University of Alberta

PERFORMANCE ANALYSIS OF RECENT REAL-TIME HEURISTIC SEARCH THROUGH SEARCH-SPACE CHARACTERIZATION

by

Daniel Huntley

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

Master of Science

Department of Computing Science

Spring 2012
Edmonton, Alberta

Abstract

Recent real-time heuristic search algorithms have demonstrated outstanding performance in video game pathfinding. However, their applications have been thus far limited to that domain. We proceed with the aim of facilitating wider applications of real-time search by fostering a greater understanding of the performance of recent algorithms. We first introduce several algorithm-independent complexity measures for search spaces and correlate their values with algorithm performance. The complexity measures are statistically shown to be strong predictors of algorithm performance across a set of commercial video game maps. We then extend this analysis to a wider variety of search spaces in the first formal application of state of the art real-time search to domains outside of video game pathfinding.

Acknowledgements

I would like to extend my sincere thanks to my friends and family for their support during my research. I would also like to thank the other members of IRCL for their feedback and assistance. In addition, I thank Ramon Lawrence for his feedback in the early stages of this research. I would also like to acknowledge the financial support of iCORE and NSERC.

I would like to thank all of the members of my examination committee for volunteering their time and efforts. Lastly, I owe a great thanks to my supervisor Vadim Bulitko. Without his continued guidance and support, this research would have been impossible.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Heuristic search is a mainstay of artificial intelligence research. A demand for quickly generated solutions to search problems gave rise to a sub-field of investigation: *real-time heuristic search*. Real-time heuristic search algorithms make decisions in constant time, independent of the size of the problem being solved.

Recent algorithms have demonstrated exceptional academic performance in video game pathfinding. However, despite being formulated for general search, most of these algorithms have not been applied to a broader selection of problems. In preliminary experimentation, we found that many of the algorithms yielded mixed or poor results in other search spaces, such as combinatorial search puzzles [20]. Motivated by this mixed performance, we seek to establish a way of empirically characterizing search spaces based on their suitability for different real-time heuristic search algorithms. This would assist algorithm selection, and provide insight for the development of future algorithms.

In this chapter we discuss the goals of our research. We then describe our specific contributions, and outline the layout of the remainder of this document.

1.1 Motivation

There are three major goals motivating our research. We address all of these goals through an analysis of real-time heuristic search performance, focusing on how performance is affected by search space features. First, we seek to build a greater understanding of where current real-time heuristic search is and is not effective. Second, we wish to demonstrate that this knowledge can facilitate algorithm selection and parameterization. Finally, we want to provide a way for other researchers to characterize benchmark search problems for the development of future real-time heuristic search algorithms.

1.2 Contributions

This document makes the following contributions. First, we present a set of complexity measures which are useful for characterizing search space complexity as it pretains to the performance of modern real-time search algorithms. Some of these complexity measures are original to this work, and some have been adapted from the literature.

We then empirically link the values of the collected complexity measures to the performance of modern database-driven real-time search algorithms. We begin with an examination of algorithm performance in pathfinding on a varied set of videogame maps. This has served as the traditional test bed for subgoaling real-time search [10] [8] [25]. This examination demonstrates a statistical correlation between solution suboptimality and complexity measure values. It also shows that machine learning can be used to build a models to predict algorithm performance and facilitate algorithm parameterization.

We continue with a parallel examination of algorithm performance beyond videogame pathfinding. This study is performed using mazes and road maps. To our knowledge, this is the first time that these algorithms have been applied to these domains. These additional search spaces represent an incremental step towards more general spaces, and introduce several of the challenges that must be addressed if contemporary algorithms are to be successfuly adapted for broader domains such as planning.

Chapter 2

Problem Formulation

In this chapter we provide definitons for the terminology that will be used throughout the remainder of this document. We formally define heuristic search problems, and present our methodology for measuring the effectiveness of heuristic search algorithms.

2.1 Heuristic Search Problems

For the purposes of this document, we define a *search problem* as the following:

- S - a set of vertices, each being a representative of a distinct *state* in the problem model;
- E - a set of *transitions*, or weighted directed edges, on $S \times S$;
- $s_{\text{start}} \in S$ - the start state;
- $s_{\text{goal}} \in S$ - the goal state;
- h - a heuristic function; $h(s_1, s_2)$ is the heuristic distance between states s_1 and $s_2 \in S$; $h(s)$ is the heuristic distance between state $s \in S$ and the closest goal state

The graph comprising S and E is called the *search space*. A single search space may accomodate many possible search problems. *Search space size* is defined as $|S|$. *Branching factor* is defined as $|E|/|S|$, the mean number of edges exiting any state in the search space. State s_2 is said to be *reachable* from state s_1 if there exists a set of edges (s_1, s_i), (s_i, s_{i+1}), ... (s_{i+n}, s_2) connecting s_1 to s_2 in the search space. A search space is *connected* if every state is reachable from every other state. A *"dead-end"* is a state which has incoming edges, but no outgoing edges.

A *solution* to a search problem consists of the ordered set of edges that forms a path from s_{start} to the goal state s_{goal}. This is intuitively the set of decisions that an agent must make, or the *plan* that it must execute, to traverse from an initial state to a goal state. The *length* of a solution is the number of edges comprising the solution, while the *cost* of a solution is the sum of the weights of those edges. An *optimal solution* to a search problem is one where there does not exist another solution with a lower cost. A *suboptimal solution* is any solution to a search problem which is not optimal.

The heuristic function h may be used by an agent to inform decisions when searching for a solution to a problem. Here we define two specific types of heuristic functions:

- h_0 - the initial heuristic function; $h_0(s_1, s_2)$ is the estimated minimal cost between states s_1 and $s_2 \in S$; $h_0(s)$ is the estimated minimal distance between state $s \in S$ and the goal state;
- h^* - the optimal cost function; $h^*(s_1, s_2)$ is the actual minimal cost between states s_1 and $s_2 \in S$; $h^*(s)$ is the actual minimal distance between state $s \in S$ and the goal state.

We refer to the class of algorithms designed to solve search problems as *heuristic search algorithms*. An agent executing a heuristic search algorithm is a *heuristic search agent*. A heuristic search algorithm that is guaranteed to find a solution if one exists is called *complete*. An algorithm that is guaranteed to find an optimal solution is called an *optimal* algorithm. *Learning* algorithms are those which make updates to their heuristic function during execution. In these instances, h will denote the current heuristic function. A heuristic function h is *admissible* if and only if $h(s) \leq h^*(s)$ $\forall s \in S$.

2.2 Search Performance

We will use two major metrics to assess the performance of a heuristic search algorithm in this document: solution suboptimality and pre-computation time. These measures are collectively referred to as *search performance*. We define *solution suboptimality* as the ratio of the cost of a solution to the cost of the optimal solution. For example, if a given solution has cost 5 and the optimal solution has cost 4, then the solution suboptimality is 1.25.

Some of the algorithms that we discuss require pre-computation time to construct a database for the given search space. We refer to this as *database construction time* or *pre-computation time*. Any such preparatory behaviour performed by a search agent is refered to as being *offline*, whereas any work done during active solving of specific search problems is refered to as being *online*.

Chapter 3

Real-time Search Algorithms

In this chapter, we discuss several recent real-time heuristic search algorithms and the historical foundation of those algorithms. We also define hill-climbability, and present the algorithm for checking hill-climbability which is used throughout this document.

Real-time heuristic search algorithms operate under enforced time constraints on their decision making. This limit is by definition a constant independent of search space size. Every search decision by a real-time agent must be made within this fixed amount of time. A real-time agent typically interleaves phases of *planning* and *execution* to traverse from the start state to a goal state. In a planning phase, the agent has a bounded amount of time to select a single edge out of the current state. This is typically done by expanding a *frontier*, a collection of nearby states, for consideration. In an execution phase, the agent moves along the selected edge to the neighbour state. The current state is updated to be the neighbour state, and a new planning phase begins.

A common approach among the algorithms we discuss is to augment online performance with the offline pre-computation of a search-space-specific database. These databases are usually used to provide one or several intermediate goals, or *subgoals*, for use during search. When an appropriate subgoal is found, search is directed towards that state rather than the original global goal. This approach often improves search performance, since heuristic accuracy is typically higher for states which are closer together.

All of the real-time heuristic search algorithms that we discuss are suboptimal (i.e., do not guarantee optimal solutions). We refer to the class of algorithms that are not real-time as *conventional* heuristic search methods. Popular search methods in this class include both optimal algorithms such as *A** [14] and *IDA** [24], and suboptimal algorithms such as *weighted-A** and *HPA** [6]. These search methods typically perform by finding an entire solution to a problem offline before an agent executes the corresponding plan online. This document does not focus on conventional heuristic search, but some methods are discussed briefly.

3.1 LRTA*: The Foundation

Learning real-time A* *(LRTA*)* [23] is the first real-time heuristic search algorithm we discuss. LRTA* serves as the foundation for three of the subsequent real-time search algorithms discussed in this chapter.

Pseudo-code for LRTA* is given as Algorithm 1. The agent begins by initializing its current location s to the start state s_{start} (line 1). Next, a *frontier* of sucessor states is generated surrounding s (line 3). This frontier includes all states reachable via a maximum of d transitions from s. d is defined as the *frontier-depth*, and is given as a constant to LRTA*.

Once the frontier is generated, the agent selects the most promising state, s' (line 4). The chosen state is that which minimizes the function $g(s') + h(s')$, where $g(s')$ is the cost of an optimal path from s to s' within the search frontier. To deter state revisitation, the heuristic value $h(s)$ is updated to $g(s, s') + h(s')$ (line 5). The agent then moves its position one step along the path towards s' (line 6), and a new frontier is expanded. These steps are repeated until the agent's current state matches the goal state.

Despite the learning step at line 5, LRTA* is prone to frequent state revisitation. This tendency has been named *scrubbing* [7], since the agent appears to "scrub" back and forth over small regions of the search space to fill in heuristic depressions. Scrubbing behaviour is detrimental for two major reasons. First, state revisitation necessarily increases suboptimality of solutions. Second, scrubbing in applications such as videogame pathfinding is visually unappealing and reduces player immersion. This is a major barrier preventing LRTA* from being applied in commercial videogames.

Algorithm 1 LRTA*($s_{\text{start}}, s_{\text{goal}}, d$)

1: $s \leftarrow s_{\text{start}}$
2: **while** $s \neq s_{\text{goal}}$ **do**
3: generate successor states of s up to d edges away, generating a frontier
4: find a frontier state s' with the lowest $g(s, s') + h(s')$
5: $h(s) \leftarrow g(s, s') + h(s')$
6: change s one step towards s'
7: **end while**

3.1.1 Hill-climbing

We define a *hill-climbing* (HC) agent as a greedy LRTA*-like agent which performs no heuristic updates and only uses the immediate neighbours of the current state to build the search frontier. Hill-climbing is not complete. Search is terminated if the agent ever reaches a state with a heuristic value less than or equal to all surrounding states. To detect search problems where one state is HC-reachable from another, that is, where a hill-climbing agent will find a solution, we use Algorithm 2 [8]. Note that this algorithm does not guarantee that the HC path will be optimal, but only that it can be found by a hill-climbing agent.

Algorithm 2 HC-Reachable(s_1, s_2, b)

```
1: s ← s_1; i ← 0
2: while s ≠ s_2 and i < b do
3:    generate immediate successor states of s, generating a frontier
4:    if h(s) ≤ h(s') for all s' in the frontier then
5:       terminate search
6:    end if
7:    find a frontier state s' with the lowest g(s') + h(s', s_2)
8:    s ← s'; i ← i + 1
9: end while
```

3.2 D LRTA*

Dynamic LRTA* (*D LRTA**) [10] was designed to mitigate the scrubbing behaviour of LRTA*. Two improvements are made over the original LRTA*: dynamic selection of search depth d, and case-based subgoaling. Both of these ends are accomplished with the aid of a pre-computed database generated offline, before the D LRTA* agent is tasked with solving search problems online.

The first component of the D LRTA* database is a system that dynamically supplies a search depth d to the agent. D LRTA* proposes two methods for selecting this search depth. The first method relies on training a classifier that takes as input information about the heuristic function and recent search history, and returns a search depth as output. The alternative method builds an abstracted version, or *pattern database* of the state space. For each pair of abstract states, an optimal search depth is computed and stored in a lookup table. For reasons discussed in Chapter 6, our implementation of D LRTA* uses a fixed search depth of 1 rather than one of these two dynamic approaches proposed by Bulitko et al.

Algorithm 3 $D \leftarrow$ build DLRTA*(ℓ)

```
1: apply a clique abstraction to the search space ℓ times
2: for every abstract state a_i do
3:    s_i ← representative state for a_i
4:    run Dijkstra's algorithm from s_i
5:    store shortest path to all other representative states
6: end for
7: for every state s_i representative of a_i do
8:    for every state s_j representative of a_j do
9:       s_subgoal ← first state on shortest path from s_i to s_j outside of a_i
10:      store s_subgoal for (a_i,a_j) in D
11:   end for
12: end for
```

The second component of the D LRTA* database provides dynamic subgoals. Pseudo-code for building this component is given in Algorithm 3. First, a clique-abstraction is applied to the search space ℓ times (line 1). The result is that every ground-level state in the search space is mapped to a corresponding abstract state. Every abstract state a_i is also given a single representative state s_i in the original search space. Since not all of the search spaces we examine are map-based, a random

ground-level state is chosen as the representative for each abstract state, rather than the centroid ground-level state.

After building the abstraction, Dijkstra's algorithm [12] is used to find the optimal path between every pair of representative states (lines 2-6). Finally, for each of these paths from s_i to s_j, we find the first state on the path that moves into a new abstract state (line 9). This first state, s_{subgoal}, is selected and stored as the subgoal for the given pair of abstract states, a_i and a_j (line 10). Whenever a D LRTA* agent is in abstract state a_i and is searching for a goal in abstract state a_j, the subgoal s_{subgoal} is used.

The foundation for the online component of D LRTA* (Algorithm 4) is very similar to the original LRTA*. The principal changes are at line 3. Rather than expanding a frontier to a static depth towards the global goal $s_{\text{global goal}}$, the agent consults the database D for an appropriate depth d and subgoal s_{subgoal}. Search is then directed towards s_{subgoal} rather than towards $s_{\text{global goal}}$. An example of D LRTA* in action is given in Figure 3.1.

Algorithm 4 D LRTA*($s_{\text{start}}, s_{\text{global goal}}, D$)

1: $s \leftarrow s_{\text{start}}$
2: **while** $s \neq s_{\text{goal}}$ **do**
3: select search depth d and goal s_{subgoal} from database D
4: generate successor states of s up to d steps away, generating a frontier
5: find a frontier state s' with the lowest $g(s, s') + h(s', s_{\text{subgoal}})$
6: $h(s, s_{\text{subgoal}}) \leftarrow g(s, s') + h(s', s_{\text{subgoal}})$
7: change s one step towards s'
8: **end while**

3.3 kNN LRTA*

The next database-driven real-time algorithm we discuss is k-nearest neighbour LRTA* (*kNN LRTA**) [8]. The primary advantage kNN LRTA* offers over D LRTA* is that it does not require a complete enumeration of all states in the search space like D LRTA* does. Rather than partitioning the search space into abstract regions, kNN LRTA* creates database records for random pairs of states across the search space. Online, the agent selects the nearest, or most appropriate, database entry for the search problem at hand.

The offline construction of a kNN LRTA* database proceeds according to Algorithm 5. The construction is parameterized by N, the number of desired entries in the database. To create one of the N records, we first select a pair of random states $(s_{\text{start}}, s_{\text{goal}})$ (line 3). An optimal path p from s_{start} to s_{goal} is found using A* . If no solution exists, or if the solution has length less than 3, a new random pair of states is selected. Otherwise, we generate a database entry Γ_p by compressing p into a chain of subgoals (lines 6-7). This process is repeated for each of the N records. Example offline database construction by kNN LRTA* is demonstrated in Figure 3.2.

The compression of an optimal solution into a chain of subgoals is performed using Algorithm

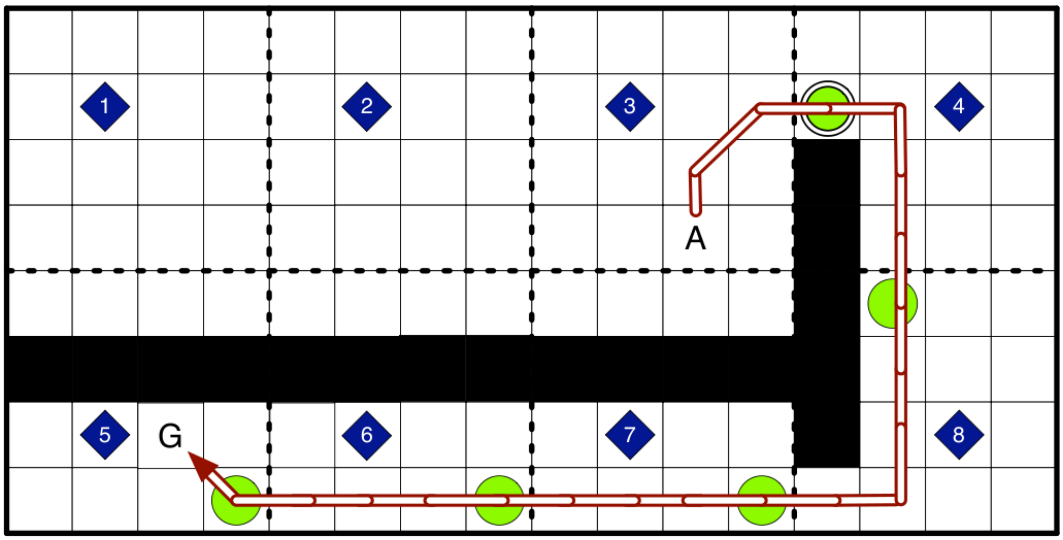

Figure 3.1: Sample execution of D LRTA*. A is the current location of the D LRTA* agent and G is the goal state. Abstract states are the groups of cells separated by dashed lines. Representative states are demarked by blue diamonds. Subgoal states that will be used along the red path are indicated by green circles. From its current position, the D LRTA* agent will search for the subgoal stored for the pair of abstract states $(3, 5)$, indicated with a double outline. Figure taken from Bulitko et al. [10].

6. The subgoals are selected from the states $(s_1, ...s_t)$ in a path p such that each subgoal is reachable from the previous one via simple hill-climbing. The first and final states are automatically included in the chain of subgoals. γ is the set of indices of states included in the chain of subgoals. Beginning one index beyond the last subgoal added to γ, we search for the furthest index i which is HC-reachable from the last subgoal. This is achieved via a binary search. The indices l and r track a decreasing window on the set of candidate indices. Through repeated HC-reachability checks (line 8), we repeatedly decrease the size of the window by half, until only a single state i remains. We then add i to γ. This is repeated until the index t of the final state s_t is added to γ (line 2).

The online component of kNN LRTA* is presented in Algorithm 7. Much like D LRTA*, kNN

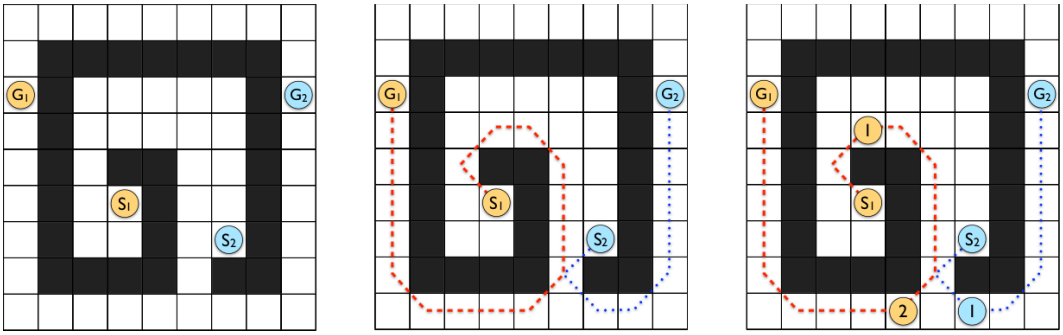

Figure 3.2: Demonstration of the offline component of kNN LRTA*. **Left:** Two random pairs of states are selected. **Centre:** The optimal path is found between the pairs of states. **Right:** The optimal paths are compressed into chains of subgoals. Figure taken from Bulitko et al. [8]

Algorithm 5 build kNN LRTA*(N)

```
1: subgoal database ← ∅
2: for n = 1, ..., N do
3:    generate a random pair of states (s_start, s_goal)
4:    compute an optimal p from s_start to s_goal with A*
5:    if p ≠ ∅ and |p| ≥ 3 then
6:       Γ_p ← compress(p)
7:       add Γ_p to the subgoal database
8:    end if
9: end for
```

Algorithm 6 $\Gamma \leftarrow$ compress($(s_1, ..., s_t)$)

```
1: γ ← (1)
2: while t ∉ γ do
3:    i ← end(γ) +1
4:    l ← i + 1
5:    r ← t
6:    while l ≤ r do
7:       m ← ⌊(l+r)/2⌋
8:       if HC-Reachable(s_end(γ), s_m) then
9:          i ← m
10:         l ← m + 1
11:      else
12:         r ← m − 1
13:      end if
14:   end while
15:   γ ← γ ∪ (i)
16: end while
17: Γ ← s_γ
```

LRTA* is an adapted form of LRTA*. However, rather than receiving a single subgoal at a time from the database, kNN LRTA* receives a stack of subgoals r (line 5). If no subgoal is available because no suitable record was found, $s_{\text{global goal}}$ is placed in r. Every time the agent's location is updated, kNN LRTA* compares the new location s to the top of r. If the agent has reached s_{subgoal} (or if $s_{\text{global goal}}$ is being used) the top entry in r is discarded (line 13). If $s_{\text{global goal}}$ was being used, kNN LRTA* consults the database for a new subgoal in the next iteration of the planning phase (lines 4-6).

Algorithm 8 details the online method for selecting the most appropriate chain of subgoals based on the agent's current location s and the goal $s_{\text{global goal}}$. First, the records in the database are sorted in ascending order by their dissimilarity to the current search problem (line 1). The dissimilarity of record r_i is defined as $\max(h(s, s_{\text{first}}), h(s_{\text{last}}, s_{\text{global}}))$, where s_{first} and s_{last} are the first and last states respectively in the chain of states constituting record r_i. We then iterate through the sorted list of records until we find one where s_{first} is HC-reachable from s, and $s_{\text{global goal}}$ is HC-reachable from s_{last} (lines 2-8). The first such record is returned for use by kNN LRTA*. If no such record is available, search defaults to the global goal (line 9). Notice that only the first M records are

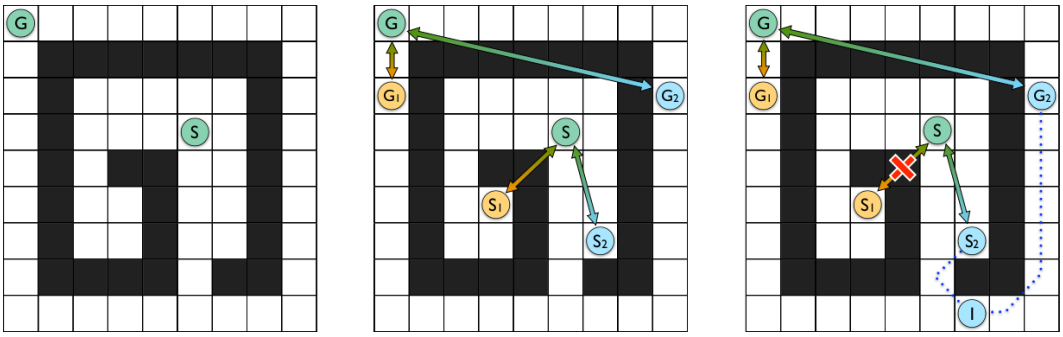

Figure 3.3: Demonstration of the online component of kNN LRTA*. **Left:** A kNN LRTA* is tasked with finding a path from S to G. **Centre:** The two subgoal records built in Figure 3.2 are considered for use. (S_1,G_1) is ranked as more similar than (S_2,G_2) to (S,G). **Right:** S_1 is not hill-climbing reachable from S. (S_2,G_2) is the chosen database record. Figure taken from Bulitko et al. [8]

Algorithm 7 kNN LRTA*(s_{start}, $s_{\text{global goal}}$, d)

$s \leftarrow s_{\text{start}}$
$r \leftarrow \emptyset$
while $s \neq s_{\text{goal}}$ **do**
  if $r = \emptyset$ **then**
    $r \leftarrow$ knnLRTAchoose(s, $s_{\text{global goal}}$)
  end if
  $s_{\text{subgoal}} \leftarrow$ top of r
  generate successor states of s up to d steps away, generating a frontier
  find a frontier state s' with the lowest $g(s, s') + h(s', s_{\text{subgoal}})$
  $h(s, s_{\text{subgoal}}) \leftarrow g(s, s') + h(s', s_{\text{subgoal}})$
  change s one step towards s'
  if top of $r = s$ **or** top of $r = s_{\text{global goal}}$ **then**
    pop from r
  end if
end while

considered for selection. $M < N$ is a constant chosen to increase the online speed of kNN LRTA*, since HC-reachability checks are computationally expensive. Example online operation of kNN LRTA* is demonstrated in Figure 3.3.

