is not easy to solve for $\mathbf{Y}$ efficiently. However, a change of variables allows the optimization to be rewritten in terms of a kernel matrix, which results in a convex objective. This section goes into the technical detail of how to accomplish this, however the reformulated optimization is identical to the one above, and although necessary to make the problem tractable, it adds little to the understanding of the problem.

3.4.1 Semidefinite Formulation

Let $\mathbf{K} = \mathbf{Y}\mathbf{Y}^\top$ be the matrix of inner products between the vector representations of the states, so that entry $K_{ij} = \mathbf{y}_i \cdot \mathbf{y}_j = \mathbf{y}_i^\top \mathbf{y}_j$. Note the squared distances between points can be directly expressed in terms of the entries of this kernel matrix:

$$\left\|\mathbf{y}_i - \mathbf{y}_j\right\|^2 = (\mathbf{y}_i - \mathbf{y}_j)^\top (\mathbf{y}_i - \mathbf{y}_j) \tag{3.16}$$

$$= \mathbf{y}_i^\top \mathbf{y}_i - 2\mathbf{y}_i^\top \mathbf{y}_j + \mathbf{y}_j^\top \mathbf{y}_j \tag{3.17}$$

$$= K_{ii} - 2K_{ij} + K_{jj} \tag{3.18}$$

Now, because the optimization on line 3.15 only refers to $\mathbf{Y}$ using the distances between points (i.e., $\left\|\mathbf{y}_i - \mathbf{y}_j\right\|$), the objective and constraints can both be rewritten entirely in terms of this kernel matrix $\mathbf{K}$:

$$\begin{aligned} \underset{\mathbf{K}}{\textbf{maximize}} \quad & \sum_{i,j} W_{ij}(K_{ii} - 2K_{ij} + K_{jj}) \\ \textbf{subject to} \quad & \forall (i,j) \in E,\ K_{ii} - 2K_{ij} + K_{jj} \leq \delta(i,j)^2, \\ & \mathbf{K} \succeq 0. \end{aligned} \tag{3.19}$$

Here, since both sides of the inequality in each local admissibility constraint are positive, an equivalent constraint has instead been imposed on the squared distances (heuristic and true). Also note a positive semidefinite constraint $\mathbf{K} \succeq 0$ was added. This importantly ensures that there exists some $\mathbf{Y}$ such that $\mathbf{K} = \mathbf{Y}\mathbf{Y}^\top$, or in other words that $\mathbf{K}$ represents inner products in some Euclidean space.

Lastly, since the optimization as stated is indifferent to translations of the points in $\mathbf{Y}$, a unique solution is forced (up to rotation) by adding a "centering" constraint. This constraint is defined as $\sum_{i,j} W_{ij}K_{ij} = 0$ (or equivalently $\text{Tr}(\mathbf{WK}) = 0$) which also serves to further simplify the objective (3.19) by zeroing the cross term. Letting $\Delta_\mathbf{W}$ be a diagonal matrix with the sums of the rows of $\mathbf{W}$ on the diagonal, we arrive at the following optimization problem:

$$
\begin{array}{lll}
\underset{\mathbf{K}}{\textbf{maximize}} & \text{Tr}(\mathbf{K}\Delta_\mathbf{W}) & (3.20) \\
\textbf{subject to} & \forall (i,j) \in E,\ K_{ii} - 2K_{ij} + K_{jj} \leq \delta(i,j)^2 & \textbf{local admissibility} \\
& \text{Tr}(\mathbf{WK}) = 0 & \textbf{“centering”} \\
& \mathbf{K} \succeq 0 & \textbf{positive semidefinite}
\end{array}
$$

This rewritten optimization involves a linear objective and linear constraints, plus a non-linear, but *convex*, semidefinite constraint. Such semidefinite programs can be solved using a standard toolbox solver such as SDPT3 [58].

3.4.2 Recovering a d-dimensional Embedding

The optimization on line 3.20 produces a kernel matrix $\mathbf{K}$, but we want the Euclidean vectors in $\mathbf{Y}$. This entails kernel principal component analysis, which involves centering the kernel to get $\mathbf{K}' = (I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top)\mathbf{K}(I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top)$ then taking the singular value decomposition (SVD) of $\mathbf{K}'$:

$$\mathbf{Y}\mathbf{Y}^\top = \mathbf{K}' = \mathbf{U}\mathbf{V}\mathbf{U}^\top, \tag{3.21}$$

Note the SVD is a fundamental matrix operation [27] and is available in many numerical computing libraries. Subsequently we can determine $\mathbf{Y}$ as follows:

$$\mathbf{Y}\mathbf{Y}^\top = \mathbf{U}\sqrt{\mathbf{V}}\sqrt{\mathbf{V}}U^\top \tag{3.22}$$

$$= \mathbf{U}\sqrt{\mathbf{V}}\sqrt{\mathbf{V}^\top}\mathbf{U}^\top \tag{3.23}$$

$$= \mathbf{U}\sqrt{\mathbf{V}}(\mathbf{U}\sqrt{\mathbf{V}})^\top \tag{3.24}$$

$$\mathbf{Y} = \mathbf{U}\sqrt{\mathbf{V}} \tag{3.25}$$

Line 3.23 uses the fact that $\mathbf{V}$ is diagonal, implying $\mathbf{V} = \mathbf{V}^\top$. The resulting eigenvectors $\mathbf{U}$, scaled by their real-valued eigenvalues $\sqrt{\mathbf{V}}$, correspond to the columns of $\mathbf{Y}$ (3.25), thereby recovering the underlying Euclidean embedding.

The number of non-zero entries in $\mathbf{V}$ may be prohibitively large (n in the worst case) but the optimization tends to keep this number small, as we will discuss in the next section. Under precise memory constraints, a reasonable way to create a representation with d values per state is to choose the d eigenvectors with the largest associated eigenvalues (the top *principal components*). This is because eigenvectors with small associated eigenvalues have small effects on the resulting heuristic estimates, so less is lost by ignoring them.[3]

3.4.3 Maximum Variance Unfolding

This exact optimization problem is not new. It is a weighted generalization of Maximum Variance Unfolding (MVU) [60], which was originally introduced for nonlinear dimensionality reduction. True to its name, MVU has a nice visual interpretation. Imagine neighboring states in the search graph being connected by rods. A rod can pass through other rods, it can be compressed, or it can grow to a length no greater than the edge cost. Once these rods are all in place, the structure is pulled apart as far as the rods allow. The result is an embedding of points in Euclidean space whose squared pairwise distances are maximized, thus maximizing the variance. The observation that maximizing variance often results in low-dimensional embeddings (i.e., small d) corresponds to its proposed use in dimensionality reduc-

[3]This resulting d-dimensional heuristic is a *linear* projection of the high-dimensional *source* heuristic to a lower-dimensional principal subspace.

tion. As noted, this is a fortunate side-effect for our application. While MVU has been applied in a number of dimensionality reduction applications, such as visualization [60], sensor networks [61], and mapping [7], its connection to identifying admissible and consistent heuristics in search had not been observed previously.

3.5 Analysis

Heuristic admissibility and consistency are vital to the guaranteed solution optimality of many search algorithms, like A* [28], IDA* [33] and their derivatives. Here we prove these hold for the optimal Euclidean heuristics defined according to the complete procedure described in the previous section, including the dimensionality reduction step of Section 3.4.2. We then consider the complexity of the proposed optimization, and cite some recent efforts to scale it.

3.5.1 Admissibility and Consistency

While the optimization is constrained to produce admissible and consistent heuristics, the complete procedure took a subset of d principal components. We show this reduction in dimensionality preserves admissibility and consistency.

Theorem 3.2. *If* $\mathbf{Y}$ *is defined as the first* d *principal components of* $\mathbf{K}$ *then* $\mathbf{Y}$ *is admissible and consistent.*

Proof. According to Lemma 3.1, we only need to show that $\mathbf{Y}$ is locally admissible. Let $\tilde{\mathbf{Y}}$ contain all n principal components of $\mathbf{K}$ – as a result, $\mathbf{K} = \tilde{\mathbf{Y}}\tilde{\mathbf{Y}}^\top$. Moreover, since $\mathbf{Y}$ is the first d principal components, we can represent $\tilde{\mathbf{Y}}$ as $[\mathbf{Y}\ \mathbf{Y}']$ for some $\mathbf{Y}'$ which contains the remaining principal components:

$$h(i,j)^2 = \|\mathbf{y}_i - \mathbf{y}_j\|^2 = (\mathbf{y}_i - \mathbf{y}_j)^\top(\mathbf{y}_i - \mathbf{y}_j) \tag{3.26}$$

$$\leq (\mathbf{y}_i - \mathbf{y}_j)^\top(\mathbf{y}_i - \mathbf{y}_j) + (\mathbf{y}'_i - \mathbf{y}'_j)^\top(\mathbf{y}'_i - \mathbf{y}'_j) \tag{3.27}$$

$$= (\tilde{\mathbf{y}}_i - \tilde{\mathbf{y}}_j)^\top(\tilde{\mathbf{y}}_i - \tilde{\mathbf{y}}_j) \tag{3.28}$$

$$= K_{ii} - 2K_{ij} + K_{jj} \tag{3.29}$$

Line 3.27 holds since the dot-product of real-valued vectors is always non-negative in value. Meanwhile, the constraints on $\mathbf{K}$ on line 3.20 guarantee for $(i,j) \in E$

that $K_{ii} - 2K_{ij} + K_{jj} \leq \delta(i,j)^2$. Thus, for any $(i,j) \in E$ we have that $h(i,j)^2 \leq \delta(i,j)^2$, so $h(i,j) \leq \delta(i,j)$ since both are non-negative. $\square$

3.5.2 Complexity of Optimization

The semidefinite program (3.20) scales linearly with the number of points, but has a time complexity in $O(N^3)$, where N is the number of constraints, and space complexity in $O(n^2)$ to store instantiations of the kernel matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$, where n is the number of states. Recall that in our formulation one constraint is required per edge in the search graph. In practical terms, performing the optimization on a search graph with thousands of states with octile connectivity requires several hours of computation, and therefore limits its broader applicability.

Note however the introduction of Euclidean heuristics has since led to attempts to scale the underlying optimization to larger problems [11], rendering the approach applicable to planning [10] and certain goal-oriented pathfinding problems [38].

3.6 Evaluation

In this evaluation of optimal Euclidean heuristics, we consider three experimental domains exhibiting different dimensionality. The performance metrics are *a)* an average count of the number of nodes the A* search algorithm [28] expands, bucketed by solution length, and *b)* the total CPU runtime required by A* to complete all problems, *including* reconstructing the action sequence. Note that drastically more computation is needed for longer paths than shorter ones (even though longer paths tend to be less common) and so performance on long paths is a key detail here.

3.6.1 Cube World

The *Cube World* is a synthetic domain, generalizing the octile grid-world to three dimensions. A perfect heuristic for this domain is obvious by its construction, but for purposes of demonstration we examine building heuristics from scratch. The specific graph considered is a large cube whose sides measure 20 units each, subdivided into 8,000 unit cubes that can be transitioned between if they share a vertex, as

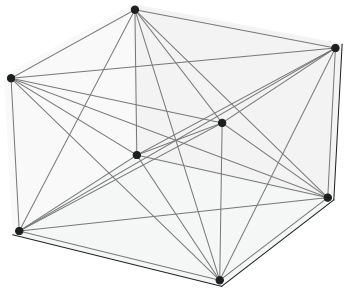


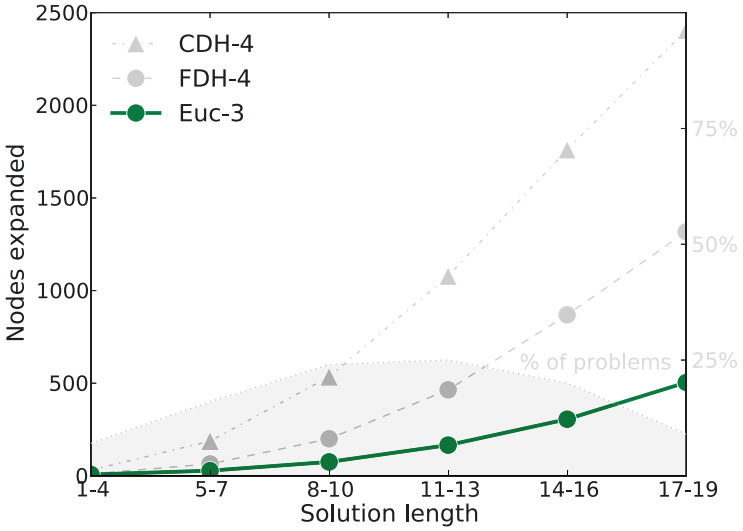
Figure 3.2: Illustration of pairwise connections in the *Cube World*.

illustrated by the subgraph in Figure 3.2. The edge costs are the distances between the cube centers. The embedding from a uniformly weighted optimal Euclidean heuristic is simply the positions of the centers of the unit cubes, occupying three dimensions (**Euc-3**). Meanwhile, two alternative heuristics are considered as baselines for comparison. The first is a set of *differential heuristics* with four pivots in the four corner states on the bottom half of the large cube (**CDH-4**). The second is a set of differential heuristics with four pivots chosen using the *Farthest* selection algorithm [24] to situate the pivots (**FDH-4**) in a complementary fashion: the first pivot is placed in a state farthest from a random seed state, and each subsequent pivot is placed in a state most distant from the previous pivots.

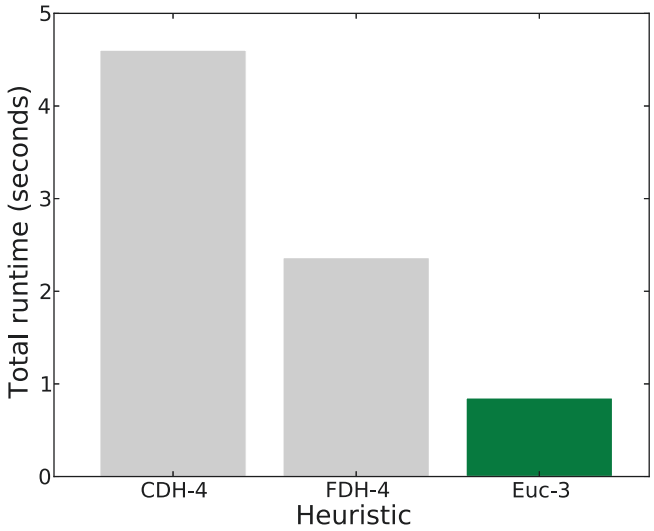
The node expansions and runtime performance of these heuristics are compared together in Figure 3.3, with Figure 3.3a additionally showing the distribution of problems of different lengths. This search space is intrinsically multidimensional, and a heuristic based on the three-dimensional embedding significantly, although not surprisingly, outperforms the heuristics from the combined one-dimensional embeddings (i.e., differential heuristics) – all in fewer dimensions.

3.6.2 Word Search

The *Word Search* domain is a unit-cost state space of four-letter words taken from a computer dictionary. The goal is to change a start word into a goal word by changing one letter at a time. For example, there is an edge between the words ‘corn’ and ‘cord’, but no edge between the words ‘cord’ and ‘worm’, as illustrated in Figure 3.4. The largest connected subgraph of these words is used to define a search graph which has 54,752 edges spanning 4,820 states.



(a) A* average node expansions.



(b) A* total CPU runtimes.

Figure 3.3: *Cube World* results comparing differential heuristics with the pivots placed in the four corners (CDH-4), differential heuristics with the pivots placed using the *Farthest* algorithm (FDH-4), and the optimal Euclidean heuristic in 3 dimensions (Euc-3). Figure 3.3a shows the average node expansions bucketed by solution length, and indicates the distribution of problems of different lengths with the shaded area. Figure 3.3b shows the runtime totals required by each of the heuristics.

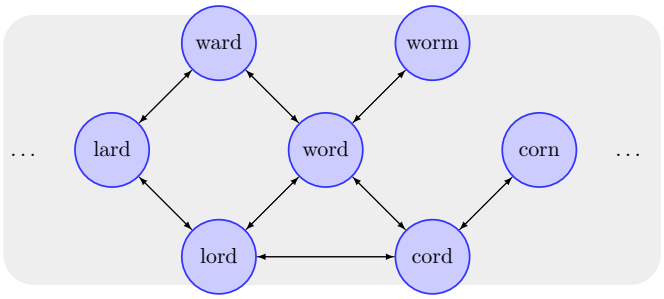


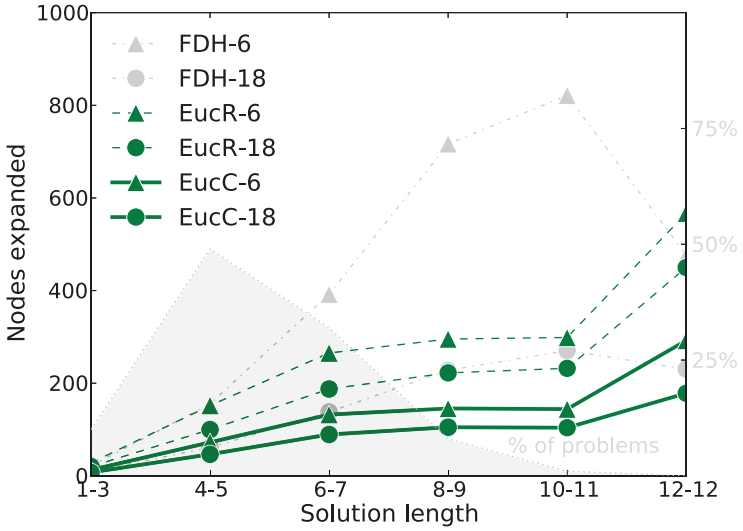
Figure 3.4: Illustrative sketch of a subgraph of the *Word Search* domain.

A Euclidean heuristic is determined under a uniform weight matrix (with the first three dimensions of this embedding shown in Figure 3.6) and we evaluate its 6 and 18 dimensional variations. Note since the transitions are all of unit cost, the *ceiling* of any fractional Euclidean heuristic value can be taken (denoted **EucC-6** and **EucC-18**), which retains admissibility and consistency. To quantify the effect this has, we also evaluate the heuristic while rounding (up *or* down) to the nearest integer, also admissible and consistent (denoted **EucR-6** and **EucR-18**). As a baseline for comparison, the *Farthest* algorithm is used to situate pivots for differential heuristics in sets of 6 and 18 (denoted **FDH-6** and **FDH-18**).

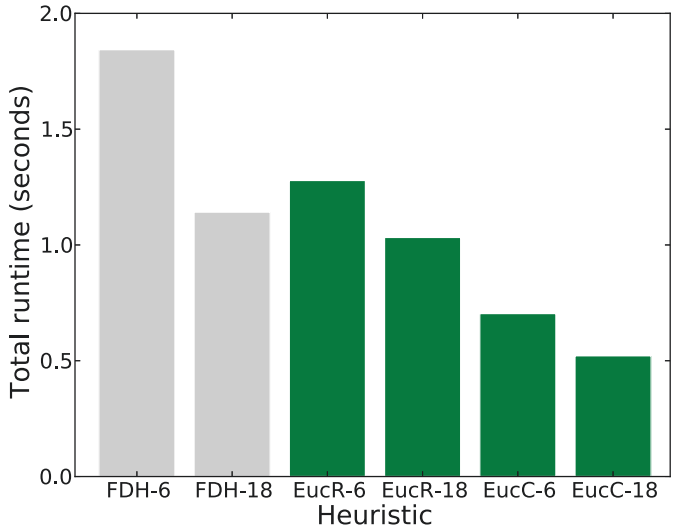
The results in Figure 3.5 show that, across the vast majority of paths, the multidimensional Euclidean heuristic offers a substantial improvement over both combined sets of differential heuristics. Moreover, the *EucC* heuristics show a pronounced advantage over the *EucR* heuristics, both in terms of node expansions and runtime. Differential heuristics do tend to excel on the longest problems, but this is common when using the *Farthest* algorithm. Here, for example, over half of these longest problems (of which there are roughly 180) start or end on the state for the word 'upas', which is often chosen as a pivot point.

3.6.3 Pathfinding

A final evaluation looks at a grid-based map from BioWare's 2009 *Dragon Age: Origins* with 5,129 states. Cardinal transitions between adjacent open cells are defined as having unit cost, while diagonals cost 1.5, but the agent cannot cut corners (move diagonally between cells which are not in a 4-clique).



(a) A* average node expansions.



(b) A* total runtimes.

Figure 3.5: *Word Search* results for A* across 10,000 random problems comparing sets of 6 and 18 differential heuristics placed using the farthest algorithm (FDH) to Euclidean heuristics in 6 and 18 dimensions. Euclidean heuristics are used both with and without the ceiling operator (EucR and EucC respectively). Figure 3.5a shows average node expansions and an indication of the distribution of the problems of each length, and Figure 3.5b shows the total runtimes.

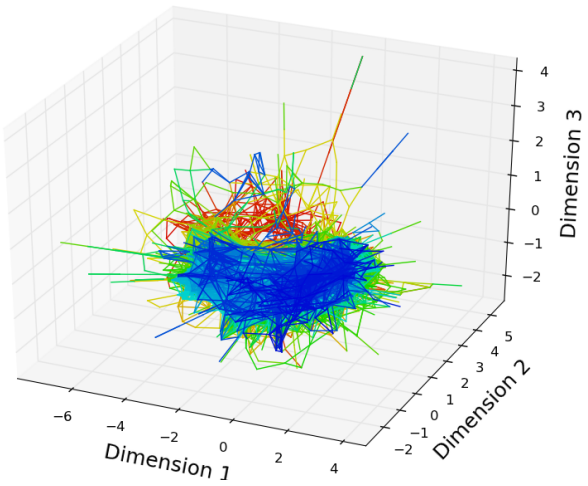
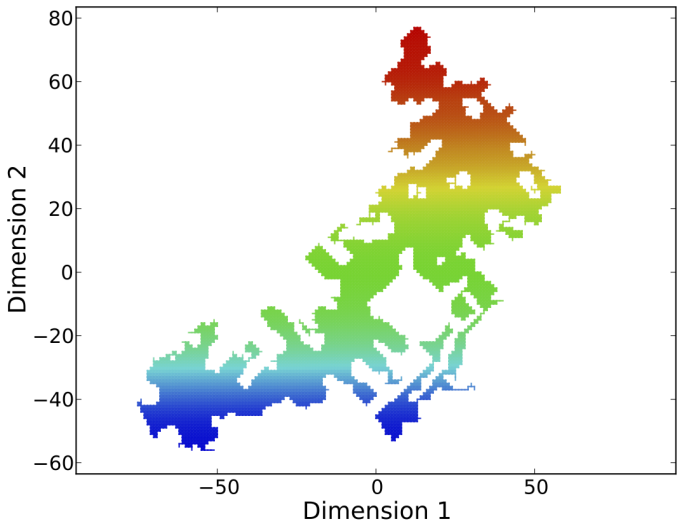


Figure 3.6: Embedding showing the top (in terms of total variance) 3 dimensions of the optimal Euclidean heuristic for the *Word Search* domain. Subsequent dimensions continue to show significant variance.

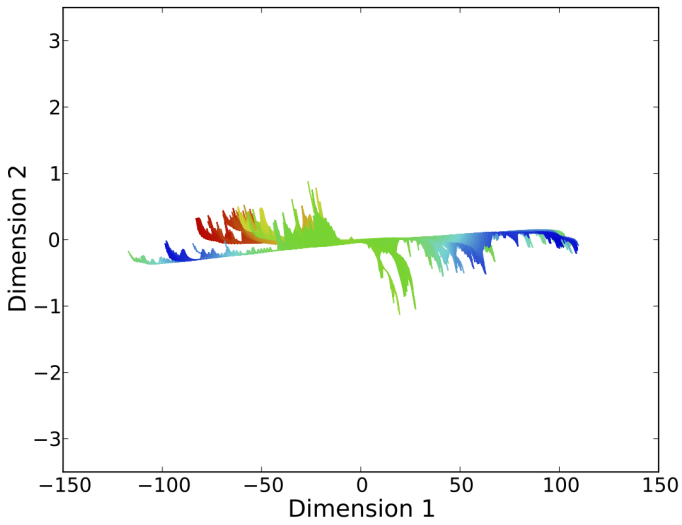
The multidimensional Euclidean heuristic for this domain is determined under a uniform weight matrix yielding an embedding as shown in Figure 3.7b, and is mapped to 1 and 3 dimensions. Since the distance between any two states must be some multiple of 0.5, any fractional Euclidean heuristic can be rounded up to the nearest full half. Competing sets of differential heuristics in sets of 1 and 3 are situated using the *Farthest* algorithm. Note that grid-world domains come equipped with a default *octile* heuristic – the distance between points in the original layout (Figure 3.7a) assuming no obstacles. This heuristic is combined with each of the preceding by taking the maximum, and is also evaluated on its own.

Results on a standard set of 530 benchmark problems are tallied in Figure 3.8. It turns out that ORZ200D, like most *Dragon Age* maps, has low intrinsic dimensionality, featuring one long (albeit curved) central "corridor." As most paths only require good heuristics between the states in this corridor, a good one-dimensional heuristic suffices to capture the most crucial pairwise distances. In this case, the Euclidean heuristic in one dimension (**Euc-1**) achieves significant gains over a single differential heuristic (**FDH-1**), both in terms of node expansions and runtime.

Unfortunately, the Euclidean heuristic in 3 dimensions (**Euc-3**) gives negligible gains over Euc-1; both of these optimal Euclidean heuristics are marginally

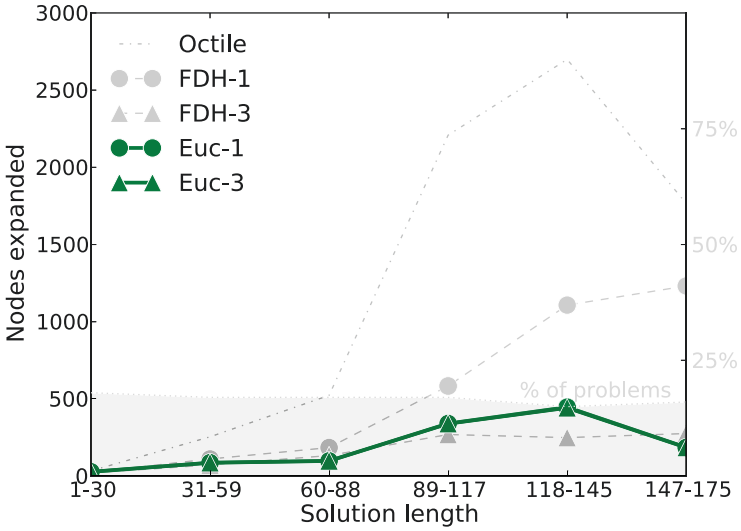


(a) Original graph layout for ORZ200D.

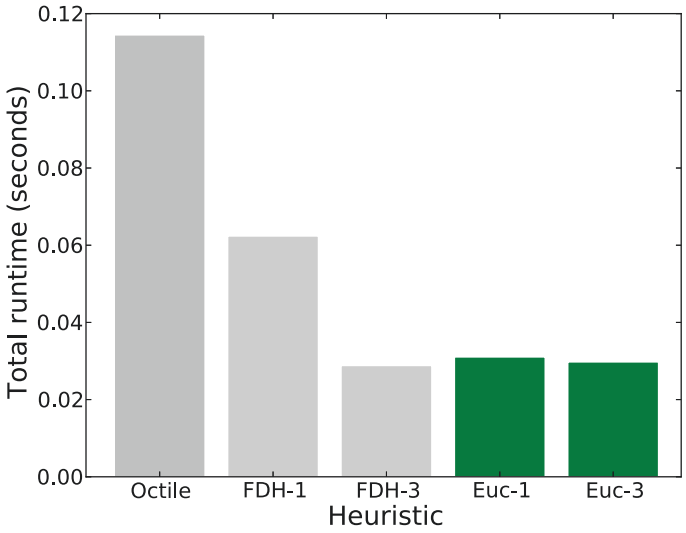


(b) Euclidean heuristic, first two dimensions.

Figure 3.7: *Dragon Age: Origins*' ORZ200D, default (i.e., original) and optimal Euclidean layouts, with colors showing the corresponding regions in each figure. Note the scale of the y axis in Figure 3.7b is magnified to show detail.



(a) A* average node expansions.



(b) A* total runtimes.

Figure 3.8: A* pathfinding results for standard benchmark problems in *Dragon Age*'s ORZ200D. We compare the default (Octile) heuristic to sets of 1 and 3 differential heuristics (FDH) and Euclidean heuristics (DH) in 1 and 3 dimensions. Figure 3.8a shows average node expansions as well as the distribution of the different problem lengths, and Figure 3.8b shows runtime totals for each heuristic.

outperformed by a set of 3 differential heuristics (**FDH-3**). This is because the associated embedding essentially has only one descriptive dimension, as shown in Figure 3.7b, corresponding to the aforementioned corridor. Storage of the less descriptive dimensions is wasteful compared to storing another differential heuristic. Since Euc-3 took hours to compute (as compared to miliseconds for FDH-3), this is an area that invites targeted improvements in the next chapter.

3.7 Summary

This chapter presented a novel approach to constructing admissible and consistent heuristics as the solution to a constrained optimization problem – one that has already been studied in the field of dimensionality reduction. Since the optimization provides information about the fundamental dimensionality of a search space, it shed new light on the nature of heuristic construction. In particular, it was observed that search graphs with higher intrinsic dimensionality (such as 2, 3, or more dimensions) tend to achieve poor coverage with a low number of differential heuristics using a standard placement algorithm, but we found good multidimensional heuristics with our approach. Conversely, graphs with a low fundamental dimensionality are particularly suited to low-dimensional heuristics.

The observed connection between heuristic search and dimensionality reduction appears highly profitable. Agent-specific transition restrictions (spatial, legal, bureaucratic) which can *globally* alter the flow of feasible and optimal paths are accommodated by *local* constraints. Even problems with intuitively challenging topologies – such as a difficult to visualize state space, variable transition costs, or teleportation – pose no added challenge to the design of the semidefinite program we described. Heuristic construction is a natural application for manifold learning techniques and optimization methods, and the ensuing chapters will continue to explore their intersection.

Chapter 4

Enhanced Differential Heuristics
via Semidefinite Programming

This chapter extends the ideas in the preceding chapter, placing the popular and successful approach of differential heuristics (also known as Lipschitz embeddings) under the lens of optimization. We will explore in detail a specific result published in 2011 [52], and in doing so will reveal a new way to view differential heuristics, as well as a preliminary approach for defining an *enhanced* differential heuristic.

4.1 Motivation

The empirical comparisons of Chapter 3 showed Euclidean heuristics can outperform differential heuristics, but those experiments were most successful on inherently multidimensional domains – to which Euclidean heuristics are well suited.

In some domains, a multidimensional Euclidean heuristic will fare worse than a carefully chosen set of differential heuristics. Section 3.6.3 showed just such an instance, where 3 differential heuristics together gave better heuristics than a monolithic 3-dimensional Euclidean heuristic for grid-based pathfinding. In par-

Data structure/representation	Points on the line ($\mathbf{y} \in \mathbb{R}^n$)
Heuristic lookup function	$h(i,j) = \lvert y_i - y_j \rvert$
Optimization method(s)	Semidefinite programming and the SVD
Solution optimality	Global[+]

[+] Followed by dimensionality reduction using PCA.

Table 4.1: Key characteristics of Enhanced Differential Heuristics.

ticular, the Euclidean heuristics stretched and flattened the map along its central corridor – an effect that can be linked to MVU's aggressive dimensionality reduction tendencies [60] – while the remaining 2 dimensions were merely wasted space. The differential heuristics captured distance information that the Euclidean heuristic was forced to sacrifice, thereby achieving better search performance.

This inspires an attempt to somehow combine those two approaches, and so this chapter sets out to "reverse engineer" differential heuristics by reinterpreting them as an optimal configuration of points under a well-defined objective. The analysis yields a counterintuitive result, but facilitates the design of a new *enhanced* differential heuristic that can be built using the semidefinite program of Chapter 3 (c.f. Table 4.1). An empirical evaluation on the standard pathfinding benchmarks of over 50 maps from BioWare's 2009 *Dragon Age: Origins* shows that this new approach to building heuristics can outperform its unoptimized counterpart.

4.2 Differential Heuristics

Recall that a differential heuristic consists of the cached distances $\delta(i, p)$ from all states i in a search space to a pivot state p. An admissible heuristic between any states i and j is the difference between their distances to p:

$$h(i, j) \equiv |\delta(i, p) - \delta(j, p)| \tag{4.1}$$

Differential heuristics are an effective and popular class of memory-based heuristics, and are typically described as an application of the triangle inequality to cached search data, which simplifies proofs of admissibility and consistency. But they can also be viewed as a one-dimensional *graph embedding* $\mathbf{y}$. Let $\mathbf{y}$ be a n-vector with the ith entry y_i defined as $\delta(i, p)$. Then the heuristic lookup (4.1) can be written:

$$h(i, j) \equiv |y_i - y_j| \tag{4.2}$$

i.e., $\mathbf{y}$ is defined as having the pivot point y_p at the origin, and every other state on the same side of the same axis at its true shortest path distance from the pivot. Pursuant to this interpretation, this chapter proves differential heuristics are in fact optimal Euclidean heuristics under a carefully chosen weight matrix $\mathbf{W}_{\text{diff}}$. They are therefore a special case of the optimization framework presented in Chapter 3.

4.3 Optimization Interpretation

Differential heuristic are conventionally built using Dijkstra's algorithm [17] to compute and store the true shortest path distance from the pivot to every other state, or a breadth first search when the transition costs are uniform. But what is the specific optimization problem being solved?

4.3.1 Visual Interpretation

One way to view differential heuristics is as if every point y_i being moved "away" from the pivot until it is at its true distance to the pivot. This is accurate, but incomplete: it must also be true of $\mathbf{Y}$ that all of the points lie on the same straight line (and therefore occupy a single dimension) and moreover that all of the points lie on the same side of the pivot (and as such they are not distributed to either side of the pivot). The *visual* interpretation of this phenomena is that the search graph is being "folded" at the pivot to occupy one dimension (Figure 4.1).

Differential heuristics therefore appear to be the result of a combination of two forces. The first force pushes all of the points as far as possible *away* from the pivot, and the second force draws all of the non-pivot points *toward* each other. Correspondingly, an optimizer trying to emulate a differential heuristic should *maximize* the distances between the pivot and the other states but also somehow – and with less emphasis – *minimize* the pairwise distances between the non-pivot states. The next section will go into formalizing these intuitive notions.

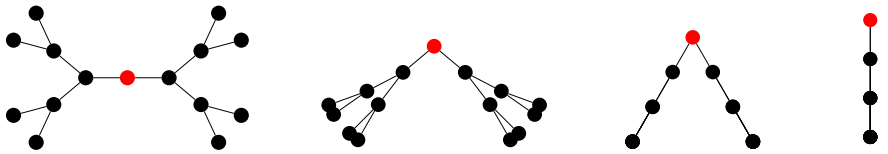


Figure 4.1: A differential heuristic can be thought of as "folding" the search graph onto the line, yielding the rightmost configuration of points on a line.

4.3.2 Objective

Let $\mathbf{Y}$ specify a multidimensional Euclidean heuristic, E the set of edges in a given search graph, and $\mathbf{W}$ a weight matrix. Recall the optimization of Chapter 3:

$$\begin{aligned} \underset{\mathbf{Y}}{\textbf{maximize}} \quad & \sum_{i,j} W_{ij} \left\| \mathbf{y}_i - \mathbf{y}_j \right\|^2 \\ \textbf{subject to} \quad & \forall (i,j) \in E \quad \left\| \mathbf{y}_i - \mathbf{y}_j \right\| \leq \delta(i,j), \end{aligned} \tag{4.3}$$

where $\mathbf{Y}$ is a d by n matrix of real values, as defined in Section 3.2.

The claim is that a differential heuristic will arise when this optimization is parametrized by a special weight matrix $\mathbf{W}_{\text{diff}}$, with zeros on the diagonal, ones on the non-diagonal entries of the pivot's row and column (thereby rewarding the optimizer for maximizing $\|y_i - y_p\|$, where p is the pivot) and *negative* values of $\frac{-1}{n-1}$ elsewhere (thereby rewarding the optimizer for minimizing $\|y_i - y_j\|$ for all $i \neq p, j \neq p$). If index 1 refers to the pivot, then this matrix appears as follows:

$$\mathbf{W}_{\text{diff}} = \begin{bmatrix} 0 & 1 & 1 & 1 & \cdots & 1 \\ 1 & 0 & \frac{-1}{n-1} & \frac{-1}{n-1} & \cdots & \frac{-1}{n-1} \\ 1 & \frac{-1}{n-1} & 0 & \frac{-1}{n-1} & \cdots & \frac{-1}{n-1} \\ 1 & \frac{-1}{n-1} & \frac{-1}{n-1} & 0 & \ddots & \frac{-1}{n-1} \\ \vdots & \vdots & \vdots & \ddots & \ddots & \frac{-1}{n-1} \\ 1 & \frac{-1}{n-1} & \frac{-1}{n-1} & \frac{-1}{n-1} & \frac{-1}{n-1} & 0 \end{bmatrix} \tag{4.4}$$

The next section proves that solving the optimization (4.3) with $\mathbf{W}_{\text{diff}}$ will yield a differential heuristic up to translation and rotation. The reader can also safely advance to Section 4.5, which suggests a simple enhancement to $\mathbf{W}_{\text{diff}}$.

4.4 Equivalence to Differential Heuristics

In this section, we will prove an equivalence between differential heuristics (for a given pivot state) and the optimal Euclidean heuristic of Chapter 3. First, without loss of generality, assume the state index 1 always refers to the pivot state.

Theorem 4.1. *Any solution Y to the optimization problem* (4.3) *with weight matrix* $\mathbf{W}_{diff}$ (4.4) *is collinear (one-dimensional) with the following heuristic values:*

$$h(i,j) = |y_i - y_j| = |\delta(i,1) - \delta(j,1)| \tag{4.5}$$

i.e., Y meets the definition of a differential heuristic (4.1).

The proof of Theorem 4.1 is broken up in the following sections. As a summary overview, the trivial border case of zero-weighted search graphs is first dismissed. Next, if y is not collinear (i.e., "rank 1" or one-dimensional up to rotation) then an alternative embedding can be found that strictly improves the objective. Finally, for any one-dimensional y, unless $y_i = \delta(i, 1)$ the objective can be improved.

4.4.1 Trivial Case

Theorem 4.1 is immediate if $\forall i, j \;\; \delta(i, j) = 0$; here the only feasible solution to the optimization (4.3) is $y = 0$, and the differential heuristic and optimal Euclidean heuristic (which is of rank 0 in this particular case) are the same.

All other cases will assume there is at least one point k where $\forall i, \; \delta(i, k) > 0$.

4.4.2 Collinearity with Pivot at Boundary

Next it will be shown that the optimal embedding y must be collinear with the pivot y_1 on a boundary by proving that any embedding *without* this property can be strictly improved simply by rotating the points about the pivot.

Lemma 4.1. *Any solution y to the optimization problem* (4.3) *with weight matrix $\mathbf{W}_{diff}$ must be collinear with the pivot on a boundary.*

Proof. Without loss of generality, let $y_1 = 0$. For purposes of contradiction, suppose y is *not* collinear or does *not* have the pivot y_1 at a boundary.

Let $y' \in \mathbb{R}^n$ be a one-dimensional embedding where $y'_i = \|y_i\|$. This simple operation can be thought of as the result of rotating the points in y about the origin so they lie on the same side of the same axis, thereby ensuring both that y'_1 is a boundary point and that y' is collinear. For particular states i and j:

$$|y'_i - y'_j| = |\|y_i\| - \|y_j\|| \leq \|y_i - y_j\| \leq \delta(i, j) \tag{4.6}$$

The first inequality is the reverse triangle inequality, and the second is the admissibility condition on y, proving y' is admissible and so a feasible solution to (4.3).

Next, the change in the objective between y' and y is:

$$f(y') - f(y) = \sum_{ij} W_{ij}|y'_i - y'_j|^2 - \sum_{ij} W_{ij}\|y_i - y_j\|^2 \tag{4.7}$$

$$= \sum_{i>1,j>1} W_{ij}\left(|y'_i - y'_j|^2 - \|y_i - y_j\|^2\right) \tag{4.8}$$

$$= \frac{-1}{n-1} \sum_{i>1,j>1} \left(|y'_i - y'_j|^2 - \|y_i - y_j\|^2\right) \tag{4.9}$$

Line 4.8 restricts the indices to reflect that the only distances that change are between non-pivot states ($i > 1$, $j > 1$) and line 4.9 swaps in the associated weights of $\frac{-1}{n-1}$. If y_i and y_j are not collinear with y_1, the reverse triangle inequality is strict:

$$|y'_i - y'_j| = |\|y_i\| - \|y_j\|| < \|y_i - y_j\| \tag{4.10}$$

$$\Rightarrow |y'_i - y'_j|^2 - \|y_i - y_j\|^2 < 0 \tag{4.11}$$

And if y_i and y_j *are* collinear with y_1, but y_1 is not a boundary point:

$$|y'_i - y'_j| = |\|y_i\| - \|y_j\|| < \|y_i\| + \|y_j\| = \|y_i - y_j\| \tag{4.12}$$

$$\Rightarrow |y'_i - y'_j|^2 - \|y_i - y_j\|^2 < 0 \tag{4.13}$$

Together (4.11) and (4.13) imply the sum over all $i > 1, j > 1$ must be negative,

$$\sum_{i>1,j>1} \left(|y'_i - y'_j|^2 - \|y_i - y_j\|^2\right) < 0 \tag{4.14}$$

and therefore substitution into (4.9), which is scaled by a negative factor of $\frac{-1}{n-1}$, implies strict improvement in the objective:

$$f(y') - f(y) = \frac{-1}{n-1} \sum_{i>1,j>1} \left(|y'_i - y'_j|^2 - \|y_i - y_j\|^2\right) > 0 \tag{4.15}$$

y' therefore has a greater objective value than y under $\mathbf{W}_{\text{diff}}$, contradicting the optimality of y and proving the lemma. □

4.4.3 Maximum Distances from the Pivot

Having established that y must be one-dimensional with y_1 on a boundary, it remains to show that the distance between any point y_i and the pivot y_1 must take on the largest admissible value $\delta(i, 1)$, resulting in heuristic values of $h(i, j) = |\delta(i, 1) - \delta(j, 1)|$ – thus exactly matching the DH definition (4.1).

Lemma 4.2. *A solution Y to* (4.3) *under weight matrix $\mathbf{W}_{diff}$ embeds every state i at its true distance from the pivot; i.e., $y_i = \delta(i, 1)$.*

Proof. Without loss of generality, assume $y_1 = 0$ and therefore $y_i \geq 0$. For purposes of contradiction, suppose there exists a state $k \neq 1$ such that $y_k \neq \delta(k, 1)$. Then $y_k < \delta(k, 1)$, since otherwise it would violate local admissibility.

Let y' be the one-dimensional embedding with $y'_i = y_i$ (for all $i \neq k$) but where y_k has been increased to $y'_k = \delta(k, 1)$. Denote by $\epsilon = y'_k - y_k$ the distance by which y_k is off from k's true distance to the pivot. The difference in the objective is:

$$
\begin{aligned}
f(y') - f(y) &= \sum_{ij} W_{ij}|y'_i - y'_j|^2 - \sum_{ij} W_{ij}|y_i - y_j|^2 && (4.16)\\
&= 2\sum_{i} W_{ik}|y'_i - y'_k|^2 - 2\sum_{i} W_{ik}|y_i - y_k|^2 && (4.17)\\
&= 2\left(y_k'^2 - y_k^2 + \sum_{i>1} W_{ik}\left(|y'_i - y'_k|^2 - |y_i - y_k|^2\right)\right) && (4.18)\\
&= 2\left((y_k + \epsilon)^2 - y_k^2 + \sum_{i>1} W_{ik}\left(|y'_i - y'_k|^2 - |y_i - y_k|^2\right)\right) && (4.19)\\
&= 2\left(\epsilon^2 + 2\epsilon y_k - \frac{1}{n-1}\sum_{i>1;i\neq k}\left(|y_i - y'_k|^2 - |y_i - y_k|^2\right)\right) && (4.20)
\end{aligned}
$$

Here, line 4.17 isolates the nonzero differences (i.e., between k and every other state), line 4.18 extracts the distance between the pivot and k from the summation, and line 4.20 applies the definition of $\mathbf{W}_{\text{diff}}$ and the fact that $y'_i = y_i$.

Suppose the only point not equal to $y_1 = 0$ is y_k. For a particular y_i:

$$
\begin{aligned}
|y_i - y'_k|^2 - |y_i - y_k|^2 &= |y'_k|^2 - |y_k|^2 && (4.21)\\
&= \epsilon^2 + 2y'_k y_k - 2y_k^2 && (4.22)\\
&= \epsilon^2 + 2(\epsilon + y_k)y_k - 2y_k^2 && (4.23)\\
&= \epsilon^2 + 2\epsilon y_k, && (4.24)
\end{aligned}
$$

where line 4.22 is just an application of the equality $\epsilon^2 = |y'_k|^2 + |y_k|^2 - 2y'_k y_k$. This

implies a strict inequality on the summation over all $i > 1; i \neq k$:

$$\sum_{i>1;i\neq k} |y_i - y'_k|^2 - |y_i - y_k|^2 = \sum_{i>1;i\neq k} \epsilon^2 + 2\epsilon y_k \tag{4.25}$$

$$= (n-2)(\epsilon^2 + 2\epsilon y_k) \tag{4.26}$$

$$< (n-1)(\epsilon^2 + 2\epsilon y_k) \tag{4.27}$$

The term $(n-2)$ on line 4.26 is due to i being neither the pivot nor k. Otherwise, there exists a point $y_s > y_1$ where $s \neq k$. For a particular y_i:

$$|y_i - y'_k|^2 - |y_i - y_k|^2 = (y'_k - y_i)^2 - (y_k - y_i)^2 \tag{4.28}$$

$$= y_k'^2 - 2y_i y'_k + y_i^2 - y_k^2 + 2y_i y_k - y_i^2 \tag{4.29}$$

$$= (y_k + \epsilon)^2 - 2y_i(y_k + \epsilon) - y_k^2 + 2y_i y_k \tag{4.30}$$

$$= y_k^2 + 2\epsilon y_k + \epsilon^2 - 2y_i y_k - 2y_i\epsilon - y_k^2 + 2y_i y_k \tag{4.31}$$

$$= 2\epsilon y_k + \epsilon^2 - 2y_i\epsilon \tag{4.32}$$

$$= \epsilon^2 + 2\epsilon(y_k - y_i) \tag{4.33}$$

For $i = s$ then $y_i > 0$, giving a strict upper bound on (4.33) of $\epsilon^2 + 2\epsilon y_k$. For all other i this bound is not strict, but altogether the sum over all $i > 1; i \neq k$ is:

$$\sum_{i>1;i\neq k} |y_i - y'_k|^2 - |y_i - y_k|^2 = \sum_{i>1;i\neq k} \epsilon^2 + 2\epsilon(y_k - y_i) \tag{4.34}$$

$$< \sum_{i>1;i\neq k} \epsilon^2 + 2\epsilon y_k \tag{4.35}$$

$$< (n-1)(\epsilon^2 + 2\epsilon y_k) \tag{4.36}$$

Substituting (4.27) or (4.36) into (4.20) shows strict improvement in the objective,

$$f(y') - f(y) > 2\left(\epsilon^2 + 2\epsilon y_k - (\epsilon^2 + 2\epsilon y_k)\right) = 0$$

and therefore Y is not optimal since y' satisfies the constraints and has a higher objective value. This is a contradiction and therefore $y_i = \delta(i, 1)$ for all i. □

Thus the heuristics arising from the optimization (4.3) under weight matrix $\mathbf{W}_{\text{diff}}$ contain the same points as a differential heuristic, up to translation and rotation.

4.5 Optimization Approach

The design of $\mathbf{W}_{\text{diff}}$ (4.4) suggests that differential heuristics optimize a strange objective. The negative weights *reward* the optimizer for *minimizing* all pairwise distances, except for the small fraction of distances to the pivot. This effect can in part be mitigated by careful pivot selection, but the fact remains that DHs are, in some sense, designed to give poor heuristic values between a majority of the states.

4.5.1 Visual Interpretation

How can this insight be used to enhance differential heuristics? To facilitate creating complementary heuristics based on multiple pivots, the optimizer should *still* move all points far from the pivot. But it should also be rewarded, rather than penalized, for any gains made in the *other* heuristic values. The visual interpretation is that this could prevent the graph from being unnecessarily folded together as illustrated in Figure 4.1, and instead flattened uniformly to the line as illustrated in Figure 4.2.

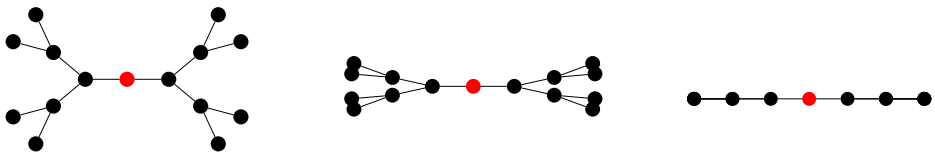


Figure 4.2: Visual interpretation of an *enhanced* differential heuristic: this simple search graph is "flattened" evenly onto the line, rather than "folded" around a point.

4.5.2 Enhanced Objective

This interpretation suggests enhancing a differential heuristic by solving the optimization problem (4.3) using an alternative weight matrix $\mathbf{W}^{+}_{\text{diff}}$, where the negative entries of $\frac{-1}{n-1}$ in $\mathbf{W}_{\text{diff}}$ (4.4) are replaced with small positive values $\epsilon > 0$. If index

1 refers to the pivot, then this matrix appears as follows:

$$\mathbf{W}^{+}_{\text{diff}} = \begin{bmatrix} 0 & 1 & 1 & 1 & \cdots & 1 \\ 1 & 0 & \epsilon & \epsilon & \cdots & \epsilon \\ 1 & \epsilon & 0 & \epsilon & \cdots & \epsilon \\ 1 & \epsilon & \epsilon & 0 & \ddots & \epsilon \\ \vdots & \vdots & \vdots & \ddots & \ddots & \epsilon \\ 1 & \epsilon & \epsilon & \epsilon & \epsilon & 0 \end{bmatrix} \tag{4.37}$$

The corresponding optimization problem is simply:

$$\begin{aligned} \textbf{maximize}_{Y} \quad & \sum_{i,j} (\mathbf{W}^{+}_{\text{diff}})_{ij} \|y_i - y_j\|^2 \\ \textbf{subject to} \quad & \forall (i,j) \in E \quad \|y_i - y_j\| \leq \delta(i,j) \end{aligned} \tag{4.38}$$

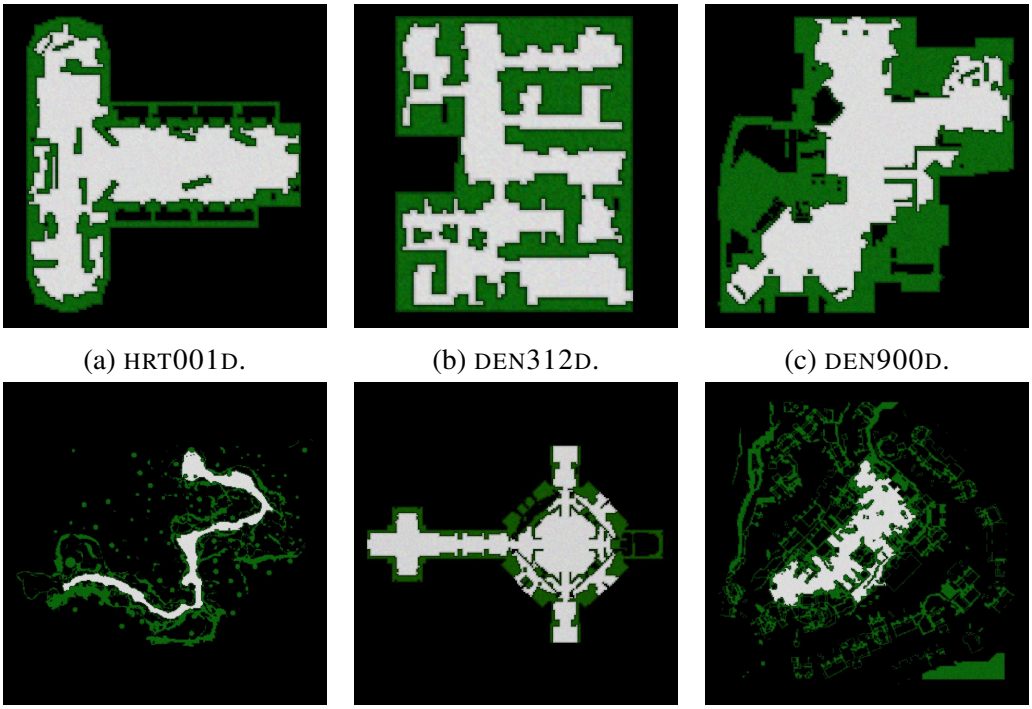
This problem (4.38) can no longer be solved with Dijkstra's algorithm, but it is a special case of Euclidean heuristic optimization and so can *still* be solved tractably – and optimally – by converting it into a semidefinite program as in Section 3.4.1. While the semidefinite formulation has the potential to create high-dimensional heuristics, it retains its dimensionality reducing tendencies. As a result the results can typically be projected down to a single dimension via principal component analysis, all while retaining admissibility and consistency – as proved in Section 3.5.1.