3.4 HCDPS

Both kNN LRTA* and D LRTA* are prone to the same scrubbing behaviour as LRTA*, either due to unavailable or inappropriate subgoals. Scrubbing can be viewed as a side effect of the sometimes slow heuristic learning process that these algorithms perform. Hill-Climbing Dynamic Programming Search (HCDPS) aims to avoid scrubbing by removing this learning component, and by guaranteeing the availability of a subgoal record [25].

Similar to the previous two algorithms, HCDPS constructs a database offline to inform search decisions. The first step in this database construction involves partitioning the entire search space into a set of abstract regions called *HC regions*. Each HC region is a set of states, with one state

Algorithm 8 $r \leftarrow$ knnLRTAchoose$(s, s_{\text{global goal}})$

1: $(r_1, ..., r_N) \leftarrow$ database records from most to least similar
2: **for** $i = 1, ..., M$ **do**
3: retrieve $r_i = (s_{\text{first}}, ..., s_{\text{last}})$
4: **if** HC-Reachable(s, s_{first}) and HC-Reachable$(s_{\text{last}}, s_{\text{global goal}})$ **then**
5: $r \leftarrow r_i$
6: return
7: **end if**
8: **end for**
9: $r \leftarrow (s, s_{\text{global goal}})$

Figure 3.4: HCDPS partitioning of a videogame map. Figure taken from Lawrence et al. [25]

designated as the *seed state*. Every state in a HC region is mutually HC-reachable with the seed state. In practice, the required HC checks are constrained to a constant maximum distance. An example partition of the search space by HCDPS is presented in Figure 3.4. The second step in the database construction involves computing a path between the representative seed states of each pair of HC regions. This path is then converted into a chain of subgoals and stored as a database record.

Pseudo-code for the partitioning scheme is presented in Algorithm 9. The partitioning P is initialized to the empty set, and $S_{\text{unpartitioned}}$ is initialized as the set of all states in the search space (lines 1-2). While states remain that have not been assigned to a HC region in P, we select a random state s_{seed} from $S_{\text{unpartitioned}}$ and designate it as a new seed state. We then grow a *maximal* HC region R from s_{seed} via breadth first search (line 5). An HC region is maximal when no adjacent states can be added to the region, either since they are not mutually HC-reachable to the seed state, or because of the bound on the HC distance. Only states remaining in $S_{\text{unpartitioned}}$ are eligible to be added to R. s_{seed} and R are recorded to the partition P (line 7) and the loop continues.

The procedure for generating a subgoal database from the search space partitioning is presented in Algorithm 10. HCDPS does not calculate the true optimal path between every pair of represen-

Algorithm 9 $P \leftarrow$ partition(S)

1: $P \leftarrow \emptyset$
2: $S_{\text{unpartitioned}} \leftarrow S$
3: **while** $|S_{\text{unpartitioned}}| > 0$ **do**
4: select a random state s_{seed} from $S_{\text{unpartitioned}}$
5: $R \leftarrow$ maximal HC region grown from s_{seed}
6: $S_{\text{unpartitioned}} \leftarrow S_{\text{unpartitioned}} - R$
7: $P \leftarrow P \cup (s_{\text{seed}}, R)$
8: **end while**

tative seed states. Instead, we use dynamic programming to assemble composite paths by chaining together optimal paths between seed states of neighbouring HC regions. The resulting algorithm is essentially an adapted version of the Floyd-Warshall algorithm which finds a path between every pair of seed states in the partition P. First, HCDPS finds the optimal path between seed states of adjacent HC regions (line 3), and compresses those paths (line 4) using the method in Algorithm 6. Once this is done for all pairs of neighbouring HC regions, the paths between all other HC regions are constructed using dynamic programming (lines 7-13).

Algorithm 10 $D \leftarrow$ generateSubgoals(P)

1: **for** each seed state s_i in P **do**
2: **for** each seed state s_j of an adjacent HC region **do**
3: path $\leftarrow$ optimal path from s_i to s_j using A*
4: subgoals$[i][j] \leftarrow$ compress(path)
5: **end for**
6: **end for**
7: **for** k from 1 to $|P|$ **do**
8: **for** i from 1 to $|P|$ **do**
9: **for** j from 1 to $|P|$ **do**
10: subgoals$[i][j] \leftarrow$ the shorter of subgoals$[i][j]$ and subgoals$[i][k]$ + subgoals$[k][j]$
11: **end for**
12: **end for**
13: **end for**

The online HCDPS agent is presented in Algorithm 11. It is similar to the kNN LRTA* agent presented in Algorithm 7 with a few exceptions. The first difference is that subgoals are guaranteed to be available. To select an appropriate database record for a search problem, we simply look up the assigned HC regions for the start and goal state pair (line 2). The second difference is that a search depth of 1 is always used when expanding a search frontier (line 5). The final difference is the elimination of the learning step. Since HCDPS provides subgoals that can guide an agent to any goal via hill-climbing alone, there is never a need to update heurisic values.

3.5 TBA*

The final real-time algorithm we discuss in this document is *time-bounded A** (TBA*) [3]. Unlike the previous three algorithms discussed, TBA* does not make use of a subgoal database. However,

Algorithm 11 HCDPS($s_{\text{start}}, s_{\text{global goal}}, D$)

```
1: s ← s_start
2: r ← subgoal stack for (s_start, s_global goal) from D
3: while s ≠ s_goal do
4:    s_subgoal ← top of r
5:    generate successor states of s, generating a frontier
6:    find a frontier state s' with the lowest g(s, s') + h(s', s_subgoal)
7:    s ← s'
8:    if s = top of r then
9:       pop from r
10:   end if
11: end while
```

recent literature has included TBA* in the discussion of state of the art real-time heuristic search [25], and so we include it in our research.

Pseudo-code for TBA* is presented in Algorithm 12. Similar to the above LRTA*-based algorithms, TBA* interleaves planning (lines 6–20) and execution (lines 21-31). The planning phase makes use of a series of resource-limited iterations of conventional A* (line 7). In each iteration, only a fixed number of node expansions are permitted, as parameterized by R. However, the open and closed lists are stored in the variable L between iterations. This specialized A* search is performed in every planning phase until a goal state is found.

Next, the most promising state (that with the lowest A* f cost) is selected from L (line 11). TBA* then traces the current lowest-cost path from the most promising state back to the start state s_{start} (line 13). This is performed by repeatedly consulting a table of "next hop" pointers stored by the specialized A* agent. This tracing is also resource-limited, and may be split between iterations of TBA* if necessary. When a trace is completed, TBA* has a new plan P_{follow} for the agent to follow in the execution phase (line 15). In the next iteration, TBA* will then begin tracing another path from the new most promising state. This process of building a new best path for TBA* to follow is repeated until a complete path to s_{goal} is found.

In the execution phase, the agent attempts to move along the current P_{follow}. However, if the agent is not currently on the path P_{follow}, it will instead take a step back towards s_{start} (line 25). Eventually, the agent's backtracing will intersect with P_{follow}, and it can resume moving towards the goal. In the worst case, this does not happen until the agent has returned all the way to s_{start}.

Sample execution of TBA* is depicted in Figure 3.5. In this example, the TBA* agent is beginning at state S and searching for state G. The cloud-shaped rings represent the states on the open list after three successive planning phases. The darkly colored states labeled a, b and c are the most-promising states after each planning phase, and the dotted lines represent the paths traced back to S. In the first two execution phases, the agent moves from S to 1 and from 1 to 2. In the third execution phase, the agent moves back to 1, trying to reach the new path leading from S to c.

Our implementation of TBA* includes enhancements presented by the original authors. Rather than always backtracing along the former path until an intersection with the new path is found, TBA*

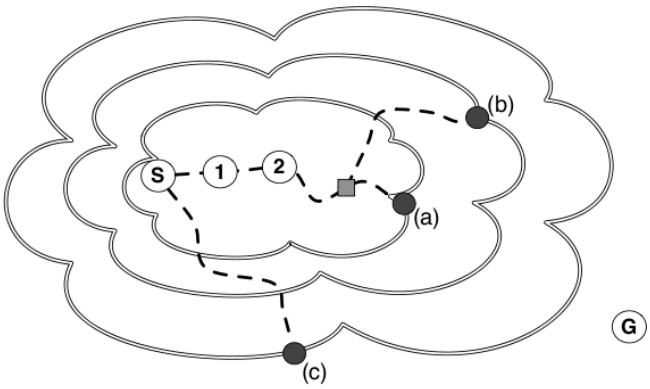

Figure 3.5: An example of TBA* execution. Figure taken from Björnsson et al. [25]

will perform a limited A* search for a shortcut to the new path. This is done by initializing the open list of the limited A* agent with the g values associated with the new path and searching backwards towards the agent's current location. Another enhancement mitigates the backtracking associated with frequently switching to a new more promising path. The agent only elects to begin following a new path when the cumulative cost of the path is at least as high as that of the current path.

Algorithm 12 TBA*($s_{\text{start}}, s_{\text{goal}}, R$)

```
1:  solutionFound ← false
2:  solutionFoundAndTraced ← false
3:  doneTrace ← true
4:  s ← s_start
5:  while s ≠ s_goal do
6:    if not solutionFound then
7:      solutionFound ← A*(L, s_start, s_goal, R)
8:    end if
9:    if not solutionFoundAndTraced then
10:     if doneTrace then
11:       P_new ← L.mostPromisingState()
12:     end if
13:     doneTrace ←traceBack(P_new, s, R)
14:     if doneTrace then
15:       P_follow ← P_new
16:       if P_follow.back() = s_goal then
17:         solutionFoundAndTraced ← true
18:       end if
19:     end if
20:   end if
21:   if P_follow.contains(s) then
22:     s ← P_follow.popFront()
23:   else
24:     if s ≠ s_start then
25:       s ← L.stepBack(s)
26:     else
27:       s ← s_prev
28:     end if
29:   end if
30:   s_prev ← s
31:   move agent to s
32: end while
```

Chapter 4

Related Work

There has been significant past research on analyzing search space complexity and prediction of search algorithm performance. A modest subset of this work has focused on real-time heuristic search in particular. In this section we explore some of this related work.

4.1 Real-time Heuristic Search Analysis

Koenig first presented motivation for analysis of real-time search performance [22]. Koenig indicated that, unlike conventional search methods, real-time search was poorly understood. As a preliminary effort, he discussed the impact of domain, heuristic and algorithm properties on search behaviour, noting that real-time search and conventional search are affected quite differently by these factors. Specifically, Koenig stated the following:

> In general, [...] not much is known about how domain properties affect the plan-execution time of real-time search methods, and there are no good techniques yet for predicting how well they will perform in a given domain. This is a promising area for future research.

Similarly, Koenig stated that there were no strong methods for predicting the comparative performance of multiple different real-time search algorithms on a given planning task. As an example, he compares the disparate performances of LRTA* and the similar algorithm *Node Counting* on a selected set of domains. Koenig observed that, despite comparable typical case performance over thousands of trials, worst case solution costs for Node Counting are substantially more expensive than for LRTA*. Furthermore, he indicated the difficulty of predicting these degenerate cases of Node Counting.

As stated, Koenig's early analysis of real-time search was limited to LRTA* and select variants. We seek to extend analysis to the more contemporary class of database-driven algorithms discussed in Chapter 3. The major motivations that we take from Koenig's work are that real-time search behaviour differs greatly not only from conventional search, but also among different real-time algorithms and search spaces. Therefore, our system for characterizing algorithm performance

discussed in Chapter 5 is designed specifically with properties of contemporary real-time algorithms in mind.

Bulitko and Lee performed a large scale analysis of numerous real-time heuristic search algorithms, including LRTA*, ϵ-LRTA*, SLA* and γ-trap [9]. They developed LRTS, a unified framework for these four algorithms, and performed a large scale empirical study across several search spaces. As motivation, they cited the present difficulty of appropriately selecting algorithms and parameters from the available pool.

Four of the five real-time search algorithms that we explore in detail have been developed in the time after the work of Bulitko and Lee. We therefore find ourselves again faced with a similar problem of an abundance of algorithms of which the relative merits have only been briefly explored in a small selection of search spaces [25]. While we do not take the approach of combining the algorithms into a single framework, we do share the motivation of facilitating algorithm selection and parameterization.

4.2 Characterizing Search Spaces

We root our efforts to analyze real-time search performance in building an understanding of search space features that influence search behaviour. This is largely motivated by the disparate performance in initial experiments applying recent real-time methods to a wider variety of search spaces [20]. Understanding search spaces can serve two important purposes. First, we can more make more informed decisions when selecting or designing algorithms for a given search space. Second, we can more consistently compare the performance of new algorithms by establishing benchmark search spaces with well understood characteristics.

Two of the features that we present in Chapter 5 are derived from the work of Ishida. Ishida identified that given the necessarily commital behaviour of real-time search algorithms, they are more susceptible to local heuristic topography than conventional search methods [21]. To measure this topography empirically, Ishida provides the following definitions:

> A *heuristic depression* is a set of connected states with heuristic values less than or equal to those of the set of immediate and completely surrounding states. A heuristic depression is *locally maximal*, when no single surrounding state can be added to the set; if added, the set does not satisfy the condition of a heuristic depression. [21]

As alluded to in Chapter 3, heuristic depressions can cause scrubbing to occur in LRTA*-based search methods. Ishida performed experiments that enumerated heuristic depressions in mazes and sliding tile puzzles. He also provided intuition as to how these search space features might affect real-time search performance in terms of the number of heuristic updates performed.

Ishida also conducted an empirical comparison of the performance of LRTA* and two variants, *Real-Time A** (RTA*) [23] and *Local Consistency Maintenance* (LCM) [29]. The analysis focused

on comparing the learning efficiency of these algorithms when multiple trials are allowed (i.e., as the heuristic converges) or when different initial heuristic functions are used.

More recently, Hoffman extended a similar study of heuristic topology[1] to general planning [17] [18]. On a set of 13 well-established planning benchmarks, Hoffman enumerated topological heuristic features and domain properties to sort the benchmarks into a complexity taxonomy. This is a somewhat similar approach to our ranking of search spaces based on complexity in Chapters 6 and 8, although Hoffman does not compute empirical correlations to performance. Hoffman concluded that the benchmark taxonomy would shed insight on the relative levels of success of heuristic search planning in these benchmarks. He also claimed that it could inform subsequent improvement of heuristic functions, allow prediction of planning performance and assist in developing more formally challenging benchmark problems.

Another of the complexity measures we use to characterize search spaces is inspired by the research of Mizusawa and Kurihara [28]. They successfully demonstrated a strong link between search performance in gridworld pathfinding domains and two "hardness measures": initial heuristic error and probability of solution existence. They define initial heuristic error for a search problem as

$$E = \sum_{s \in S'} h^*(s) - h_0(s)$$

where S' is the set of all states on some path between connecting the start and goal states. [2]

The search spaces used by Mizusawa and Kurihara are generated randomly by making random cells in the gridworld untraversable. The percentage of untraversable cells is called the *obstacle ratio*. Since the obstacles are placed randomly, solutions are not guaranteed to exist for search problems. The entropy

$$H = -p \log_2 p - (1-p) \log_2 (1-p)$$

is used as their measure of likeliness of solution existence.

Mizusawa and Kurihara demonstrated that E, H, and LRTA* and RTA* solution cost are all maximized at a similar obstacle ratio of approximately 41% in square, randomly generated maps. Unlike Mizusawa and Kurihara, the search problems we consider are guaranteed to have solutions. However, their system of using complexity measures to characterize search spaces with respect to search performance was informative to our own research.

Rayner et. al recently examined dimensionality as an instrinsic property of search spaces [30]. They found that by considering the dimensionality of a search space, they were able to gain insights as to what classes of heuristic function would be appropriate for that search space. As future work, we would like to incorporate dimensionality into the selection of complexity measures we present in Chapter 5.

[1] The terms topography and topology are used interchangeably in the literature when discussing heuristic functions.

[2] In a fully connected search space, S' is equivalent to S, the set of all states.

4.3 Predicting Search Performance

Previous work has been conducted to predict the performance of simpler real-time heuristic search algorithms. Citing important applications in planning (e.g., as part of the Heuristic Search Planner [5] and the Fast-Forward Planner [19]), Lòpez sought to model the efficiency of heuristic hill-climbing by modelling the algorithm as a Markov process [26]. Using several sizes of the sliding tile puzzle, the model could reasonably predict the likelihood that a hill-climbing agent reaches a target state in a number of moves equal to the initial heuristic value. Lòpez states that this model is useful not only as a predictor of simple real-time search performance, but as a gauge of heuristic accuracy.

One of the complexity measures we present in Chapter 5 bears similarity to Lòpez's work. We consider the probability with which a hill-climbing agent successfuly reaches a target state. Unlike Lòpez, we do not differentiate between cases where the path taken by the agent is more or less expensive than the heuristic estimate.

Chapter 5

Complexity Measures

To prepare for the application of recent real-time heuristic search to new domains, we sought to first find a method of empirically characterizing the challenges that new search spaces would bring. It is suspected intuitively that a maze is more "complex" than an open room for an agent to navigate, but this notion of complexity is not as evident in search spaces which are not easily visualized.

Our goals for this research are threefold. We first seek to facilitate algorithm selection. Second, we wish to build an understanding of search space features that will inform subsequent algorithm development. Third, we wish to support an improved system of characterizing the difficulty of benchmark problems used in real-time search experimentation.

In this chapter we present a set of complexity measures that are used to quantify the innate features of a search space. The complexity measures are specifically designed to assess the suitability of existing real-time search algorithms to a given search space. We also discuss how these measures should be computed in practice.

5.1 Domain-Independent Complexity Measures

This work uses a set of eight domain-independent complexity measures [20]. All of the measures are calculated independently of any algorithm, and may thus be useful for assessing the suitability of several different algorithms for a particular search space. For presentational clarity, we discuss the measures as they are calculated for search spaces with a single goal state. However, all of the measures could be easily adapted to serve search spaces with multiple goals.

1. **HC Region Size** - the mean number of states per abstract region when the search space is partitioned using the abstraction method of HCDPS. This abstraction method is presented in Algorithm 9 in Section 3.4.

2. **HC Probability** - the probability that a randomly selected state is HC-reachable from another randomly selected state. HC-reachability is checked using Algorithm 2 from Section 3.1.1.

3. **Scrubbing Complexity** - the mean number of visits among all states receiving at least one visit in the solution returned by an LRTA* agent. This measure is intended to model real-time search behaviour when no sub-goals are made available to the agent.

4. **Path Compressibility** - the mean number of subgoal states when the solution to a random search problem is compressed using Algorithm 6.

5. **A*-Difficulty** - the mean number of states on the A* closed list after solving a random search problem, scaled by the length of the solution. This measure has been previously used to measure the complexity of search problems for conventional search.

6. **Heuristic Error** - Adapted from [28]. Heuristic error is the average cumulative difference in value between h_0 and h^* across all reachable states sampled over a random set of goal states.

7. **Total Depression Width** - The mean number of depressed states for a random goal state. States within the heuristic depression containing the goal state are excluded from consideration, since that depression is not inhibitive for real-time search. This measure is intended to model the likelihood that a real-time agent will become temporarily trapped during search.

8. **Depression Capacity** - The mean sum of depths across all depressed states for a random goal state. Again, the heuristic depression containing the goal state is not considered. This measure is intended to model not only the likelihood for a real-time agent to become temporarily trapped, but also the duration for which it will be trapped.

5.2 Computing Complexity Measures in Practice

There are two major considerations when computing the values of these measures for a search space. First, we must make sure that we are sampling across a sufficient number of goal states or search problems for the measured value to be representative of the search space. Second, we must calculate the measures in a way that avoids introducing bias towards search space size. In this section we discuss how we address both concerns.

5.2.1 Sufficient Sampling

To determine an appropriate sample size to calculate each measure, we perform repeated sampling until the current observed mean becomes relatively stable. As an aid we used *stability graphs*, which plot the current sample number against the current observed mean. After a certain number of samples, we observe a decrease in fluctuation of the current mean value. This number of samples is then used to compute that complexity measure in practice. A complete set of eight stability graphs for a sample search space is presented in Figure 5.1.

There is no constant sample size that will be sufficient for all search spaces. However, this method serves as a reasonable guide for selecting appropriate sample sizes. Larger search spaces,

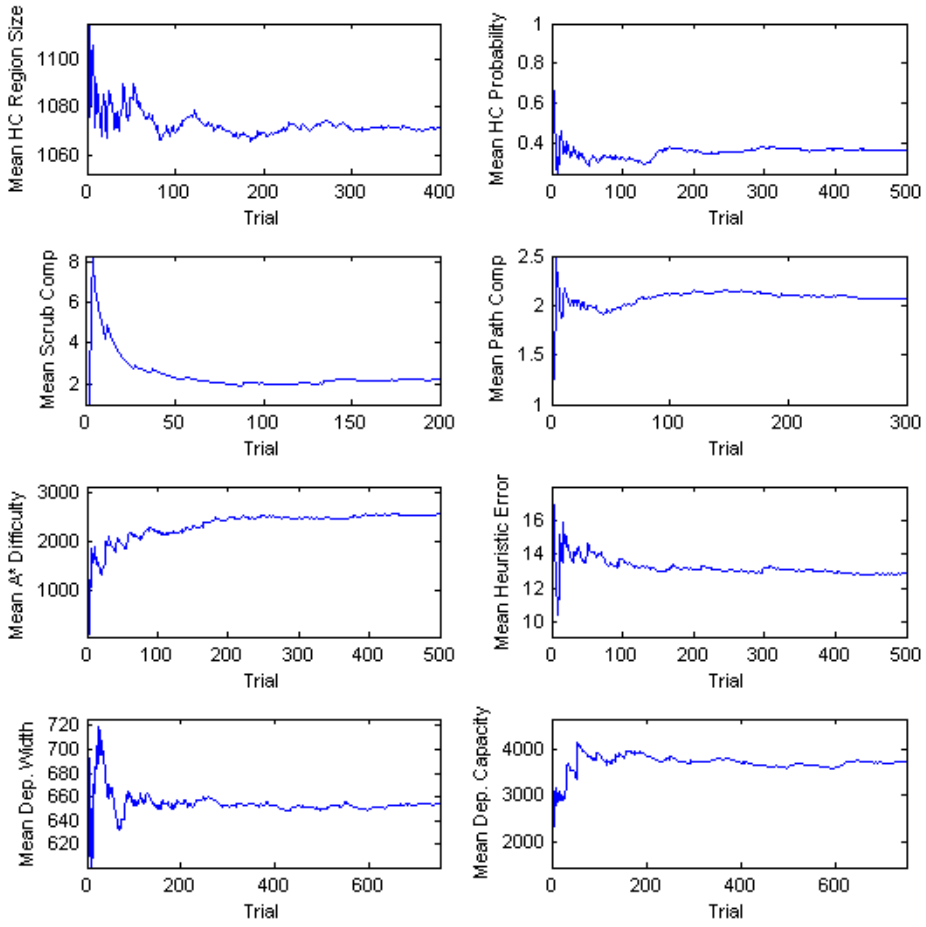

Figure 5.1: Stability plots for a sample videogame pathfinding search space.

and those with diverse regional complexity, will require a larger number of samples. Luckily, many common search spaces exhibit a similar degree of complexity across the whole search space. As an example, see Figure 5.2. This figure depicts a small Towers of Hanoi search space with 5 pegs and 3 discs. Due to symmetries in the search space definition, large sections of the search space are repeated, reducing the number of samples that we will require to measure the complexity.

5.2.2 Localized Complexity

The ultimate purpose of the complexity measures is to allow efficient comparison of multiple search spaces. Our aim is to compare these search spaces on the basis of local complexity. To ensure that the measures are effectively capturing local complexity rather than being confounded by search space size, we can constrain the calculation of the complexity measures to a fixed portion of the search space. To accurately reflect the entirety of the search space, we can then perform repeated

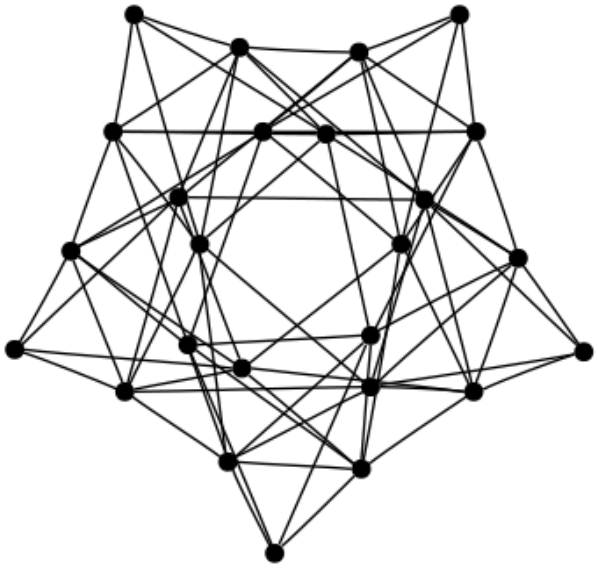

Figure 5.2: A Towers of Hanoi search space. Notice the large amount of symmetry in the search space.

sampling across several such random portions of the search space. The random sub-spaces can be generated via a bounded breadth-first search originating at a randomly selected state. After repeating the sampling across a variety of bounded regions, we compute the aggregate complexity measure as the mean of the collected measure values for the sampled regions. It is important to ensure that the subspaces are of the same size for all of the search spaces being compared in this way.

Aside from preventing a bias to search space size, this sampling technique can improve the efficiency of calculating the complexity measures. This is of particular importance when applying the measures to large search spaces. For example, calculating scrubbing complexity on a large video game map can be very expensive, since solving even a single instance of LRTA* takes a non-trivial amount of time. We can instead solve a larger number of LRTA* problems in the smaller sub-spaces.

The final benefit of this technique is that it allows the complexity measures to be computed for implicitly defined search spaces which are too large to fit in memory. Instead of expanding the entire search space at once, we expand only one bounded portion at a time. The complexity measures are computed for that bounded portion, and the process is repeated.

Chapter 6

Applications to Videogame Pathfinding

In this chapter we present experimental evidence linking the performance of real-time heuristic search algorithms on videogame pathfinding to the values of the complexity measures presented in Chapter 5. We begin by describing our selection of search spaces for experimentation. We then detail our methodology for measuring search performance.

We make use of two distinct methods for linking the complexity measures to search performance. The first method, which we explore in this Chapter, is to compute the rank correlation between the values of a complexity measure and the performance of an algorithm for a given search space. This method allows us to identify which search space features can significantly impact the performance of an algorithm. This is useful for understanding the limitations of current algorithms, and for gauging the relative difficulty of search spaces for an algorithm.

Our second method involves using machine learning to build predictive models of search space performance. This demonstrates that we can use the values of the complexity measures to assist in algorithm selection and parameterization. This second method is explored in Chapter 7.

6.1 Pathfinding as a Search Problem

Videogame pathfinding has been been the traditional testbed for the recent real-time heuristic search algorithms we examine. Pathfinding in commercial videogames is very resource limited, often limited to a specific amount of time (often less than 1 ms) [8]. It is therefore a natural application for real-time search. In this section we formally describe videogame pathfinding as a search problem as defined in Chapter 2.

A videogame pathfinding search space, or *map*, consists of a grid of cells that are either travsersable or untraversable. An untraversable cell is also referred to as an *obstacle*. An agent may move from its current cell to any cardinally or diagonally adjacent travserable cell. The cost of every cardinal transition is 1, and the cost of every diagonal transition is 1.4. We use *octile distance*

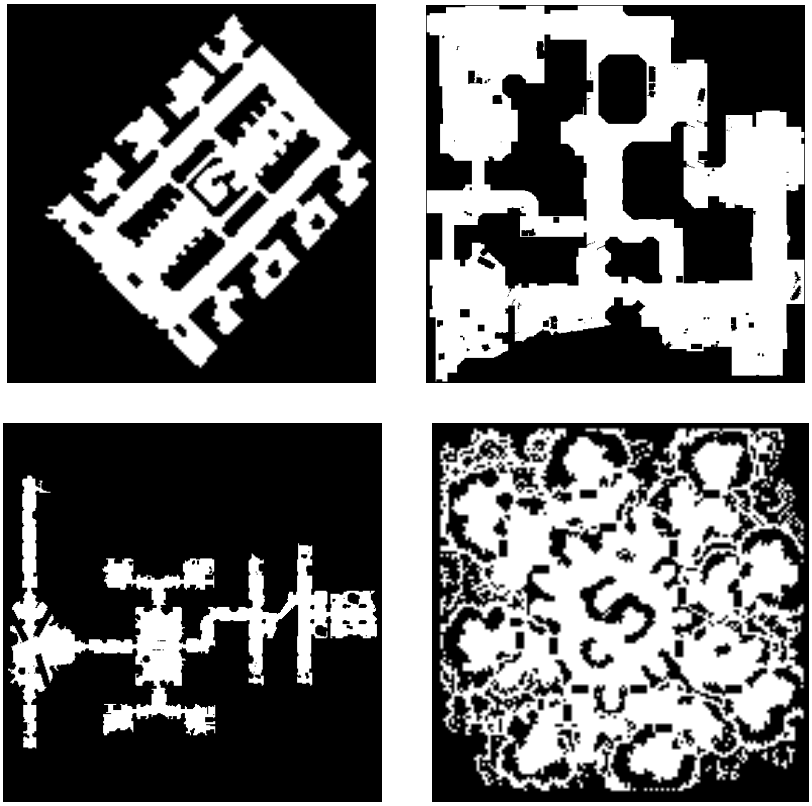

Figure 6.1: Sample maps from the video games (clockwise from top left) *Baldur's Gate*, *Counter-strike: Source*, *Warcraft 3* and *Dragon Age: Origins.*

as our heuristic function. The octile distance between states s_1 and s_2 is defined as

$$\text{octile}(s_1, s_2) = ||x_1 - x_2| - |y_1 - y_2|| + 1.4 \times \min(|x_1 - x_2|, |y_1 - y_2|)$$

where the state s_i is located in row x_i and column y_i of the grid. All of the videogame pathfinding problems we discuss have a single start state and a single goal state, although different problems on the same map may have differing start and goal states.

6.2 Experimental Design

To establish a quantifiable link between the complexity measures presented in Chapter 5 and the performance of real-time heuristic search, we first conducted experiments in pathfinding across a sample of video game maps. The experiments presented in this chapter are conducted across 20 video game maps, with 5 from each of *Baldur's Gate* [1], *Counter-strike: Source* [33], *Warcraft 3* [4] and *Dragon Age: Origins* [2]. Maps from the first three games have been previously used in recent real-time heuristic search literature [8, 25]. Most of these maps are available online through Nathan Sturtevant's Moving AI Lab [32]. Four example maps are presented in Figure 6.1. The complete set of maps is presented in Appendix A.

As discussed in Chapter 2, we consider two basic measures of a real-time search algorithm's performance: solution suboptimality and database computation time.[1] Both of these measures can differ vastly based on the properties of a search space.

To ensure that search suboptimality was not influenced by search space size, we first scaled all of the maps to approximately 50000 states. We then generated 10 sub-maps per map by randomly selecting samples of exactly 20000 states from each map. Sub-maps were generated via breadth-first search, originating at a random state in the search space. We treated each of these $20 \times 10 = 200$ sub-maps as a whole and distinct search space for our experiments. The intent of this sampling technique was to increase the number of available data points 10-fold, while retaining maps that were sufficiently similar by inspection to full-size game maps, and yet diverse enough to establish trends in search space complexity.

To compute the solution suboptimality of each algorithm, we generated a random set of 250 pathfinding problems on each sub-map, and solved each problem with each of the three algorithms. This number of problems was chosen to cover a sufficient sample of typical search problems, while balancing experimental time constraints. The problems were constrained to have solutions of length at least 10 to avoid the inclusion of trivially easy search problems.

6.2.1 Algorithm Implementation Details

In this chapter we examine five real-time heuristic search algorithms: LRTA*, D LRTA*, kNN LRTA*, HCDPA and TBA*. All of our algorithm implementations exist within a common framework. While this may reduce algorithm efficiency in some cases, it helps reduce any discrepencies in performance due to differences in tie-breaking procedures, data structures used etc.

In order to make the trends in algorithm performance most visible, algorithm parameters were chosen to yield a wide spread of performance. If algorithms are parameterized to produce the lowest cost solutions, then observed suboptimality tends towards 1, obscuring intrinsic differences in the search spaces. Therefore, all parameters were chosen so that very few problems would be solved optimally. LRTA* was run with a lookahead depth of $d = 1$. D LRTA* ran with $\ell = 5$. kNN LRTA* used $N = 1000$ and $M = 10$. HCDPS ran with $r = 1$. All on-line HC checks were limited to 250 steps. TBA* was run with a resource limit of $R = 5$.

6.3 Correlation

To empirically link the complexity measures to search performance, we calculate the Spearman rank correlation coefficients between the average solution suboptimality and a complexity measure's values [31]. The coefficient reflects the tendency for two variables to increase or decrease monotonically together, possibly in a non-linear fashion. The ability to gauge non-linear correlation was our main motivation for selecting Spearman correlation over other measures such as Pearson correlation.

[1] Since LRTA* and TBA* do not utilize subgoal databases they are omitted from our discussion of precomputation time.

Let X and Y be two sets of data. Let x_i be the rank, or position in descending order, of the i^{th} element in X. Define y_i analogously. Then Spearman correlation is defined as

$$\text{corr}(X, Y) = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}}$$

where $\bar{x}$ is the mean rank for all x_i, and $\bar{y}$ is the mean rank for all y_i.

We compute two sets of correlations: ρ_{mean} using *mean* suboptimality, and ρ_{median} using *median* suboptimality. The use of median suboptimality is to mitigate the impact of outliers: rare but vastly suboptimal solutions. Let $\bar{A}_i(m_k)$ and $\tilde{A}_i(m_k)$ be the mean and median solution suboptimality respectively when algorithm i is run on the set of 250 problems for sub-map k. Also let $C_j(m_k)$ be the value of complexity measure j for sub-map k.

$$\rho^{ij}_{\text{mean}} = \text{corr}([\bar{A}_i(m_1), ..., \bar{A}_i(m_{200})], [C_j(m_1), ..., C_j(m_{200})])$$

$$\rho^{ij}_{\text{median}} = \text{corr}([\tilde{A}_i(m_1), ..., \tilde{A}_i(m_{200})], [C_j(m_1), ..., C_j(m_{200})])$$

All correlation values that we report were computed using the MATLAB Statistics Toolbox [27].

For each pair of one of the five algorithms and one of the eight complexity measures, we compute the two sets of correlation coefficients using 200 data points (one for each sub-map). The closer the correlation coefficient is to $+1$ or -1, the higher the associaton of the algorithm's performance to the complexity measure. The complete sets of correlation coefficients and corresponding *p*-values are presented in Tables 6.4 and 6.5, and discussed in the following subsections. A complete set of scatterplot graphs depicting the relationships between mean and median algorithm suboptimality and the complexity measure values is presented in Appendix B in Figures B.5 and B.6.

6.3.1 Correlation Among Algorithms

Our first step was to test our hypothesis that some real-time search algorithms respond differently to certain search space features than others. In other words, we wanted to demonstrate that search problems can not simply be sorted on a one-dimensional continuum of complexity. A certain search problem may be very difficult for one algorithm, and yet comparatively easy for another.

To demonstrate this, we calculated the Spearman rank correlation between the solution suboptimality for each of the five algorithms in a pairwise fashion. The correlations between the mean suboptimalities are presented in Table 6.1 and between the median suboptimalities in 6.2. The corresponding graphical plots are presented in Figures B.1 and B.2.

We observed very few strong correlations in performance in the mean case. The only strong correlation was between TBA* and LRTA* ($\rho = 0.857$), with a moderate correlation observed between knn LRTA* and HCDPS ($\rho = 0.538$).

In contrast, we observed typically stronger correlations in the median case. HCDPS and kNN LRTA* ($\rho = 0.821$), HCDPS and LRTA* ($\rho = 0.714$), and kNN LRTA* and LRTA* ($\rho = 0.763$) are all substantially more correlated than in the mean case, and TBA* and LRTA* remain fairly

	LRTA*	D LRTA*	kNN LRTA*
LRTA*	–	0.4862, $< 10^{-10}$	0.182, 1.01×10^{-2}
D LRTA*	0.4862, $< 10^{-10}$	–	0.263, 1.77×10^{-4}
kNN LRTA*	0.182, 1.01×10^{-2}	0.263, 1.77×10^{-4}	–
HCDPS	0.140, 4.89×10^{-2}	0.163, 2.09×10^{-2}	0.538, $< 10^{-10}$
TBA*	0.857, $< 10^{-10}$	0.425, $< 10^{-10}$	0.284, 5.00×10^{-5}

	HCDPS	TBA*
LRTA*	0.140, 4.89×10^{-2}	0.857, $< 10^{-10}$
D LRTA*	0.163, 2.09×10^{-2}	0.425, $< 10^{-10}$
kNN LRTA*	0.538, $< 10^{-10}$	0.284, 5.00×10^{-5}
HCDPS	–	0.210, 2.87×10^{-3}
TBA*	0.210, 2.87×10^{-3}	–

Table 6.1: Spearman rank correlation coefficients ρ for *mean* solution suboptimality. p-values for statistical significance are given in italics.

	LRTA*	D LRTA*	kNN LRTA*
LRTA*	–	-0.154, 2.91×10^{-2}	0.714, $< 10^{-10}$
D LRTA*	-0.154, 2.91×10^{-2}	–	0.071, 3.19×10^{-1}
kNN LRTA*	0.714, $< 10^{-10}$	0.071, 3.19×10^{-1}	–
HCDPS	0.763, $< 10^{-10}$	-0.018, 7.97×10^{-1}	0.821, $< 10^{-10}$
TBA*	0.764, $< 10^{-10}$	-0.324, 3.25×10^{-6}	0.446, $< 10^{-10}$

	HCDPS	TBA*
LRTA*	0.763, $< 10^{-10}$	0.764, $< 10^{-10}$
D LRTA*	-0.018, 7.97×10^{-1}	-0.324, 3.25×10^{-6}
kNN LRTA*	0.821, $< 10^{-10}$	0.446, $< 10^{-10}$
HCDPS	–	0.584, $< 10^{-10}$
TBA*	0.584, $< 10^{-10}$	–

Table 6.2: Spearman rank correlation coefficients ρ for *median* solution suboptimality. p-values for statistical significance are given in italics.

correlated ($\rho = 0.764$). By considering only median solution suboptimality, we effectively remove from consideration any degenerate cases where the algorithms return vastly suboptimal solutions. Therefore, the algorithms are more frequently observed to achieve optimal or near optimal median solution costs, and the overall variance in median suboptimality is lower. We hypothesize that this "tightening of the pack" is responsible for the higher correlations. We therefore postulate that the relative disparities in algorithm performance manifest more apparently in outlying cases.

6.3.2 Correlation Among Complexity Measures

Note that not all of the complexity measures are independent. In fact, some of the complexity measures are quite highly correlated. However, we feel that the subtle differences between the highly correlated measures warrant the inclusion of all of these measures. Similar to the inter-algorithm comparison in the previous section, we present the pairwise Spearman rank correlation of all of the complexity measures in Table 6.3. A scatterplot of the relationships between the complexity

measures is presented in Figure B.3 and continued in Figure B.4.

	HC Region Size	HC Probability	Scrubbing Complexity
HC Region Size	–	0.759, $< 10^{-10}$	-0.117, 1.00×10^{-1}
HC Probability	0.759, $< 10^{-10}$	–	-0.529, $< 10^{-10}$
Scrubbing Complexity	-0.117, 1.00×10^{-1}	-0.529, $< 10^{-10}$	–
Path Compressibility	-0.697, $< 10^{-10}$	-0.920, $< 10^{-10}$	0.602, $< 10^{-10}$
A* Difficulty	-0.055, 4.36×10^{-1}	-0.514, $< 10^{-10}$	0.918, $< 10^{-10}$
Heuristic Error	-0.068, 3.40×10^{-1}	-0.540, $< 10^{-10}$	0.945, $< 10^{-10}$
Depression Width	-0.435, $< 10^{-10}$	-0.611, $< 10^{-10}$	0.696, $< 10^{-10}$
Depression Capacity	0.072, 3.11×10^{-1}	-0.178, 1.15×10^{-2}	0.633, $< 10^{-10}$

	Path Compressibility	A* Difficulty	Heuristic Error
HC Region Size	-0.697, $< 10^{-10}$	-0.055, 4.36×10^{-1}	-0.068, 3.40×10^{-1}
HC Probability	-0.920, $< 10^{-10}$	-0.514, $< 10^{-10}$	-0.540, $< 10^{-10}$
Scrubbing Complexity	0.602, $< 10^{-10}$	0.918, $< 10^{-10}$	0.945, $< 10^{-10}$
Path Compressibility	–	0.613, $< 10^{-10}$	0.624, $< 10^{-10}$
A* Difficulty	0.613, $< 10^{-10}$	–	0.983, $< 10^{-10}$
Heuristic Error	0.624, $< 10^{-10}$	0.983, $< 10^{-10}$	–
Depression Width	0.622, $< 10^{-10}$	0.621, $< 10^{-10}$	0.629, $< 10^{-10}$
Depression Capacity	0.230, 1.05×10^{-3}	0.594, $< 10^{-10}$	0.606, $< 10^{-10}$

	Depression Width	Depression Capacity
HC Region Size	-0.435, $< 10^{-10}$	0.072, 3.11×10^{-1}
HC Probability	-0.611, $< 10^{-10}$	-0.178, 1.15×10^{-2}
Scrubbing Complexity	0.696, $< 10^{-10}$	0.633, $< 10^{-10}$
Path Compressibility	0.622, $< 10^{-10}$	0.230, 1.05×10^{-3}
A* Difficulty	0.621, $< 10^{-10}$	0.594, $< 10^{-10}$
Heuristic Error	0.629, $< 10^{-10}$	0.606, $< 10^{-10}$
Depression Width	–	0.737, $< 10^{-10}$
Depression Capacity	0.737, $< 10^{-10}$	–

Table 6.3: Spearman rank correlation coefficients ρ between complexity measure values. p-values for statistical significance are given in italics.

We observe that HC region size is fairly correlated to HC probability ($\rho = 0.759$) and path compressibility ($\rho = 0.697$). Likewise, HC probability and path compressibility are observed to be very highly correlated ($\rho = 0.920$). These three complexity measures are all directly dependent on the movement of a hill-climbing agent through the search space.

We do, however, propose that there are intuitive differences in what these complexity measures are capturing. Unlike the other two measures, HC region size is measuring a space, rather than along a single path. HC region size differs more appreciably from the other two measures in search spaces with many narrow corridors, versus search spaces with open regions. This behaviour is discussed in Chapter 8. Additionally, we hypothesize that HC region size is a locally focused measure of hill-climbability among local groups of states, whereas HC probability measures hill-climbability between states that are arbitrarily positioned across the search space. HC probability provides a qualitative measure of hill-climbing performance, whereas path compressibility provides a quantitative measure. The former only distinguishes between problems which a hill-climbing agent can and

	LRTA*	D LRTA*	kNN LRTA*
HC Region Size	0.045, 5.23×10^{-1}	-0.207, 3.27×10^{-3}	$\mathbf{-0.804}$, $< 10^{-10}$
HC Probability	-0.433, $< 10^{-10}$	-0.309, 8.29×10^{-6}	-0.708, $< 10^{-10}$
Scrubbing Complexity	$\mathbf{0.955}$, $< 10^{-10}$	$\mathbf{0.451}$, $< 10^{-10}$	0.216, 2.21×10^{-3}
Path Compressibility	0.498, $< 10^{-10}$	0.327, 2.37×10^{-6}	0.617, $< 10^{-10}$
A* Difficulty	0.854, $< 10^{-10}$	0.328, 2.43×10^{-6}	0.136, 5.57×10^{-2}
Heuristic Error	0.884, $< 10^{-10}$	0.352, 3.80×10^{-7}	0.152, 3.16×10^{-2}
Depression Width	0.663, $< 10^{-10}$	0.447, $< 10^{-10}$	0.503, $< 10^{-10}$
Depression Capacity	0.647, $< 10^{-10}$	0.359, 2.21×10^{-7}	0.066, 3.51×10^{-1}

	HCDPS	TBA*
HC Region Size	-0.458, $< 10^{-10}$	-0.124, 8.06×10^{-2}
HC Probability	$\mathbf{-0.515}$, $< 10^{-10}$	-0.513, $< 10^{-10}$
Scrubbing Complexity	0.168, 1.72×10^{-2}	0.878, $< 10^{-10}$
Path Compressibility	0.428, $< 10^{-10}$	0.572, $< 10^{-10}$
A* Difficulty	0.108, 1.28×10^{-1}	$\mathbf{0.883}$, $< 10^{-10}$
Heuristic Error	0.128, 6.99×10^{-2}	0.882, $< 10^{-10}$
Depression Width	0.332, 1.61×10^{-6}	0.705, $< 10^{-10}$
Depression Capacity	0.190, 7.90×10^{-1}	0.613, $< 10^{-10}$

Table 6.4: Spearman rank correlation coefficients ρ_{mean} for *mean* solution suboptimality against complexity measure values. p-values for statistical significance are given in italics. The strongest correlation for each algorithm is in bold.

cannot solve, whereas path compressibility provides a gauge of how many failures a hill-climbing agent would encounter.

Heuristic error exhibited a strong correlation to scrubbing complexity ($\rho = 0.945$) and A* difficulty ($\rho = 0.983$). This is unsurprising, since higher magnitude heuristic errors will naturally correspond to a larger number of states being entered or considered by an LRTA* or A* agent. A* difficulty and scrubbing complexity are also very highly correlated ($\rho = 0.918$). We suspect this is due to their mutual sensitivity to inaccurate heuristics.

6.3.3 LRTA*

We observed a high correlation between the mean suboptimality of LRTA* and scrubbing complexity ($\rho_{\text{mean}} = 0.955$). This is natural, since scrubbing complexity is directly derived from LRTA*. A high correlation to heuristic error is also observed ($\rho_{\text{mean}} = 0.884$), which we attribute to the link between high magnitude heuristic errors and repeated state revisitation in LRTA*. The moderate correlation to depression width ($\rho_{\text{mean}} = 0.663$) and depression capacity ($\rho_{\text{mean}} = 0.647$) fits with prior literature that links the presence of heuristic depressions to poor LRTA* performance [9].

When we consider median suboptimality, LRTA* exhibits higher correlations to the HC-related measures path compressibility ($\rho_{\text{median}} = 0.852$) and HC probability ($\rho_{\text{mean}} = -0.841$). In the median case, LRTA* is not as hampered by scrubbing. By removing outliers, we are removing the cases where LRTA* must perform excessive state revisitation. We believe that this similarity in behaviour of LRTA* and a hill-climbing agent on easier search problems causes these higher

	LRTA*	D LRTA*	kNN LRTA*
HC Region Size	-0.609, $< 10^{-10}$	-0.211, 2.68×10^{-3}	-0.762, $< 10^{-10}$
HC Probability	-0.841, $< 10^{-10}$	0.073, 3.03×10^{-1}	$\mathbf{-0.832}$, $< 10^{-10}$
Scrubbing Complexity	0.669, $< 10^{-10}$	-0.377, 4.69×10^{-8}	0.320, 3.75×10^{-6}
Path Compressibility	$\mathbf{0.852}$, $< 10^{-10}$	-0.071, 3.18×10^{-1}	0.727, $< 10^{-10}$
A* Difficulty	0.656, $< 10^{-10}$	-0.429, $< 10^{-10}$	0.252, 3.16×10^{-4}
Heuristic Error	0.656, $< 10^{-10}$	$\mathbf{-0.443}$, $< 10^{-10}$	0.278, 6.61×10^{-5}
Depression Width	0.728, $< 10^{-10}$	-0.127, 7.35×10^{-2}	0.504, $< 10^{-10}$
Depression Capacity	0.352, 3.28×10^{-7}	-0.248, 4.06×10^{-4}	0.046, 5.18×10^{-1}

	HCDPS	TBA*
HC Region Size	-0.639, $< 10^{-10}$	-0.176, 1.29×10^{-2}
HC Probability	$\mathbf{-0.828}$, $< 10^{-10}$	-0.609, $< 10^{-10}$
Scrubbing Complexity	0.485, $< 10^{-10}$	0.862, $< 10^{-10}$
Path Compressibility	0.809, $< 10^{-10}$	0.665, $< 10^{-10}$
A* Difficulty	0.449, $< 10^{-10}$	0.889, $< 10^{-10}$
Heuristic Error	0.467, $< 10^{-10}$	$\mathbf{0.892}$, $< 10^{-10}$
Depression Width	0.548, $< 10^{-10}$	0.651, $< 10^{-10}$
Depression Capacity	0.170, 1.58×10^{-2}	0.542, $< 10^{-10}$

Table 6.5: Spearman rank correlation coefficients ρ_{median} for *median* solution suboptimality against complexity measure values. p-values for statistical significance are given in italics. The strongest correlation for each algorithm is in bold.

correlations in the median case.