4.6 Evaluation

This evaluation considers a total of 59 maps from BioWare's 2009 *Dragon Age: Origins*. Three representative examples of these maps are shown in Figure 4.3. Each such map is based on a grid with between 168 and 6,240 contiguous open cells, among which an agent can travel cardinally (at unit cost) or diagonally (at cost 1.5, enabling any heuristic to be rounded up to the nearest full half), and cutting corners is once again disallowed to mimic restrictions on an agent with volume. A third-party set of standard benchmark problems which span varying lengths and degrees of difficulty is associated with each individual map [55].[1]

Five sets of heuristics are compared on this domain. First are sets of 1 and 3 differential heuristics (denoted **FDH-1** and **FDH-3**) where the *Farthest* placement

[1] Map data and benchmark problems are publically available at http://www.movingai.com/.



(a) HRT001D. (b) DEN312D. (c) DEN900D.

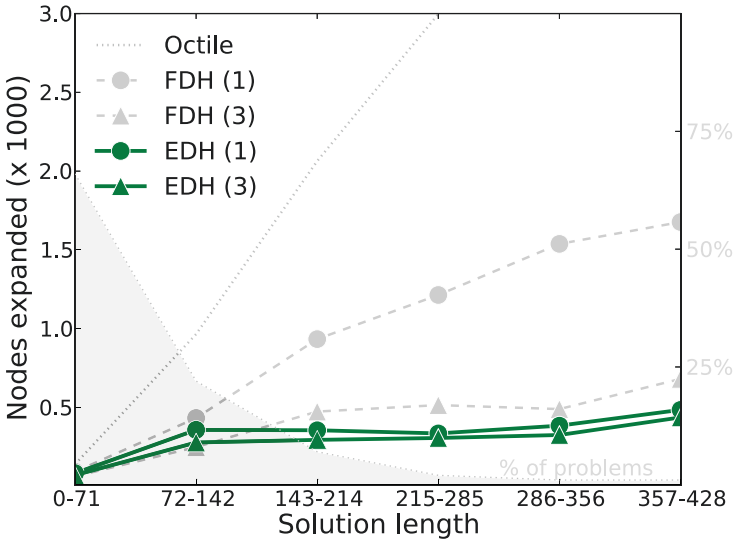
(d) BRC300D. (e) ORZ301D. (f) ORZ200D.

Figure 4.3: Example *Dragon Age: Origins* maps used in the evaluation. The agent can move freely between the open (white) cells, with darker cells blocked off and inaccessible. Most maps feature open rooms and long "central corridors".

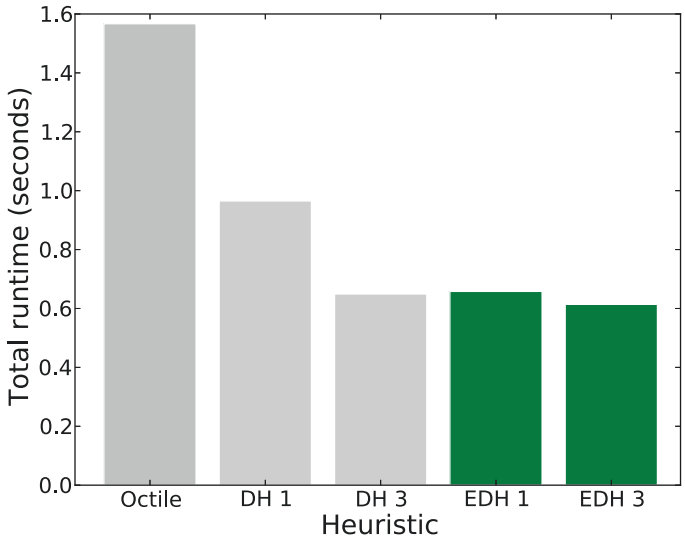
algorithm is used to situate the pivots: the first pivot is placed in a state farthest from a random seed state and each subsequent pivot is placed in a state most distant from the previous pivots. Next are sets of 1 and 3 *enhanced* differential heuristics (denoted **EDH-1** and **EDH-3**). Each EDH set uses the same pivots as the corresponding FDH set, but here the nth EDH uses weights $\mathbf{W}^{+}_{\text{diff}}$ based on the nth DH pivot to solve the optimization problem (4.38), where the non-pivot ϵ entries in $\mathbf{W}^{+}_{\text{diff}}$ are set[2] to 10^{-3} Finally, the default octile heuristic is evaluated on its own, and is combined with each of the preceding by maximizing over all heuristic values.

Figure 4.4 shows the node expansions and elapsed runtime of A* [28]. In Figure 4.4a, there is a clear trend of EDHs leading to fewer node expansions than either the default or differential heuristics, especially on longer paths. On average, FDH-3 appears to have a slight edge over EDH-3 on the shortest paths (i.e., those whose

[2]It can often be better to tune these augmented values to the map in question. Nevertheless, the use of a fixed value of 10^{-3} yielded generally good results across the entire test suite.



(a) A* node expansions for each heuristic.



(b) A* runtimes for each heuristic.

Figure 4.4: A* pathfinding results for standard benchmark problems on 59 *Dragon Age* maps. We compare the default (Octile) heuristic to sets of 1 and 3 differential heuristics (FDH) and sets of 1 and 3 *enhanced* differential heuristics (EDH). Figure 4.4a shows average node expansions as well as the distribution of the different problem lengths, and Figure 4.4b shows runtime totals for each heuristic.

solution length is less than 142), and there are individual maps for which FDH-3 provides better heuristics overall than EDH-3, but these counterexamples are in the minority. Meanwhile, the elapsed runtime results shown in Figure 4.4b mirror these gains shown in the node expansions, although somewhat less dramatically. This is possibly due to the relative infrequency of long paths in the benchmark problems but, in sum, optimal Euclidean heuristics based on $\mathbf{W}^{+}_{\text{diff}}$ show a marked improvement over differential heuristics, and represent an approach worth pursuing.

4.7 Summary

This chapter "reverse engineered" the popular and effective approach of differential heuristics by identifying their implied optimality criterion. This analysis consequently showed the Euclidean heuristic optimization of Chapter 3 generalizes differential heuristics beyond a single dimension. It also revealed an unobserved distance minimizing bias in the implied objective of differential heuristics. And lastly, it facilitated an "enhanced", optimization-driven approach to building one-dimensional heuristics: modifying the differential heuristic objective to give small rewards (rather than penalties) to the optimizer for moving *all* points apart.

While differential heuristics can be built more quickly and can scale to larger problems than the enhanced counterparts that have been introduced here, an empirical study showed that the *concept* of differential heuristics can be improved upon, at least in the important target domain of video game pathfinding. This success motivates the forthcoming investigations into alternative ways to construct strong one-dimensional heuristics, which also forgo the need to truncate a higher-dimensional embedding to a single dimension.

Chapter 5

Line Heuristic Optimization
via Alternating Maximization

This chapter introduces *Line Heuristics*, and an efficient local search procedure based on successive linear progams for defining good heuristics in this space. Later, these heuristics will turn out to be a special case of a more general framework – a case that will be addressed in detail Chapter 6. However, for purposes of clarity, it is helpful to first address this special case and identify its connections in the literature.

5.1 Motivation

The methods of the preceding two chapters are two-step: high-dimensional 'source' embeddings are determined as the result of semidefinite programs, and then are dimensionality-reduced *post factum* using principal component analysis (PCA). Computationally hard problems are neatly avoided by so separating these two steps, but when PCA is forced to discard dimensions with nonzero variance, the separation can cause lost optimality with respect to the original objective (i.e., PCA will find the optimal linear projection, but cannot find a superior nonlinear projection).

Data structure/representation	Points on the line ($\mathbf{y} \in \mathbb{R}^n$)
Heuristic lookup function	$h(i,j) = \lvert y_i - y_j \rvert$
Optimization method(s)	Local search (successive linear programs)
Solution optimality	Local$^+$

$^+$ With logarithmic bounds under the uniformly weighted objective.

Table 5.1: Key characteristics of the Line Heuristic framework.

The stricter the dimensionality constraint (e.g., when $d = 1$, yielding what we will call line heuristics), the greater the potential for lost optimality. This warrants developing a better understanding of how to construct line heuristics directly, without separating the embedding and dimensionality reduction steps.

Working within a single dimension will induce an NP-hard optimization problem, but some guarantees on optimality can still be established. In particular, we will find it possible to state modest probabilistic bounds on a line heuristic by adopting efficient methods from the metric embedding theory literature [6, 37]. Moreover, a novel algorithm will be presented to incrementally improve a one-dimensional embedding until it is *locally optimal* via a succession of linear programs. Table 5.1 shows a summary overview of this framework.

The proposed approach has a number of appealing properties, including its tendency to scale better than SDPs even as it achieves the *anytime* property, its monotonic convergence, and its propensity to create integer embeddings. Later, in Chapter 6, it will also serve as the basis for constructing strong *directed* heuristics, as represented by points on the line. Empirical results will also show that it can give large search performance enhancements on several undirected graphs.

5.2 Line Heuristics

We start with a formal introduction of Line Heuristics as they appear in this chapter.

Definition 5.2.1 (Line Heuristics)**.** Line heuristics are heuristic values that can be computed as absolute distances between points on the real line. Each state i in a search graph with n states is associated with a single real number y_i, and the heuristic lookup is defined as:

$$h(i, j) \equiv |y_i - y_j| \tag{5.1}$$

As in previous chapters, we define $\mathbf{y}$ as the n-vector whose entries are these points:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{5.2}$$

This vector determines a set of heuristic values between all pairs of points, becoming our optimization variable and the primary unit of analysis in this chapter.

Note that heuristics using this lookup (5.1) already exist in the literature (e.g., differential heuristics [56], where each y_i is defined as the optimal path length from state i to a single pivot state, as well as the *enhanced* differential heuristics of Chapter 4 that relied on our framework of Euclidean Heuristic Optimization).

5.3 Optimization Problem

The general problem of arranging points on the line to best match some given distance data has been studied from several perspectives in computing, including data mining [29], bioinformatics [32], and combinatorics [50]. The heuristic construction setting presents a new set of challenges and considerations.

In particular, to determine the "best" line heuristic $\mathbf{y}$, we will follow the pattern of the preceding two chapters and define it as a solution to the following:

$$\begin{aligned} \underset{\mathbf{y}}{\textbf{minimize}} \quad & \mathcal{L}(\mathbf{y}) \\ \textbf{subject to} \quad & \mathbf{y} \text{ is admissible and consistent} \end{aligned} \tag{5.3}$$

That is, $\mathbf{y}$ should be arranged in such a way that the resulting heuristic values encode the search graph distances with the least amount of loss, while being admissible and consistent. These constraints and the objective will be examined in turn.

5.3.1 Constraints

The admissibility and consistency conditions for a line heuristic $\mathbf{y}$ are as follows:

$$\forall\, i, j \quad |y_i - y_j| \leq \delta(i, j) \qquad \textbf{admissibility} \tag{5.4}$$

$$\forall\, i, j, k \quad |y_i - y_j| \leq \delta(i, k) + |y_j - y_k| \qquad \textbf{consistency} \tag{5.5}$$

where $\delta(i, j)$ is the cost of the shortest path from i to j. These constraints apply to all pairs and triples of states, but can be simplified by applying Lemma 3.1:

Theorem 5.1. *A line heuristic is admissible and consistent if and only if it is locally admissible (i.e., distances between neighbors are non-overestimating).*

Proof. The space of absolute distances between all pairs of a set of n points on the line is a simple example of a finite metric space, well-known to satisfy the triangle inequality.[1] It follows by Lemma 3.1 that a line heuristic $\mathbf{y}$ is therefore admissible and consistent if and only if it is locally admissible. □

Consequent to Theorem 5.1, global admissibility and consistency can be achieved by imposing just one nonlinear constraint per edge in the search graph:

$$\forall (i,j) \in E \;\; |y_i - y_j| \leq \delta(i,j) \tag{5.6}$$

For convenience, each such constraint (5.6) will be rewritten as a pair of *equivalent* linear constraints as follows:

$$\forall (i,j) \in E \;\; |y_i - y_j| \leq \delta(i,j) \;\; \Leftrightarrow \;\; \begin{cases} y_i - y_j \leq \delta(i,j) \\ y_j - y_i \leq \delta(i,j) \end{cases} \tag{5.7}$$

Further, if we represent each undirected edge as two *directed* edges of equal weight (i.e., $(i,j) \in E \Leftrightarrow (j,i) \in E$) then line 5.7 can be written more concisely as:

$$\forall (i,j) \in E \;\; y_i - y_j \leq \delta(i,j), \tag{5.8}$$

which is the constraint set that will be used throughout this chapter.

5.3.2 Objective

It is well known that an admissible search algorithm must expand every state encountered whose heuristic is low enough to suggest it may be on an optimal path to the goal [2]. A suitable loss is therefore the weighted sum of errors between the resulting heuristic values and the true distances, for all pairs across n states:[2]

$$\mathcal{L}(\mathbf{y}) = \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} \, |\delta(i,j) - h(i,j)| \tag{5.9}$$

$$= \sum_{i,j} W_{ij} \, |\delta(i,j) - |y_i - y_j|| \tag{5.10}$$

As before, the entries in the weight matrix $\mathbf{W}$ enable specifying the relative importance of each pair of states. A good general heuristic arises from defining $\mathbf{W}$ as the

[1] This space is also a Euclidean space, and so Theorem 3.1 applies as well.

[2] Throughout this chapter, i and j always index the summations from 1 to n, as on on line 5.9.

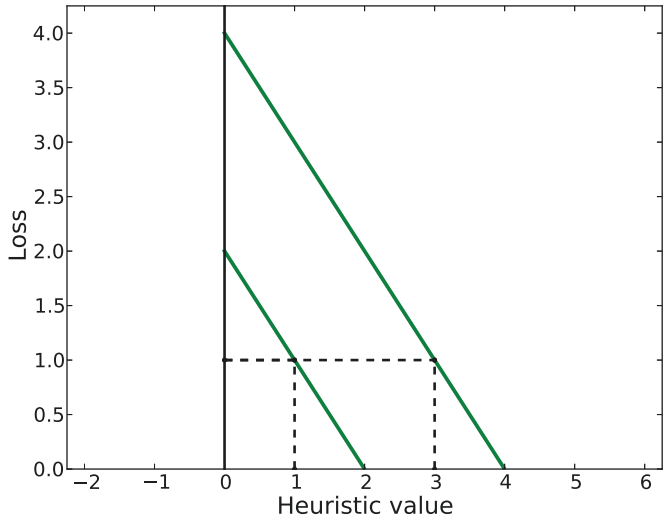


Figure 5.1: Lines showing how the error in a line heuristic between two states interacts with loss. Since the heuristic is an absolute distance, it cannot be less than 0, and the admissibility condition guarantees no negative loss can be incurred.

uniform weight matrix $\mathbf{W}_{\text{unif}}$, which is 1 everywhere but 0 on the diagonal:

$$\mathbf{W}_{\text{unif}} \equiv \mathbf{e}\mathbf{e}^\top - \mathbf{I} \tag{5.11}$$

where $\mathbf{e}$ is the vector of ones. This special case allows us to cite existing probabilistic bounds, which will be discussed in detail later in Section 5.4. But, as usual, $\mathbf{W}$ can also be used to indicate specific points of interest such as common start and goal locations. Alternatively, since many more nodes are expanded during search on longer paths, weight can be added to far away pairs of points.[3]

Example. The sensitivity of this loss to errors in the heuristic is shown by example in Figure 5.1. If the true distance between two states is 2, the loss follows the bottom line and becomes 0 when the heuristic meets its target value of 2. If the true distance is 4, the loss similarly follows the top line. Note the same magnitude of error in both cases contributes equally to the loss, whatever the true distance. In other words, if both heuristics have an error of 1, both contribute 1 to the total loss, which is indicated by the dashed lines in the figure.

[3]Note the following distinction: the optimal Euclidean heuristics in Chapter 3 already implicitly upweight longer paths, since the loss is computed by comparing *squared* distance values.

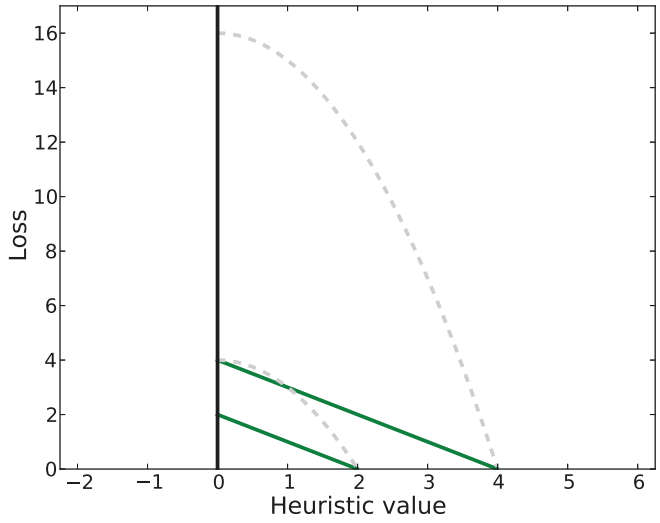


Figure 5.2: Plot showing a comparison to the loss used in Chapter 3.

This loss should be contrasted with the quadratic loss used in Chapter 3 (Figure 5.2), which grows in magnitude as the underlying true distance increases; the maximum error when the true distance is 4 is significantly greater in the latter case.

Simplification. Due to the admissibility condition, the heuristic between states i and j, $h(i,j)$ is always less than their true distance, $\delta(i,j)$, and the loss (5.10) can be simplified by breaking it up as follows:

$$\mathcal{L}(\mathbf{y}) = \sum_{i,j} W_{ij} \, |\delta(i,j) - |y_i - y_j|| \tag{5.12}$$

$$= \sum_{i,j} W_{ij} \, (\delta(i,j) - |y_i - y_j|) \tag{5.13}$$

$$= \sum_{i,j} W_{ij} \, \delta(i,j) - \sum_{i,j} W_{ij} |y_i - y_j| \tag{5.14}$$

Since the first term on line 5.14 does not depend on $\mathbf{y}$, it can be safely ignored without affecting the choice of $\mathbf{y}$, implying minimizing the following is equivalent:

$$\mathcal{L}(\mathbf{y}) \equiv -\sum_{i,j} W_{ij} \, |y_i - y_j| \tag{5.15}$$

Therefore $\mathcal{L}(\mathbf{y})$ can be minimized by *maximizing* the sum of weighted pairwise absolute distances. Combining this newly simplified objective and the constraints:

$$\begin{aligned} \underset{\mathbf{y}}{\textbf{maximize}} \quad & \sum_{i,j} W_{ij} \left| y_i - y_j \right| \\ \textbf{subject to} \quad & \forall (i,j) \in E \;\; y_i - y_j \leq \delta(i,j) \end{aligned} \tag{5.16}$$

5.3.3 Hardness

Even though both the objective and constraints have been simplified to some extent, there is unlikely to be an efficient, polynomial time method for finding optimal solutions to this optimization problem (5.16). In fact, many linear graph arrangement problems are NP-hard [16], and this case will prove to be no exception.

Theorem 5.2. *The optimization problem* (5.16) *is NP-hard.*

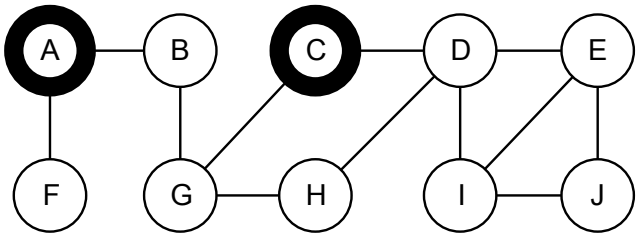
Proof. Saxe proved in 1979 that deciding whether an undirected, integer-weighted graph G can be losslessly embedded to the line is NP-hard [54]. Reducing this decision problem to the optimization problem (5.16) is straightforward. If G can be losslessly embedded to the line, then by definition an optimal embedding $\mathbf{y}^*$ can achieve zero loss under (5.10). It can then be verified in polynomial time that $\mathbf{y}^*$'s pairwise distances exactly match the graph's true distances, thereby answering Saxe's decision problem in polynomial time. □

It should be emphasized that Theorem 5.2 is not due to our choice of loss (5.10), but is due to our choice to work directly with a rank-constrained target space (i.e., the line). Any loss function that accounts for the errors between all pairs of states, including the loss introduced in Chapter 3, enables a similar reduction.

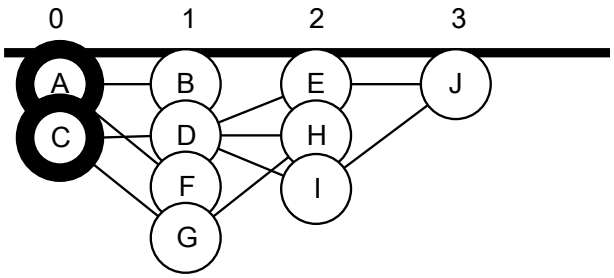
5.4 Bounded Solutions

Theorem 5.2 implies determining a globally optimal line heuristic can become intractable for large graphs. Fortunately, the optimization problem (5.16) resembles the known problem of finding contractive embeddings of undirected weighted graphs. Many forms of this problem have been studied,[4] but a solution can be based

[4] See work by Dhamdhere, Håstad *et al.*, and Rabinovich [15, 32, 50].



(a) Original graph with unit edge costs.



(b) Resulting embedding locations on the line.

Figure 5.3: Illustration of a Lipschitz embedding. The top figure shows a typical arrangement of a graph for visualization purposes. The bottom figure shows how a Lipschitz embedding aligns those same vertices to the line.

on a popular and broadly applicable approach studied by Bourgain [6] and subsequently Linial *et al.* [37]. The result is a polynomial time algorithm that produces admissible and consistent heuristics with a probabilistic bound on solution quality.

5.4.1 Lipschitz Embeddings

The main tool used is an easily computed embedding that is often referred to as a *Lipschitz embedding*. These embeddings have been used, for instance, in similarity search [29] and bioinformatics [36], and can be viewed as a generalization of differential heuristics [56]. In a Lipschitz embedding, each y_i is defined as vertex i's shortest path distance to the nearest vertex in a subset of the vertices $R \subseteq V$ called a *reference set*:

$$y_i \leftarrow \delta(i, R) \equiv \min_{j \in R} \delta(i, j) \tag{5.17}$$

In practice, this vector $\mathbf{y}$ can be efficiently computed by modifying Dijkstra's algorithm [17] to solve the multi-source shortest path problem rooted at the *set* R (rather than the canonical single-source shortest path problem).

Example. An example Lipschitz embedding is shown in Figure 5.3. The depicted graph (Figure 5.3a) has unit edge costs, and the reference set R is defined as containing the two filled vertices, A and C. The shortest-path distance from each vertex i to the nearest reference vertex – either A or C – defines a point on the line y_i to which that vertex i is mapped (Figure 5.3b). The heuristics resulting from taking pairwise distances are admissible and consistent, which the following will establish.

Lemma 5.1. *For every two points i, j, and an arbitrary reference set $R \subseteq V$, the following "generalized" form of the triangle inequality holds:*

$$\delta(i, R) - \delta(j, R) \leq \delta(i, j) \tag{5.18}$$

Proof. Let $q = \arg\min_{k \in R} \delta(j, k)$ for an arbitrary point j and reference set R:

$$\delta(i, R) - \delta(j, R) = \delta(i, R) - \delta(j, k) \tag{5.19}$$

$$\leq \delta(i, k) - \delta(j, k) \tag{5.20}$$

$$\leq \delta(i, j) \tag{5.21}$$

where the inequality on line 5.21 is just the triangle inequality applied to the true shortest path distances. A trivial extension, assuming symmetric distances (i.e., $\delta(i, j) = \delta(j, i)$), is that reordering the indices also gives the following:

$$\delta(j, R) - \delta(i, R) \leq \delta(j, i) = \delta(i, j) \tag{5.22}$$

Thus proving this generalization of the triangle inequality. □

Theorem 5.3. *A one-dimensional Lipschitz embedding* $\mathbf{y}$ *based on reference set R (defined as on line 5.17 gives admissible and consistent heuristics on the line.*

Proof. Simple substitution as well as applying Lemma 5.1 implies:

$$h(i, j) = |y_i - y_j| = |\, \delta(i, R) - \delta(j, R)| \tag{5.23}$$

$$= \max\{\delta(i, R) - \delta(j, R), \delta(j, R) - \delta(i, R)\} \leq \delta(i, j) \tag{5.24}$$

where line 5.24 is implied by the inequalities on lines 5.21 and 5.22 together. This establishes admissibility, and therefore Theorem 5.1 implies the heuristic defined by the distances between the points in $\mathbf{y}$ must also be consistent. □

Having proved admissibility and consistency must always hold, the main difficulty in generating good Lipschitz embeddings lies in choosing the reference set R.

Algorithm 5.1: Returns a line heuristic with bounded average distortion.

Input: An undirected search graph $G = (V, E)$.
Output: $\mathbf{y} \in \mathbb{R}^n$, an admissible and consistent embedding to the line.

```
q ← randInt(1, log n);
R ← randSubset(V, q);
for i from 1 to |V| do
    y_i ← min_{j∈R} δ(i, j) ;
return y;
```

5.4.2 Probabilistic Approach due to Bourgain/Linial *et al.*

Suppose R is selected from among a certain distribution (2^k vertices chosen at random, where k is chosen from $[1, 2, ..., \log n]$), as summarized in Algorithm 5.1. Linial *et al.* [37], following work by Bourgain [6], prove that every pairwise distance in the resulting Lipschitz embedding can be expected to achieve:

$$\forall i, j \quad \mathbb{E}\left[|y_i - y_j|\right] \geq \frac{\delta(i, j)}{40 \log n} \tag{5.25}$$

Substituting line 5.25 into the objective (5.16) under $\mathbf{W}_{\text{unif}}$ (5.11) yields the bound:

$$\mathbb{E}\left[\sum_{i,j} W_{ij}|y_i - y_j|\right] = \mathbb{E}\left[\sum_{i,j} |y_i - y_j|\right] \tag{5.26}$$

$$\geq \sum_{i,j} \frac{\delta(i, j)}{40 \log n} \tag{5.27}$$

Observe that the bound (5.27) is at least as strong as an approximation guarantee – that is, it is with respect to the *true* distances, not just the best obtainable distances.

Algorithm 5.1 achieves a probabilistic bound on the objective while never considering that objective explicitly; it also requires nothing more than the solution of a multi-source shortest path problem through the state space. This simplicity is to the algorithm's credit, and it is therefore worth citing and studying empirically. Nevertheless, in practice, most embeddings generated using this method can be greatly improved by conducting a local search. , as described in Section 5.5.

5.5 Optimization Approach

This section gives an efficient algorithm for finding at least locally optimal line heuristics. This necessitates a simple reformulation of the objective that adds an extra optimization variable, but will show that any given line heuristic can be improved upon until it is at least locally optimal by way of a succession of linear programs interwoven with a sorting operation. Subsequent analysis of this method will reveal a number of novel properties.