6.3.4 D LRTA*

Despite having the same underlying agent as LRTA* and kNN LRTA*, D LRTA* exhibits no strong correlations with the presented complexity measures. The interaction between the clique abstraction and the heuristic topology of the map can be complex, even among ground-level states within a common abstract region. Very suboptimal solutions are usually tied to scrubbing behavior within an abstract region. However, the frequency of these cases is only weakly linked to overall scrubbing complexity ($\rho_{\text{mean}} = 0.451$) and to heuristic error ($\rho_{\text{mean}} = 0.352$).

Finding a computationally efficient predictor of D LRTA* performance remains an open research goal.

6.3.5 kNN LRTA*

In the mean case, kNN LRTA* performance is most correlated to HC region size ($\rho_{\text{mean}} = -0.804$) and HC probability ($\rho_{\text{mean}} = -0.708$). Since database records can only be used when they are HC-reachable relative to the start and goal states, a lower HC probability results in a lower chance of finding an appropriate record, causing search to fall back on LRTA* and therefore yielding a higher suboptimality. HC region size is suspected to be a marginally stronger predictor of record availability than HC probability since it is a more locally focused measure than HC probability. Since only the M most similar kNN LRTA* database records are considered for use, localized hill-climbability

will be more related to a database record being available.

In the median case, similar correlations are observed to HC probability ($\rho_{\text{median}} = -0.832$) and HC region size ($\rho_{\text{median}} = -0.762$). Path compressibility also has a somewhat higher correlation ($\rho_{\text{mean}} = 0.727$).

6.3.6 HCDPS

HCDPS performance is most correlated to HC probability ($\rho_{\text{mean}} = -0.515$) and the other HC-based measures. We were initially surprised that HC region size did not correlate more highly to the performance of HCDPS in the mean case, since HC region size is computed using the same abstraction method as in HCDPS databases. However, it appears that in the mean case, HC region size has a twofold relationship with solution suboptimality for HCDPS. Larger HC regions typically lead to lower suboptimality. This is due to the method that HCDPS uses to pre-compute subgoal records. When passing through fewer abstract regions, as is expected when region sizes are larger, the database record will be generated using fewer constituent paths, and is expected to be closer to optimal. However, if HC regions are too large, suboptimality can increase as the HC agent will be forced to deviate from the optimal path to pass through representative states.

In the median case, HCDPS correlates most strongly to HC probability ($\rho_{\text{median}} = -0.828$) and path compressibility ($\rho_{\text{median}} = 0.809$), and correlates more highly with HC region size than in the mean case ($\rho_{\text{median}} = -0.639$). We take this as evidence that in typical cases larger HC regions result in lower suboptimality, while in highly suboptimal cases, larger HC regions can cause increased suboptimality, matching the effects described above.

HCDPS performance is poorly correlated to depression capacity ($\rho_{\text{mean}} = 0.190$, $\rho_{\text{median}} = 0.170$), scrubbing complexity ($\rho_{\text{mean}} = 0.168$, $\rho_{\text{median}} = 0.485$) and heuristic error ($\rho_{\text{mean}} = 0.128$, $\rho_{\text{median}} = 0.467$). Since HCDPS does not perform heuristic updates, there is never a need to revisit states to fill in heuristic depressions. Therefore it is unsurprising that complexity measures which gauge the magnitude of heuristic innacuracy and state-revisitation are not correlated to HCDPS performance.

6.3.7 TBA*

The mean suboptimality of TBA* is most highly correlated to A* difficulty ($\rho_{\text{mean}} = 0.883$). This follows from the dependence of TBA* on a bounded A* agent. The more node expansions required by the A* agent, the more execution phases that will occur with TBA* following a potentially incomplete path. We attribute the similarly high correlations of TBA* to heuristic error ($\rho_{\text{mean}} = 0.892$) and scrubbing complexity ($\rho_{\text{mean}} = 0.862$) to the high correlation between these two measures and A* difficulty. TBA* also has a moderately high correlation to depression width ($\rho_{\text{mean}} = 0.705$).

In the median case, TBA* remains highly correlated to A* difficulty ($\rho_{\text{median}} = 0.889$), heuristic error ($\rho_{\text{median}} = 0.892$) and scrubbing complexity ($\rho_{\text{median}} = 0.862$). This leads us to believe that

there are no drastic differences between typical and degenerate behaviour of TBA* relative to the eight complexity measures we present.

6.3.8 Correlation to Database Construction Time

Solution suboptimality is not the only important measure of real-time search performance that is affected by search space features. The amount of time required for database precomputation can differ substantially even among search spaces of the same size. Therefore, we also examined how the complexity measures correlate to precomputation time. Table 6.6 presents the Spearman correlation ρ_{time} between complexity measure values and the database precomputation time in seconds.

	D LRTA*	kNN LRTA*	HCDPS
HC Region Size	-0.544, $< 10^{-10}$	-0.016, 8.19×10^{-1}	0.477, $< 10^{-10}$
HC Probability	-0.632, $< 10^{-10}$	-0.491, $< 10^{-10}$	$\mathbf{0.708}$, $< 10^{-10}$
Scrubbing Complexity	0.509, $< 10^{-10}$	0.911, $< 10^{-10}$	-0.344, 5.97×10^{-7}
Path Compressibility	$\mathbf{0.723}$, $< 10^{-10}$	0.580, $< 10^{-10}$	-0.665, $< 10^{-10}$
A* Difficulty	0.582, $< 10^{-10}$	$\mathbf{0.988}$, $< 10^{-10}$	-0.342, 7.31×10^{-7}
Heuristic Error	0.556, $< 10^{-10}$	0.982, $< 10^{-10}$	-0.357, 2.07×10^{-7}
Depression Width	0.427, $< 10^{-10}$	0.601, $< 10^{-10}$	-0.444, $< 10^{-10}$
Depression Capacity	0.188, 7.56×10^{-3}	0.606, $< 10^{-10}$	-0.177, 1.22×10^{-2}

Table 6.6: Spearman rank correlation coefficients ρ_{time} between complexity measure values and database computation time. p-values for statistical significance are given in italics.

D LRTA* construction time is most correlated to path compressibility ($\rho_{\text{mean}} = 0.723$). kNN LRTA* is highly correlated to A* difficulty ($\rho_{\text{time}} = 0.988$). The most time consuming component of kNN LRTA* database construction requires the optimal solving of a fixed number of search problems with A*. Therefore, the more difficult these problems are to solve for A*, the longer the time required to build the database. HCDPS database construction is most correlated to HC probability ($\rho_{\text{mean}} = 0.708$).

Chapter 7

Predictive Modelling

In the preceding chapter, we established a statistical link between the complexity measures and real-time search performance. Given the values of the complexity measures for two search spaces, we have an idea of how well a real-time heuristic search algorithm will perform in one search space relative to the other. This information alone solidifies our understanding of what search space features the five algorithms are sensitive to, and will be useful for characterizing search benchmarks. However, we would also like to apply the complexity measures to assist in algorithm selection and parameter tuning. To this end, we now adopt a predictive approach to modelling search performance with the complexity measures.

7.1 Experimental Design

Using machine learning, we construct a predictive model of search performance. As input, this model takes the values of the complexity measures computed for a given search space. As output, the model returns a predicted value of a chosen performance metric (e.g., solution suboptimality). A depiction of this model is presented in Figure 7.1.

For each metric of search performance, we build two classes of models. For the first class of models, we discretize the output metric into 10 bins with equal frequency in the training data. The output prediction is then a classification into one of these 10 bins. For the second class of models,

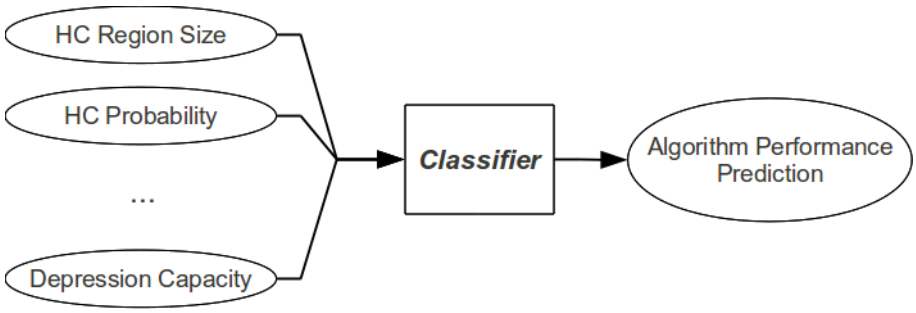

Figure 7.1: Predictive model of search performance.

the output is simply a numerical prediction of the search performance metric.

For each class of model, we tested a selection of classifiers within the WEKA framework [13]. We use the naive *ZeroR* classifier as a bassline predictor. In the first class, *ZeroR* always outputs the first bin. In the second class, *ZeroR* always outputs the mean value of the output metric observed in the training data.

The models that we build are trained and tested on the video game pathfinding data described in Section 6.2. All trials are performed with 10-fold cross-validation. For the first class of models, we report the percentage of cases where the correct bin was predicted for the output search metric. A greater percentage represents higher classifier accuracy. For the second class of models, we report the root-mean-square error (RMSE) and the relative root-square error (RRSE) of the output metric. RRSE is the percentage size of the error relative to the error of the *ZeroR* classifier. A smaller RMSE and RRSE correpsond to higher classifier accuracy.

Classifier	LRTA*	D LRTA*	kNN LRTA*	HCDPS	TBA
BayesNet	48%	18.5%	28.5%	15.5%	32%
MultilayerPerceptron	50.5%	**24%**	**37%**	19.5%	**39.5%**
AdaBoostM1	19.5%	18.5%	18%	17%	20%
Bagging	49%	22%	35.5%	16%	35.5%
ClassificationViaRegression	**54%**	21%	31.5%	**22.5%**	33%
RandomCommittee	51.5%	23%	29.5%	14.5%	**39.5%**
DecisionTable	45%	18.5%	30.5%	17.5%	31.5%
J48	44.5%	21%	28.5%	18%	32.5%
ZeroR	10%	10%	10%	10%	10%

Table 7.1: Classification accuracy for discretized mean suboptimality. The most accurate classifier for each algorithm is given in bold.

Classifier	LRTA*	D LRTA*	kNN LRTA*
SimpleLinearRegression	**8.975**, *36.22%*	**6.273**, *76.24%*	0.1452, *91.49%*
LeastMedSq	23.45, *57.57%*	8.555, *103.9%*	**0.1399**, *88.12%*
LinearRegression	13.21, *32.43%*	6.563, *79.77%*	0.1412, *88.99%*
MultilayerPerceptron	17.55, *43.07%*	13.70, *166.5%*	0.1706, *107.5%*
ZeroR	40.74, *100%*	8.230, *100%*	0.1587, *100%*

Classifier	HCDPS	TBA*
SimpleLinearRegression	0.0335, *83.48%*	0.5925, *58.05%*
LeastMedSq	0.0317, *79.11%*	0.4973, *48.72%*
LinearRegression	**0.0317**, *78.98%*	**0.4631**, *45.37%*
MultilayerPerceptron	0.0350, *87.34%*	0.5531, *54.18%*
ZeroR	0.0401, *100%*	1.021, *100%*

Table 7.2: Prediction error (RMSE) for raw mean suboptimality. RRSE is given in italics. The most accurate predictor for each algorithm is given in bold.

7.2 Predicting Mean Suboptimality

Table 7.1 presents the accuracy when predicting discretized mean suboptimality. We observe that LRTA* (54%), TBA* (39.5%) and kNN LRTA* (37%) have the highest peak classification accuracies. This is in keeping with our observations in Table 6.4, where these three algorithms had the highest observed correlations to the complexity measures. Conversely, D LRTA* (24%) and HCDPS (22.5%) have lower peak accuracies. However, in all cases, we are able to achieve a higher accuracy than the uninformed *ZeroR* classifier (10%).

Table 7.2 presents the RMSE and RRSE when predicting continuous mean suboptimality. We observe the lowest minimum error rates for LRTA* (RRSE = 36.22%) and TBA* (RRSE = 45.37%). kNN LRTA* (RRSE = 88.12%) is not as successfuly predicted as in the discretized case.

7.3 Predicting Median Suboptimality

Classifier	LRTA*	D LRTA*	kNN LRTA*	HCDPS	TBA
BayesNet	43%	15.5%	41.5%	29%	31%
MultilayerPerceptron	48%	21%	**47%**	31%	35%
AdaBoostM1	20%	17.5%	20%	19.5%	19%
Bagging	47.5%	**23.5%**	42%	**35.5%**	34%
ClassificationViaRegression	48%	15%	45%	30.5%	36.5%
RandomCommittee	**49%**	21.5%	38.5%	31%	37%
DecisionTable	39.5%	15%	36.5%	27.5%	29%
J48	48.5%	15.5%	41%	30%	**38.5%**
ZeroR	10%	10%	10%	10%	10%

Table 7.3: Classification accuracy for discretized median suboptimality. The most accurate classifier for each algorithm is given in bold.

Classifier	LRTA*	D LRTA*	kNN LRTA*
SimpleLinearRegression	6.994, *92.81%*	0.1313, *94.67%*	0.0285, *50.29%*
LeastMedSq	7.524, *99.86%*	0.1283, *91.81%*	0.0240, *42.31%*
LinearRegression	**6.564**, *87.11%*	**0.1257**, *89.98%*	0.0227, *40.13%*
MultilayerPerceptron	9.149, *121.4%*	0.1431, *102.4%*	**0.0226**, *39.88%*
ZeroR	7.535, *100%*	0.1397, *100%*	0.0566, *100%*

Classifier	HCDPS	TBA*
SimpleLinearRegression	0.0175, *53.57%*	0.6409, *53.46%*
LeastMedSq	0.0176, *54.03%*	0.5198, *43.35%*
LinearRegression	**0.0173**, *52.86%*	**0.5194**, *43.33%*
MultilayerPerceptron	0.0238, *72.90%*	0.6725, *56.09%*
ZeroR	0.0326, *100%*	1.199, *100%*

Table 7.4: Prediction error (RMSE) for raw median suboptimality. RRSE is given in italics. The most accurate predictor for each algorithm is given in bold.

Table 7.3 presents the accuracy when predicting discretized median suboptimality. We again ob-

serve that LRTA* (49%), kNN LRTA* (47%) and TBA* (38.5%) have the highest peak classification accuracies. In the case of kNN LRTA* and HCDPS (35.5%), we observe a higher classification accuracy than in the mean case. This corresponds to the higher correlations observed with median suboptimality for these algorithms, as presented in Table 6.5.

Table 7.4 presents the RMSE and RRSE when predicting continuous median suboptimality. We observe the lowest minimum error rates for kNN LRTA* (RRSE = 39.88%), TBA* (RRSE = 43.33%) and HCDPS (RRSE = 52.86%).

Classifier	D LRTA*	kNN LRTA*	HCDPS
BayesNet	21%	67.5%	18.5%
MultilayerPerceptron	**32**%	63%	21.5%
AdaBoostM1	19%	19.5%	18.5%
Bagging	31%	**69**%	**25.5**%
ClassificationViaRegression	29%	61%	24%
RandomCommittee	27.5%	66%	24%
DecisionTable	20%	62.5%	18.5%
J48	27.5%	62%	22%
ZeroR	10%	10%	10%

Table 7.5: Classification accuracy for discretized database construction time. The most accurate classifier for each algorithm is given in bold.

Classifier	D LRTA*	kNN LRTA*	HCDPS
SimpleLinearRegression	12.97, *69.46%*	0.3369, *16.00%*	0.5362, *75.49%*
LeastMedSq	13.24, *70.90%*	0.3007, *14.29%*	0.5346, *75.26%*
LinearRegression	**11.19**, *59.94%*	**0.2672**, *12.70%*	**0.5113**, *71.99%*
MultilayerPerceptron	12.64, *67.71%*	0.3580, *17.01%*	0.6813, *95.92%*
ZeroR	18.68, *100%*	2.105, *100%*	0.7103, *100%*

Table 7.6: Prediction error (RMSE) for raw database construction time. RRSE is given in italics. The most accurate predictor for each algorithm is given in bold.

7.4 Predicting Database Construction Time

Table 7.5 presents the accuracy when predicting discretized database construction time. We observe the highest peak classification accuracy for kNN LRTA* (69%). In Table 6.6, kNN LRTA* database construction time had the highest observed correlations. Note though that D LRTA* (32%) and HCDPS (25.5%) also exhibit peak classification accuracies well above the *ZeroR* classifier.

Table 7.6 presents the RMSE and RRSE when predicting continuous database construction time. Similarly to in the discrete case, the error rate for kNN LRTA* (12.70%) is lowest, follwed by D LRTA* (59.94%) and then HCDPS (71.99%).

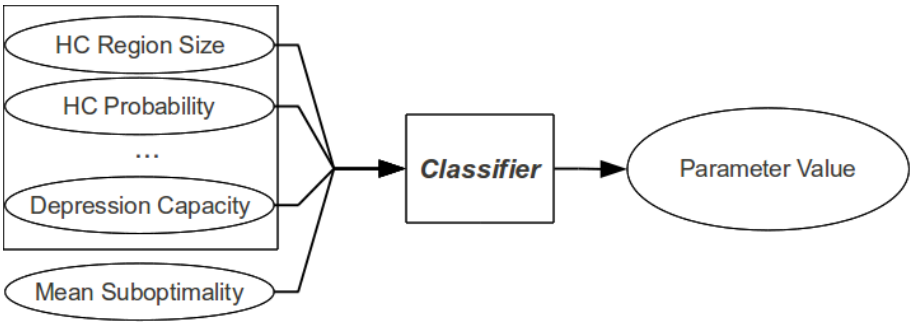

Figure 7.2: Predictive model for assisting algorithm parameterization.

7.5 Assisting Algorithm Parameterization

In the previous models we use the values of complexity measures to predict the search performance of a given algorithm for a search space. However, if we instead include *desired* search performance as an input to the model, we can predict appropriate values for algorithm parameters. Figure 7.2 depicts a predictive model for assisting algorithm parameterization.

As an example, we use this approach to predict appropriate kNN LRTA* database sizes that will achieve a desired mean suboptimality. Using the same search problems described in Section 6.2, we ran kNN LRTA* with database sizes of 500, 1000, 2500, 5000, 10000 and 20000, for $6 \times 200 = 1200$ datapoints. We then trained classifiers to output database size, using the values of the 8 complexity measures for a map and the mean kNN LRTA* suboptimality for that map as input.

Table 7.7 presents the accuracy when predicting database size discretized into 6 bins. Table 7.8 presents the RMSE and RRSE when predicting continuous database size. All experiments are performed with 10-fold cross-validation. In the discrete case, the peak classification accuracy is 50.42%. In the continuous case, the lowest RRSE is (52.54%). In both cases, we are able to outperform the *ZeroR* classifier.

Classifier	Accuracy
BayesNet	34.17%
MultilayerPerceptron	49.33%
AdaBoostM1	30.33%
Bagging	$\mathbf{50.42\%}$
ClassificationViaRegression	48.58%
RandomCommittee	39.83%
DecisionTable	39.12%
J48	48.92%
ZeroR	16.67%

Table 7.7: Classification accuracy for kNN LRTA* database size. The most accurate classifier for each algorithm is given in bold.

Classifier	RMSE, *(RRSE)*
SimpleLinearRegression	6469.66, *94.79%*
Bagging	**3585.58**, *52.54%*
LinearRegression	6840.78, *94.96%*
MultilayerPerceptron	4130.0811, *51.76%*
ZeroR	6824.65, *100%*

Table 7.8: Prediction error (RMSE) for kNN LRTA* database size. RRSE is given in italics. The most accurate predictor for each algorithm is given in bold.

Chapter 8

Beyond Video Game Pathfinding

In the previous two chapters we presented a set of complexity measures for characterizing search spaces, and demonstrated their relationship to the performance of five real-time heuristic search algorithms in the domain of video game pathfinding. In this chapter, we seek to extend these results to two additional classes of search problems: pathfinding in mazes and road maps. This is the first study to apply TBA*, kNN LRTA* and HCDPS to either of these domains, or any search space outside of video game pathfinding.

We begin by formally describing the two new classes of search problems. We then conduct experiments using the same correlation approach as in the previous chapter. To mitigate the influence of differing search space sizes, we again apply the sampling method described in Section 6.2. All of the experiments in this chapter are performed using sub-spaces of $20,000$ states.

For this chapter we omit the algorithm D LRTA*. This decision is in part due to the poor correlations observed to D LRTA* in the previous chapter. Additionally, computing a D LRTA* database becomes exceedingly expensive in general domains with an unknown branching factor. For D LRTA* to be successfuly applied to domains beyond video game pathfinding, we recommend that alternative abstraction techniques be used. By selecting a more appropriate abstraction method, one can avoid the high expense associated with building a clique-expansion in search spaces with a high branching factor. We have not confirmed what effect alternative abstraction methods would have on the suboptimality of D LRTA* solutions. This is suggested as future work.

8.1 Road Maps

Finding short paths in real-time on road maps is a common demand for consumer electronics such as personal and automotive GPS devices. Similar to video game pathfinding, this is an application where speed of search and quality of paths are critical to a positive user experience. The road maps that we examine are essentially arbitrary graphs. The edge weights in the graph are real-valued, and represent the geographical distance between the states the edge connects. For each state, we also have integer longitude and latitude values that are used for heuristic computation.

We use four maps from the 9^{th} DIMACS Implementation Challenge [11]. The road maps have between $264,346$ and $1,070,376$ states, and between $733,846$ and $2,712,798$ edges. For each of the four maps, we generated 25 sub-spaces for a total of 100 submaps. We solved 250 search problems on each map. We use the geographical Euclidean distance between two nodes as our heuristic. For states s_1 and s_2,

$$h(s_1, s_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

where x_i and y_i are the longitude and latitude of state s_i respectively. Each search problem has a single start state and a single goal state chosen randomly from the search space. Plots of the experimental data are presented in Appendix C.

Note that HCDPS is omitted from this section. Constructing an HCDPS database for these road maps exceeded the available amount of memory. The HCDPS partitioning scheme generates too many abstract regions for road maps. This causes the number of required subgoal entries to increase beyond a feasible level. This is reflected in the very low HC region size observed for road maps (5 to 15 states on average).

8.1.1 Correlation Among Complexity Measures

Table 8.1 presents the observed Spearman rank correlations between the complexity measures for road maps. Most of the correlations match those observed for video game pathfinding, and so we referback to the rationale provided in Section 6.3.2. However, there are several instances where the observed correlation differs by a substantial margin ($> \pm 0.300$). In particular, HC region size and depression capacity exhibited higher correlations in road maps than in video game maps.

8.1.2 Correlation Among Algorithms

Table 8.2 presents the correlation between the mean suboptimality of the algorithms for road maps. Table 8.3 presents the correlation between the median suboptimality of the algorithms for road maps. In both cases, a very high correlation is observed between LRTA* and kNN LRTA* suboptimality. In the vast majority of road map problems, kNN LRTA* is unable to find a suitable database record, and thus falls back on LRTA*. Therefore, the two algorithms are observed to have very similar performance. This is reflected in the very low observed HC probability for road maps (< 0.03).