5.5.1 Sign Matrix Formulation

Define $\mathbf{S} \in \{-1, 0, 1\}^{n \times n}$ as an auxiliary *sign* matrix with the following conditions:

$$\forall i \neq j \quad S_{ij} \in \{1, -1\} \qquad \textbf{unit entries} \tag{5.28}$$

$$\forall i \neq j \quad S_{ij} = 1 \Leftrightarrow S_{ji} = -1 \qquad \textbf{antisymmetric} \tag{5.29}$$

$$\forall i \quad S_{ii} = 0 \qquad \textbf{“hollow”} \tag{5.30}$$

Note for any pair i, j that the term $S_{ij}(y_i - y_j)$ is maximized whenever:

$$S_{ij}(y_i - y_j) = \text{sign}(y_i - y_j)(y_i - y_j) = |y_i - y_j| \tag{5.31}$$

where the sign operator gives -1 if its argument is negative, 1 if its argument is positive, and 0 otherwise. Thus $\mathbf{S}$ implies a *permutation* of the points, and can be used to rewrite the optimization objective (5.16) as follows:

$$\begin{aligned} \underset{\mathbf{S},\mathbf{y}}{\textbf{maximize}} \quad & \sum_{i,j} W_{ij}\, S_{ij}\, (y_i - y_j) \\ \textbf{subject to} \quad & \forall (i,j) \in E \;\; y_i - y_j \leq \delta(i,j), \\ & \mathbf{S} \text{ obeys conditions (5.28) through (5.30).} \end{aligned} \tag{5.32}$$

Condition 5.29 can be used to rewrite the objective as follows:

$$\sum_{i,j} W_{ij} S_{ij} (y_i - y_j) = \sum_{i,j} W_{ij} S_{ij} y_i - \sum_{i,j} W_{ij} S_{ij} y_j \tag{5.33}$$

$$= \sum_i y_i \sum_j W_{ij} S_{ij} - \sum_i y_i \sum_j W_{ji} S_{ji} \tag{5.34}$$

$$= \sum_i y_i \left(\sum_j W_{ij} S_{ij} + \sum_j W_{ji} S_{ij} \right) \tag{5.35}$$

$$= \sum_i y_i \sum_j (W_{ij} + W_{ji}) S_{ij} \tag{5.36}$$

Line 5.33 splits the sum into a difference of two terms, while line 5.34 swaps the labels of the indices i and j in the second term. Next, line 5.35 factors the sum over $\mathbf{y}$ out of both terms. Line 5.36 finally applies the antisymmetric property (5.29). This expression can also be further simplified using vector notation:

$$= \mathbf{y}^\top ((\mathbf{W} + \mathbf{W}^\top) \circ \mathbf{S}) \mathbf{e} \tag{5.37}$$

$$\equiv \mathbf{c}^\top \mathbf{y} \tag{5.38}$$

Here the operator denoted $\circ$ is a Hadamard (entrywise) product, e is the vector of ones, and on line 5.38, $\mathbf{c} = ((\mathbf{W} + \mathbf{W}^\top) \circ \mathbf{S})\mathbf{e}$ is a vector in n dimensions. This reformulation will prove essential to the following local search algorithm.

5.5.2 Alternating Maximization

Since the optimization problem is NP-hard (Theorem 5.2) there is unlikely to be an efficient (i.e., polynomial-time) solution for $\mathbf{y}$ and $\mathbf{S}$ simultaneously. But independent of the other, each variable has a polynomial-time solution. These imply simple update rules, together comprising the local search procedure in Algorithm 5.2.

Note there are a number of efficient ways to initialize the local search, with example seeds including random/zero configurations, differential heuristics,[5] and Lipschitz embeddings (Section 5.4). More involved graph embedding methods, including the Euclidean heuristic optimization of Chapter 3 are also possibilities.

[5] If the optimization is parametrized with the *enhanced* differential heuristic weight matrix $\mathbf{W}^+_{\text{diff}}$ (Chapter 4) an appropriate choice may be the corresponding differential heuristic.

Update Step 1: Updating the Sign Matrix $\mathbf{S}$

Given any line heuristic $\mathbf{y}$, an optimal sign matrix $\mathbf{S}$ can be determined by sorting the entries of $\mathbf{y}$ and using the resulting (strict) ranking r to define $\mathbf{S}$ as:

$$S_{ij} = \begin{cases} 0 & \text{if } i = j \\ 1 & \text{if } r_i > r_j \\ -1 & \text{else} \end{cases} \tag{5.39}$$

Defining $\mathbf{S}$ as on line 5.39 only involves a sorting operation followed by determining the entries in $\mathbf{S}$, and so the problem can in general be solved in $O(n^2)$ time. This step corresponds to lines 3 through 9 in Algorithm 5.2.

Example. Let $\mathbf{y} = [5, 2, 0, 4, 2]^\top$ be a line heuristic. Although two sorted orderings of $\mathbf{y}$ are feasible due to the repeat entries in $\mathbf{y}$, one possible strict ordering is $\mathbf{r} = [4, 1, 0, 3, 2]^\top$, yielding the sign matrix:

$$\mathbf{S} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ -1 & 0 & 1 & -1 & -1 \\ -1 & -1 & 0 & -1 & -1 \\ -1 & 1 & 1 & 0 & 1 \\ -1 & 1 & 1 & -1 & 0 \end{bmatrix} \tag{5.40}$$

If $\mathbf{W} = \mathbf{W}_{\text{unif}}$ as defined on line 5.11 then the linear coefficient vector is:

$$\mathbf{c} = ((\mathbf{W} + \mathbf{W}^\top) \circ \mathbf{S})\mathbf{e} = [4, -2, -4, 2, 0]^\top \tag{5.41}$$

Thus establishing the first half of the iterative algorithm.

Update Step 2: Updating the Embedding $\mathbf{y}$

Given a feasible sign matrix $\mathbf{S}$, the best heuristic $\mathbf{y}$ is a solution to the following:

$$\begin{aligned} \underset{\mathbf{y}}{\textbf{maximize}} \quad & \mathbf{c}^\top \mathbf{y} \\ \textbf{subject to} \quad & \forall (i,j) \in E \;\; y_i - y_j \leq \delta(i,j) \\ & y_1 = 0 \end{aligned} \tag{5.42}$$

where once again $\mathbf{c} = ((\mathbf{W} + \mathbf{W}^\top) \circ \mathbf{S})\mathbf{e}$. Note a "centering" constraint has been imposed on y_1 to resolve translation invariance, but does not affect the resulting heuristic values. This update step corresponds to lines 11 and 12 in Algorithm 5.2.

Algorithm 5.2: Defines a line heuristic that is at least locally optimal.

Input: A search graph $G = (V, E)$, an initial ("seed") embedding $\mathbf{y} \in \mathbb{R}^n$, and a weight matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$.

Output: $\mathbf{y} \in \mathbb{R}^n$, an admissible and consistent heuristic.

repeat
  // Update step for $\mathbf{S}$
  Define r as a sorted ordering of $\mathbf{y}$;
  for i *from* 1 *to* n **do**
    $S_{ii} \leftarrow 0$;
    for j *from* $i + 1$ *to* n **do**
      if $r_i > r_j$ **then** $S_{ij} \leftarrow 1$;
      else $S_{ij} \leftarrow -1$;
      $S_{ji} \leftarrow -S_{ij}$;
  // Update step for $\mathbf{y}$
  $c \leftarrow ((W + W^\top) \circ S)^\top e$;
  Define $\mathbf{y}$ as a solution to the linear program (5.43);
until convergence;
return $\mathbf{y}$

This optimization problem (5.42) is in fact a linear program, and can be solved in polynomial time using widely available standard toolbox solvers such as the IBM CPLEX optimizer. The canonical LP formulation is as follows:

$$\begin{aligned} \underset{\mathbf{y}}{\textbf{maximize}} \quad & \mathbf{c}^\top \mathbf{y} \\ \textbf{subject to} \quad & \mathbf{A}\mathbf{y} \leq \mathbf{b} \end{aligned} \tag{5.43}$$

with $\mathbf{A} \in \mathbb{R}^{|E|+1 \times n}$. The first $|E|$ rows of $\mathbf{A}$ define the admissibility constraints for each edge $(i, j) \in E$, enumerated by index p. That is, A_{ik} is 0 everywhere except: $A_{pi} = 1$ and $A_{pj} = -1$, with $b_p = \delta(i, j)$. The last row of $\mathbf{A}$ captures the "centering" constraint on y_1: that is, $A_{1,p+1} = 1$ and $A_{i>1,p+1} = 0$, with $b_{p+1} = 0$. Note each row of $\mathbf{A}$ contains no more than two nonzero entries, and when there *are* two nonzero entries, they are of opposite opposite sign.

5.6 Analysis

Algorithm 5.2 has some useful properties, including its propensity to generate integer embeddings. It can also be interrupted after any iteration without violating admissibility or consistency, and so is an *anytime* algorithm. Due to the constraints on the linear program, the resulting heuristic will always be admissible and consistent, but its monotonic convergence implies it finds better solutions over time.

5.6.1 Integer Embedding Theorem

The following describes cases in which the optimization method of Section 5.5.2 is guaranteed to produce integer embeddings. From a practical standpoint, integer values are well suited to storage and compression, and their use circumvents the inefficiency and imprecision that often accompanies floating point arithmetic.

Theorem 5.4. *If the transition costs in the search graph are all integer valued, then a solution to the linear program* (5.43) *will only have integer coordinates (*$\mathbf{y} \in \mathbb{Z}^n$*).*

Proof. Given the linear program $\max_{\mathbf{y}} \mathbf{c}^\top \mathbf{y} : \mathbf{A}\mathbf{y} \leq \mathbf{b}$, an important result in combinatorial optimization is the following: if $\mathbf{A}$ is totally unimodular (TU) and $\mathbf{b}$ is integral, then said linear program has integral optima *for any* $\mathbf{c}$. But note the LP (5.43) defines the entries of $\mathbf{b}$ as the transition costs between neighbors (or 0 in the case of the constraint $y_1 = 0$). Therefore $\mathbf{b}$ is integral whenever the transition costs are integral. Moreover, a useful result by Hoffman and Kruskal [30] can be used to prove the constraints on $\mathbf{y}$ imply $\mathbf{A}$ is a TU matrix, since the following sufficient conditions hold: *a*) each entry of $\mathbf{A}$ is either 0, 1, or -1; *b*) each row[6] of $\mathbf{A}$ has at most two nonzero entries; and *c*) any two nonzero entries in a given row of $\mathbf{A}$ have opposite signs, altogether proving the proposition. □

[6]Hoffman and Kruskal's result is commonly quoted as referring to the *columns* of $\mathbf{A}$, but this is without loss of generality since the TU property is a property of the determinants of all square submatrices of $\mathbf{A}$. Transposing a matrix does not change its determinant.

5.6.2 Convergence and Termination

Assuming the pairwise distance between any two points is defined and finite and $\mathbf{W}$ is non-negative, the following shows that Algorithm 5.2 *must* converge to a local optimum, and moreover that it must terminate in finite time.

Theorem 5.5. *Algorithm 5.2 is monotonically convergent.*

Proof. The proof will show both update rules monotonically increase the objective (5.38), toward a finite maximum. First, let $\mathbf{S}$ and $\mathbf{S}'$ be sign matrices before and after the sign update (Algorithm 5.2, lines 2–9). The difference in the objective is:

$$\sum_{i,j} W_{ij} S'_{ij}(y_i - y_j) - \sum_{i,j} W_{ij} S_{ij}(y_i - y_j) \tag{5.44}$$

$$= \sum_{i,j} W_{ij} \left(S'_{ij}(y_i - y_j) - S_{ij}(y_i - y_j) \right) \tag{5.45}$$

Whenever $S_{ij} = S'_{ij}$ it must be the case that $S_{ij}(y_k - y_\ell) = S'_{ij}(y_k - y_\ell)$. Moreover, $S_{ij} \neq S'_{ij}$ implies $S_{ij}(y_k - y_\ell) = -|y_k - y_\ell| \leq S'_{ij}(y_k - y_\ell)$, since the offdiagonal entries of $\mathbf{S}$ and $\mathbf{S}'$ must be 1 or -1 as stated in (5.28). Altogether, $S'_{ij}(y_k - y_\ell) - S_{ij}(y_k - y_\ell) \geq 0$, for all i, j, and substituting into line 5.45 gives

$$\sum_{i,j} W_{ij} \left(S'_{ij}(y_i - y_j) - S_{ij}(y_i - y_j) \right) \geq 0 \tag{5.46}$$

if $\mathbf{W}$ is non-negative. Thus the sign update rule leads to monotonic gains. Next, the update rule for $\mathbf{y}$ maximizes $\mathbf{c}^\top \mathbf{y}$ by definition (5.42), so is at least as good as any feasible alternative, proving monotonicity of the second update rule.

Since the admissibility condition upper bounds the objective by a constant C,

$$\forall\, i, j \quad |y_i - y_j| \leq \delta(i, j) \tag{5.47}$$

$$\Rightarrow \sum_{i,j} W_{ij} |y_i - y_j| \leq \sum_{i,j} W_{ij}\, \delta(i, j) < C, \tag{5.48}$$

then Algorithm 5.2 must be monotonically convergent. □

Theorem 5.6. *Algorithm 5.2 terminates in finite time.*

Proof. The sign matrix $\mathbf{S}$ is a discrete matrix variable, and so if the search graph contains a finite number of states n, then the number of unique n-by-n sign matrices m must also be finite. By the pigeonhole principle, at least one sign matrix will

be repeated after $m + 1$ iterations of the update rules. Assuming the LP solver is deterministic, the same $\mathbf{y}$ will then be generated at least twice as well. According to Proposition 5.5, the objective improves monotonically across iterations, so repetition of $\mathbf{y}$ implies the objective value must not have changed in the intervening steps, which would cause the algorithm to terminate (line 13). □

5.7 Evaluation

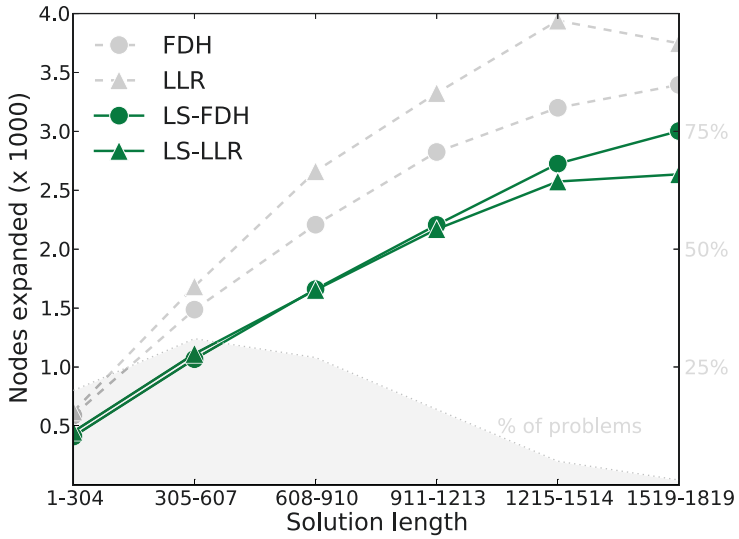
The next section explores three empirical issues. First, a synthetic *Spider Search* domain is used to compare a locally optimal line heuristic to some alternatives, while also evaluating different ways of seeding that optimization. Next, the *Word Search* domain of Chapter 3 is revisited, examining key issues *vis a vis* Proposition 5.4. Finally, we explore parametrizing the weight matrix $\mathbf{W}$ by drawing inspiration from the enhanced differential heuristics of Chapter 4, and build multiple *complementary* line heuristics for a larger suite of pathfinding domains.

5.7.1 Spider Search

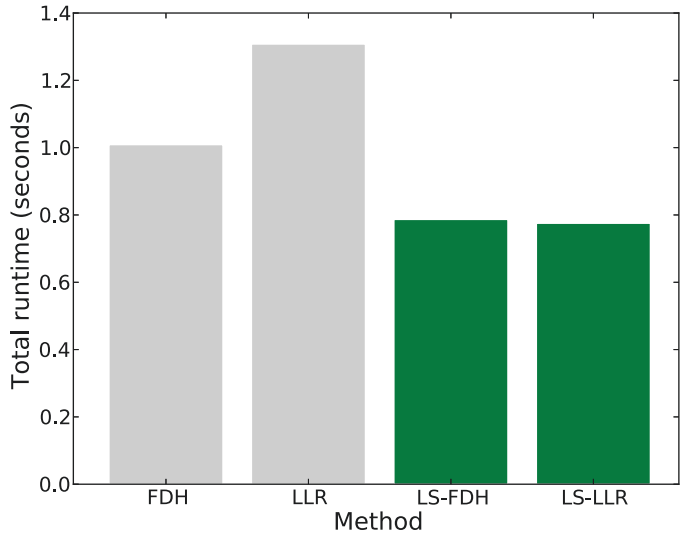
A t-*spider* is a tree whose root node is of degree $t \geq 3$, while all other nodes – comprising the "legs" – have degree 1 or 2 (Figure 5.5). The agent's task is to transition from a start node in the tree to a goal node in the smallest possible number of steps. This domain serves as a simple but illustrative example of a search graph that does not map perfectly to the line; we therefore cannot expect a line heuristic to exactly capture all of the distances, but our optimization approach is hypothesized to outperform existing alternatives.

The specific graph we experiment with is a 15-spider whose legs have between 1 and 1,000 states determined uniformly at random, yielding a total of 7,459 states. Several alternative heuristics are compared. First is the best[7] of ten differential heuristics selected using the *Farthest* algorithm (**FDH**); next is the best of ten using the probabilistic approach of Linial *et al.* as in Section 5.4 (**LLR**); finally, local search as described in Section 5.5, parametrized by a uniform weight matrix, and seeded with both of the former (**LS-FDH** and **LS-LLR** respectively).

[7]Here, the "best" is determined as the heuristic resulting in the fewest total node expansions.



(a) Bucketed average node expansions.



(b) A* total CPU runtime.

Figure 5.4: *Spider Search* results for A* across 10,000 random problems. A differential heuristic placed using the farthest algorithm (FDH), the probabilistic approach of Linial *et al.* (LLR), and locally optimal variants of each (LS-FDH and LS-LLR respectively) are compared. Figure 5.4a shows average node expansions and the distribution of problem lengths, and Figure 5.4b shows runtime totals.

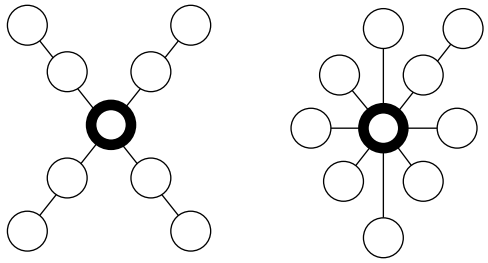


Figure 5.5: Illustration of a 4-spider and an 8-spider.

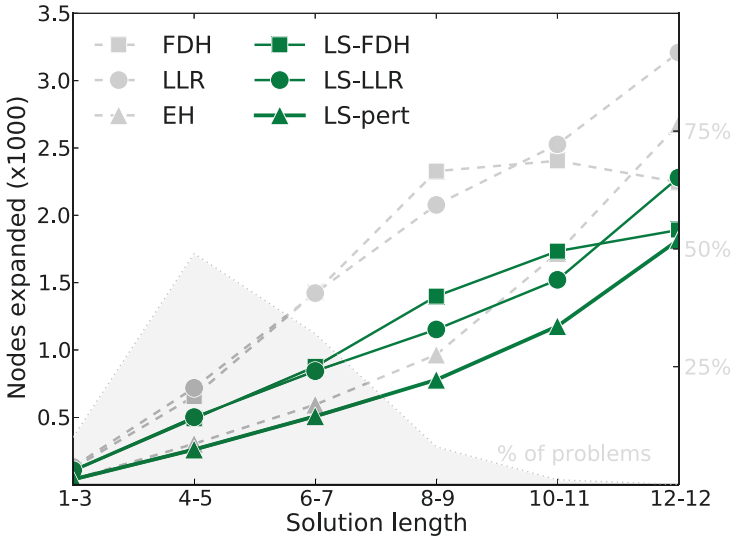
The average node expansions and total runtime by A* [28] (including the time to reconstruct the optimal move sequence) are tallied across a suite of 10,000 random problem instances in Figure 5.4. The local search variations outperform their seeds in terms of both of these metrics, and the choice of of starting seed appears negligible in this domain. Local optimality alone appears to be the greatest contributing factor influencing performance, albeit on a very simple domain.

5.7.2 Word Search

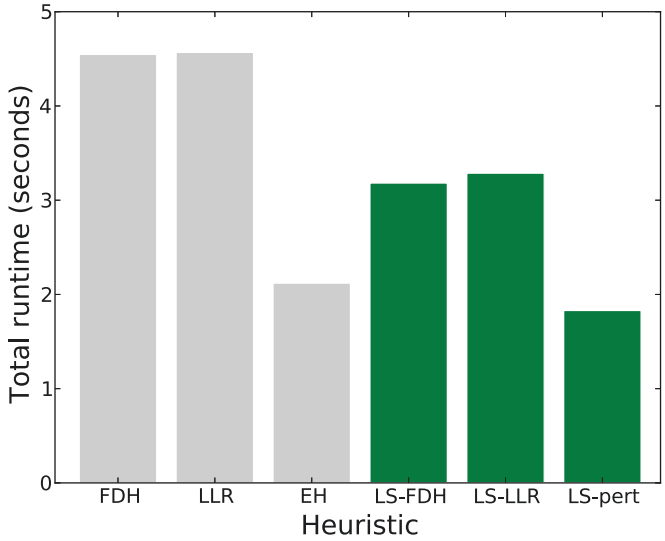
Next we revisit the *Word Search* domain of Chapter 3, a densely connected state space of 4,820 four-letter words with edges connecting words differing by one letter. The Euclidean heuristic optimization in Section 3.6.2 suggested this domain was inherently multidimensional; correspondingly, Euclidean heuristics in six and eighteen dimensions strongly outperformed sets of six and eighteen differential heuristics. But what if we are restricted to a one-dimensional heuristic?

The following compares several one-dimensional heuristics, including the best of ten differential heuristics using the *Farthest* algorithm (**FDH**); the best of ten of Linial *et al.*'s probabilistic approach (**LLR**); locally optimal variations of each of the former; and an optimal Euclidean heuristic in one dimension (**EH**). Any fractional heuristics given by the latter are rounded *up* to the next integer value.

Average runtimes and node expansions by A* on a suite of 10,000 random problems are shown in Figure 5.6. Overall, the results mimic those for the Spider Search domain, with the two LS variations outperforming their unoptimized counterparts. But the EH outperforms both by a significant margin. Its success may be due to its use of a different objective, which was solved to *global* optimality, but we also



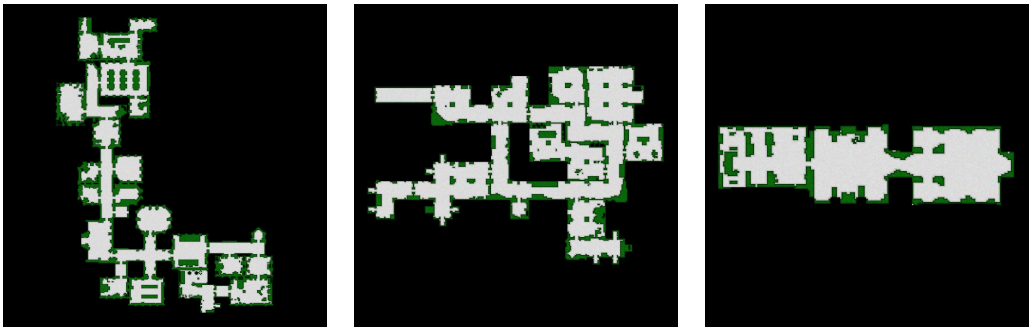
(a) Bucketed average node expansions.



(b) A* total CPU runtime.

Figure 5.6: *Word Search* results for A* across 10,000 random problems. A differential heuristic placed using the farthest algorithm (FDH), the probabilistic approach of Linial *et al.* (LLR), Euclidean heuristics using a ceiling operator (EH), locally optimal variants of each FDH and LLR (LS-FDH, LS-LLR), and a 'perturbed' variant of LS-LLR (LS-pert) are compared. Figure 5.6a shows average node expansions and the distribution of the problem lengths, and Figure 5.6b shows runtime totals.

(a) LAK303D. (b) LAK304D. (c) LAK506D.



(d) DEN005D. (e) DEN011D. (f) RMTST01.

Figure 5.7: Example *Dragon Age: Origins* maps used in the evaluation. Most maps are several times larger than the maps used in the evaluation in Chapter 4.

know from Section 3.6.2 that rounding its fractional values *up* is having a significant impact. Proposition 5.4 implies that the *LS* variants cannot benefit from this effect since they must give integer heuristics – but can it be induced?

To answer this, we perturb one of the locally optimal heuristics (LS-LLR) off its integer coordinates by selecting a random entry y_s and adding or subtracting 10^{-3} as long as this: *a)* preserves admissibility; and *b)* increases the sum of pairwise heuristics when a ceiling is applied. This is repeated 10^7 times. No claims can be made to to this procedure's optimality, and its incremental effect is smaller than that of the local search procedure. Moreover, its use should be weighed against the benefits of pure integer representations. But the resulting heuristic (**LS-pert**) performs significantly better and achieves comparable runtimes to its EH competitor.

5.7.3 Pathfinding

The last problem set returns to the domain of grid-based pathfinding. The parameters are as originally laid out in Section 3.6.3: the agent can move between any adja-

cent open cells, cardinal moves have unit cost, diagonals cost 1.5, and corner-cutting is disallowed as a means to emulate volume restrictions. Because Algorithm 5.2 scales better than the semidefinite programs of Chapter 4, we consider some larger maps: the problem set is comprised of all standard benchmark problems [55] on contiguous maps with between 168 and 18,890 and states (see Figure 5.7).

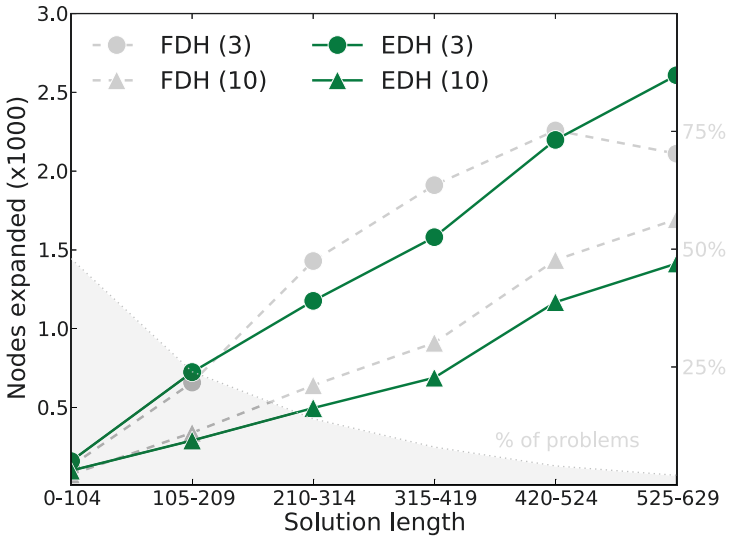
Guided by Section 4.6, we consider the problem of building multiple complementary line heuristics. In that section we observed that *enhanced* differential heuristics (EDHs) chosen with the *Farthest* placement algorithm were high performing in this domain. The following attempts to repeat that success by parametrizing the optimization (5.16) with $\mathbf{W}^{+}_{\text{diff}}$, yielding an iterative linear programming approach to EDHs. While the optimization is nonconvex, each unoptimized differential heuristic provides a good seed for the search for its corresponding EDH.