8.1.3 Correlation Between Complexity and Performance

Table 8.4 presents the correlation between the mean suboptimality of the algorithms and the complexity measure values for road maps. Table 8.5 presents the correlation between the median suboptimality of the algorithms and the complexity measure values for road maps. In both the mean and median case, LRTA* suboptimality is correlated to similar measures as in video game pathfinding, with the highest correlation to scrubbing complexity ($\rho_{\text{mean}} = 0.878$, $\rho_{\text{median}} = 0.839$). The higher

	HC Region Size	HC Probability	Scrubbing Complexity
HC Region Size	–	0.683, $< 10^{-10}$	$\mathbf{-0.569}$, $< 10^{-10}$
HC Probability	0.683, $< 10^{-10}$	–	-0.611, $< 10^{-10}$
Scrubbing Complexity	$\mathbf{-0.569}$, $< 10^{-10}$	-0.611, $< 10^{-10}$	–
Path Compressibility	-0.706, $< 10^{-10}$	-0.764, $< 10^{-10}$	0.830, $< 10^{-10}$
A* Difficulty	$\mathbf{-0.364}$, 2.14×10^{-4}	-0.468, 9.33×10^{-7}	0.731, $< 10^{-10}$
Heuristic Error	$\mathbf{-0.667}$, $< 10^{-10}$	-0.732, $< 10^{-10}$	0.892, $< 10^{-10}$
Depression Width	$\mathbf{-0.936}$, $< 10^{-10}$	-0.654, $< 10^{-10}$	0.570, $< 10^{-10}$
Depression Capacity	$\mathbf{-0.710}$, $< 10^{-10}$	$\mathbf{-0.746}$, $< 10^{-10}$	0.836, $< 10^{-10}$

	Path Compressibility	A* Difficulty	Heuristic Error
HC Region Size	-0.706, $< 10^{-10}$	$\mathbf{-0.364}$, 2.14×10^{-4}	$\mathbf{-0.667}$, $< 10^{-10}$
HC Probability	-0.764, $< 10^{-10}$	-0.468, 9.33×10^{-7}	-0.732, $< 10^{-10}$
Scrubbing Complexity	0.830, $< 10^{-10}$	0.732, $< 10^{-10}$	0.892, $< 10^{-10}$
Path Compressibility	–	0.653, $< 10^{-10}$	$\mathbf{0.955}$, $< 10^{-10}$
A* Difficulty	0.653, $< 10^{-10}$	–	0.696, $< 10^{-10}$
Heuristic Error	$\mathbf{0.955}$, $< 10^{-10}$	0.696, $< 10^{-10}$	–
Depression Width	0.644, $< 10^{-10}$	0.454, 2.67×10^{-6}	0.607, $< 10^{-10}$
Depression Capacity	$\mathbf{0.924}$, $< 10^{-10}$	0.561, 1.90×10^{-9}	$\mathbf{0.954}$, $< 10^{-10}$

	Depression Width	Depression Capacity
HC Region Size	$\mathbf{-0.936}$, $< 10^{-10}$	$\mathbf{-0.710}$, $< 10^{-10}$
HC Probability	-0.654, $< 10^{-10}$	$\mathbf{-0.746}$, $< 10^{-10}$
Scrubbing Complexity	0.570, $< 10^{-10}$	0.835, $< 10^{-10}$
Path Compressibility	0.644, $< 10^{-10}$	$\mathbf{0.924}$, $< 10^{-10}$
A* Difficulty	0.454, 2.67×10^{-6}	0.561, 1.90×10^{-9}
Heuristic Error	0.607, $< 10^{-10}$	$\mathbf{0.954}$, $< 10^{-10}$
Depression Width	–	0.636, $< 10^{-10}$
Depression Capacity	0.636, $< 10^{-10}$	–

Table 8.1: Spearman rank correlation coefficients ρ between complexity measure values for road maps. p-values for statistical significance are given in italics. Correlations substantially different from those observed for video game pathfinding are indicated in bold.

correlation between LRTA* and HC region size in road maps ($\rho_{\text{mean}} = -0.431$, $\rho_{\text{median}} = -0.680$) parallels the increased correlation between HC region size and scrubbing complexity in Table 8.1.

kNN LRTA* is most correlated to scrubbing complexity ($\rho_{\text{mean}} = -0.431$, $\rho_{\text{median}} = -0.680$), A* difficulty ($\rho_{\text{mean}} = 0.773$, $\rho_{\text{median}} = 0.743$) and heuristic error ($\rho_{\text{mean}} = 0.761$, $\rho_{\text{median}} = 0.818$). In video game pathfinding, these measures had a very low correlation to kNN LRTA* suboptimality. We attribute this difference to the lower likelihood that kNN LRTA* can find a subgoal in road maps, as discussed in Section 8.1.2.

TBA* suboptimality in road maps is most correlated to A* difficulty ($\rho_{\text{mean}} = 0.536$, $\rho_{\text{median}} = 0.491$). As discussed in the previous chapter, this is due to the foundation of TBA* being a depth-limited A* agent.

	LRTA*	kNN LRTA*	TBA*
LRTA*	–	0.970, $< 10^{-10}$	0.189, 5.85×10^{-2}
kNN LRTA*	0.970, $< 10^{-10}$	–	0.220, 2.80×10^{-2}
TBA*	0.189, 5.85×10^{-2}	0.220, 2.80×10^{-2}	–

Table 8.2: Spearman rank correlation coefficients ρ_{mean} for *mean* solution suboptimality for road maps. p-values for statistical significance are given in italics.

	LRTA*	kNN LRTA*	TBA*
LRTA*	–	0.984, $< 10^{-10}$	0.137, 1.73×10^{-1}
kNN LRTA*	0.984, $< 10^{-10}$	–	0.133, 1.86×10^{-1}
TBA*	0.137, 1.73×10^{-2}	0.133, 1.86×10^{-1}	–

Table 8.3: Spearman rank correlation coefficients ρ_{median} for *median* solution suboptimality for road maps. p-values for statistical significance are given in italics.

	LRTA*	kNN LRTA*	TBA*
HC Region Size	-0.431, 9.62×10^{-6}	-0.528, 2.83×10^{-8}	0.071, 4.85×10^{-1}
HC Probability	-0.402, 3.38×10^{-5}	-0.502, 1.04×10^{-7}	0.017, 8.70×10^{-1}
Scrubbing Complexity	$\mathbf{0.878}$, $< 10^{-10}$	$\mathbf{0.908}$, $< 10^{-10}$	0.051, 6.13×10^{-1}
Path Compressibility	0.596, $< 10^{-10}$	0.697, $< 10^{-10}$	-0.001, 9.90×10^{-1}
A* Difficulty	0.737, $< 10^{-10}$	0.773, $< 10^{-10}$	$\mathbf{0.536}$, 1.57×10^{-8}
Heuristic Error	0.680, $< 10^{-10}$	0.761, $< 10^{-10}$	0.009, 9.29×10^{-1}
Depression Width	0.510, 9.96×10^{-8}	0.592, $< 10^{-10}$	0.048, 6.32×10^{-1}
Depression Capacity	0.590, $< 10^{-10}$	0.689, $< 10^{-10}$	-0.100, 3.17×10^{-1}

Table 8.4: Spearman rank correlation coefficients ρ_{mean} for *mean* solution suboptimality against complexity measure values for road maps. p-values for statistical significance are given in italics. The strongest correlation for each algorithm is in bold.

	LRTA*	kNN LRTA*	TBA*
HC Region Size	-0.680, $< 10^{-10}$	-0.733, 2.83×10^{-8}	0.133, 4.85×10^{-1}
HC Probability	-0.648, $< 10^{-10}$	-0.699, 1.04×10^{-7}	0.059, 8.70×10^{-1}
Scrubbing Complexity	$\mathbf{0.839}$, $< 10^{-10}$	$\mathbf{0.844}$, $< 10^{-10}$	-0.007, 6.13×10^{-1}
Path Compressibility	0.759, $< 10^{-10}$	0.812, $< 10^{-10}$	-0.030, 9.90×10^{-1}
A* Difficulty	0.739, $< 10^{-10}$	0.743, $< 10^{-10}$	$\mathbf{0.491}$, 1.57×10^{-8}
Heuristic Error	0.776, $< 10^{-10}$	0.818, $< 10^{-10}$	-0.016, 9.29×10^{-1}
Depression Width	0.773, $< 10^{-10}$	0.807, $< 10^{-10}$	-0.031, 6.32×10^{-1}
Depression Capacity	0.740, $< 10^{-10}$	0.783, $< 10^{-10}$	-0.143, 3.17×10^{-1}

Table 8.5: Spearman rank correlation coefficients ρ_{median} for *median* solution suboptimality against complexity measure values for road maps. p-values for statistical significance are given in italics. The strongest correlation for each algorithm is in bold.

8.2 Mazes

Mazes are defined identically to the video game maps used in Chapter 6. However, search problems in a maze are generally more challenging than in a video game map. Video game maps are typically composed of open regions where obstacles may frequently have no impact on direct agent movement. In contrast, mazes consist of many winding narrow passages, often forcing an agent to take a convoluted path to reach the goal state. The practical manifestation of this tendency is reduced

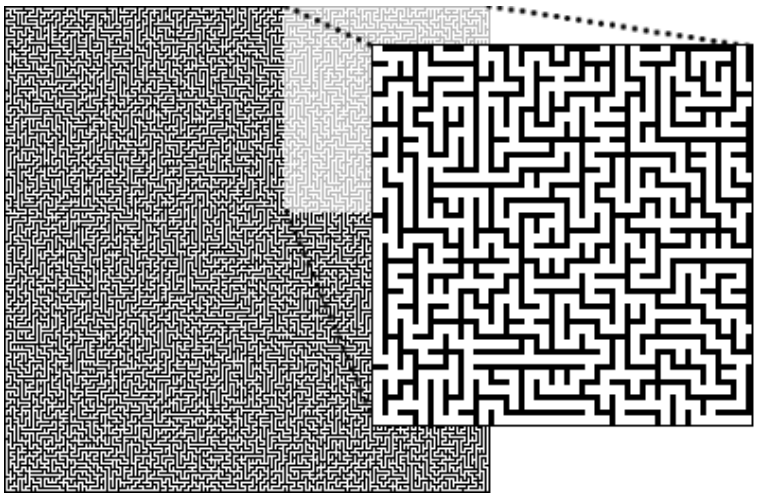

Figure 8.1: An example 512×512 maze, with a section enlarged for visibility. This maze has a corridor width of 1.

heuristic accuracy.

Another property that differentiates mazes from video games results from corridor width. The *corridor width* of a maze is defined as the width in cells of the narrow passages constituting the maze. The mazes we explore have corridor widths of 1, 2, 4 and 8. As a result, the branching factor of states is typically smaller in mazes than in video game maps. When in a narrow corridor, an agent

We use a set of 20 distinct random mazes [32], with 5 mazes for each corridor width. The mazes are 512×512 cells in size. An example maze is presented in Figure 8.1. For each maze we generated 5 sub-spaces of $20,000$ states, for a total of 100 sub-mazes. We solved 250 search problems in each sub-maze. Plots of the experimental data are presented in Appendix D.

8.2.1 Correlation Among Complexity Measures

Table 8.6 presents the observed correlations between the complexity measures for mazes. The observed correlations for mazes differ more significantly from video game pathfinding than those for the road maps did. For example, several of the correlations to HC region size are much higher. We suspect that this is due to the clustering of data according to corridor width. Mazes with a common corridor width typically exhibit a similar HC region size. This can be seen in Figures D.3 and D.4. The differences observed for HC region size in search spaces with narrow corridors match our suspicions about the locality of HC region size discussed in the previous chapter.

The observed correlations for scrubbing complexity are lower than for video game pathfinding. Scrubbing complexity appears to have a similar spread of values for each different corridor width (as seen in Figures D.3 and D.4). In contrast, other measures are generally more disparate accross different corridor widths. This decreases the observed correlations for scrubbing complexity.

Perhaps the most interesting result is that the directions of the correlations for A* difficulty and for depression capacity are reversed from those observed in video game pathfinding. We attribute

	HC Region Size	HC Probability	Scrubbing Complexity
HC Region Size	–	0.899, $< 10^{-10}$	-0.049, 6.26×10^{-1}
HC Probability	0.899, $< 10^{-10}$	–	$\mathbf{-0.109}$, 2.79×10^{-1}
Scrubbing Complexity	-0.049, 6.26×10^{-1}	$\mathbf{-0.109}$, 2.79×10^{-1}	–
Path Compressibility	-0.924, $< 10^{-10}$	-0.900, $< 10^{-10}$	$\mathbf{0.206}$, 3.99×10^{-2}
A* Difficulty	$\mathbf{0.420}$, 1.59×10^{-5}	$\mathbf{0.321}$, 1.13×10^{-3}	$\mathbf{0.378}$, 1.19×10^{-4}
Heuristic Error	$\mathbf{-0.837}$, $< 10^{-10}$	-0.829, $< 10^{-10}$	$\mathbf{0.337}$, 6.56×10^{-4}
Depression Width	$\mathbf{-0.921}$, $< 10^{-10}$	-0.890, $< 10^{-10}$	$\mathbf{0.065}$, 5.23×10^{-1}
Depression Capacity	$\mathbf{0.951}$, $< 10^{-10}$	$\mathbf{0.890}$, $< 10^{-10}$	$\mathbf{-0.060}$, 5.53×10^{-1}

	Path Compressibility	A* Difficulty	Heuristic Error
HC Region Size	-0.924, $< 10^{-10}$	$\mathbf{0.421}$, 1.59×10^{-5}	$\mathbf{-0.837}$, $< 10^{-10}$
HC Probability	-0.900, $< 10^{-10}$	$\mathbf{0.321}$, 1.13×10^{-3}	-0.829, $< 10^{-10}$
Scrubbing Complexity	$\mathbf{0.206}$, 3.99×10^{-2}	$\mathbf{0.378}$, 1.19×10^{-4}	$\mathbf{0.337}$, 6.56×10^{-4}
Path Compressibility	–	-0.209, 3.71×10^{-2}	0.962, $< 10^{-10}$
A* Difficulty	$\mathbf{-0.209}$, 3.71×10^{-2}	–	$\mathbf{-0.015}$, 8.79×10^{-1}
Heuristic Error	0.962, $< 10^{-10}$	$\mathbf{-0.015}$, 8.79×10^{-1}	–
Depression Width	$\mathbf{0.905}$, $< 10^{-10}$	$\mathbf{-0.372}$, 1.58×10^{-4}	0.833, $< 10^{-10}$
Depression Capacity	$\mathbf{-0.932}$, $< 10^{-10}$	0.435, 7.61×10^{-6}	$\mathbf{-0.840}$, $< 10^{-10}$

	Depression Width	Depression Capacity
HC Region Size	$\mathbf{-0.921}$, $< 10^{-10}$	$\mathbf{0.951}$, $< 10^{-10}$
HC Probability	-0.886, $< 10^{-10}$	$\mathbf{0.890}$, $< 10^{-10}$
Scrubbing Complexity	$\mathbf{0.065}$, 5.23×10^{-1}	$\mathbf{-0.060}$, 5.53×10^{-1}
Path Compressibility	0.905, $< 10^{-10}$	$\mathbf{-0.932}$, $< 10^{-10}$
A* Difficulty	$\mathbf{-0.372}$, 1.58×10^{-4}	0.435, 7.61×10^{-6}
Heuristic Error	0.833, $< 10^{-10}$	$\mathbf{-0.840}$, $< 10^{-10}$
Depression Width	–	$\mathbf{-0.865}$, $< 10^{-10}$
Depression Capacity	$\mathbf{-0.865}$, $< 10^{-10}$	–

Table 8.6: Spearman rank correlation coefficients ρ between complexity measure values for mazes. p-values for statistical significance are given in italics. Correlations substantially different from those observed for video game pathfinding are indicated in bold.

this observation to the Yule-Simpson effect. Despite a positive trend among the data among mazes of a fixed corridor width, the overall data exhibits a negative trend. This indicates a potential danger of using a correlation-based analysis of the complexity measures without also considering the distribution of the data, especially when diverse search spaces are being considered.

8.2.2 Correlation Among Algorithms

Table 8.7 presents the correlation between the mean suboptimality of the algorithms for mazes. Table 8.8 presents the correlation between the median suboptimality of the algorithms for mazes. The observed correlations are analogous to those observed in video game pathfinding, with the exception of kNN LRTA* where the direction of the correlations is reversed. We again attribute this to the Yule-Simpson effect. The correlations for mazes of a single fixed corridor width are in line with those observed in video game maps.

	LRTA*	kNN LRTA*
LRTA*	–	-0.528, 2.80×10^{-8}
kNN LRTA*	-0.528, 2.80×10^{-8}	–
HCDPS	0.277, 5.41×10^{-3}	-0.422, 1.46×10^{-5}
TBA*	0.780, $< 10^{-10}$	-0.810, $< 10^{-10}$

	HCDPS	TBA*
LRTA*	0.277, 5.41×10^{-3}	0.780, $< 10^{-10}$
kNN LRTA*	-0.422, 1.46×10^{-5}	-0.810, $< 10^{-10}$
HCDPS	–	0.549, 5.49×10^{-9}
TBA*	0.549, 5.49×10^{-9}	–

Table 8.7: Spearman rank correlation coefficients ρ_{mean} for *mean* solution suboptimality for maze pathfinding. p-values for statistical significance are given in italics.

	LRTA*	kNN LRTA*
LRTA*	–	-0.522, 4.10×10^{-8}
kNN LRTA*	-0.522, 4.10×10^{-8}	–
HCDPS	0.238, 1.70×10^{-2}	-0.340, 5.46×10^{-4}
TBA*	0.716, $< 10^{-10}$	-0.829, $< 10^{-10}$

	HCDPS	TBA*
LRTA*	0.238, 1.70×10^{-2}	0.716, $< 10^{-10}$
kNN LRTA*	-0.340, 5.46×10^{-4}	-0.829, $< 10^{-10}$
HCDPS	–	0.474, 6.30×10^{-7}
TBA*	0.474, 6.30×10^{-7}	–

Table 8.8: Spearman rank correlation coefficients ρ_{median} for *median* solution suboptimality for maze pathfinding. p-values for statistical significance are given in italics.

8.2.3 Correlation Between Complexity and Performance

Table 8.9 presents the correlation between the mean suboptimality of the algorithms and the complexity measure values for mazes. Table 8.10 presents the correlation between the median suboptimality of the algorithms and the complexity measure values for mazes. We again observe a manifestation of the Yule-Simpson effect: the direction of most correlations for LRTA*, HCDPS and TBA* are reversed from those observed in video game pathfinding.

For kNN LRTA*, HC region size ($\rho_{\text{mean}} = -0.840$, $\rho_{\text{median}} = -0.847$), HC probability ($\rho_{\text{mean}} = -0.807$, $\rho_{\text{median}} = -0.827$) and path compressibility ($\rho_{\text{mean}} = 0.810$, $\rho_{\text{median}} = 0.820$) have similarly high correlations to those in video game maps. We suspect that this is due to the way that kNN LRTA* provides subgoals during search. The corridor width does not significantly alter the ability of kNN LRTA* to find appropriate subgoals.

	LRTA*	kNN LRTA*
HC Region Size	$0.786, < 10^{-10}$	$\mathbf{-0.840}, < 10^{-10}$
HC Probability	$0.745, < 10^{-10}$	$-0.807, < 10^{-10}$
Scrubbing Complexity	$0.325, 1.05 \times 10^{-3}$	$0.203, 4.27 \times 10^{-2}$
Path Compressibility	$-0.759, < 10^{-10}$	$0.810, < 10^{-10}$
A* Difficulty	$0.429, 1.02 \times 10^{-5}$	$-0.328, 9.26 \times 10^{-4}$
Heuristic Error	$-0.651, < 10^{-10}$	$0.745, < 10^{-10}$
Depression Width	$\mathbf{-0.790}, < 10^{-10}$	$0.806, < 10^{-10}$
Depression Capacity	$0.788, < 10^{-10}$	$-0.814, < 10^{-10}$

	HCDPS	TBA*
HC Region Size	$0.466, 1.42 \times 10^{-6}$	$0.922, < 10^{-10}$
HC Probability	$0.440, 4.67 \times 10^{-6}$	$0.898, < 10^{-10}$
Scrubbing Complexity	$-0.384, 9.09 \times 10^{-5}$	$-0.183, 6.83 \times 10^{-2}$
Path Compressibility	$-0.551, 4.86 \times 10^{-9}$	$\mathbf{-0.978}, < 10^{-10}$
A* Difficulty	$0.031, 7.62 \times 10^{-1}$	$0.283, 4.52 \times 10^{-3}$
Heuristic Error	$\mathbf{-0.583}, < 10^{-10}$	$-0.935, < 10^{-10}$
Depression Width	$-0.456, 2.39 \times 10^{-6}$	$-0.903, < 10^{-10}$
Depression Capacity	$0.447, 3.98 \times 10^{-6}$	$0.928, < 10^{-10}$

Table 8.9: Spearman rank correlation coefficients ρ_{mean} for *mean* solution suboptimality against complexity measure values for maze pathfinding. p-values for statistical significance are given in italics. The strongest correlation for each algorithm is in bold.

	LRTA*	kNN LRTA*
HC Region Size	$0.733, < 10^{-10}$	$\mathbf{-0.847}, < 10^{-10}$
HC Probability	$0.660, < 10^{-10}$	$-0.827, < 10^{-10}$
Scrubbing Complexity	$0.312, 1.64 \times 10^{-3}$	$0.103, 3.09 \times 10^{-1}$
Path Compressibility	$-0.651, < 10^{-10}$	$0.820, < 10^{-10}$
A* Difficulty	$0.597, < 10^{-10}$	$-0.348, 4.24 \times 10^{-4}$
Heuristic Error	$-0.506, 1.21 \times 10^{-7}$	$0.754, < 10^{-10}$
Depression Width	$-0.726, < 10^{-10}$	$0.809, < 10^{-10}$
Depression Capacity	$\mathbf{0.754}, < 10^{-10}$	$-0.814, < 10^{-10}$

	HCDPS	TBA*
HC Region Size	$0.390, 6.11 \times 10^{-5}$	$0.917, < 10^{-10}$
HC Probability	$0.382, 8.87 \times 10^{-5}$	$0.888, < 10^{-10}$
Scrubbing Complexity	$-0.344, 4.62 \times 10^{-4}$	$-0.167, 9.70 \times 10^{-2}$
Path Compressibility	$-0.464, 1.20 \times 10^{-6}$	$\mathbf{-0.970}, < 10^{-10}$
A* Difficulty	$0.069, 4.97 \times 10^{-1}$	$0.303, 2.27 \times 10^{-3}$
Heuristic Error	$\mathbf{-0.489}, 2.42 \times 10^{-7}$	$-0.925, < 10^{-10}$
Depression Width	$-0.382, 8.84 \times 10^{-5}$	$-0.899, < 10^{-10}$
Depression Capacity	$0.384, 7.86 \times 10^{-5}$	$0.925, < 10^{-10}$

Table 8.10: Spearman rank correlation coefficients ρ_{median} for *median* solution suboptimality against complexity measure values for maze pathfinding. p-values for statistical significance are given in italics. The strongest correlation for each algorithm is in bold.

Chapter 9

Discussion

In this chapter we conduct a discussion of the results presented in this work. We organize the discussion according to the three major goals of our research presented in Chapter 1. We also discuss how the research could be extended to make additional contributions towards these goals. Finally, we suggest future work in the field of real-time heuristic search.

9.1 Understanding Algorithm Performance

The statistical correlations we observe in Chapters 6 and 8 provide insight into how search space features affect the performance of real-time heuristic search. While the importance of these features was already known, we now have a way of empirically measuring the degree to which they are affect the performance a specific algorithm.

9.2 Facilitating Algorithm Use

We have described a system by which one can inform the decision of which real-time algorithm is suitable for a given search space. Using the models we presented, one can automatically predict the performance of an algorithm on a novel search space. One can also use a similar model to determine algorithm parameters which will yield the approximate desired level of performance. Previously there was no way to accomplish this without expert knowledge. Given the large investment of time required to compute databases for subgoal-driven search in large search spaces, this is a valuable tool.

Several improvements could be made to bolster the strength of the predictions made by the models we present. The performance of some algorithms, such as D LRTA*, was not predicted as accurately as others. We suspect that this could be improved by finding a better way to extract search space features which affect D LRTA* performance. We tested additional measures to try and measure the effect of the D LRTA* clique abstraction on search performance. However, these measures were expensive to compute and did not yield significantly stronger results.

We would also like to test the ability of a single model to make predictions accross several domain types. We would also like to build such a model across search spaces of differing sizes, including search spaces size as an input to the predictive model.

9.3 Characterizing Benchmarks

We propose that the eight complexity measures we have presented are appropriate for characterizing benchmarks for real-time heuristic search. Having a universal way of empirically measuring the complexity of search spaces is a useful tool for algorithm development and evaluation. The complexity measures can be applied to any search space, and have been statistically demonstrated to impact the performance of real-time heuristic search.

We are aware of other research being simultaneously conducted to establish similar complexity measures for characterizing benchmarks [32]. However, to our knowledge, this work has not yet been formally published. Additionally, this independent work is aimed at conventional heuristic search, rather than real-time heuristic search.

9.4 Future Work

During the course of this work, other researchers have independently developed two real-time heuristic search algorithms which actively use heuristic depression information to adjust search decisions. The first, aLSS-LRTA* [16] uses a scheme to flag and avoid states which have been discovered online to belong to a heuristic depression. The authors do not present a comparison of the performance of this algorithm to contemporary subgoaling algorithms such as DLRTA*, kNN LRTA* and HCDPS.

The second algorithm (which is unnamed by the authors) [15] is a modified version of LRTA* that uses subgoals placed according to the presence of heuristic depressions. The subgoal database is a set of trees constructed by performing a modified version of Dijkstra's algorithm. The subgoal trees store states which are identified as exits from heuristic depressions. Unfortunately, one subgoal tree must be computed for every state in the search space. We therefore suspect that this algorithm will scale poorly to larger search spaces.