The following heuristics are compared. Baseline sets of 3 and 10 differential heuristics are built using the *Farthest* selection algorithm to select pivots (denoted *FDH-3/10*). Next are sets of 3 and 10 *enhanced* differential heuristics (denoted *EDH-3/10*). The nth EDH uses weights $\mathbf{W}^{+}_{\text{diff}}$ based on the nth FDH pivot to solve the optimization problem (4.38), where the ϵ entries in $\mathbf{W}^{+}_{\text{diff}}$ are set to $1/n$. The "default" octile heuristic is combined with each of the preceding heuristics by maximizing over all heuristic values, and is also evaluated on its own.

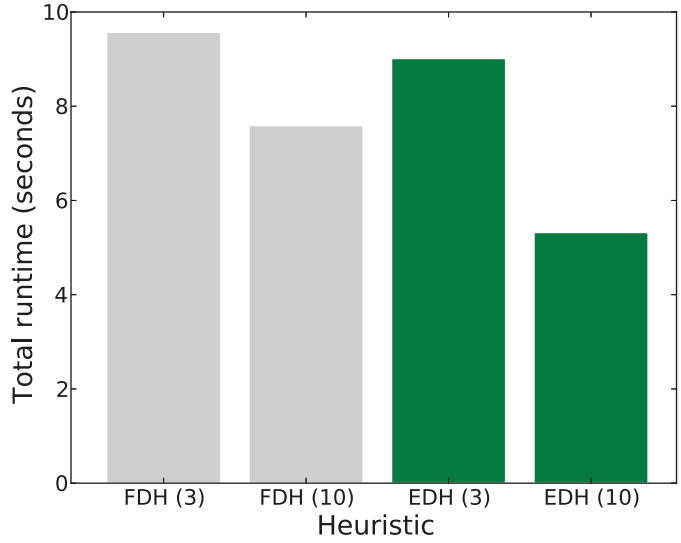
The results, tallied in Figure 5.8, show significant improvements of the *EDH* variants over their *FDH* seeds, both in terms of node expansions and runtime.

5.8 Summary

This chapter presented a novel, optimization-driven approach to building admissible and consistent heuristics represented by points on the line. This is akin to an embedding problem with a dimensionality constraint (i.e., one dimension), resulting in an NP-hard optimization problem, but this chapter presented an efficient, approximate solution based on two insights. The first was that, as with the Euclidean heuristics of Chapter 3, local admissibility constraints imply global admissibility and consistency guarantees. The second was the reformulation of an intuitive objective (i.e.,



(a) Bucketed average node expansions.



(b) A* total CPU runtime.

Figure 5.8: A* pathfinding results for standard benchmark problems on *Dragon Age* maps. We compare sets of 3 and 10 differential heuristics (FDH) to sets of *enhanced* differential heuristics (EDH). Figure 5.8a shows average node expansions as well as the distribution of the different problem lengths, and Figure 5.8b shows runtime totals for each heuristic.

sum of errors) as a function of two variables, each with a simple, independent update rule. Together these insights fostered a monotonically convergent, anytime algorithm that terminates on locally optimal solutions.

A subsequent empirical evaluation showed competitive search performance over alternatives; these successes furthermore carried over into a setting requiring multiple *complementary* line heuristics, which was achieved by adopting Chapter 4's scheme for improving a set of differential heuristics based on an effective pivot layout. Significantly, the ideas in this chapter pave the way for a new type of general *directed* heuristic in one dimension, which will be introduced in the next chapter.

Chapter 6

Hinge Heuristic Optimization
for Asymmetric Domains

Without departing from the special but fundamental case of representing a heuristic as a set of points embedded on the line, we will now launch an investigation into a more general family of asymmetric heuristics. We call this family of heuristics *hinge heuristics*, and show that these heuristics capture and generalize the line heuristic optimization of Chapter 5, as well as a wider family of directed heuristics.

6.1 Motivation

In a *directed* search graph, the transitions between two states can be one-way or have differing cost depending on the direction the agent is travelling. Examples include moving a mass across terrain (Figure 6.1), navigating road networks where traffic laws can disproportionately affect the travel distances between points [24], or imitative pathfinding resulting from some machine-learned cost map [51].

Unfortunately, undirected (or *symmetric*) heuristics of the kind studied in the preceding chapters may be ineffective in such domains, because an accurate un-

Data structure/representation	Points on the line ($\mathbf{y} \in \mathbb{R}^n$)
Heuristic lookup function	$h(i,j) = (y_i - y_j)^+$
Optimization method(s)	Local search (successive linear programs)
Solution optimality	Local$^+$

$^+$ With logarithmic bounds under the uniformly weighted objective.

Table 6.1: Key characteristics of the Hinge Heuristic framework.

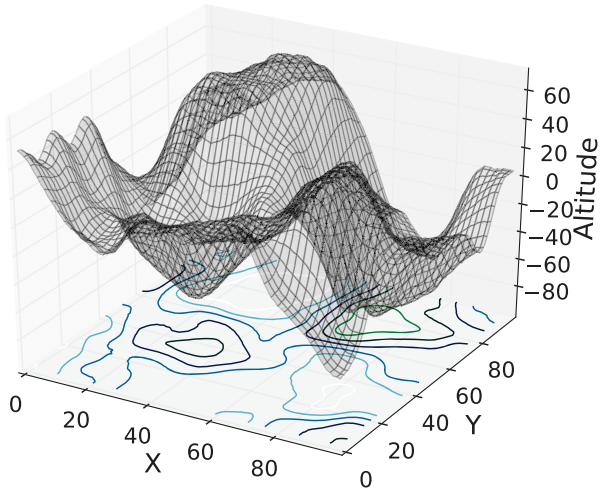


Figure 6.1: Mesh plot and contour map of a 100m-by-100m "terrain" domain.

derestimating heuristic in one direction can be arbitrarily poor in the other. For instance, if the cost for travelling from state a to b is 100 and the cost from b to a is 1, a symmetric admissible heuristic can (at best) give $h(a, b) = h(b, a) = 1$.

As a step toward addressing this vital issue, this chapter will introduce a new family of asymmetric heuristics. This family generalizes the line heuristic optimization of Chapter 5 and, accordingly, this chapter can be viewed as a logical extension of that chapter. While the resulting optimization problem is shown to be NP-hard, the bounds established by Linial *et al.* [37], as summarized in Section 5.4, are extended to cover this new case. Moreover, the alternating maximization technique of Section 5.5.2 remains fully applicable while retaining its convergence and integer embedding properties. We tie these developments down with empirical results on a number of demonstrative asymmetric domains, which show the practical benefits of such an approach and the value in building strong asymmetric heuristics.

6.2 Hinge Heuristics

Hinge heuristics describe a family of asymmetric heuristic functions: a heuristic value between two states is simply the difference between a pair of points on the real line, but is scaled by a constant factor depending on the pair's left/right orientation. In particular, let each state i in a search graph $G = (V, E)$ be associated with a point

$y_i \in \mathbb{R}$ in the vector $\mathbf{y}$. Given a constant "scaling" parameter $\sigma \geq 1$, the heuristic between two states i and j is defined as:

$$h^\sigma(i,j) = \max\{y_i - y_j, (y_j - y_i)/\sigma\} \tag{6.1}$$

The lookup can equivalently be written as:

$$h^\sigma(i,j) = \begin{cases} |y_i - y_j| & \text{if } y_i > y_j \\ |y_i - y_j|/\sigma & \text{else} \end{cases} \tag{6.2}$$

that is, it is simply an absolute value that gets *scaled down* by a factor of σ, depending on whether y_i appears to the left or to the right of y_j on the line.

The variable σ gives hinge heuristics the ability to express varying degrees of asymmetry to best fit a specific target domain. Note that when $\sigma > 1$, the distance between half of the pairs of points will necessarily be scaled down, but this could enable the points in $\mathbf{y}$ to move much farther apart from one-another while remaining admissible. In other words, a hinge heuristic can forfeit representing a small heuristic value for a short path from states b to a so as to better represent a large heuristic value for a long path from a to b. (It is up to the optimizer to make this tradeoff across all pairs of states, negotiating which pairs to best express.)

Consider some examples. If $\sigma = 1$, the heuristic is the absolute distance between two points, and is well-suited to undirected domains; this precisely matches the definition of line heuristics studied in Chapter 5. When σ takes on finite values greater than 1, the heuristics are only "partially" directed, which can befit certain target domains that, likewise, only have "partial" asymmetries in the distances between states (we note partially directed heuristics seem relatively untouched in the literature). Finally, as $\sigma \to \infty$, the heuristic becomes a difference thresholded at 0; this particular lookup appears in the literature in the contexts of "directed differential heuristics" [53], or the individual components of the ALT algorithm [24].

6.3 Optimization Problem

We present a general optimization approach to defining good admissible and consistent hinge heuristics. We assume σ has been defined *a priori* (and otherwise that

a line search will be performed to find a suitable value). As in earlier chapters, the admissibility and consistency constraints and objective are examined in turn.

6.3.1 Constraints

Recall the following define the admissibility and consistency of a hinge heuristic:

$$\forall i, j \quad h^\sigma(i,j) \leq \delta(i,j) \tag{6.3}$$

$$\forall i, j, k \quad h^\sigma(i,j) \leq \delta(i,k) + h^\sigma(j,k) \tag{6.4}$$

where $\delta(i,j)$ is the shortest path distance between i and j. These conditions apply to all pairs and triples in the state space but, as in preceding chapters, we can show that satisfying a subset of them satisfies all of them for any σ.

Theorem 6.1. *A hinge heuristic defined by any vector* $\mathbf{y}$ *and scaling parameter* σ *is globally consistent if and only if it is locally admissible.*

Proof. This proof relies on Lemma 3.1, which showed that any heuristic obeying the triangle inequality is globally consistent if and only if it is locally admissible.

While the heuristic lookup (6.2) relaxes the metric axioms of discernibility and symmetry, it *does* retain the triangle inequality.[1] To show this, (i.e., $h^\sigma(i,j) \leq h^\sigma(i,k) + h^\sigma(k,j)$ for all i, j, k), let y_i, y_j, and y_k be any three points on the line. We will exhaustively enumerate all possible cases. First, if $y_i \leq y_j$:

$$h^\sigma(i,j) = |y_i - y_j|/\sigma \leq |y_i - y_k|/\sigma + |y_k - y_j|/\sigma \tag{6.5}$$

$$\leq h^\sigma(i,k) + h^\sigma(k,j), \tag{6.6}$$

where the inequality on line 6.5 is the triangle inequality on absolute differences. Similarly, if $y_j \leq y_k \leq y_i$ then:

$$h^\sigma(i,j) = |y_i - y_j| \leq |y_i - y_k| + |y_k - y_j| \tag{6.7}$$

$$= h^\sigma(i,k) + h^\sigma(k,j) \tag{6.8}$$

Furthermore, if $y_j \leq y_i \leq y_k$ then:

$$h^\sigma(i,j) = y_i - y_j \leq y_k - y_j = h^\sigma(k,j) \tag{6.9}$$

$$\leq h^\sigma(i,k) + h^\sigma(k,j) \tag{6.10}$$

[1] It also retains non-negativity, yielding what is sometimes called a *quasimetric*.

where the inequality on line 6.9 holds since $y_i \leq y_k$. Using a similar argument, if $y_k \leq y_j \leq y_i$ then:

$$h^\sigma(i,j) = y_i - y_j \leq y_i - y_k = h^\sigma(i,k) \tag{6.11}$$

$$\leq h^\sigma(i,k) + h^\sigma(k,j) \tag{6.12}$$

thus proving, through exhaustive examination, that the triangle inequality holds. By Lemma 3.1, a hinge heuristic is globally consistent if it is locally admissible. □

From the optimizer's standpoint, Theorem 6.1 implies that global admissibility and consistency of h^σ can be achieved simply by local constraints on $\mathbf{y}$:

$$\forall (i,j) \in E \quad h^\sigma(i,j) \leq \delta(i,j) \tag{6.13}$$

By substitution into the heuristic lookup (6.2), this implies two linear constraints for every one of the directed[2] edges in the search graph's set of edges, E:

$$\forall (i,j) \in E \quad y_i - y_j \leq \delta(i,j), \text{ and} \tag{6.14}$$

$$\forall (i,j) \in E \quad y_j - y_i \leq \sigma\, \delta(i,j), \tag{6.15}$$

representing an enormous reduction over the constraints between all pairs and all triples of states originally implied by lines 6.3 and 6.4.[3]

6.3.2 Objective

The objective is once again captured by a scalar loss function, $\mathcal{L} : \mathbf{y} \to \mathbb{R}^+$, which maps all of the errors in the heuristic to a single real value. As in Chapter 5, we will use a weighted sum of the absolute errors in the heuristics values:

$$\mathcal{L}(\mathbf{y}) = \sum_{i,j} W_{ij} \left| \delta(i,j) - h^\sigma(i,j) \right| \tag{6.16}$$

where $\mathbf{W} \in \mathbb{R}^{n \times n}$ is the pairwise weight matrix, a user-specified parameter that can be used to code the relative importance of certain routes and goal (which also facilitates the building of sets of multiple *complementary* heuristics, per Chapter 4).

[2] We assume undirected graphs are coded as directed graphs with symmetric edges everywhere.

[3] Note if $\sigma = 1$, lines 6.14 and 6.15 simply constrain the absolute distance between two points. Conversely, as $\sigma \to \infty$, the second inequality is vacuous and can be omitted from the constraint set.

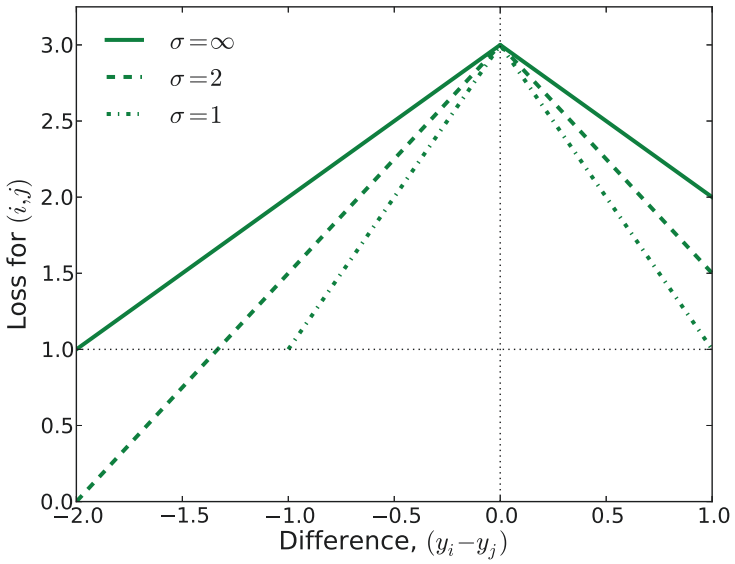


Figure 6.2: Sketch of how loss (i.e., $|\delta(i,j) - h^\sigma(i,j)| + |\delta(j,i) - h^\sigma(j,i)|$) interacts with the difference between two points $y_i - y_j$ when their true distances to each other are $\delta(i,j) = 1$ and $\delta(j,i) = 2$ respectively.

Examples. A sketch of how this loss interacts with the difference between two points, $y_i - y_j$, is shown in Figure 6.2. In this example, we assume the true (asymmetric) distances between the two points are $\delta(i,j) = 1$ and $\delta(j,i) = 2$, and assume the weight on the errors for this pair of points is $W_{ij} = W_{ji} = 1$. Only the domain for which the heuristic is admissible is shown and, for completeness, we also show the loss for different values of σ:

- When $\sigma = 1$, the heuristic lookup (6.2) defines an absolute value of the difference between y_i and y_j. Note there is no arrangement of the two points for which the loss (6.16) can reach 0, and a loss of 1 must be incurred even in the optimal case. The best strategy is to place the points as far apart as admissibly possible so that $h^\sigma(i,j) = h^\sigma(j,i) = 1$, implying a loss of $|h^\sigma(i,j) - \delta(i,j)| + |h^\sigma(j,i) - \delta(j,i)| = |1-1| + |1-2| = 1$.

- As $\sigma \to \infty$, the heuristic lookup defines a difference thresholded at 0. Once again, we are unable to arrange these two points in such a way that the loss incurred is 0. Instead, the best strategy is to express the larger of the two distances (i.e., $\delta(j,i) = 2$) by placing y_i at a distance of 2 units to the left

of y_j so that $h^\sigma(i,j) = 0$ and $h^\sigma(j,i) = 2$, once again implying a loss of $|h^\sigma(i,j) - \delta(i,j)| + |h^\sigma(j,i) - \delta(j,i)| = |0-1| + |2-2| = 1$.

- However, when $\sigma = 2$, there *is* in fact an arrangement of the points y_i and y_j under which the loss incurred reaches 0. In particular, we can place y_i two units to the left of y_j so that $h^\sigma(i,j) = 1$ and $h^\sigma(j,i) = 2$. The loss incurred is $|h^\sigma(i,j) - \delta(i,j)| + |h^\sigma(j,i) - \delta(j,i)| = |1-1| + |2-2| = 0$ – a perfect heuristic in both directions, suggesting the choice of σ can matter.

Simplification. Having established our objective function, we can now work on simplifying it using the same transformation that we have used in previous chapters. Since $\mathbf{y}$ is constrained to be admissible (i.e., $\delta(i,j) \geq h^\sigma(i,j)$), taking an absolute value on line 6.16 is unnecessary, and we can rewrite the loss as:

$$\mathcal{L}(\mathbf{y}) = \sum_{i,j} W_{ij} \left(\delta(i,j) - h^\sigma(i,j)\right) \tag{6.17}$$

$$= \sum_{i,j} W_{ij}\, \delta(i,j) - \sum_{i,j} W_{ij} h^\sigma(i,j) \tag{6.18}$$

Note the first term on line 6.18 does not depend on $\mathbf{y}$, so the optimizer can therefore ignore that term, and instead simply attempt to maximize the second term. Furthermore, observe that for all i, j, each hinge heuristic value $h^\sigma(i,j)$ and its converse $h^\sigma(j,i)$ always sum together (without loss of generality) to the following:

$$h^\sigma(i,j) + h^\sigma(j,i) = |y_i - y_j| + |y_i - y_j|/\sigma = \left(1 + \tfrac{1}{\sigma}\right) |y_i - y_j| \tag{6.19}$$

Thus the weighted sum of heuristics over *all* such pairs can be equivalently written:

$$\sum_{i,j} W_{ij} h^\sigma(i,j) = \sum_{i,j} \left(1 + \tfrac{1}{\sigma}\right) W_{ij} |y_i - y_j| \tag{6.20}$$

$$= \left(1 + \tfrac{1}{\sigma}\right) \sum_{i,j} W_{ij} |y_i - y_j| \tag{6.21}$$

From the optimizer's standpoint, the multiplicative term $\left(1 + \frac{1}{\sigma}\right)$ is just a constant that can also be safely ignored. Combining the constraints (6.14, 6.15) and this objective yields the following optimization problem, which is linear in its constraints

but nonlinear in its objective:

$$\begin{aligned} \underset{\mathbf{y}}{\textbf{maximize}} \quad & \sum_{i,j} W_{ij}|y_i - y_j| \\ \textbf{subject to} \quad & \forall (i,j) \in E \quad y_i - y_j \leq \delta(i,j) \\ & \forall (i,j) \in E \quad y_j - y_i \leq \delta(i,j)\sigma \end{aligned} \tag{6.22}$$

Fortunately, this is *exactly* the optimization objective given for the line heuristic optimization of Chapter 5. The local search procedure described in Section 5.5 can therefore be applied without modification, implying any given hinge heuristic can be incrementally improved via a terminating succession of linear programs until it is locally optimal. The approach inherits the propensity for integer embeddings (Theorem 5.4), its convergence and termination properties (Theorems 5.5 and 5.6) and, unfortunately, the following hardness result.

Theorem 6.2. *The optimization problem* (6.22) *is NP-hard.*

Proof. Let G be an arbitrary, *undirected* search graph, and let $\sigma = 1$. The optimization problem on line (6.22) is now the line heuristic optimization problem studied in Chapter 5; by Theorem 5.2, this problem is NP-hard. But this describes a special case of the optimization (6.22), and so that problem must also be NP-hard. □

Despite Theorem 6.2, it is indeed possible to generate bounded, *approximate* solutions to our general optimization problem (6.22).

6.4 Bounded Solutions

In general, the implication of Theorem 6.2 is that we will generally require time exponential in the size of the input search graph to find an optimal solution to the problem (6.22). However, in this section we show it is still possible to design a simple and efficient algorithm to generate solutions with a modest probabilistic bound on their quality. The following result builds upon the work of Bourgain [6] and Linial *et al.* [37] (first summarized in Section 5.4), and begins by extending standard Lipschitz embeddings with a notion of *direction*.

6.4.1 Directed Lipschitz Embeddings

Our bound will rely on a new kind of embedding that can be used to construct admissible and consistent hinge heuristics. A *directed* Lipschitz embedding is a basic extension of a standard Lipschitz embedding[4] to the case of directed graphs. Much like standard Lipschitz embeddings, a directed Lipschitz embedding is defined in terms of the true shortest path distances from each state to the nearest state in a reference set. But in this case, the meaning of the term "nearest" takes on additional semantics. In particular, for a given reference set R, two unique (and possibly very different) directed Lipschitz embeddings can be defined:

1. Each point y_i is defined as the true distance from state i *to* the reference set,

$$y_i = \delta(i, R) = \min_{j \in R} \delta(i, j). \tag{6.23}$$

2. Each point y_i is defined as the true distance *from* the reference set to i,

$$y_i = -\delta(R, i) = -\min_{j \in R} \delta(j, i). \tag{6.24}$$

Note the negation of the values (6.24), which will be necessary to prove admissibility and consistency in the resulting heuristic values.

Once such an embedding $\mathbf{y}$ has been built using one (and only one) of the above rules, a heuristic between two points can be defined by the hinge distance (6.2). For certain values of σ, these resulting heuristics are also admissible and therefore consistent. Before proving this statement, we provide a simple lemma.

Lemma 6.1. *For any two points i and j in an embedding $\mathbf{y}$ of the directed graph $G = (V, E)$, and a reference set $R \subseteq V$, the following inequality holds:*

$$\delta(R, i) - \delta(R, j) \leq \delta(j, i) \tag{6.25}$$

Proof. Let $q = \arg\min_{q \in R} \delta(q, i)$:

$$\delta(R, i) - \delta(R, j) = \delta(q, i) - \delta(R, j) \tag{6.26}$$

$$\leq \delta(q, i) - \delta(q, j) \tag{6.27}$$

$$\leq \delta(j, i) \tag{6.28}$$

[4]Refer to Section 5.4.1 for an introduction to Lipschitz embeddings, with a detailed example.

where the inequality on line 6.27 arises since $\delta(R, j) \leq \delta(q, j)$ (by definition 6.24), and the inequality on line 6.28 is simply due to the basic form of the triangle inequality applied here to the true shortest path distances in G. $\square$

(Note that Lemma 6.1 is similar to, but not the same as, the result in Lemma 5.1. We will use both of these Lemmas to construct a proof for the following theorem.)

Theorem 6.3. *Let* $\mathbf{y}$ *be a directed Lipschitz embedding of a search graph* $G = (V, E)$ *defined using either rule* (6.23) *or rule* (6.24)*. If* $\sigma \geq \delta(i, j)$ *for any pair of states* i, j*, then* $\mathbf{y}$ *defines an admissible and consistent hinge heuristic over* G*.*

Proof. Lemma 3.1 states any heuristic obeying the triangle inequality is consistent as long as it is admissible; Theorem 6.1 established that the triangle inequality holds for hinge heuristics, and so it remains to show that $\mathbf{y}$ is admissible for any states i and j and any reference set R. We consider each rule (6.23 and 6.24) in turn:

1. Suppose the rule on line 6.23 is used and $y_i = \delta(i, R)$. There are now two independent possibilities. In the first, $y_i \geq y_j$, yielding:

$$h^\sigma(i, j) = |y_i - y_j| = (y_i - y_j) \tag{6.29}$$
$$= \delta(i, R) - \delta(j, R) \tag{6.30}$$
$$\leq \delta(i, j) \tag{6.31}$$

   where line 6.29 is a result of the definition of the heurustic lookup (6.2), and line 6.31 is due to the "generalized" triangle inequality proved in Lemma 5.1. Otherwise $y_i < y_j$, which gives:

$$h^\sigma(i, j) = |y_i - y_j|/\sigma = (y_j - y_i)/\sigma \tag{6.32}$$
$$= (\delta(j, R) - \delta(i, R))/\sigma \tag{6.33}$$
$$\leq \delta(j, i)/\sigma \tag{6.34}$$
$$\leq \delta(i, j) \tag{6.35}$$

   where line 6.32 follows the heuristic lookup (6.2), line 6.34 is due to Lemma 5.1, and line 6.35 is due to our working constraint that $\sigma \geq \delta(j, i)/\ \delta(i, j)$.

2. Next, suppose the rule on line 6.24 is used and $y_i = -\delta(R, i)$. Again, there are two independent possibilities. In the first, $y_i \geq y_j$, yielding:

$$h^\sigma(i,j) = |y_i - y_j| = (y_i - y_j) \tag{6.36}$$
$$= (-\delta(R,i)) - (-\delta(R,j)) \tag{6.37}$$
$$= \delta(R,j) - \delta(R,i) \tag{6.38}$$
$$\leq \delta(i,j) \tag{6.39}$$

where line 6.36 is the heurustic lookup (6.2), and line 6.39 is due to the inequality proved in Lemma 6.1. Otherwise $y_i < y_j$, which gives:

$$h^\sigma(i,j) = |y_i - y_j|/\sigma = (y_j - y_i)/\sigma \tag{6.40}$$
$$= ((-\delta(R,j)) - (-\delta(R,i)))/\sigma \tag{6.41}$$
$$= (\delta(R,i) - \delta(R,j))/\sigma \tag{6.42}$$
$$\leq \delta(j,i)/\sigma \tag{6.43}$$
$$\leq \delta(i,j) \tag{6.44}$$

where line 6.40 follows the heuristic lookup (6.2), line 6.43 once again follows from Lemma 6.1, and line 6.44 is a result of our constraint on σ (i.e., that $\sigma \geq \delta(j,i)/\,\delta(i,j)$ for any i, j).