As future work, we suggest the development of a modified version of kNN LRTA* that uses search space information to guide the placement of database entries, rather than distributing them randomly. We suggest building from kNN LRTA*, since it has the favorable property of not requiring a complete search space enumeration. This could be essential for adapting database-driven real-time heuristic search to very large domains such as planning. One idea is to have kNN LRTA* select multiple random candidate entries at a time, and only store the entry deemed to be in the most complex area, as computed by an appropriate complexity measure.

Chapter 10

Conclusion

In this document, we have explored the performance of real-time heuristic search as it is affected by the properties of search spaces. We began by formally defining heuristic search and real-time search. We then comprehensively reviewed a selection of state of the art real-time heuristic search algorithms. We discussed the common approach of many of these algorithms to compute a domain-specific database that provides subgoals to guide the search agent. We explored the efforts of other researchers to understand real-time heuristic search performance, and discussed why our work was significant in this context.

We then presented our main vehicle for characterizing search spaces and understanding algorithm performance: a set of eight domain-independent complexity measures. After explaining how the complexity measures can be computed in practice, we examined how they relate to the performance of real-time heuristic search in videogame pathfinding. We demonstrated a statistical link between the measures and the performance of five real-time algorithms. We then showed how the measures could be used to predict the performance of algorithms, and to assist in algorithm parameterization. Our examination of videogame pathfinding was followed by an extension of the complexity measures to mazes and road maps. This was the first such examination of database-driven real-time search in these domains. Finally, we suggested avenues for future research, and discussed how this work could inform the future development of real-time heuristic search algorithms.

Bibliography

[1] BioWare Corp. Baldur's Gate, 1998.

[2] BioWare Corp. Dragon Age: Origins, 2009.

[3] Yngvi Björnsson, Vadim Bulitko, and Nathan R. Sturtevant. Tba*: Time-bounded a*. In Craig Boutilier, editor, *IJCAI*, pages 431–436, 2009.

[4] Blizzard Entertainment. Warcraft III: Reign of Chaos, 2002.

[5] Blai Bonet and Hector Geffner. Planning as heuristic search. *Artificial Intelligence*, 129:5–33, 2001.

[6] Adi Botea, Martin Müller, and Jonathan Schaeffer. Near optimal hierarchical path-finding. *Journal of Game Development*, 1:7–28, 2004.

[7] Vadim Bulitko and Yngvi Björnsson. kNN LRTA*: Simple subgoaling for real-time search. In *AIIDE*, pages 2–7, 2009.

[8] Vadim Bulitko, Yngvi Björnsson, and Ramon Lawrence. Case-based subgoaling in real-time heuristic search for video game pathfinding. *JAIR*, 39:269 – 300, 2010.

[9] Vadim Bulitko and Greg Lee. Learning in real-time search: A unifying framework. *J. Artif. Intell. Res. (JAIR)*, 25:119–157, 2006.

[10] Vadim Bulitko, Mitja Luštrek, Jonathan Schaeffer, Yngvi Björnsson, and Sverrir Sigmundarson. Dynamic control in real-time heuristic search. *JAIR*, 32:419 – 452, 2008.

[11] Center for Discrete Mathematics & Theoretical Computer Science. 9th DIMACS Implementation Challenge - Shortest Paths. `http://www.dis.uniroma1.it/~challenge9/download.shtml`.

[12] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[13] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. In *SIGKDD Explorations*, volume 11, 2009.

[14] P E Hart, N J Nilsson, and B Raphael. A formal basis for the heuristic determination of minimum cost paths. *Ieee Transactions On Systems Science And Cybernetics*, 4(2):100–107, 1968.

[15] Carlos Hernandez and Jorge A. Baier. Fast subgoaling for pathfinding via real-time search. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS-11)*, Freiburg, Germany, June 2011.

[16] Carlos Hernandez and Jorge A. Baier. Real-time heuristic search with depression avoidance. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, Barcelona, Spain, July 2011.

[17] Jörg Hoffmann. Local search topology in planning benchmarks: An empirical analysis. In Bernhard Nebel, editor, *IJCAI*, pages 453–458. Morgan Kaufmann, 2001.

[18] Jörg Hoffmann. Local search topology in planning benchmarks: A theoretical analysis. In Malik Ghallab, Joachim Hertzberg, and Paolo Traverso, editors, *AIPS*, pages 92–100. AAAI, 2002.

[19] Jrg Hoffmann. Ff: The fast-forward planning system. *AI magazine*, 22:57–62, 2001.

[20] Daniel Huntley and Vadim Bulitko. Extending the applications of recent real-time heuristic search. In *AAAI*, page In press, 2011.

[21] Toru Ishida. *Real-time search for learning autonomous agents*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[22] Sven Koenig. Real-time heuristic search: Research issues. In *AIPS Workshop on Planning as Combinatorial Search: Propositional, Graph-Based, and Disjunctive Planning Methods*, pages 75–79, 1998.

[23] Richard Korf. Real-time heuristic search. *AIJ*, 42:189–211, 1990.

[24] Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[25] Ramon Lawrence and Vadim Bulitko. Taking learning out of real-time heuristic search for video-game pathfinding. In *Australasian Joint Conf. on AI*, pages 10–19, Adelaide, Australia, 2010.

[26] Carlos Linares Lòpez. Heuristic hill-climbing as a markov process. In Danail Dochev, Marco Pistore, and Paolo Traverso, editors, *AIMSA*, volume 5253 of *Lecture Notes in Computer Science*, pages 274–284. Springer, 2008.

[27] MathWorks. Linear or Rank Correlation - MATLAB. `http://www.mathworks.com/help/toolbox/stats/corr.html`.

[28] Masataka Mizusawa and Masahito Kurihara. Hardness measures for gridworld benchmarks and performance analysis of real-time heuristic search algorithms. *J. Heuristics*, 16(1):23–36, 2010.

[29] Joseph Pemberton and Richard E. Korf. Making locally optimal decisions on graphs with cycles. Technical report, goal Example LRTA LS *(k) with Lookahead 6 RTAA* with Lookahead 6 LRTS with Lookahead 3, 1992.

[30] D. Chris Rayner, Michael H. Bowling, and Nathan R. Sturtevant. Euclidean heuristic optimization. In Wolfram Burgard and Dan Roth, editors, *AAAI*. AAAI Press, 2011.

[31] Charles Spearman. The proof and measurement of association between two things. *AJP*, 15:7–28, 1904.

[32] Nathan Sturtevant. Moving AI Lab Pathfinding Benchmarks. `http://www.aiide.org/benchmarks/`.

[33] Valve Corporation. Counter-Strike: Source, 2004.

Appendix A

Videogame Maps

In this appendix, we present images of the complete set of commercial videogame maps used for experiments in Chapter 6.

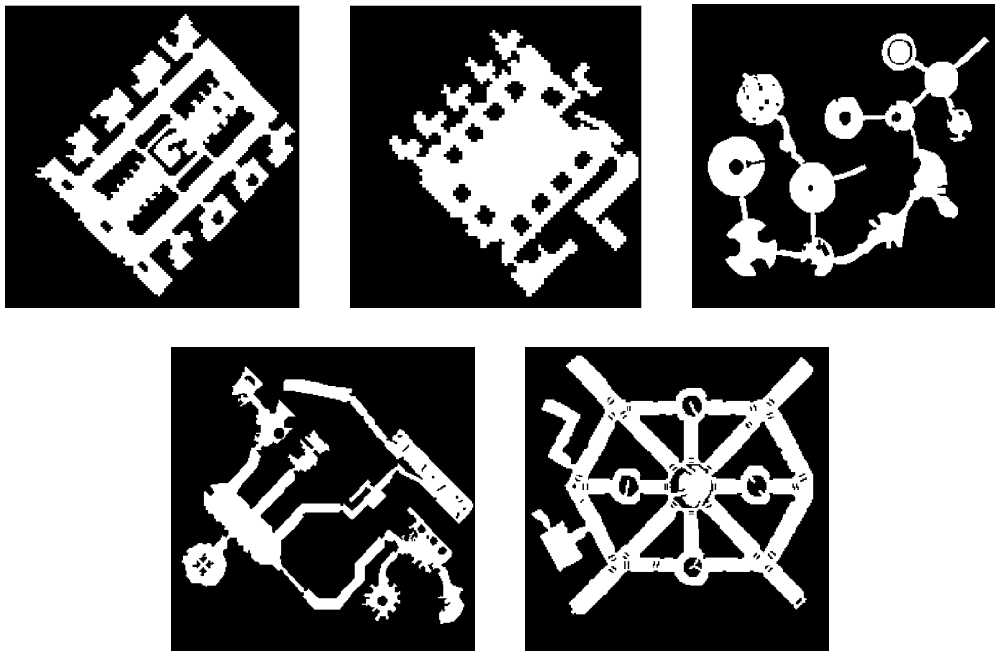

Figure A.1: Maps from the video game *Baldur's Gate* [1].

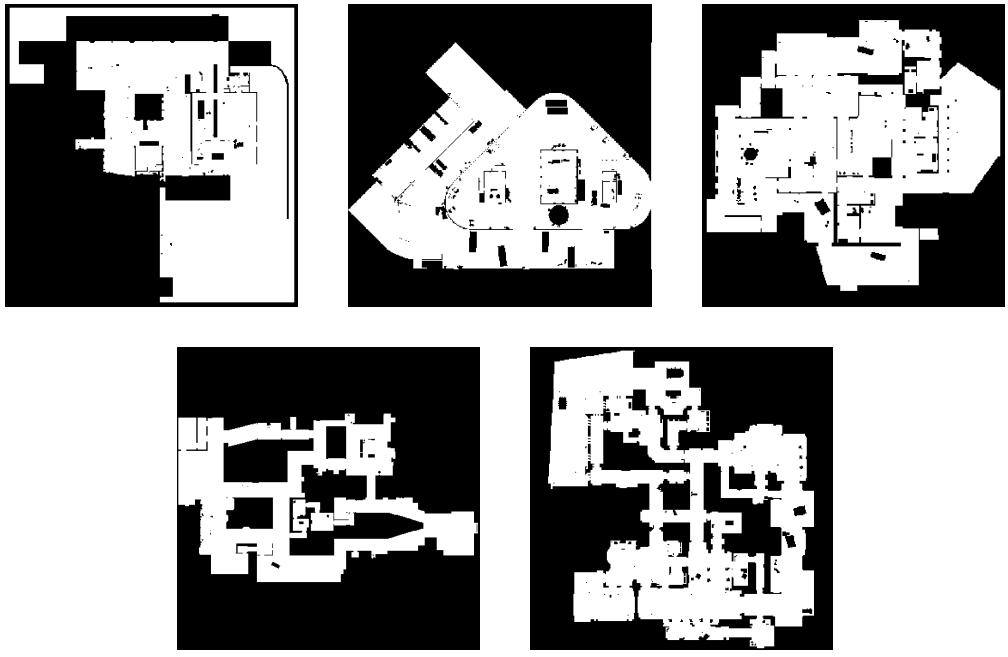

Figure A.2: Maps from the video game *Counter-strike: Source* [33].

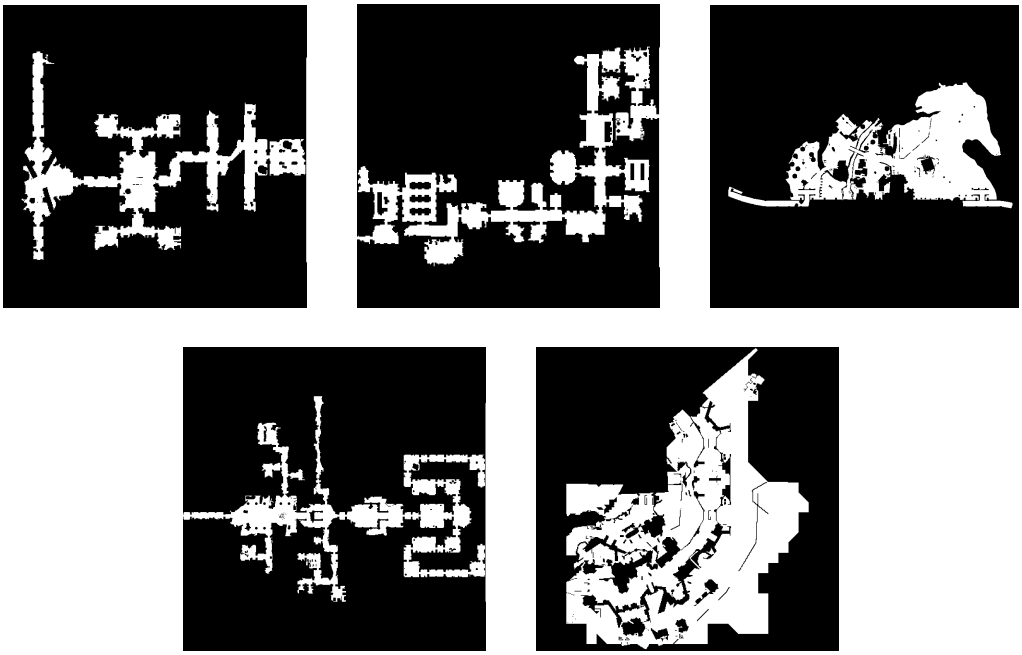

Figure A.3: Maps from the video game *Dragon Age: Origins* [2].

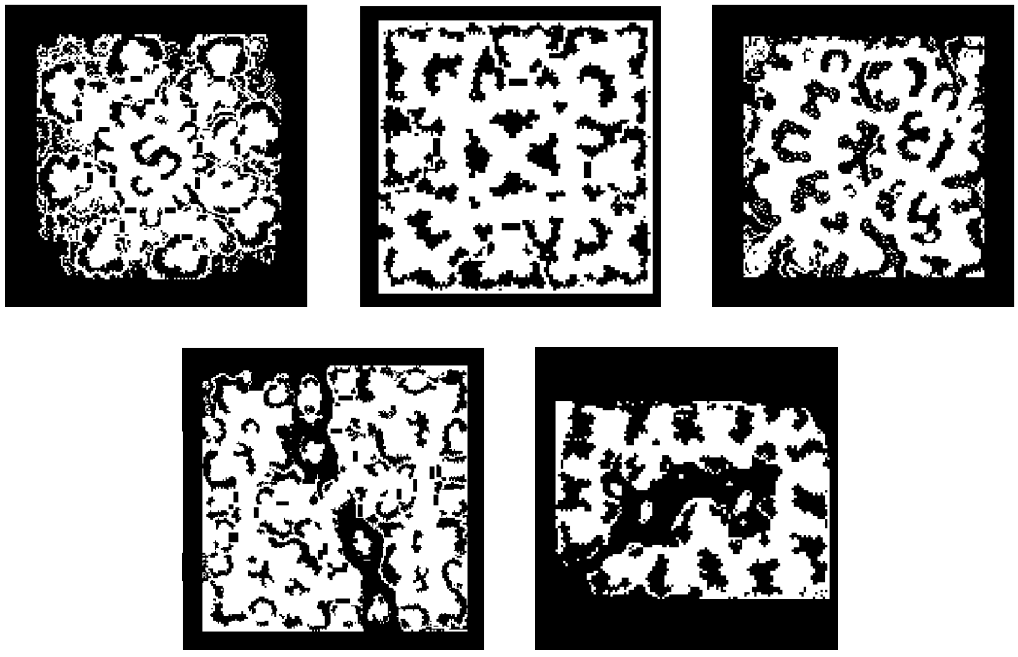

Figure A.4: Maps from the video game *Warcraft 3* [4].

Appendix B

Videogame Pathfinding Plots

In this appendix we present plots of the experimental data collected in Chapter 6.

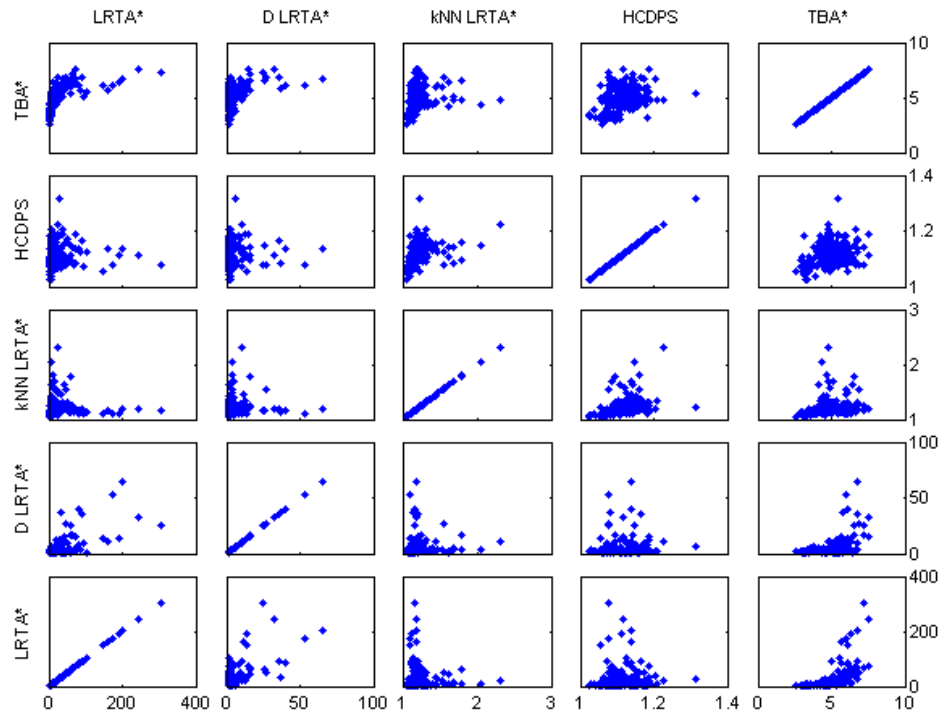

Figure B.1: Mean suboptimality by algorithm against mean suboptimality by algorithm for videogame pathfinding problems.

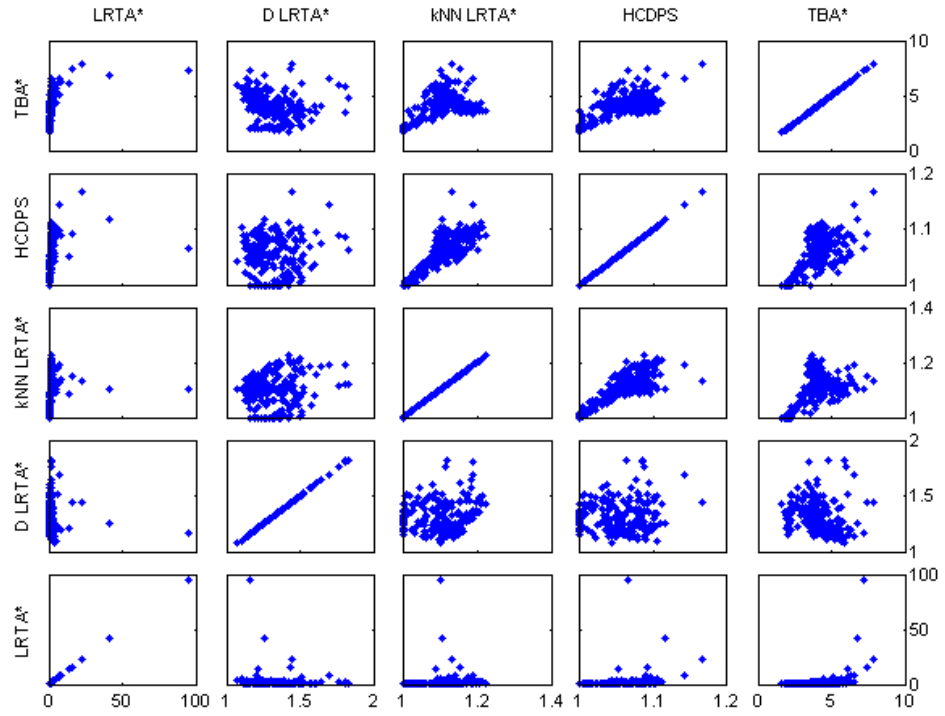

Figure B.2: Median suboptimality by algorithm against median suboptimality by algorithm for videogame pathfinding problems.

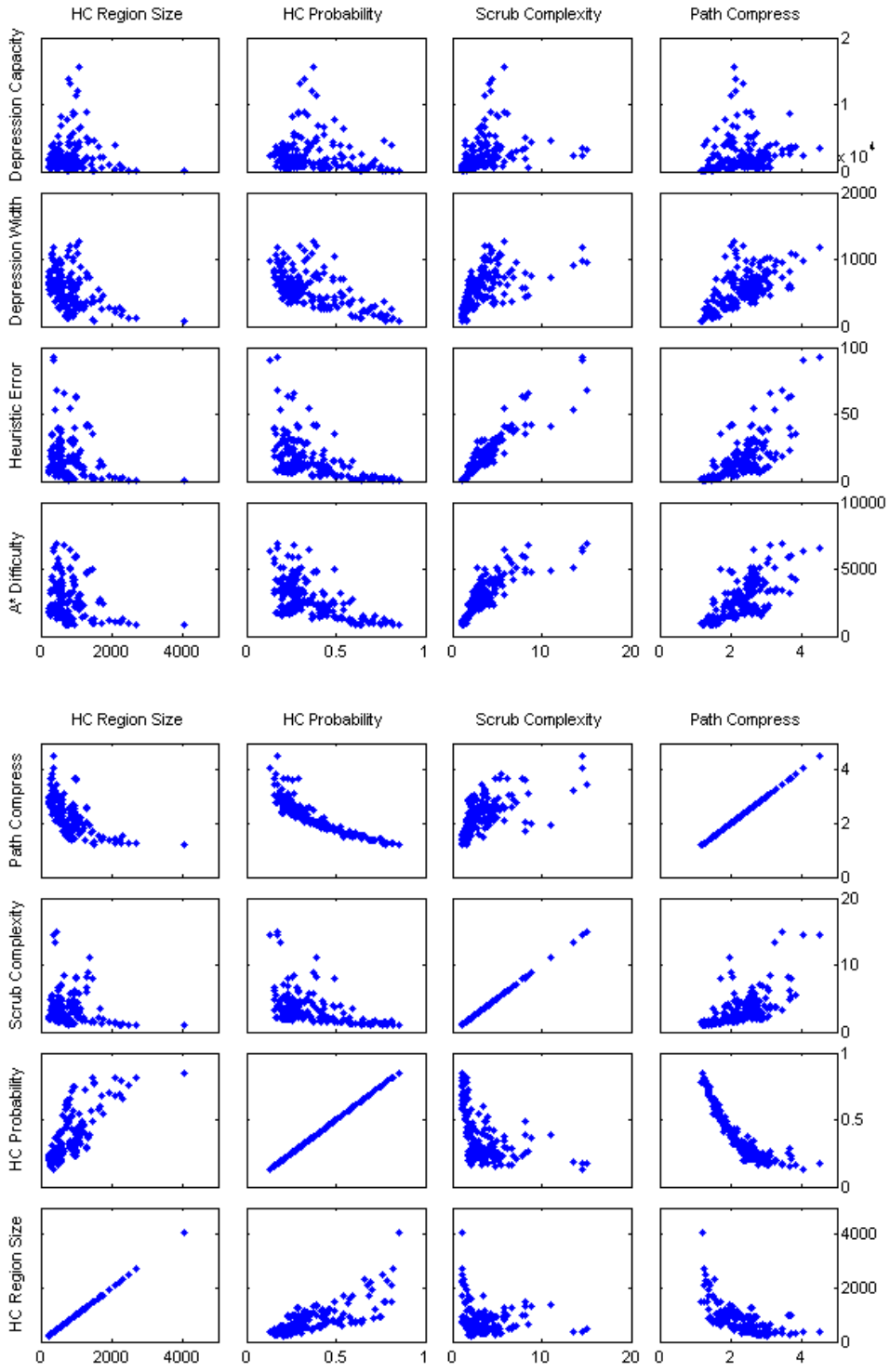

Figure B.3: Complexity measure values against other complexity measure values for videogame pathfinding problems. (Continued in Figure B.4.)

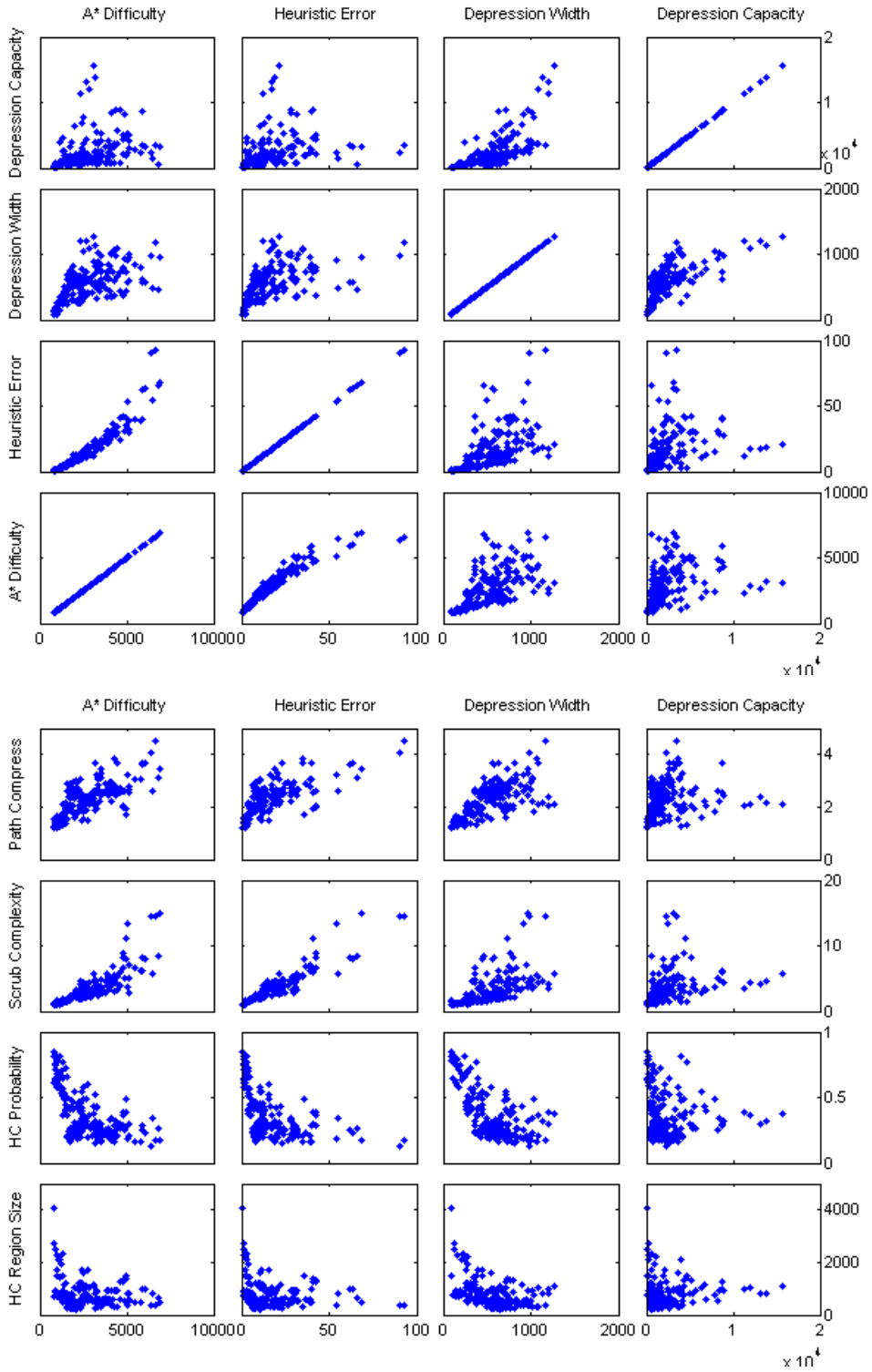

Figure B.4: *Continued:* Complexity measure values against other complexity measure values for videogame pathfinding problems.

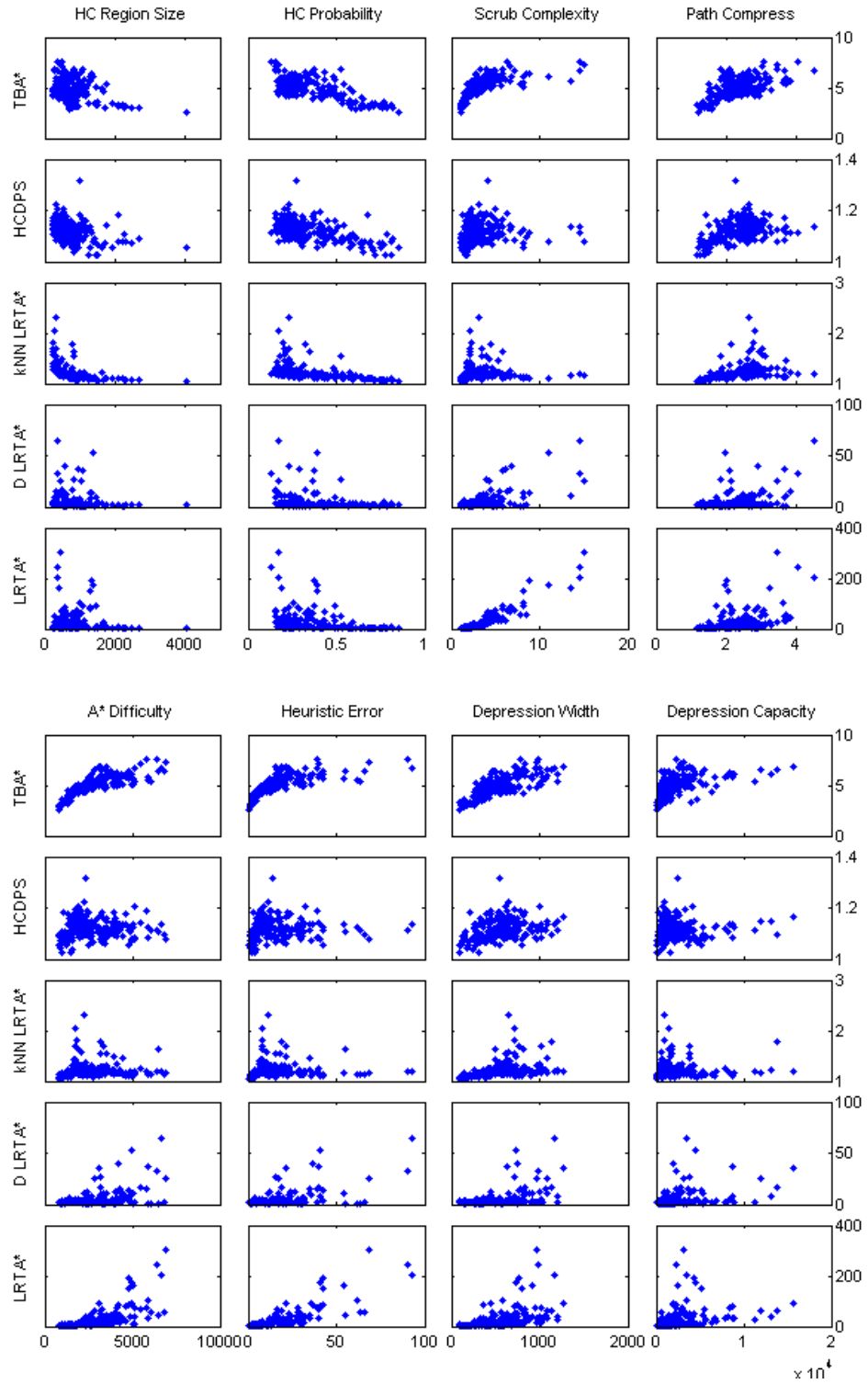

Figure B.5: Complexity measure values against mean algorithm suboptimality for videogame pathfinding problems.

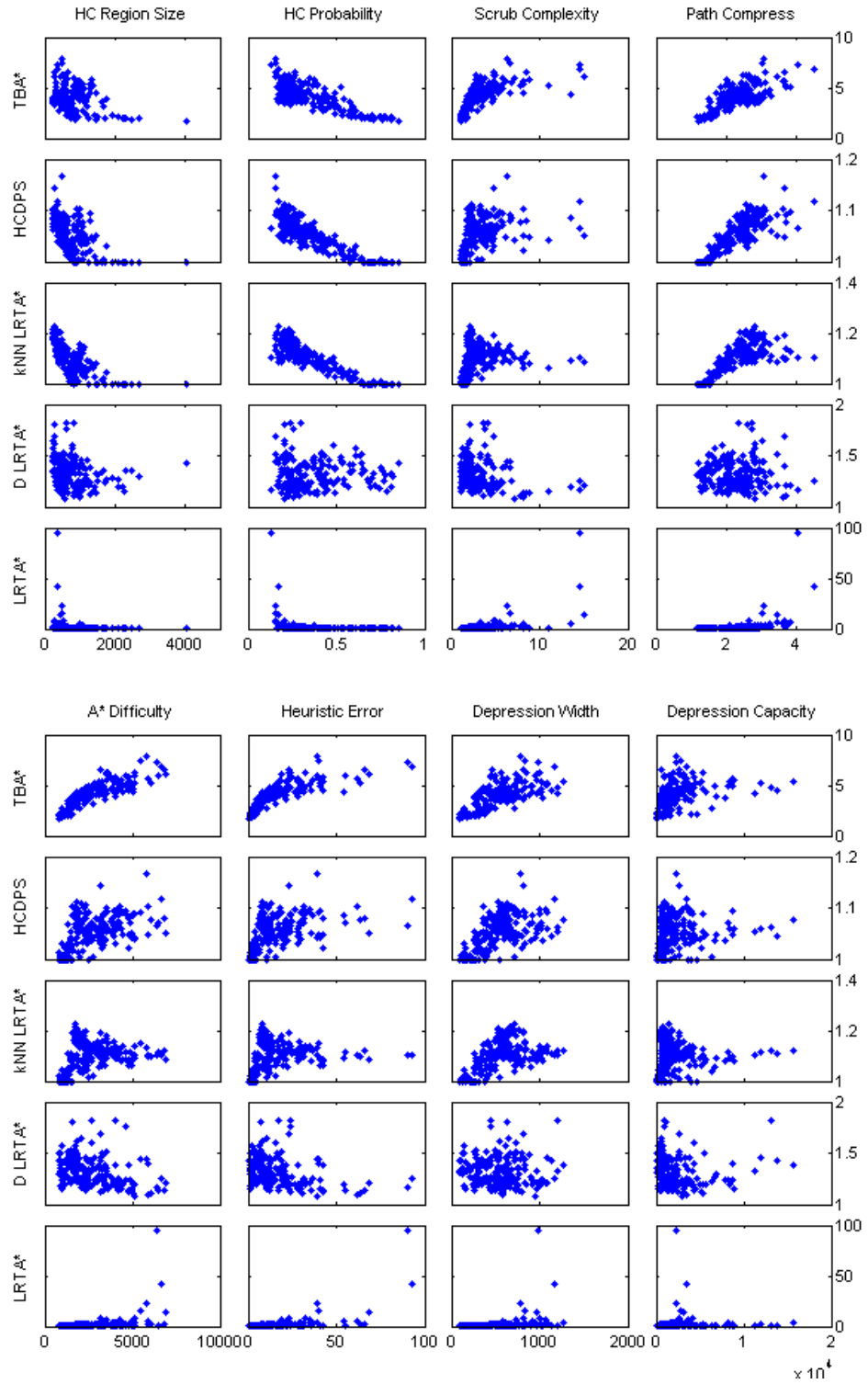

Figure B.6: Complexity measure values against median algorithm suboptimality for videogame pathfinding problems.

Appendix C

Road Map Plots

In this appendix we present plots of the experimental data for road maps in Chapter 8.

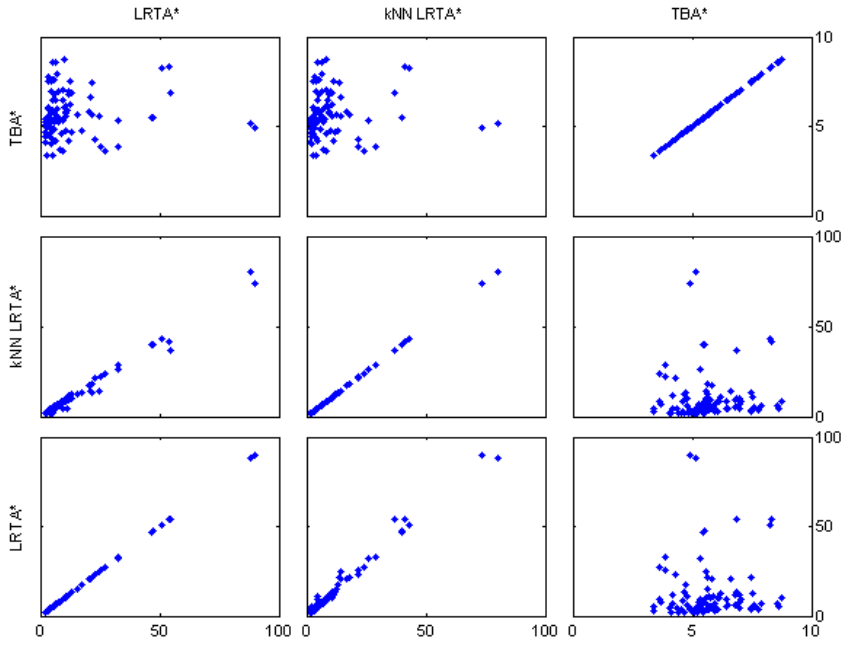

Figure C.1: Mean suboptimality by algorithm against mean suboptimality by algorithm for road map problems.

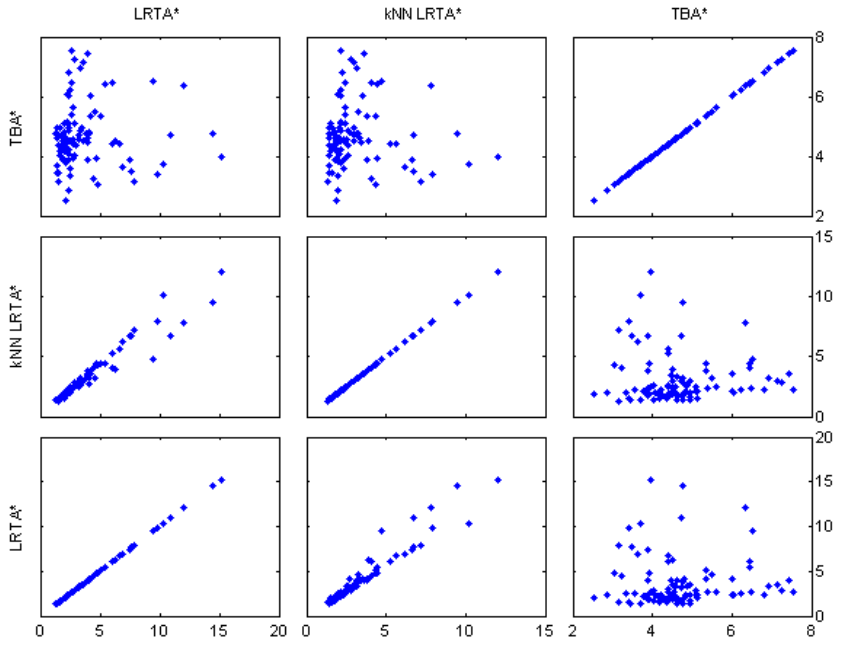

Figure C.2: Median suboptimality by algorithm against median suboptimality by algorithm for road map problems.

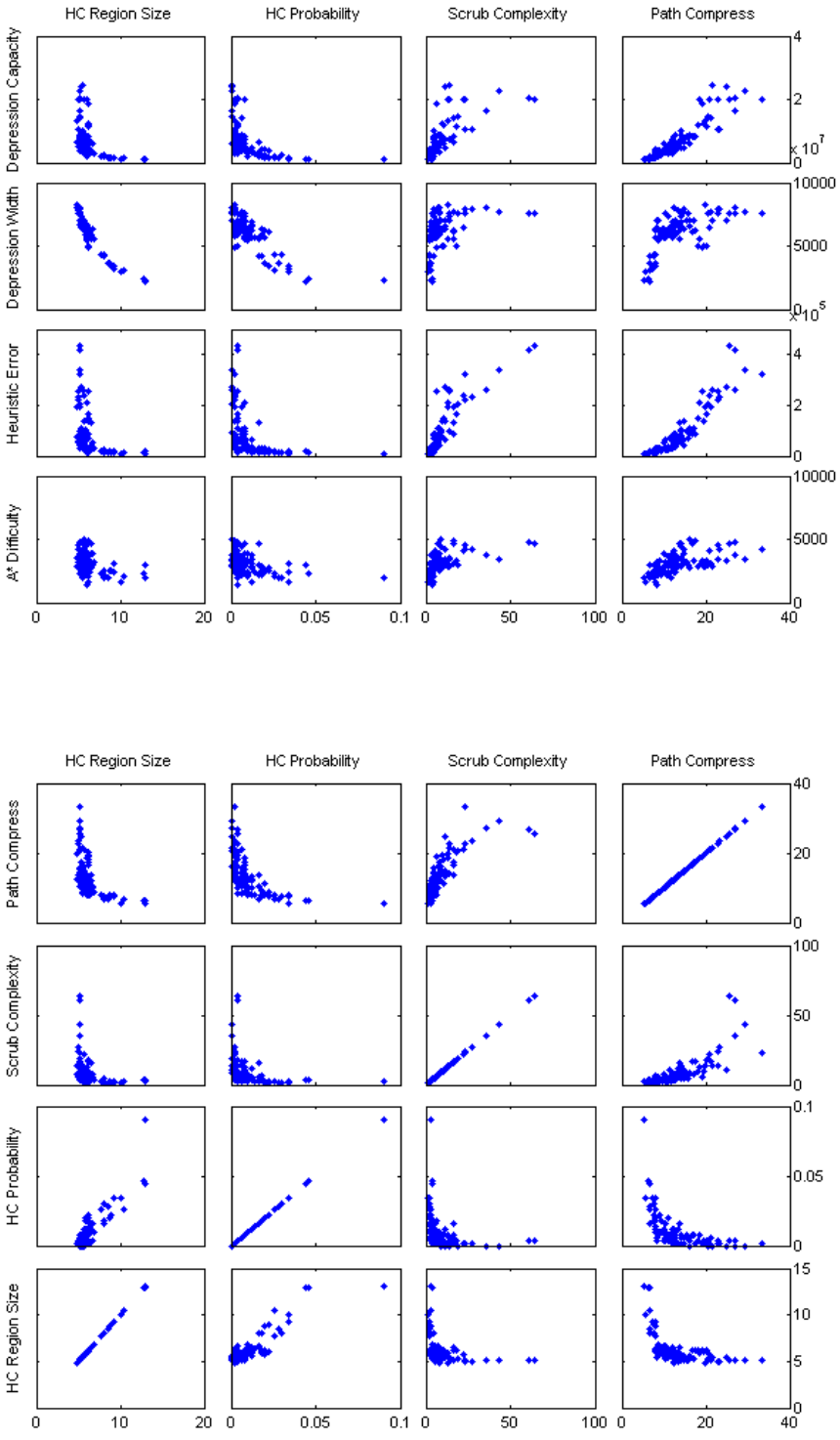

Figure C.3: Complexity measure values against other complexity measure values for road map problems. (Continued in Figure C.4.)

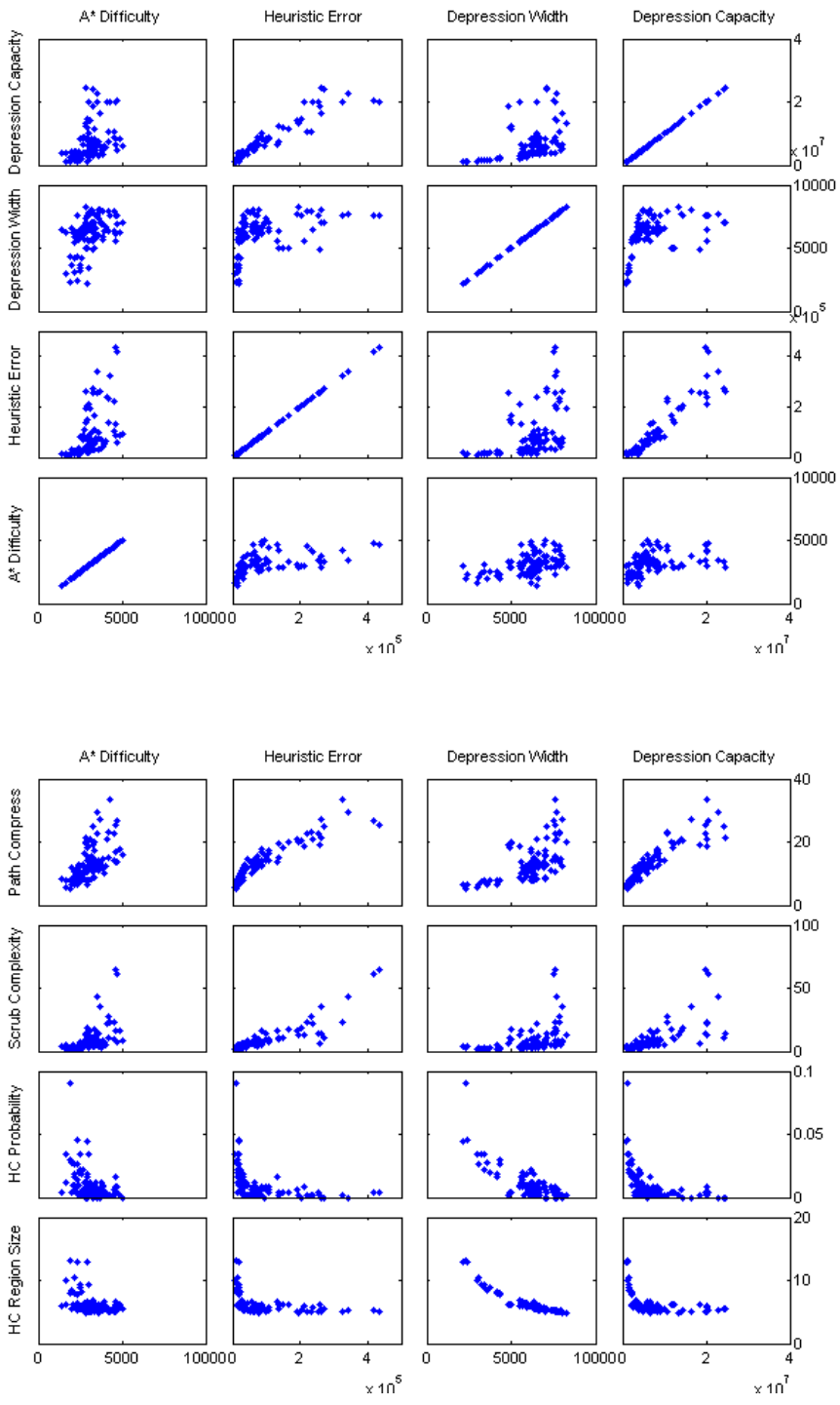

Figure C.4: *Continued:* Complexity measure values against other complexity measure values for road map problems.

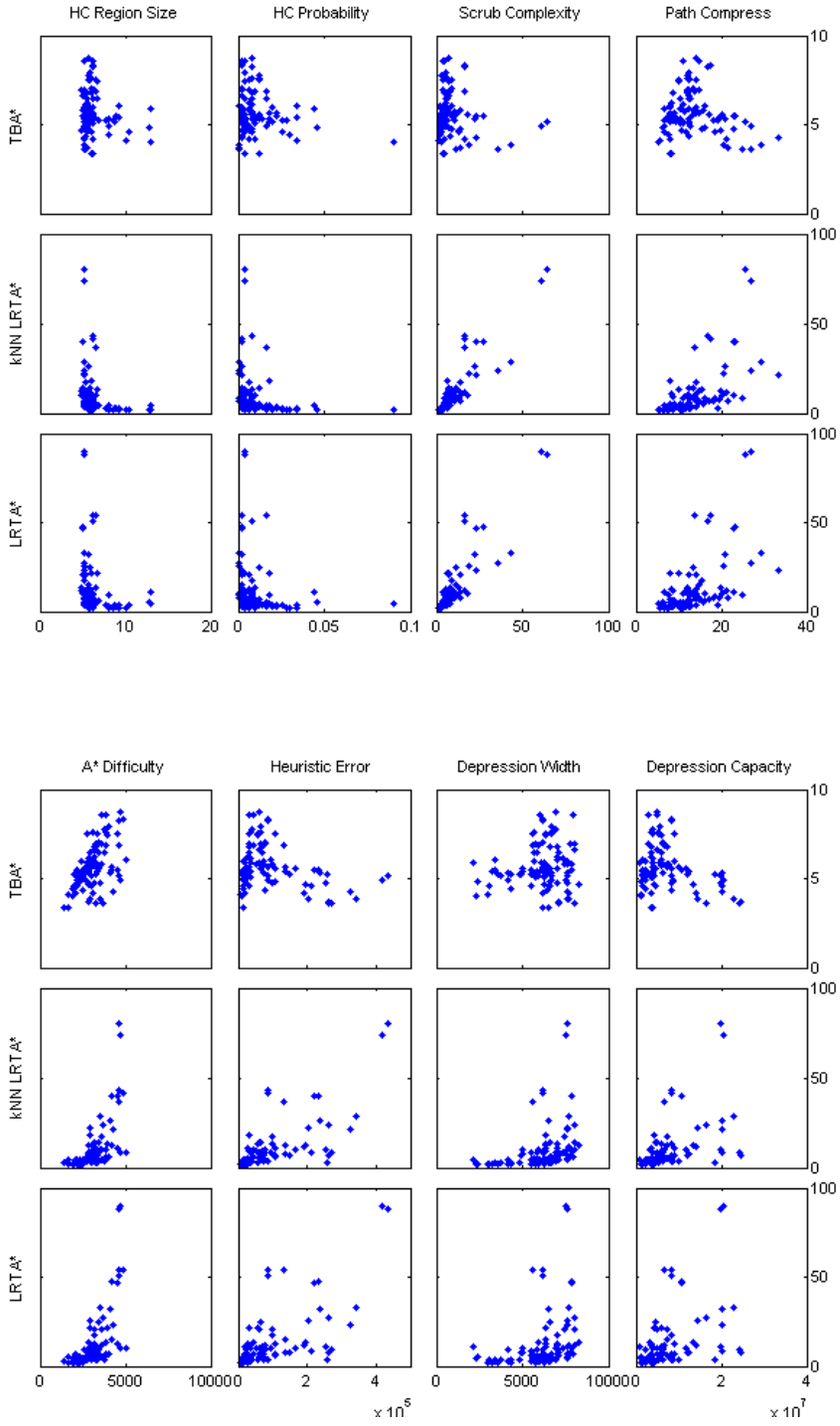

Figure C.5: Complexity measure values against mean algorithm suboptimality for road map problems.