The preceding exhaustively enumerate all possible cases, and so $\mathbf{y}$ must define an admissible and – as a result of Lemma 3.1 – globally consistent hinge heuristic. □

6.4.2 Probabilistic Approach Extending Bourgain/Linial *et al.*

Recall in Section 5.4.2 where we described how Linial *et al.* [37], following work by Bourgain [6], proved that the pairwise distances in a specific Lipschitz embedding can be expected to exceed a logarithmic bound. In this section we show how this result can be extended to cover the case of *directed* Lipschitz embeddings.

Let R be a reference set selected from among 2^k vertices chosen uniformly at random, where k is chosen from $[1, 2, \ldots, \log n]$ inclusive.[5] Next, define a *directed*

[5] This is the same random sampling method used in Chapter 5's Algorithm 5.1.

Algorithm 6.1: Returns a hinge heuristic with bounded average distortion.

Input: A directed search graph $G = (V, E)$.
Output: $\mathbf{y} \in \mathbb{R}^n$, an admissible and consistent hinge heuristic.

```
1  q ← randInt(1, log n);
2  R ← randSubset(V, q);
3  if randInt(0,1) = 0 then
4      // Compute distances to the reference set
5      for i from 1 to |V| do
6          y_i ← min_{j∈R} δ(i, j);
7  else
8      // Compute distances from the reference set
9      for i from 1 to |V| do
10         y_i ← − min_{j∈R} δ(j, i);
11 return y;
```

Lipschitz embedding $\mathbf{y}$ from R, using either the rule on line 6.23 or 6.24 *with equal probability*. This simple procedure is formalized in Algorithm 6.1, and the following proposition proves a probabilistic bound on the resulting hinge heuristics:

Theorem 6.4. *Algorithm 6.1 generates a hinge heuristic* $\mathbf{y}$ *which is expected to be within a logarithmic factor of optimal, or more specifically:*

$$\mathbb{E}\left[\sum_{i,j} h^{\sigma}(i,j)\right] \geq \frac{1+\sigma}{\sigma} \sum_{i,j} \frac{\delta(i,j)}{80 \log n} \tag{6.45}$$

Proof. Let $R \subseteq V$ be a randomly chosen reference set of 2^k vertices, and $k \in \mathbb{Z}$ chosen uniformly at random from $[1, 2, \ldots, \log n]$. Let $\mathbf{y} \in \mathbb{R}^n$ be a directed Lipschitz embedding of G, defined with *equal probability* using either (6.23) or (6.24). This $\mathbf{y}$ has effectively been generated by Algorithm 6.1. Moreover, if we assume $\sigma \geq \delta(i,j)$, then $\mathbf{y}$ is an admissible and consistent hinge heuristic by Theorem 6.3.

Next, define a new graph G' as the "symmetrization" of G as follows:

$$\beta(i,j) = \beta(j,i) = \tfrac{1}{2}(\delta(i,j) + \delta(j,i)) \tag{6.46}$$

Under this definition, (β, V) now defines a finite metric [42]: the pairwise distances between the points in G' are symmetric, non-negative, and obey the triangle

inequality. Still, the *sum* of pairwise distances across G' is the same as across G:

$$\sum_{i,j} \delta(i,j) = \sum_{i,j} \tfrac{1}{2}(\delta(i,j) + \delta(j,i)) \tag{6.47}$$

$$= \sum_{i,j} \beta(i,j) \tag{6.48}$$

Now let $\mathbf{y}'$ be a standard Lipschitz embedding of G' also based on reference set R:

$$y'_i = \beta(i,R) \tag{6.49}$$

Note the embedding $\mathbf{y}'$ has effectively been generated by Algorithm 5.1.

We approach the proof by relating $\mathbf{y}'$ (which we have bounds on) to $\mathbf{y}$. In particular, to prove the bound (6.45), note for any $i, j \in V$ we can write the expected sum of the heuristic values in both directions, $\mathrm{E}\left[h^\sigma(i,j) + h^\sigma(j,i)\right]$, as follows:

$$\mathrm{E}\left[\max\left\{y_i - y_j, \frac{y_j - y_i}{\sigma}\right\} + \max\left\{y_j - y_i, \frac{y_i - y_j}{\sigma}\right\}\right] \tag{6.50}$$

Since at most only one of $(y_i - y_j)$ and $(y_j - y_i)$ can be greater than zero, this expression can be rewritten equivalently and simplified into the following:

$$\mathrm{E}\left[\max\left\{y_i - y_j + \frac{y_i - y_j}{\sigma}, y_j - y_i + \frac{y_j - y_i}{\sigma}\right\}\right]$$

$$= \frac{1+\sigma}{\sigma}\,\mathrm{E}\left[\max\{y_i - y_j, y_j - y_i\}\right] \tag{6.51}$$

$$\frac{1+\sigma}{\sigma}\,\mathrm{E}\left[|y_i - y_j|\right] \tag{6.52}$$

We further make the computation of this expected value explicit (6.53), express it as a sum of halves (6.54), and lower bound it using the triangle inequality (6.55):

$$\frac{1+\sigma}{\sigma}\left(\tfrac{1}{2}|\,\delta(i,R) - \delta(j,R)| + \tfrac{1}{2}|\,\delta(R,i) - \delta(R,j)|\right) \tag{6.53}$$

$$= \frac{1+\sigma}{2\sigma}\left(\tfrac{1}{2}\,|\delta(i,R) - \delta(j,R)| + \tfrac{1}{2}\,|\delta(R,i) - \delta(R,j)|\right)$$

$$+ \frac{1+\sigma}{2\sigma}\left(\tfrac{1}{2}\,|\delta(j,R) - \delta(i,R)| + \tfrac{1}{2}\,|\delta(R,j) - \delta(R,i)|\right) \tag{6.54}$$

$$\geq \frac{1+\sigma}{2\sigma}\left(\tfrac{1}{2}\,|\delta(i,R) + \delta(R,i) - \delta(j,R) - \delta(R,j)|\right)$$

$$+ \frac{1+\sigma}{2\sigma}\left(\tfrac{1}{2}\,|\delta(j,R) + \delta(R,j) - \delta(i,R) - \delta(R,i)|\right) \tag{6.55}$$

Using the definitions of β (6.46) and $\mathbf{y}'$ (6.49) allows line 6.55 to be written as:

$$\frac{1+\sigma}{2\sigma}\left|\beta(i,R)-\beta(j,R)\right|+\frac{1+\sigma}{2\sigma}\left|\beta(j,R)-\beta(i,R)\right| \tag{6.56}$$

$$=\frac{1+\sigma}{2\sigma}\left|y'_i-y'_j\right|+\frac{1+\sigma}{2\sigma}\left|y'_j-y'_i\right| \tag{6.57}$$

$$=\frac{1+\sigma}{\sigma}\left|y'_i-y'_j\right| \tag{6.58}$$

Altogether implying the following holds:

$$\mathrm{E}\left[h^\sigma(i,j)+h^\sigma(j,i)\right]\geq\frac{1+\sigma}{\sigma}\ \left|y'_i-y'_j\right| \tag{6.59}$$

$$\frac{1}{2}\sum_{i,j}\mathrm{E}\left[(h^\sigma(i,j)+h^\sigma(j,i))\right]\geq\frac{1+\sigma}{2\sigma}\ \sum_{i,j}\left|y'_i-y'_j\right| \tag{6.60}$$

And therefore:

$$\mathrm{E}\left[\sum_{i,j}h^\sigma(i,j)\right]\geq\frac{1+\sigma}{2\sigma}\ \mathrm{E}\left[\sum_{i,j}\left|y'_i-y'_j\right|\right] \tag{6.61}$$

$$\geq\frac{1+\sigma}{2\sigma}\sum_{i,j}\frac{\delta(i,j)}{40\log n} \tag{6.62}$$

$$=\frac{1+\sigma}{\sigma}\sum_{i,j}\frac{\delta(i,j)}{80\log n} \tag{6.63}$$

where the inequality on line 6.62 is because G' is a *symmetric* graph and $\mathbf{y}'$ was effectively generated by Algorithm 5.1, implying Linial *et al.*'s classical bound [37] applies; thus completing the proof by proving the proposed bound (6.45). □

6.5 Evaluation

The forthcoming evaluation compares our newly derived Hinge Heuristic optimization to a number of competitors, on three directed search domains. As in preceding chapters, the performance metrics under consideration are *a)* an average count of the number of nodes the A* search algorithm [28] expands, bucketed by solution length, and *b)* the total CPU runtime required by A* to complete all problems, which includes reconstructing the action sequence.

The first of these asymmetric domains is a synthetic *Web Graph* domain, which is a connected digraph of webpages based on real-world network data. The next is

a *Terrain* domain, which is a procedurally-generated landscape comprised of cliffs and valleys, and wherein physically-inspired rules are used to dictate the action costs (e.g., travelling uphill is more expensive than travelling downhill). The third domain is the *Platformer* domain, a video-game inspired two-dimensional grid in which the agent moves left, right, up, and down, using platforms and ladders to negotiate with an artificial gravity and reach various goal locations.

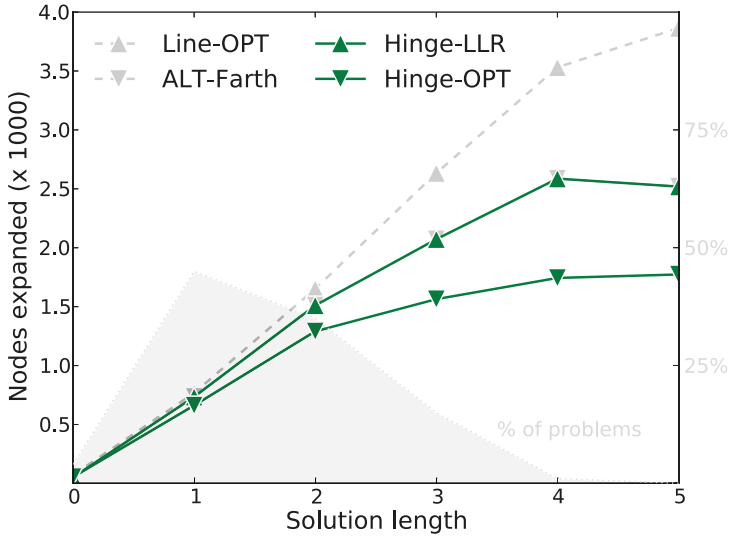
For each of these three domains, we consider four different approaches to constructing directed and undirected search heuristics. First, **Line-OPT** is an *undirected*, locally optimal line heuristic, generated as described in Chapter 5. It is defined on an undirected version of the digraph in which each directed edge of cost $\delta_G(i, j)$ is replaced by an undirected edge of cost $\min\{\delta_G(i, j), \delta_G(j, i)\}$, which yields consistent undirected heuristics for the original directed graph. This approach outperforms all other types of undirected heuristics in the same amount of memory, and so can be seen as a representative of the "undirected" approach. Second, **ALT-Farth** may be thought of as two *directed* differential heuristics placed using the *Farthest* algorithm [24, 53]. Note that this heuristic uses *double* the memory of any line or hinge heuristic, but still serves as a useful baseline for comparison. Third, **Hinge-LLR** is the *directed* probabilistic method described in Section 6.4. Because the generating algorithm is stochastic, ten instances are generated and the results of the best (in terms of nodes expanded) are reported (we report the best, and not the average, establish a more aggressive baseline). Fourth and finally, **Hinge-OPT** is a *directed* locally optimal hinge heuristic, seeded using the corresponding Hinge-LLR heuristic, and improved until it is at least locally optimal by way of the search method described in Section 5.5.2, which is run until termination.

6.5.1 Computer Science Web Graph

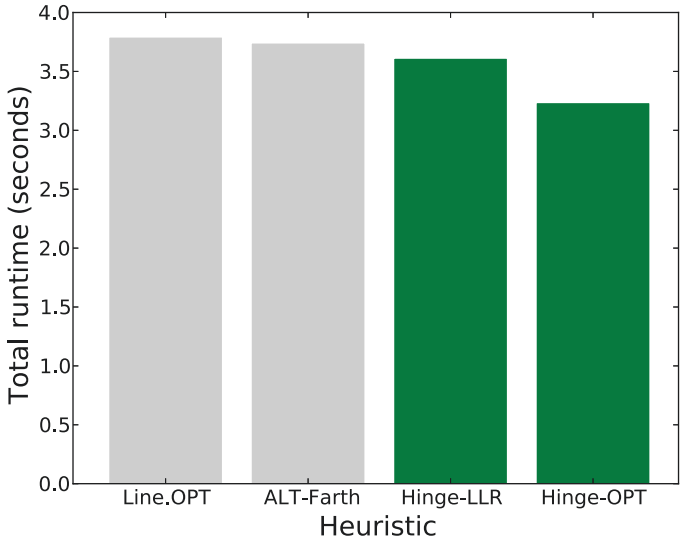
The *Web Graph* domain is a synthetic but demonstrative domain, where the search graph's nodes represent web documents sampled from the WebKB dataset.[6]

This directed generated by deploying a web crawler to recursively follow the

[6]WebKB is a publically available snapshot of the interconnected web networks of four American universities including departmental pages, faculty pages, student pages, course pages, etc.



(a) Bucketed average node expansions.



(b) A* total CPU runtime.

Figure 6.3: A* results across 10,000 randomly chosen problems in the *Web Graph* domain. Figure 6.3a shows average node expansions as well as the distribution of the different problem lengths; Figure 6.3b shows runtime totals for each heuristic.

links on every page it encounters, using the index pages for the four computer science departments as start seeds (Cornell University, the University of Texas, the University of Washington, and the University of Wisconsin-Madison). This yields a total of 4,933 pages, with each page becoming a node. Next, the web links connecting the pages become unit-cost directed edges from the originating page to the destination page; meanwhile, weighted edges in the *backward* direction are added if no such link already exists with a cost[7] of 3. Having defined the directed search, our agent is tasked with determining "relatedness" scores between 10,000 random pairs of documents: here, relatedness is defined as the minimum number of hops (i.e., links or backlinks) required to reach the second document from the first. The tally of problem lengths shown in Figure 6.3a reveals that most of these documents are highly related, and none are greater than a distance of five away from each other. In this domain, it is most congruent to set $\sigma = 3$ for the hinge heuristics, since back-edges cost three times as much as forward edges.

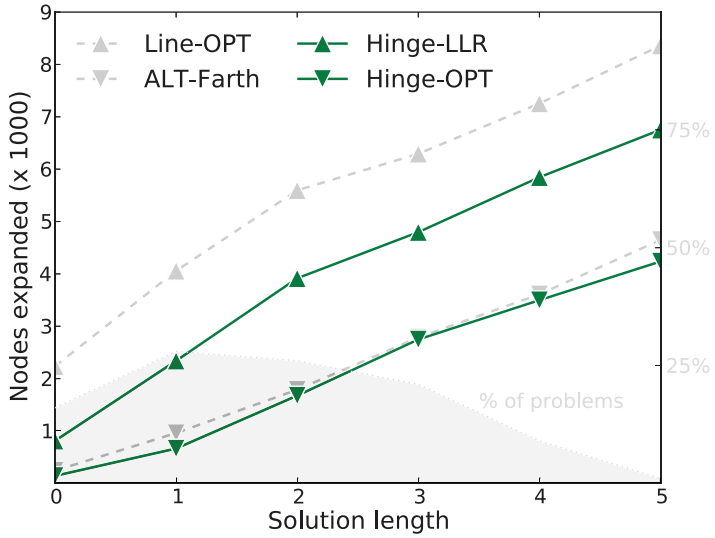
Results comparing the four competing heuristics are tallied in Figure 6.3. While locally optimal, the *undirected* line heuristic optimization of Chapter 5 is least-suited to this directed domain. Meanwhile, the ALT-Farth and Hinge-LLR heuristics perform on par with each other (however, it is worth noting that the latter is using *half* the memory of the former). Lastly, running our local optimization on the hinge heuristic (yielding the heuristic labelled Hinge-OPT) yields significant gains over the baselines, which is reflected in a modest CPU-time speedup as well.

6.5.2 Terrain Navigation

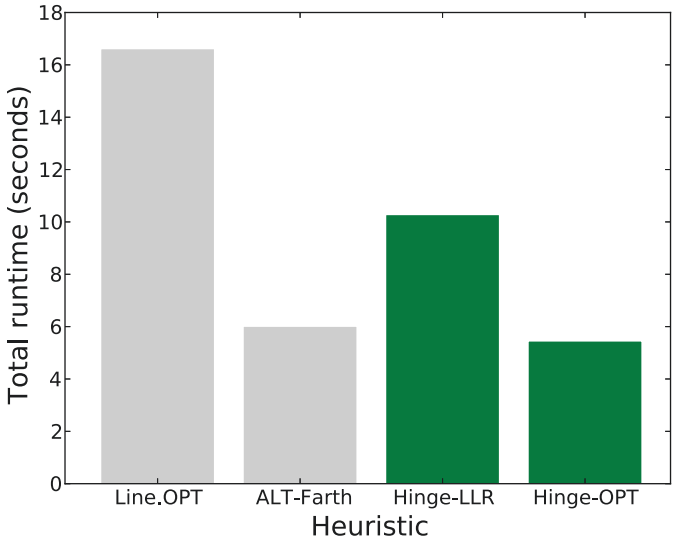
The *Terrain* domain is a state space defined over a 100m by 100m tract of terrain, procedurally generated using the Diamond-Square algorithm [20] followed by ten passes of local averaging (i.e., for each state, its height is redefined as the average of its current height and its neighbor's heights).

An illustration and contour map of this space is shown in Figure 6.1. States are defined on every square meter, and the edges between vertices are defined in both

[7]Note the directed nature of this graph: while a link may exist from one page to another, the latter may contain no link to the former. Also note that increasing the relative cost of these backward edges can serve to dramaticize the importance of using directed heuristics.



(a) Bucketed average node expansions.



(b) A* total CPU runtime.

Figure 6.4: A* pathfinding results across 10,000 randomly chosen problems in the *Terrain* domain. Figure 6.4a shows average node expansions and the distribution of the problem lengths; Figure 6.4b shows runtime totals for each.

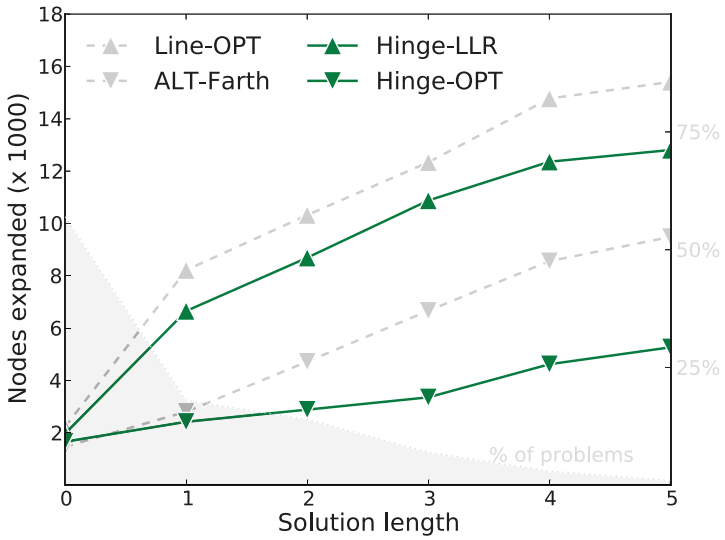
directions along cardinals and diagonals, essentially yielding an octile grid-world with topographic features. The cost of an edge is defined as the *work* (in Joules) to move a mass of one kilogram between the two points in an idealized physical simulation (there is no friction, the agent's one kilogram mass is assumed to be moving at fixed velocity, and braking to maintain a constant velocity on a downhill slope has no cost). Because of the asymmetry between the work to go downhill and uphill, a symmetric admissible heuristic cannot capture any distance information at all (travelling downhill is always free). But the flexibility afforded by a hinge heuristic enables us to make headway on representing the distances in this space: in this domain, we achieved the best results by setting $\sigma = \infty$ for the hinge heuristics (that is, the hinge heuristics we evaluate here are fully one-directional).

Results are shown in Figure 6.4. In general, we see that using one-sided heuristics in this domain leads to significantly fewer nodes expanded during search: while locally optimal, the *undirected* line heuristic optimization of Chapter 5 cannot compete, and is more than twice as expensive (both in terms of node expansions and CPU-time) than the best methods. Here, the ALT-Farth heuristics outcompete Hinge-LLR, but the result of our local optimization (Hinge-OPT) is significant gains over all the baselines, including ALT-Farth (which, it is worth emphasizing again, uses twice the memory).

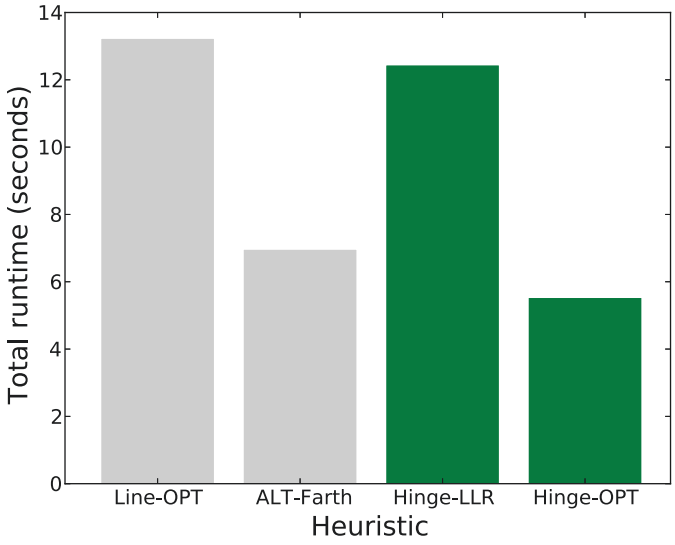
6.5.3 Platformer

The *Platformer* domain is a video-game inspired domain. The agent moves on a two-dimensional grid with specific motion laws, and must determine paths between start and goal states which minimize the number of elapsed timesteps.

Two simple examples of this domain are shown in Figure 6.6, and the particular environment we deploy the agent into is the randomly generated 200×200 platformer depicted in Figure 6.6, right. A basic set of rules dictate the agent's motion: it can walk left and right on platforms, and it can ascend and descend ladders. Note these kinds of actions only permit cardinal movement of the agent: left, right, up, and down. However, if the agent should move off the edge of a platform and into a mid-air tile, it begins a downward descent moving down by one row each timestep,



(a) Bucketed average node expansions.



(b) A* total CPU runtime.

Figure 6.5: A* pathfinding results across 10,000 randomly chosen problems in the *Platformer* domain. Figure 6.5a shows average node expansions and the distribution of the different problem lengths; Figure 6.5b shows runtime totals for each heuristic.

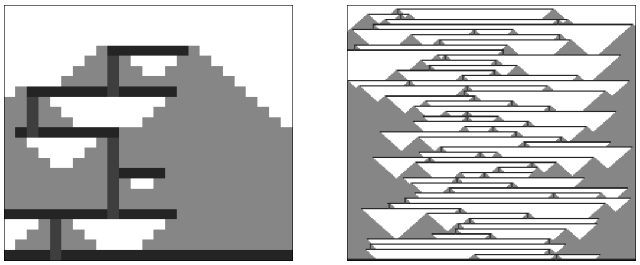


Figure 6.6: *Platformer* examples. Dark cells are platforms/ladders, grey are mid-air, and white are out of reach. The platformer on the right is used in the experiments.

until it lands on the ground, on a platform, or touches a ladder. As it descends, it can optionally control the direction of its descent to the left and right, effectively achieving off-cardinal, diagonal movement. This domain features a great deal of asymmetry in the pairwise distances between states, since the agent is able to descend (even diagonally) through mid-air states, but cannot ascend through them; accordingly, we find good results by setting $\sigma = 10$ for our hinge heuristics.

Results comparing the four competing heuristics in terms of node expansions and CPU-time are tallied (Figure 6.5). The locally optimal *undirected* line heuristic optimization of Chapter 5 fares the worst of the four methods; it is more than twice as expensive (both in terms of node expansions and CPU-time) than the locally optimal hinge heuristic (Hinge-OPT, which fares the best of the four). Meanwhile, as with the *Terrain* domain, the ALT-Farth heuristics outcompete Hinge-LLR, but we reiterate, once again, that the former require twice the memory of the former.

6.6 Summary

This chapter introduced hinge heuristics: an optimization-driven, directed approach to defining good heuristics. The approach we have proposed has three defining features. It uses a one-sided "hinge" lookup with a user-definable parameter σ that can be tuned to specific target domains; it has a probabilistic lower bound on the average distortion, the likeness of which pivots off of work by Bourgain and Linial *et al.*; and with some careful rephrasing of the objective, it accommodates an efficient local search which can be used to improve hinge heuristics until they are at least locally optimal. Accompanying numerical experiments on three different

domains showed that this method is effective at representing asymmetric distances, suggesting overall that local search and the optimization interpretation continues to be a vital part of generating good heuristics.

Thus concludes our use of numerical optimization for heuristic generation. In the next chapter, we will turn to combinatorial optimization, examining the problem of how to best choose *subsets* of heuristics when many are available.

Chapter 7

Subset Selection of Heuristics
via Submodular Maximization

This chapter presents a study of the problem of choosing a subset of a larger set of heuristics in the face of memory or computational constraints. While following a similar methodology, this topic can be viewed as complementary to those discussed in earlier chapters, and is based on work published in 2013 [53].

7.1 Motivation

In addition to the methods introduced in the previous chapters, the literature describes many other approaches to heuristic construction. Examples include pattern databases [13], a variety of memory-based and true-distance heuristics [56], and regressors [18]. Each of these is capable of generating a number of different heuristic functions based on input parameters, and can be combined by taking a maximum. But if a limit is placed on the number of heuristics one can combine (due to memory or computational requirements), then a subset selection problem arises.

This chapter presents one way in which such a subset selection problem can be viewed as an optimization problem. Loss will once again be defined as a weighted

Representation/data structure	A *subset* of heuristics ($H \subseteq C$)
Heuristic lookup	$h^H(i,j) = \max_{h_x \in H} h_x(i,j)$
Optimization method(s)	Greedy search
Solution optimality	Approximate (within 0.63 of optimal)

Table 7.1: Key characteristics of the Euclidean Heuristic framework.

sum of the differences between the true distances and the heuristic estimates. When the candidate heuristics are admissible, this loss can be translated into a submodular and monotonic *utility function*, implying that greedy selection is near-optimal. A *sample* utility function will also be proposed, under which greedy selection retains provable optimality guarantees, if the heuristics are consistent.

An empirical evaluation of this approach will show that it can outperform existing selection methods. It will also be shown to be useful for accurately comparing the use of directed and undirected heuristics, lending insight into the problem of constructing heuristics for highly directed domains.

7.2 Heuristic Subset Selection

Heuristic subset selection entails choosing a "good" subset H of a larger set of candidate heuristics C:

$$H \subseteq C = \{h_1, \ldots, h_{|C|}\} \tag{7.1}$$

It is assumed that the heuristics in H are to be combined with a set of default heuristics D by maximizing over the values given across both H and D. For states i and j, this heuristic lookup is denoted:

$$h^H(i,j) = \max_{h_x \in H \cup D} h_x(i,j) \tag{7.2}$$

In the simplest case, D contains only the zero heuristic, which gives 0 for any pair of states queried. It is further assumed that any default or candidate heuristic $h_x \in D \cup C$ is admissible (i.e., never overestimating):

$$\forall i,j, \quad h_x(i,j) \leq \delta(i,j) \tag{7.3}$$

Here, $\delta(i,j)$ is the true distance between states i and j, and thus the combined heuristic lookup h^H is guaranteed to be admissible.

7.3 Optimization Problem

Choosing the heuristic subset H can be viewed as an optimization problem:

$$\begin{aligned} \underset{H \in 2^C}{\textbf{minimize}} \quad & \mathcal{L}(H) \\ \textbf{subject to} \quad & |H| = d \end{aligned} \tag{7.4}$$

The constraint $|H| = d$ simply limits the capacity of H to a fixed-size subset of C, and the loss $\mathcal{L}(H)$ is a scalar quantity summarizing the quality of a given subset H.

7.3.1 Objective

The following scalar function is proposed to summarize the errors in H:

$$\mathcal{L}(H) = \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} \left| \delta(i,j) - h^H(i,j) \right| \tag{7.5}$$

That is, a weighted sum of errors between the resulting heuristic values and the true distances. As in previous chapters, the non-negative weight matrix $W \in \mathbb{R}^{n \times n}$ is a free parameter that can be used to define the relative importance of each pair of states, perhaps based on knowledge of frequent start and goal locations.

This loss (7.5) can be rewritten as a *utility* function $\mathcal{U}$. First, all of the heuristics in $H \cup D$ are admissible, so for all states i and j, $h^H(i,j) \leq \delta(i,j)$. Therefore the absolute value is unnecessary and the sum can be split as follows:

$$\mathcal{L}(H) = \sum_{i,j} W_{ij} \left(\delta(i,j) - h^H(i,j) \right) \tag{7.6}$$

$$= \sum_{i,j} W_{ij} \ \delta(i,j) - \sum_{i,j} W_{ij} \, h^H(i,j) \tag{7.7}$$

The leftmost term on line 7.7 does not depend on H, so minimizing $\mathcal{L}(H)$ is equivalent to maximizing the term on the right. A corresponding utility function is

$$\mathcal{U}(H) = \sum_{i,j} W_{ij} \, h^H(i,j) - \alpha, \tag{7.8}$$

where α is a normalizing constant that has *no effect* on the choice of H, but will help to clarify some of the forthcoming analysis by ensuring that $\mathcal{U}(\emptyset) = 0$. In

particular, α is a weighted sum of the contributions of the default heuristics:

$$\alpha = \sum_{i,j} W_{ij}\, h^{\emptyset}(i,j) \tag{7.9}$$

$$= \sum_{i,j} W_{ij} \max_{h_d \in D} h_d(i,j) \tag{7.10}$$

In summary, a specific utility maximization problem is faced:

$$\begin{aligned} \underset{H \in 2^C}{\textbf{maximize}} \quad & \mathcal{U}(H) \\ \textbf{subject to} \quad & |H| = d \end{aligned} \tag{7.11}$$

7.3.2 Hardness

Unfortunately, solutions to (7.11) are unlikely to be efficiently computable.

Theorem 7.1. *The optimization problem* (7.11) *is NP-hard.*[1]

Proof. A simple reduction is from the NP-complete *Vertex Cover* problem over an undirected graph (V, E). This is the problem of finding a subset of d graph vertices $T \subseteq V$ such that all edges in E are incident to at least one vertex in T.

By definition, a heuristic is a scalar function over all pairs of vertices in a (search) graph. A special case of such a function is one that only gives 1 between a vertex and its neighbors, and 0 for any other input. To reduce vertex cover, the default set of heuristics D can be defined as containing only the zero heuristic, and the candidate set C can be defined as the set $C = \{h_v : v \in V\}$ where $h_v \in C$ gives a value of 1 between vertex v and its neighbors, and 0 otherwise, i.e.:

$$h_v(i,j) = \mathbf{1}((i,j) \in E \wedge (i = v \vee j = v)) \tag{7.12}$$

where $\mathbf{1}$ is the indicator function, giving 1 when its argument is true and 0 otherwise. This implies a maximum attainable utility (7.8) of $2|E|$, corresponding to having heuristic values of 1 between every vertex and its neighbors. But if a subset of d heuristics from C captures $2|E|$ utility, then d vertices must be incident to all $|E|$ edges in the search graph, and therefore there is a vertex cover of size d as well. □

[1] A distinct result is the NP-hardness of ALT preprocessing under an edge-covering objective [3].

7.4 Approach

Despite Theorem 7.1, *greedy selection* yields a solution to the optimization problem (7.11) with a near-optimality guarantee since $\mathcal{U}$ is submodular and monotonic.

7.4.1 Submodularity

Submodularity is a diminishing returns property, aptly describing settings where marginal gains in utility fall away as more elements are added to a set. Let $A \subseteq B \subsetneq S$, let $x \in S \setminus B$, and let ϕ be a function over 2^S. ϕ is submodular if:

$$\phi(A \cup \{x\}) - \phi(A) \geq \phi(B \cup \{x\}) - \phi(B) \tag{7.13}$$

That is, the same element newly added to a subset and its superset will lead the subset to gain at least as much in value as the superset.

Lemma 7.1. *$\mathcal{U}$ is submodular.*

Proof. Let $A \subseteq B$ be sets of heuristics, and let $h_c \in C$ be a particular but arbitrary candidate heuristic function which is in neither A nor B (i.e., $h_c \in C \setminus B$). The inequality on line 7.13 can be reproduced as follows:

$$\begin{aligned}
\mathcal{U}(A \cup \{h_c\}) - \mathcal{U}(A) &= \sum_{i,j} W_{ij}\, h^{A \cup \{h_c\}}(i,j) - \sum_{i,j} W_{ij}\, h^{A}(i,j) && (7.14)\\
&= \sum_{i,j} W_{ij}\, (h^{A \cup \{h_c\}}(i,j) - h^{A}(i,j)) && (7.15)\\
&= \sum_{i,j} W_{ij} (h_c(i,j) - h^{A}(i,j))^{+} && (7.16)\\
&\geq \sum_{i,j} W_{ij} (h_c(i,j) - h^{A \cup B}(i,j))^{+} && (7.17)\\
&= \mathcal{U}(B \cup \{h_c\}) - \mathcal{U}(B) && (7.18)
\end{aligned}$$

Line 7.14 twice expands the definition of utility (7.8) with the α terms cancelling, and line 7.15 rewrites this difference of sums as a sum of differences between h_c and h^A together versus h^A alone. Line 7.16 equates this to a sum of the *positive* differences between h_c and h^A, where $(x)^+ = \max\{0, x\}$. Line 7.17 holds since h_c's individual gains over $h^{A \cup B}$ cannot exceed its gains over h^A. But $A \cup B = B$, altogether proving submodularity. □

7.4.2 Monotonicity

Monotonicity implies that adding an element to a set never leads to a decrease in value. $\mathcal{U}(H)$ is a weighted sum of maximal heuristic values, suggesting monotonicity holds. More formally, let $A \subseteq B \subseteq S$ be sets and let ϕ be a function over 2^S. ϕ is monotonic if $\phi(A) \leq \phi(B)$.

Lemma 7.2. *$\mathcal{U}$ is monotonic.*

Proof. Let $A \subseteq B$ be sets of heuristics. It must be shown that $\mathcal{U}(A) \leq \mathcal{U}(B)$.

$$\mathcal{U}(A) = \sum_{i,j} W_{ij}\, h^A(i,j) - \alpha \tag{7.19}$$

$$\leq \sum_{i,j} W_{ij}\, h^{A \cup B}(i,j) - \alpha \tag{7.20}$$

$$= \sum_{i,j} W_{ij}\, h^B(i,j) - \alpha = \mathcal{U}(B) \tag{7.21}$$

Where line 7.20 assumes non-negativity of W, thus proving monotonicity. □

7.4.3 Approximation Algorithm

Together, submodularity and monotonicity are exploitable properties that reveal simple approximation algorithms to hard problems. A reference including details for speeding up greedy selection under such functions is by Krause & Golovin [34]. In particular, Lemmas 7.1, 7.2, and a key result by Nemhauser et al. [40] imply:

Theorem 7.2 (approximation ratio)**.** *Initialize $H_0 = \emptyset$ and incrementally add heuristics by greedy selection from a set of admissible heuristics C,*

$$H_t = H_{t-1} \cup \left\{ \arg\max_{h \in C} \mathcal{U}(H_{t-1} \cup \{h\}) \right\}. \tag{7.22}$$

$\mathcal{U}(H_d)$ is greater than a factor of $(1 - 1/e) \approx 0.63$ of optimal.

A similar bound has been previously observed for the problem of selecting heuristics by Fuchs [22]. However, it applies to an edge covering measure of utility that can only be used with a specific type of heuristic, and does not incorporate an arbitrary default heuristic. While such an edge covering objective is compared to later, Theorem 7.2 applies to any – possibly heterogeneous – set of candidate heuristics C for which this measure of utility can be efficiently determined.

7.5 Sampling Method

Greedy selection can be further sped up by measuring utility with a simpler approximation to $\mathcal{U}$. The approach considered is to sum the heuristic values between only a sample of the states. Intuitively, if this sample is densely and uniformly distributed throughout the state space, then the heuristics between sample states should be strongly correlated with the heuristics between all states.

7.5.1 A Partitioning Approach

One of the many possible ways to implement sampling is to partition the state space into m mutually exclusive and collectively exhaustive regions, represented by sets of state indices, $Z_1, \ldots, Z_m$. Within each region, a single sample state is nominated, $z_i \in Z_i$. Based on these sample states, a new *sample utility* function is defined as

$$\overline{\mathcal{U}}(H) = \sum_{p=1}^{m} \sum_{q=1}^{m} \overline{W}_{pq}\, h^H(z_p, z_q) - \overline{\alpha}, \tag{7.23}$$

where the weight between sample states p and q is the sum of the weights between the states in partitions Z_p and Z_q,

$$\overline{W}_{pq} = \sum_{r \in Z_p} \sum_{s \in Z_q} W_{rs}, \tag{7.24}$$

or $\overline{W}_{pq} = |Z_p|\,|Z_q|$ if W specifies a uniform weighting, and

$$\overline{\alpha} = \sum_{p=1}^{m} \sum_{q=1}^{m} \overline{W}_{pq}\, h^{\emptyset}(z_p, z_q) \tag{7.25}$$

is a normalizing constant ensuring $\overline{\mathcal{U}}(\emptyset) = 0$.

Choosing the best partitioning for a specific search graph remains an open problem. In the experiments in Section 7.6, sample states are incrementally selected so as to *cover* the largest number of uncovered states; a state is covered if it is within t steps of a sample. When all states are covered, partitions are defined by assigning states to the nearest sample state as measured in an *undirected* version of the search graph, where the transition costs are replaced with $\delta(i,j) \leftarrow \min\{\delta(i,j), \delta(j,i)\}$. This approach can be likened to some existing methods for specifying canonical heuristics [56] and choosing pathfinding subgoals [9].

7.5.2 Analysis

This section analyzes the effect sampling has on optimality with respect to the true utility function (7.8). The foundation of this analysis is heuristic consistency. If each default and candidate heuristic $h_x \in D \cup C$ is consistent,[2] i.e.,

$$\forall i, j, k, \quad h_x(i,k) \leq h_x(j,k) + \delta(i,j), \tag{7.26}$$

then local distance information in the search graph can be used to bound the difference between $\mathcal{U}(H)$ and $\overline{\mathcal{U}}(H)$ for any H.

Lemma 7.3 (bound on sample utility)**.** *The sample utility's error is bounded by a weighted sum of the distances between samples and states in the same partition:*

$$\forall H \in 2^C, \quad |\mathcal{U}(H) - \overline{\mathcal{U}}(H)| \leq \epsilon \tag{7.27}$$

with:

$$\epsilon = 2 \sum_{p=1}^{m} \sum_{q=1}^{m} \sum_{r \in Z_p} \sum_{s \in Z_q} W_{rs}(\delta(r, z_p) + \delta(z_q, s)) \tag{7.28}$$

Proof. Since the partitions are mutually exclusive and collectively exhaustive, $\mathcal{U}(H)$ is equivalent to a sum over pairs of states between pairs of partitions:[3]

$$\mathcal{U}(H) = \sum_{i,j} W_{ij}\, h^H(i,j) - \alpha \tag{7.29}$$

$$= \sum_{p,q,r,s} W_{rs}\, h^H(r,s) - \sum_{p,q,r,s} W_{rs}\, h^{\emptyset}(r,s) \tag{7.30}$$

Since the heuristics in H are consistent, it must be true for states p, q, r, and s that:

$$h^H(r,s) \leq h^H(z_p, s) + \delta(r, z_p) \tag{7.31}$$

$$\leq h^H(z_p, z_q) + \delta(r, z_p) + \delta(z_q, s) \tag{7.32}$$

Reversing the consistency inequality similarly gives:

$$h^{\emptyset}(r,s) \geq h^{\emptyset}(z_p, s) - \delta(z_p, r) \tag{7.33}$$

$$\geq h^{\emptyset}(z_p, z_q) - \delta(z_p, r) - \delta(s, z_q) \tag{7.34}$$

[2] Consistency also implies admissibility.

[3] p and q always iterate from 1 to m, and r and s over the indices in Z_p and Z_q as on line 7.28.

Substituting lines 7.32 and 7.34 into line 7.30 establishes an upper bound on $\mathcal{U}(H)$:

$$\begin{aligned}\mathcal{U}(H) \leq & \sum_{p,q,r,s} W_{rs}(h^H(z_p, z_q) + \delta(r, z_p) + \delta(z_q, s)) \\ & - \sum_{p,q,r,s} W_{rs}(h^{\emptyset}(z_p, z_q) - \delta(z_p, r) - \delta(s, z_q)) \end{aligned} \tag{7.35}$$

By separating the terms that refer to the heuristics from those that do not refer to the heuristics, line 7.35 can be rewritten to recover the definitions of $\overline{W}, \overline{\mathcal{U}}$ and ϵ:

$$\begin{aligned} & \sum_{p,q,r,s} W_{rs}\, h^H(z_p, z_q) - \sum_{p,q,r,s} W_{rs}\, h^{\emptyset}(z_p, z_q) \\ + \sum_{p,q,r,s} W_{rs}(\delta(r, z_p) + \delta(z_q, s)) & + \sum_{p,q,r,s} W_{rs}(\delta(z_p, r) + \delta(s, z_q)) \qquad (7.36) \\ = \sum_{p,q} \overline{W}_{pq}\, h^H(z_p, z_q) & - \sum_{p,q} \overline{W}_{pq}\, h^{\emptyset}(z_p, z_q) + \epsilon \qquad (7.37) \\ & = \overline{\mathcal{U}}(H) + \epsilon \qquad (7.38) \end{aligned}$$

where the last two terms of line 7.36 are identical but with transposed indices, together equating to ϵ; thus proving $\mathcal{U}(H) \leq \overline{\mathcal{U}}(H) + \epsilon$. The lower bound proceeds similarly, and together these bounds give $|\mathcal{U}(H) - \overline{\mathcal{U}}(H)| \leq \epsilon$. □

Lemma 7.3 implies a modified greedy algorithm could be used to optimize $\overline{\mathcal{U}}$ and still retain an approximation guarantee to optimal under $\mathcal{U}$ [35]. However, this modification entails a polynomial increase in the number of iterations over the candidate set, which may be unacceptable when the set is large. Fortunately, solutions of known optimality under $\overline{\mathcal{U}}$ have a *provable* optimality ratio under $\mathcal{U}$. To see this, first consider the following general lemma.

Lemma 7.4 (additive/relative bound)**.** *Let θ and ϕ be set functions over 2^S with*

$$\min_{A:|A|=d} \theta(A) = 0, \quad \min_{A:|A|=d} \phi(A) = 0, \tag{7.39}$$

$$\forall A \in 2^S \quad |\theta(A) - \phi(A)| \leq \epsilon, \tag{7.40}$$

for some ϵ, and finite maxima at cardinality d denoted

$$\theta^* = \max_{A:|A|=d} \theta(A), \quad \phi^* = \max_{B:|B|=d} \phi(B). \tag{7.41}$$

For any set G with $|G| = d$, G's optimality with respect to ϕ is tied to its optimality with respect to θ:

$$\frac{\phi(G)}{\phi^*} \geq b \Rightarrow \frac{\theta(G)}{\theta^*} \geq b\frac{\phi(G) - \epsilon}{\phi(G) + b\epsilon} \tag{7.42}$$

Proof. The bound given on line 7.40 implies that θ^* and ϕ^* must also have bounded difference: $\theta^* - \epsilon \leq \phi(\arg(\theta^*)) \leq \phi^*$. This reveals an inequality relating θ^* to $\phi(G)$,

$$\phi(G) \geq b\phi^* \geq b(\theta^* - \epsilon) \Leftrightarrow \theta^* \leq \frac{\phi(G) + b\epsilon}{b}, \tag{7.43}$$

which lets us bound the optimality of $\theta(G)$ as

$$\frac{\theta(G)}{\theta^*} \geq \frac{\phi(G) - \epsilon}{\theta^*} \geq b\frac{\phi(G) - \epsilon}{\phi(G) + b\epsilon}, \tag{7.44}$$

thus proving the inequality. □

Next, by duplicating the reasoning in Lemmas 7.1 and 7.2, it can be easily shown that the sample utility function $\overline{\mathcal{U}}$ is both submodular and monotonic, and so it follows – just as in Theorem 7.2 – that greedy selection under $\overline{\mathcal{U}}$ finds solutions that score greater than a factor of $(1 - 1/e)$ of $\overline{\mathcal{U}}$'s optimal value. Together with Lemma 7.4, this implies the following result:

Theorem 7.3 (approximation ratio under sampling)**.** *Initialize $H_0 = \emptyset$, and incrementally add heuristics by greedy selection from a set of consistent heuristics C,*

$$H_t = H_{t-1} \cup \left\{ \arg\max_{h \in C} \overline{\mathcal{U}}(H_{t-1} \cup \{h\}) \right\}. \tag{7.45}$$

$\overline{\mathcal{U}}(H_d)$ is within a factor of $(1 - 1/e)$ of optimal, implying $\mathcal{U}(H_d)$ is within a factor of $(1 - 1/e)\frac{\overline{\mathcal{U}}(H_d) - \epsilon}{\overline{\mathcal{U}}(H_d) + \epsilon - \epsilon/e}$ of optimal.

Under the assumption that both $\overline{\mathcal{U}}(H)$ and ϵ can be efficiently computed, it is therefore easy to determine an optimality bound on H *post factum*. Note that this bound improves as $\overline{\mathcal{U}}(H)$ increases (i.e., as more heuristics are added to the set H).

7.6 Evaluation

The approach of greedy utility maximization under $\mathcal{U}$ and $\overline{\mathcal{U}}$ is tested on optimal search with A^* [28], with a focus on selecting true-distance heuristics. The number of nodes expanded during search and the total CPU time are used as general performance measures by which to make comparisons.

The evaluation is subdivided into two studies. The first is to compare *existing* selection methods on undirected search domains; the second is to use greedy utility maximization to benchmark a new type of *directed* differential heuristic for strongly directed domains, revealing encouraging results over alternative heuristic sources.

7.6.1 Undirected Domains

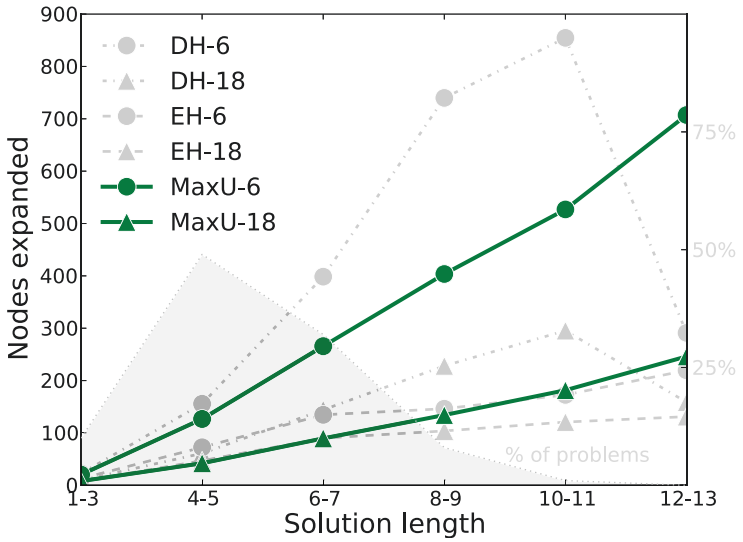
Greedy utility maximization is first applied to the selection of differential heuristics [56] for undirected domains. Recall a single DH consists of the precomputed distances from each state to a specific landmark state p and, from these distances, $h_p(i, j) = |\delta(i, p) - \delta(j, p)|$ gives consistent heuristics. Each DH requires the same amount of memory, so a cardinality constraint doubles as a constraint on memory.

Word Search

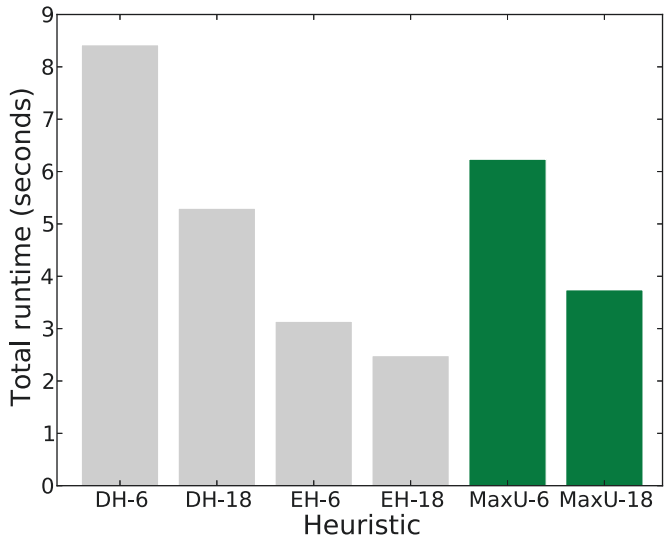
As before, the four-letter *Word Search* domain is comprised of states representing four letter words from an English dictionary. A word can be changed into another by substituting one letter at a time, resulting in 54,752 edges across 4,820 states. *Coal* could be turned into *gold* by taking the path $\langle$coal; goal; goad; gold$\rangle$.

In Chapter 3, Euclidean heuristics were shown to outperform DHs that used the *Farthest* selection algorithm [24] on precisely this search graph. In Chapter 5, a larger number of *enhanced* differential heuristics was shown to be competitive in terms of node expansions, but not runtime. Nevertheless, it remains unclear whether DHs or the algorithm that was used to select them are to blame. This distinction is to be emphasized: a strong class of heuristics may be undervalued in the absence of a good selection algorithm. This motivates testing whether DHs are powerful enough to outperform Euclidean heuristics when driven by greedy utility maximization under $\mathcal{U}$ (note utility is not sampled using $\overline{\mathcal{U}}$ here). The experiment tests sets of 6 and 18 DHs (**DH-6** and **DH-18**), and uses the same Euclidean heuristics of dimension 6 and 18 (**EH-6** and **EH-18**). The default heuristic is the zero heuristic.

The results across 50,000 problems are shown in Figure 7.1. Greedily maximizing utility (**MaxU**) gives a set of 6 DHs that are still not competitive with a 6-dimensional Euclidean heuristic. The 18 greedily chosen DHs are significantly



(a) *Word Search* A* node expansions.



(b) *Word Search* timing results.

Figure 7.1: *Word Search* node expansion and timing results comparing multidimensional Euclidean heuristics (EH) and differential heuristics chosen using the *Farthest* selection method (FDH) and the utility-maximizing objective (MaxU).

more competitive, but fall behind on the longer solution lengths. Meanwhile, the DHs placed using the *Farthest* algorithm excel only on a minority of the longest paths. This is expected of *Farthest*, since such paths tend to start or end on the landmark states. The timing results underscore these results, with the more efficient Euclidean heuristic lookup clearly dominating in terms of solution time.

Pathfinding

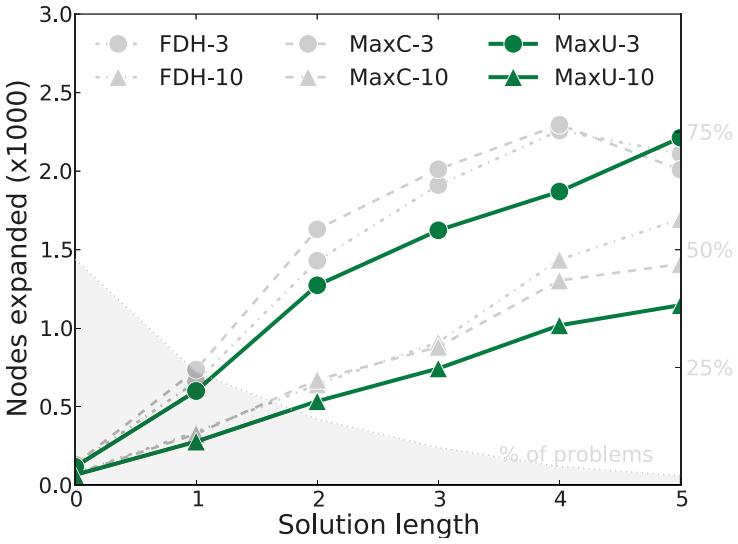
Next are the standard pathfinding benchmarks [55] on BioWare's *Dragon Age: Origins*' grid-based maps. The agent can move cardinally at cost 1 or diagonally at cost 1.5 among open grid cells, as long as doing so does not cut the corner of any closed grid cell. All benchmarks on fully connected maps with between 168 and 18,890 states are considered, using the straight-line octile distance as the default heuristic.

Three selection algorithms are compared. First is greedy utility maximization under $\overline{\mathcal{U}}$, where the partitions are defined using a radius of 1 when a map has less than 10,000 states, and a radius of 2 otherwise (**MaxU**). Next, a greedy edge-covering approach [22] is considered (**MaxC**). This measure of utility is also submodular monotonic, so greedy selection gives a similar optimality guarantee with respect that objective. But to improve its efficiency and facilitate an even comparison, this approach is modified to use the same sample states used by $\overline{\mathcal{U}}$. Last, the *Farthest* algorithm is used, which has tended to be very effective on these benchmark problems (**FDH**). Each approach is used to define sets of 3 and 10 DHs.