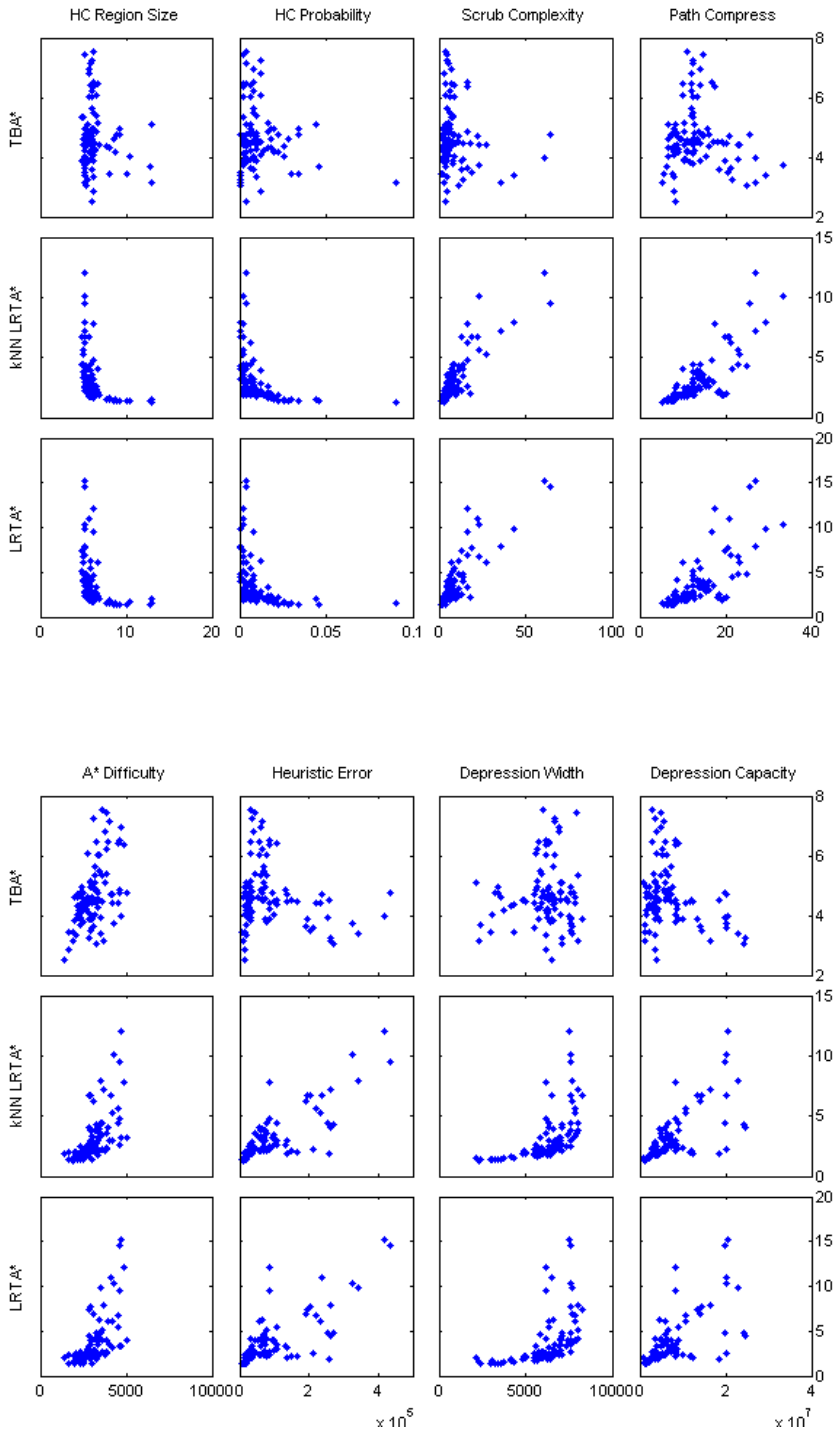

Figure C.6: Complexity measure values against median algorithm suboptimality for road map problems.

Appendix D

Maze Pathfinding Plots

In this appendix we present plots of the experimental data for maze pathfinding in Chapter 8. The apparent clustering of data points in the plots is due to the differing corridor widths in the mazes used. The data points within a cluster belong to mazes with a common corridor width.

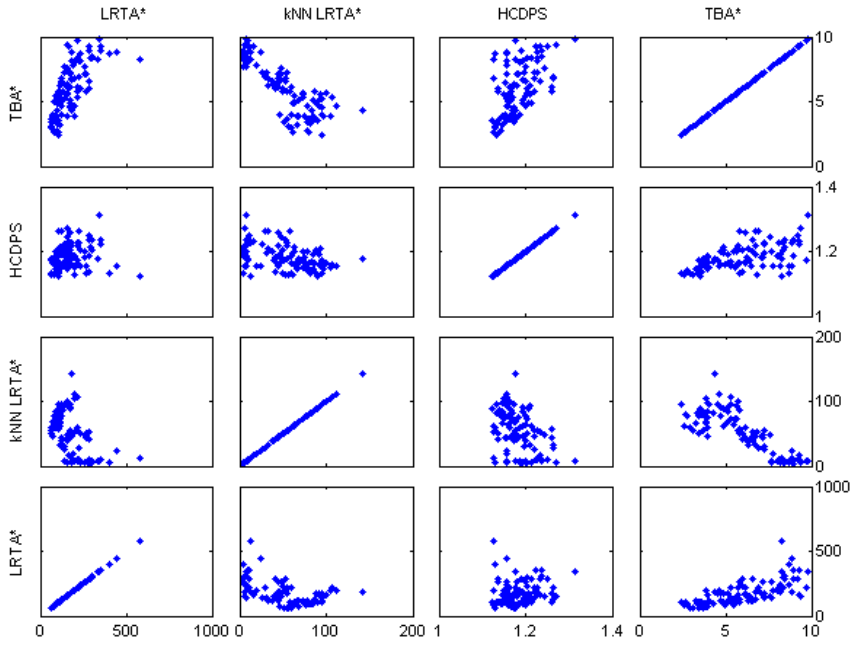

Figure D.1: Mean suboptimality by algorithm against mean suboptimality by algorithm for maze pathfinding problems.

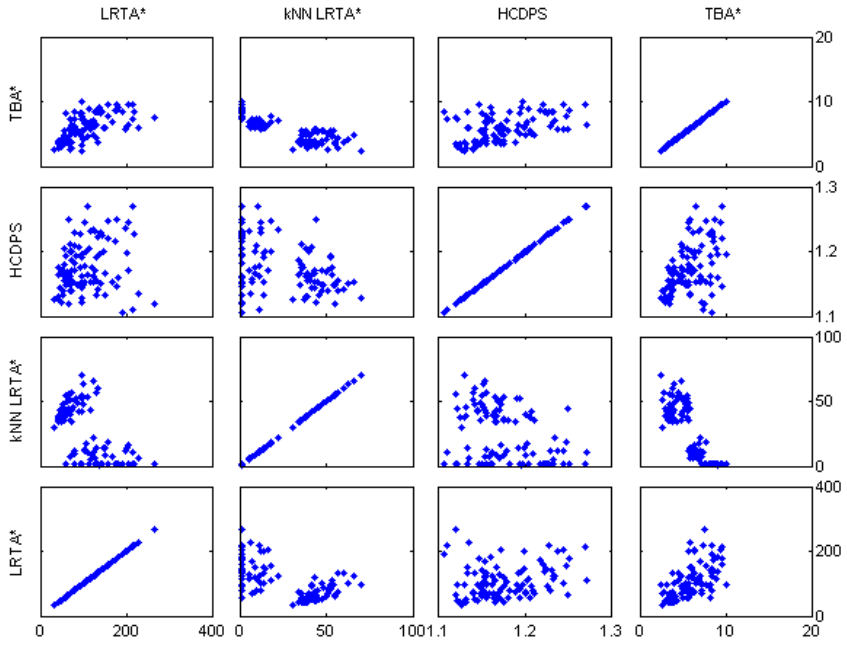

Figure D.2: Median suboptimality by algorithm against median suboptimality by algorithm for maze pathfinding problems.

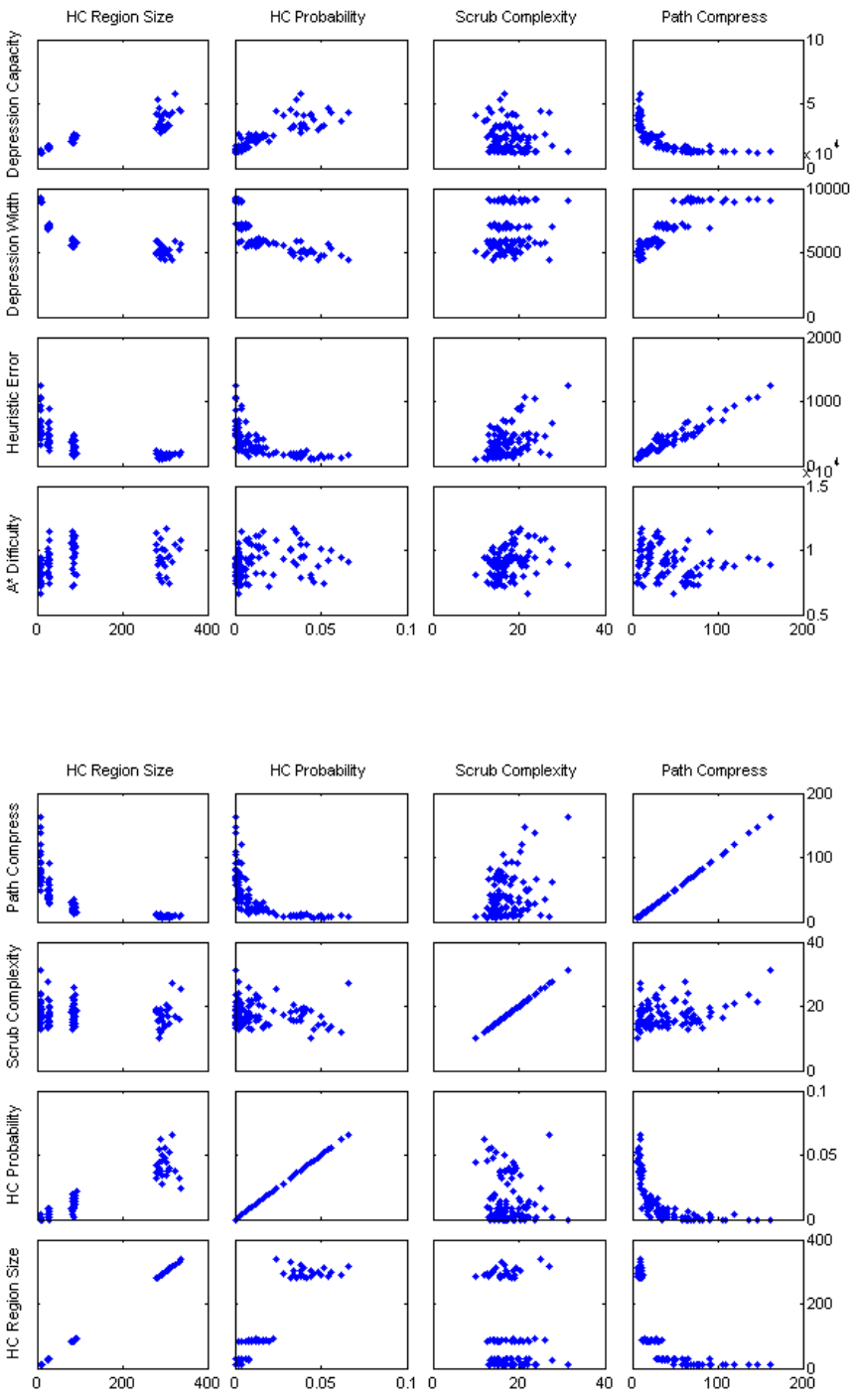

Figure D.3: Complexity measure values against other complexity measure values for maze pathfinding problems. (Continued in Figure D.4.)

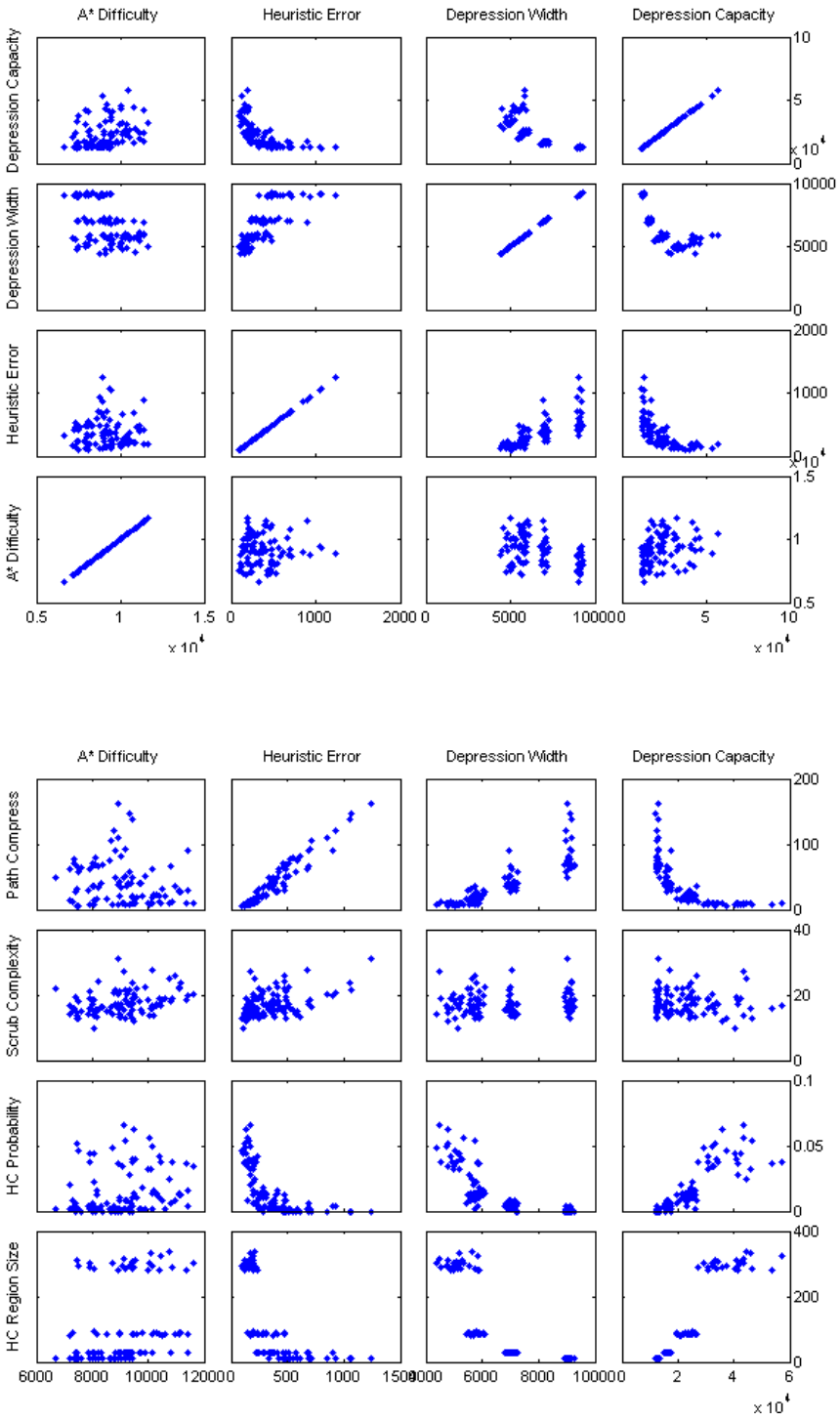

Figure D.4: *Continued:* Complexity measure values against other complexity measure values for maze pathfinding problems.

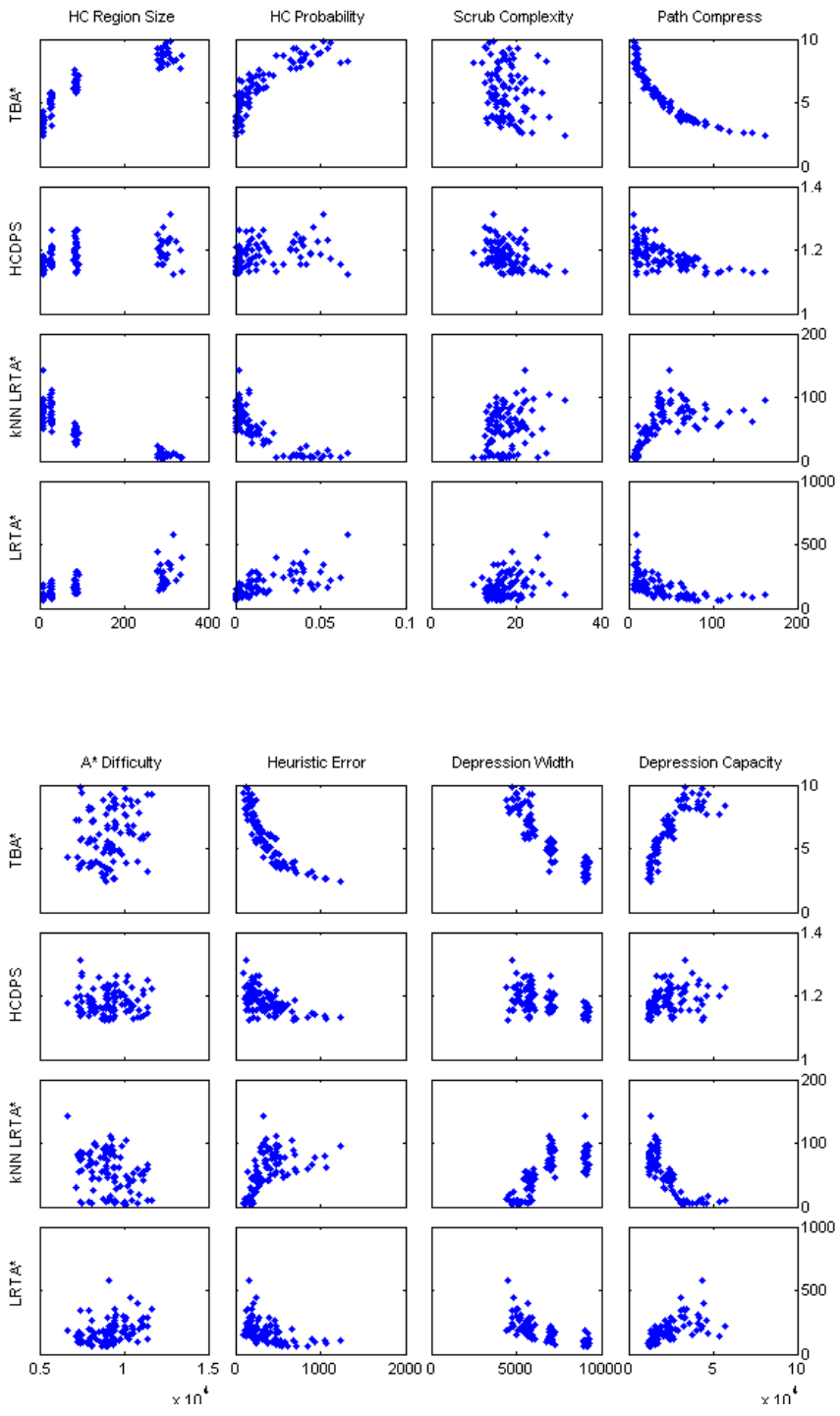

Figure D.5: Complexity measure values against mean algorithm suboptimality for maze pathfinding problems.

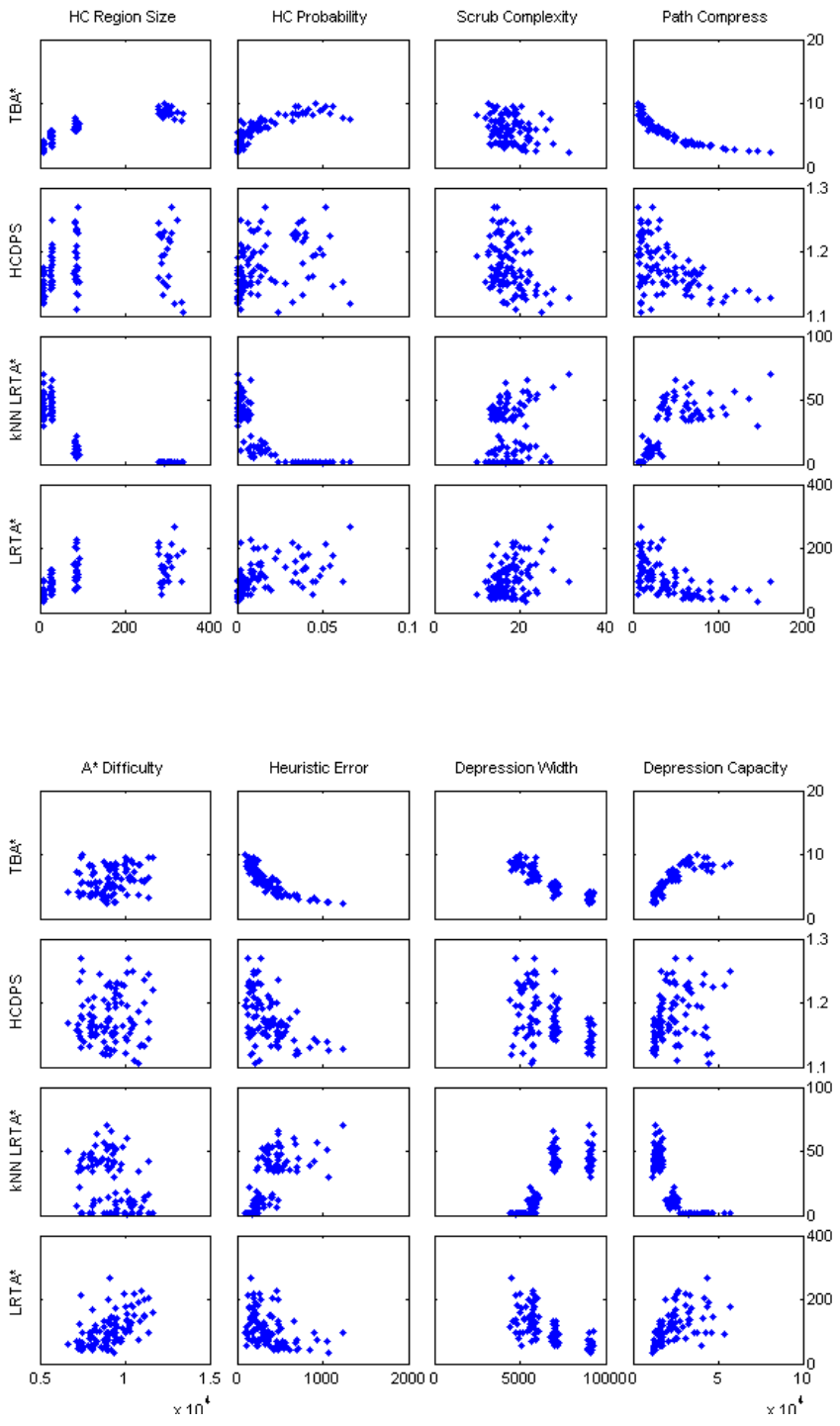

Figure D.6: Complexity measure values against median algorithm suboptimality for maze pathfinding problems.