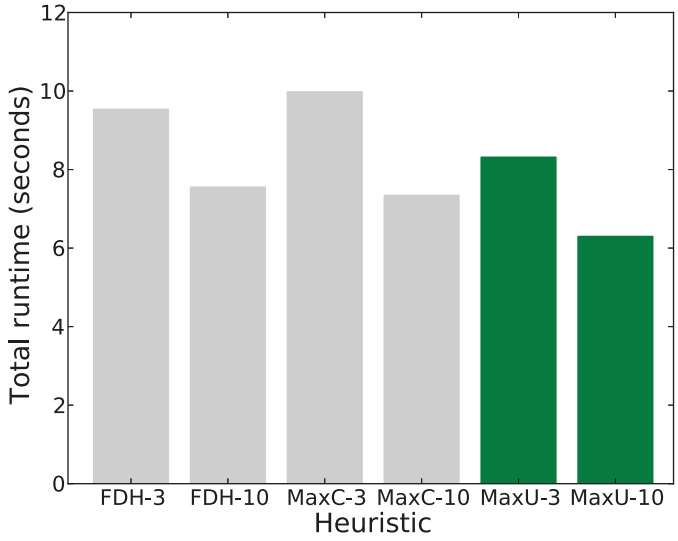
Figure 7.2 shows greedy utility maximization achieving marked improvements on the vast majority of problems. Though all three approaches have access to the default heuristic during search, our approach benefits from explicitly incorporating it into how the subset is chosen. While these results are not as great as seen in Chapter 5 on the same problems, the optimization – greedy utility maximization – finished in considerably less time and is much more easily implemented.

7.6.2 Directed Domains

Since utility maximization can be applied to *any* class of heuristics, an important practical use is to compare competing classes of heuristics on a level playing field.



(a) *Pathfinding* A* node expansions.



(b) *Pathfinding* timing results.

Figure 7.2: *Pathfinding* node expansion and timing results comparing differential heuristics chosen using the *Farthest* selection method (*FDH*), an edge-covering objective (*MaxC*), and a utility-maximizing objective (*MaxU*).

In the following, three kinds of heuristics for *directed* search graphs are compared.

Recall a search graph is *directed* if, for any two states i and j, $\delta(i,j) \neq \delta(j,i)$. DHs can still generate consistent heuristics on such graphs if the distances are computed on an undirected version of the graph with $\delta(i,j) \leftarrow \min\{\delta(i,j), \delta(j,i)\}$, but a new approach of *directed* differential heuristics will also be considered.

Directed differential heuristics (DDHs). DDHs use a pivot state p too, but each pivot defines two distinct heuristic functions. If distances are computed *to* the pivot:

$$h_{\overleftarrow{p}}(i,j) = (\delta(i,p) - \delta(j,p))^+ \tag{7.46}$$

If they are computed *from* the pivot:

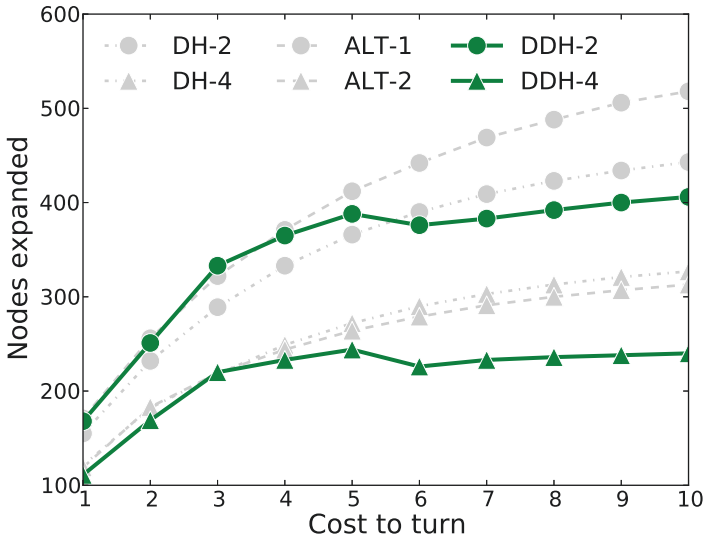
$$h_{\overrightarrow{p}}(i,j) = (\delta(p,j) - \delta(p,i))^+ \tag{7.47}$$

A search graph with n states defines $2n$ DDHs. DDHs are essentially a decoupling of the heuristics used by the ALT algorithm [24], which always stores both the *to* and *from* distances for each pivot. Given fixed memory, the space of possible DDH subsets encapsulates and is exponentially larger than the space of possible ALT subsets, but the greedy algorithm will only experience a doubling in effort.

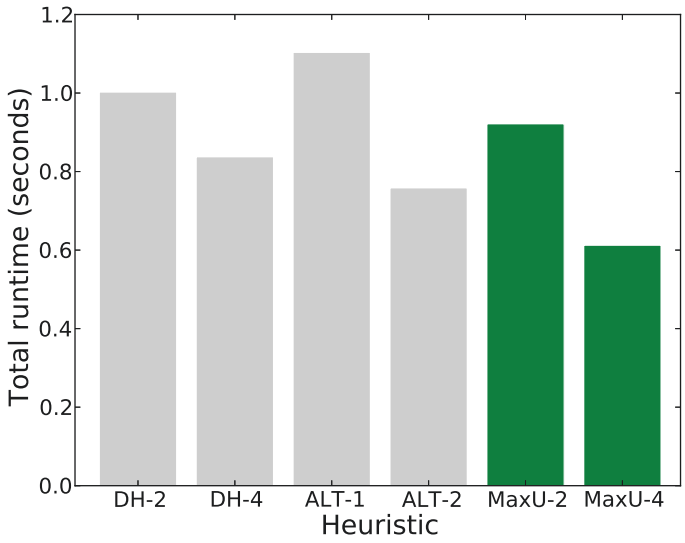
Each of the three has a defining drawback. A DH built on the undirected graph risks being based on inaccurate distances; a DDH suffers from returning 0 for at least half of the state pairs; and a single ALT heuristic occupies twice the memory of any one DH or DDH. Note that the optimality bound on our subset selection approach applies in each case, despite these major idiosyncrasies.

Oriented Pathfinding

In the first comparison of these three heuristics, a state variable is added to the *Dragon Age* map, LAK101D, to describe the agent's current heading. The agent can advance along its current heading or turn left and right in 45 degree increments, as depicted in Figure 7.4. This results in 7,342 directed edges across 2,544 states. The cost to advance is 1, and the cost to turn is a variable which offers control over the difference in cost between an action and its reverse. The default heuristic is the travel cost assuming no closed cells and no cost to turn.



(a) *Oriented* node expansions by A* for varying turn costs.



(b) *Oriented* timings when cost to turn is 10.

Figure 7.3: *Oriented Pathfinding* node expansion and timing results on 10,000 problems comparing ALT heuristics, differential heuristics (DH), and directed differential heuristics (DDH), all chosen greedily under a utility maximizing objective.

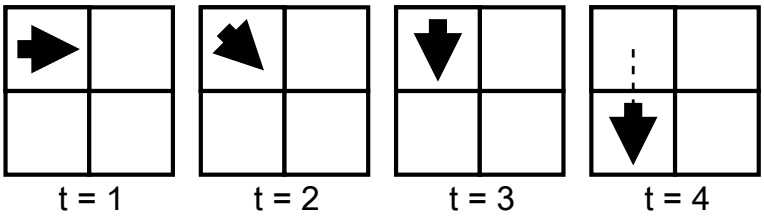


Figure 7.4: In an *oriented game map*, an east-facing agent turns before going south.

How does the cost to turn affect the performance of sets of 1 and 2 ALT heuristics, and sets of 2 and 4 DH and DDH heuristics? Each set is selected using greedy utility maximization without sampling (that is, utility is measured directly using $\mathcal{U}$), and each set uses the same amount of memory. The results across 10,000 benchmark problems are shown in Figure 7.3. The search problems become more expensive to solve as the cost to turn increases, but DDHs seem least affected by this. A tradeoff between DHs and DDHs is also evident: when the cost to turn is small, 2 DHs are preferred over 2 DDHs, but this changes as the cost increases.

Platformer

The last test domain is defined on a grid of three types of cells: platforms, ladders, and mid-air. Examples are shown in Figure 7.5. If the agent is on a platform or ladder, it can move left and right, or up and down onto ladders. If the agent occupies a mid-air state, it automatically moves *down* one cell per turn, but can optionally control its descent left or right. The default heuristic is the number of steps assuming the agent can travel freely between any adjoining cells. This domain is modelled on a popular genre of video games which has received relatively little attention as a pathfinding benchmark, but is an extreme example of a cost structure that occurs in domains with actions that are difficult to reverse.

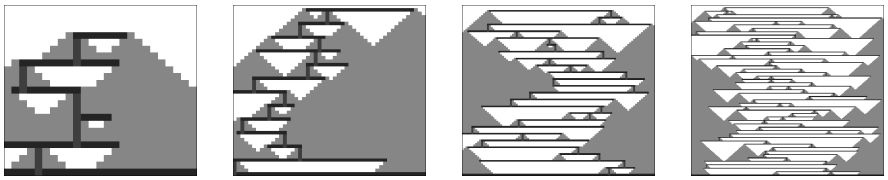
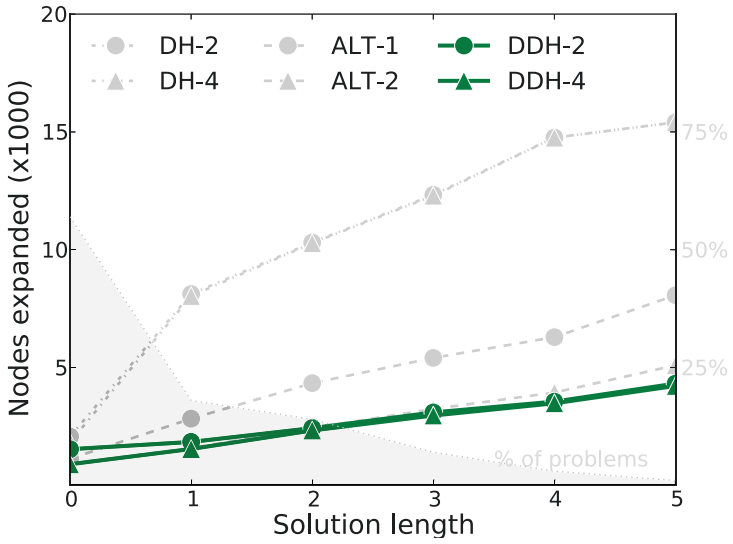
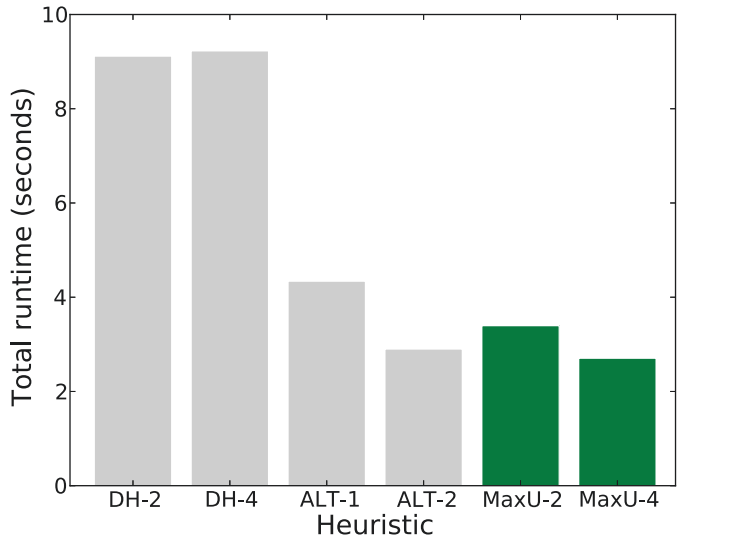


Figure 7.5: *Platformer* examples. Dark cells are platforms/ladders, grey are mid-air, and white are out of reach. The platformer on the right is used in the experiments.



(a) *Platformer* results.



(b) *Platformer* timings.

Figure 7.6: *Platformer* node expansion and timing results on 10,000 problems comparing *ALT* heuristics, differential heuristics (*DH*), and directed differential heuristics (*DDH*), all chosen greedily under a utility maximizing objective.

A random platformer is defined on a 200×200 grid with 16,248 accessible states. The states are partitioned into regions whose radius is approximately 2, and greedy maximization of $\overline{U}$ is used to choose sets of 1 and 2 ALT heuristics, and sets of 2 and 4 DH and DDH heuristics. Figure 7.6 shows the results on 10,000 random problems. These results reveal DHs to be ineffective at representing costs in this domain, due to the disparity between the cost of an action and its reverse. Meanwhile, the DDH subsets consistently outperform their ALT counterparts. With the knowledge that both are near-optimal under the same objective, one can infer that DDHs are well worth considering as an alternative class of search heuristics in domains with highly directed costs.

7.7 Summary

This chapter introduced a novel approach to the widespread problem of selecting search heuristics. It was shown that greedy selection is nearly optimal under a natural measure of utility, and furthermore that provable optimality is not sacrificed when utility is measured on just a sample of the states. These two points rely on the admissibility and consistency of the heuristics respectively – but such properties are commonly satisfied by many existing classes of heuristic functions.

The empirical study emphasized the importance of optimality under a well-motivated objective. Approximate (greedy) utility maximization partially redeemed an underperforming class of heuristics in one domain, outcompeted existing selection algorithms in another, and was instrumental in a pilot study of a promising new type of heuristic which excels in directed domains. This work has the potential to be extended in several new directions, such as by defining new measures of utility, more efficient sampling methods, and wider application to domains where heuristic subset selection defies domain expertise.

Chapter 8

Conclusion

Heuristic search algorithms rely on the presence of an accurate heuristic function to be most effective. This thesis cast the problem in a new light by defining the heuristic as the solution to an optimization problem. Several novel heuristic construction settings were considered, generally revolving around the idea of using the distances between points to encode the true shortest-path distances in a search graph.

Chapter 3 begun by considering a multidimensional Euclidean representation. Its careful choice of loss enabled us to rewrite the associated optimization problem as a semidefinite program, which facilitated tractable solutions. We also observed this optimization to be identical to one studied in the field of manifold learning, exposing a new connection between AI subfields. Empirically, this method showed strong results on inherently "multidimensional" search domains compared to differential heuristics, but gave weaker results on inherently *low*-dimensional domains.

Chapter 4 showed that the same optimization problem, carefully parametrized, could be used to *reproduce* differential heuristics. Not only did this result reveal the generality of that optimization formulation, but it also suggested a novel strategy for *improving* upon a given set of differential heuristics. This turned out to be crucial for building a competitive class of *enhanced* differential heuristics, which proved to be well-suited to inherently low-dimensional domains.

Motivated by the empirical success of these low-dimensional heuristics, Chapter 5 turned to building good heuristics directly on the line, rather than by truncating a high-dimensional heuristic as the previous chapters had. While citing existing bounds in the literature, we also developed a scalable local optimization technique

for this setting with several appealing properties. Chapter 6 continued the investigation of this special but fundamental case of building heuristics on the line, but focused on a new, flexible type of directed heuristic whose points lie in a quasimetric space. Empirical studies showed both of these low-dimensional heuristics to be effective in a number of illustrative search domains.

Having introduced several new techniques for building search heuristics, Chapter 7 turned to selecting a best subset among them. Despite the selection problem being NP-hard, we appealed to key results in the discrete optimization literature to find good approximations – approximations expected to be within a *factor of two* of optimal. This solution provided a common interface to the heuristic construction techniques that preceded, and an empirical study once again emphasized the importance of striving for optimality under a well-defined objective function.

Each of these cases followed the methodology of optimization: posing the problem as the minimization of a constrained and precise loss function, identifying exploitable properties in that loss function, and coming away with tractable solutions that can be justifiably be described as "good" in a concrete, quantitative sense. It is hoped that the effective use of this methodology, as described in this dissertation, will inspire future research into heuristic construction.

Bibliography

[1] Arfaee, S. J., Zilles, S., & Holte, R. C. (2010). Bootstrap Learning of Heuristic Functions. In *Proceedings of the Third Annual Symposium on Combinatorial Search (SoCS)*, (pp. 52–60). AAAI Press 2010.

[2] Bagchi, A., & Mahanti, A. (1983). Search Algorithms Under Different Kinds of Heuristics - A Comparative Study. *Journal of the ACM (JACM)*, *30*(1), 1–21.

[3] Bauer, R., Columbus, T., Katz, B., Krug, M., & Wagner, D. (2010). Preprocessing Speed-Up Techniques is Hard. In *Proceedings of the 7th International Conference on Algorithms and Complexity*, CIAC'10, (pp. 359–370). Berlin, Heidelberg: Springer-Verlag.

[4] Bioware (2009). Dragon Age: Origins (Video Game).

[5] Bjårnsson, Y., Enzenberger, M., Holte, R. C., & Schaeffer, J. (2005). Fringe Search: Beating A* at Pathfinding on Game Maps. In *Proceedings of the 2005 IEEE Symposium on Computational Intelligence and Games (CIG05)*, (pp. 125–132).

[6] Bourgain, J. (1985). On Lipschitz Embedding of Finite Metric Spaces in Hilbert Space. *Israel Journal of Mathematics*, *52*, 46–52.

[7] Bowling, M., Wilkinson, D., & Ghodsi, A. (2006). Subjective Mapping. In *Proceedings of the 21st National Conference on Artificial Intelligence*, vol. 2, (pp. 1569–1572). AAAI Press.

[8] Bulitko, V., & Lee, G. (2006). Learning in Real-Time Search: A Unifying Framework. *Journal of Artificial Intelligence Research*, *25*, 119–157.

[9] Bulitko, V., Rayner, C., & Lawrence, R. (2012). On Case Base Formation in Real-Time Heuristic Search. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment conference (AIIDE)*, (pp. 106–111).

[10] Chen, W., Chen, Y., Weinberger, K. Q., Lu, Q., & Chen, X. (2013). Goal-Oriented Euclidean Heuristics with Manifold Learning. In *Proceedings of the Twenty-Seventh National Conference on Artificial Intelligence (AAAI)*, (pp. 74–76). Seattle, WA, USA.

[11] Chen, W., Weinberger, K. Q., & Chen, Y. (2013). Maximum Variance Correction with Application to A*. In *Proceedings of the Thirtieth International Conference on Machine Learning (ICML)*, (pp. 302–310). Atlanta, GA, USA.

[12] Cortes, C., & Vapnik, V. (1995). Support-vector networks. In *Machine Learning*, (pp. 273–297).

[13] Culberson, J., & Schaeffer, J. (1998). Pattern Databases. *Computational Intelligence*, *14(3)*, 318–334.

[14] De Bie, T., Cristianini, N., & Rosipal, R. (2005). Eigenproblems in Pattern Recognition. *Handbook of Geometric Computing: Applications in Pattern Recognition, Computer Vision, Neuralcomputing, and Robotics*.

[15] Dhamdhere, K. (2004). Approximating Additive Distortion of Embedding into Line Metrics. In *7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, (pp. 96–104). SpringerVerlag.

[16] Díaz, J., Petit, J., & Serna, M. (2002). A Survey of Graph Layout Problems. *ACM Computing Surveys*, *34*(3), 313–356.

[17] Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, *1*(1), 269–271.

[18] Ernandes, M., & Gori, M. (2004). Likely-admissible and Sub-symbolic Heuristics. In *ECAI*, (pp. 613–617).

[19] Floyd, R. W. (1962). Algorithm 97: Shortest Path. In *Communications of the ACM*, vol. 5, (p. 345). New York, NY, USA.

[20] Fournier, A., Fussell, D., & Carpenter, L. (1982). Computer Rendering of Stochastic Models. *Communications of the ACM*, *25*(6), 371–384.

[21] Franco, S., Barley, M. W., & Riddle, P. J. (2013). A New Efficient In Situ Sampling Model for Heuristic Selection in Optimal Search. In *Australasian Conference on Artificial Intelligence*, (pp. 178–189).

[22] Fuchs, F. (2010). *On Preprocessing the ALT-Algorithm*. Master's thesis, Institute for Theoretical Informatics, Faculty of Computer Science, University of Karlsruhe.

[23] Geisberger, R., Sanders, P., Schultes, D., & Delling, D. (2008). Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *Proceedings of the Seventh Workshop on Experimental Algorithms*, vol. 5038 of *Lecture Notes in Computer Science*, (pp. 319–333). Springer.

[24] Goldberg, A. V., & Harrelson, C. (2005). Computing the Shortest Path: A* Search Meets Graph Theory. In *Symposium on Discrete Algorithms (SODA)*, (pp. 156–165).

[25] Goldberg, A. V., & Werneck, R. F. (2003). Computing Point-to-Point Shortest Paths from External Memory. In *Proceedings of the 2005 SIAM Workshop on Algorithms Engineering and Experimentation*, (pp. 26–40).

[26] Goldenberg, M., Felner, A., Sturtevant, N., & Schaeffer, J. (2010). Portal-Based True-Distance Heuristics for Path Finding. In *Proceedings of Symposium on Combinatorial Search (SoCS)*, (pp. 39–45).

[27] Golub, G. H., & Loan, C. F. V. (1996). *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press.

[28] Hart, P., Nilsson, N., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107.

[29] Hjaltason, G. R., & Samet, H. (2003). Properties of Embedding Methods for Similarity Searching in Metric Spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *25*(5), 530–549.

[30] Hoffman, A. J., & Kruskal, J. B. (1956). Integral Boundary Points of Convex Polyhedra. In H. W. Kuhn, & A. W. Tucker (Eds.) *Linear Inequalities and Related Systems, Annals of Mathematics Studies 38*, (pp. 223–246). Princeton, NJ, USA: Princeton University Press.

[31] Holte, R., Felner, A., Newton, J., Meshulan, R., & Furcy, D. (2006). Maximizing over Multiple Pattern Databases Speeds up Heuristic Search. *Artificial Intelligence Journal*, *170*, 1123–1136.

[32] Håstad, J., Ivansson, L., & Lagergren, J. (2003). Fitting Points on the Real Line and its Application to RH Mapping. *Journal of Algorithms*, *49*(1), 42 – 62.

[33] Korf, R. E. (1985). Depth-first Iterative Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence*, *27*(3), 97–109.

[34] Krause, A., & Golovin, D. (2012). Submodular Function Maximization. In *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press.

[35] Krause, A., & Guestrin, C. (2005). A Note on the Budgeted Maximization of Submodular Functions. Tech. Rep. CMU-CALD-05-103, Carnegie Mellon University.

[36] Linial, M., Linial, N., Tishby, N., & Yona, G. (1997). Global self organization of all known protein sequences reveals inherent biological signatures. *Journal of Molecular Biology*, *268*, 539 – 556.

[37] Linial, N., London, E., & Rabinovich, Y. (1995). The Geometry of Graphs and Some of Its Algorithmic Applications. *Combinatorica*, *15*, 215–245.

[38] Lu, Q., Chen, W., Chen, Y., Weinberger, K. Q., & Chen, X. (2013). Utilizing Landmarks in Euclidean Heuristics for Optimal Planning. In *Late-Breaking Track, Proceedings of the Twenty-Seventh National Conference on Artificial Intelligence (AAAI)*, (pp. 74–76). Seattle, WA, USA.

[39] Martelli, A. (1977). On the Complexity of Admissible Search Algorithms. *Artificial Intelligence*, *8*, 1–13.

[40] Nemhauser, G., Wolsey, L., & Fisher, M. (1978). An Analysis of Approximations for Maximizing Submodular Set Functions. *Mathematical Programming*, *14*(1), 265–294.

[41] Ng, T. E., & Zhang, H. (2002). Predicting Internet Network Distance with Coordinates-Based Approaches. In *IEEE Conference on Computer Communications (INFOCOM)*, (pp. 170–179).

[42] Papadopoulos, A., & Troyanov, M. (2007). Weak metrics on Euclidean domains. *JP Journal of Geometry and Topology*, *1*, 23–44.

[43] Passino, K. M., & Antsaklis, P. J. (1994). A Metric Space Approach to the Specification of the Heuristic Function for the A* Algorithm. *IEEE Transactions on Systems, Man, and Cybernetics*, *24*, 159–166.

[44] Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

[45] Pearl, J. (1988). On the Discovery and Generation of Certain Heuristics. In R. Engelmore (Ed.) *Readings from the AI Magazine*, (pp. 58–66). Menlo Park, CA, USA: American Association for Artificial Intelligence.

[46] Pennings, T. J. (2003). Do Dogs Know Calculus? *The College Mathematics Journal*, *34*(3), 178–182.

[47] Petrik, M., & Zilberstein, S. (2008). Learning Heuristic Functions through Approximate Linear Programming. In *International Conference on Automated Planning and Scheduling (ICAPS)*, (pp. 248–255).

[48] Pohl, I. (1970). Heuristic Search Viewed as Path Finding in a Graph. *Artificial Intelligence Journal (AIJ)*, *1*, 193–204.

[49] Powell, W. B. (2007). *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. John Wiley & Sons.

[50] Rabinovich, Y. (2003). On Average Distortion of Embedding Metrics into the Line. In *35th Annual ACM Symposium on Theory of Computing*, (pp. 456–462).

[51] Ratliff, N. D., Bagnell, J. A., & Zinkevich, M. A. (2006). Maximum Margin Planning. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, (pp. 729–736).

[52] Rayner, C., Bowling, M., & Sturtevant, N. (2011). Euclidean Heuristic Optimization. In *Proceedings of the Twenty-Fifth National Conference on Artificial Intelligence (AAAI)*, (pp. 81–86). San Francisco, CA, USA.

[53] Rayner, C., Sturtevant, N., & Bowling, M. (2013). Subset Selection of Search Heuristics. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, (pp. 637–643). Beijing, China.

[54] Saxe, J. B. (1979). Embeddability of Weighted Graphs in k-Space is Strongly NP-Hard. In *Allerton Conference on Circuit and System Theory*.

[55] Sturtevant, N. R. (2012). Benchmarks for Grid-Based Pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*(2), 144–148.

[56] Sturtevant, N. R., Felner, A., Barrer, M., Schaeffer, J., & Burch, N. (2009). Memory-Based Heuristics for Explicit State Spaces. *IJCAI-09*, (pp. 609–614).

[57] Thorup, M., & Zwick, U. (2001). Approximate Distance Oracles. In *Proceedings of the thirty-third annual ACM symposium on theory of computing*, STOC, (pp. 183–192). New York, NY, USA: ACM.

[58] Toh, K. C., Todd, M. J., & Tutuncu, R. H. (1999). SDPT3 – a Matlab software package for semidefinite programming. *Opt. Methods and Software*, *11*, 545–581.

[59] Warshall, S. (1962). A Theorem on Boolean Matrices. In *Journal of the ACM (JACM)*, vol. 9, (pp. 11–12). New York, NY, USA.

[60] Weinberger, K. Q., & Saul, L. K. (2006). An Introduction to Nonlinear Dimensionality Reduction by Maximum Variance Unfolding. In *Proceedings of the 21st National Conference on Artificial Intelligence*, AAAI, (pp. 1683–1686).

[61] Weinberger, K. Q., Sha, F., Zhu, Q., & Saul, L. K. (2006). Graph Laplacian Regularization for Large-Scale Semidefinite Programming. In *Advances in Neural Information Processing Systems 19*, (pp. 1489–1496).