The Compressed Differential Heuristic

Meir Goldenberg
ISE Department
Ben-Gurion University (Israel)
Mgoldenbe@yahoo.ca

Nathan Sturtevant
CS Department
University of Denver (USA)
Sturtevant@cs.du.edu

Ariel Felner
ISE Department
Ben-Gurion University (Israel)
Felner@bgu.ac.il

Jonathan Schaeffer
CS Department
University of Alberta (Canada)
Jonathan@cs.ualberta.ca

Abstract

The differential heuristic (DH) is an effective memory-based heuristic for explicit state spaces. In this paper we aim to improve its performance and memory usage. We introduce a compression method for DHs which stores only a portion of the original uncompressed DH, while preserving enough information to enable efficient search. Compressed DHs (CDH) are flexible and can be tuned to fit any size of memory, even smaller than the size of the state space. Furthermore, CDHs can be built without the need to create and store the entire uncompressed DH. Experimental results across different domains show that, for a given amount of memory, a CDH significantly outperforms an uncompressed DH.

1 Introduction and overview

A common research direction in heuristic search is to develop memory-based heuristics in the form of look-up tables built during a preprocessing phase. Pattern databases (PDBs) (Culberson and Schaeffer 1998) are a common technique, which is usually implemented for implicit exponential domains where good domain abstractions are available.

Felner et al. (2007) showed that PDBs can be compressed by merging several PDB entries into one. If the entries to be merged have similar values, then most of the information stored in the original PDB is preserved. Similar performance can be achieved using less memory or, alternatively, constant memory size can retain more information, enabling a faster search. (Felner et al. 2009) concluded that PDBs are not well-suited for explicit domains, such as map-based path finding.

True distance heuristics (TDHs) are a class of memory-based heuristics developed for explicit domains (Sturtevant et al. 2009; Felner et al. 2009; Goldenberg et al. 2010). TDHs store distances between selected pairs of states in the original state space (as opposed to distances in an abstract space stored by a PDB). This information is used to compute admissible heuristic values between any pair of states. TDH variants are distinguished by the pair selection criteria and by the way the heuristic is computed.

A simple and effective TDH is the *differential heuristic* (DH) which was independently developed and used by a number of researchers (Goldberg and Harrelson 2005)(G&H), (Ng and Zhang 2002). A set of *pivot* states P is determined. A DH stores the exact distance from every state x to *all* states $p \in P$. G&H fit DHs into the limited memory of a portable device by storing part of the information on the secondary storage device. We suggest a different approach to reducing the memory usage of DHs. Our approach can be combined with the approach of G&H to achieve additional performance gains on portable devices.

We introduce the *compressed* DH (CDH) which stores a part of the DH's look-up table (hereafter, we refer to DH as the *regular* DH to distinguish from CDH). In CDH, a subset of the $|P|$ distances are stored for each state. When a distance from a state x to a pivot p is missing from the table, it is substituted by distances that were preserved by the compression in two ways: 1) computing upper and lower bounds on the missing distances from the goal to the pivots, and 2) using BPMX (Zahavi et al. 2007) (see Section 5). Any A^*-based search algorithm can use CDH.

Experimental results across multiple domains (game maps, road maps, and a robotic arm) show that, for a given amount of memory, a CDH outperforms a regular DH of the same size. Furthermore, CDHs can be effectively used in applications, such as video games, where even the minimal amount of memory needed by a DH (i.e. the size of the domain) is not afforded. CDHs perform well because they are based on a larger DH containing a richer diversity of heuristic values. CDHs can be built without the need to create and store the entire uncompressed DH. While this paper focuses on DHs, the ideas are applicable to other types of TDHs.

2 DH and its compressed version

Given a weighted graph $G = (V, E, W)$, consider the problem of finding the shortest path between an arbitrary pair of states, $s, g \in V$ (a problem instance). Denote the cost of the shortest path between any two states $a, b \in V$ by $d(a, b)$. We assume that a base heuristic, $h_{base}(a, b)$, is defined for the domain (e.g., Manhattan distance for grid-based searches). Furthermore, we assume that $m|V|$ memory is available for storing the heuristic look-up table for some positive real number m. This table is computed once for a given input graph and is used across multiple problem instances.

Definition: Differential heuristic (DH). Choose $P \subset V$ ($|P| \ll |V|$) states called *pivots*, such that $|P| = m$. For each state $a \in V$, the look-up table of DH stores the distances from a to all pivots $p \in P$, i.e, exactly $m|V|$ distances $d(a,p)$ are stored in the table. For any given state a and pivot p, the following heuristic estimate of $d(a,g)$ with respect to p is admissible:

$$dh_p(a,g) = |d(a,p) - d(g,p)| \tag{1}$$

and the *differential heuristic* (DH) is:

$$dh(a,g) = \max\left\{h_{base}(a,g), \max_{p\in P} d_p(a,g)\right\}.$$

We refer to the DH just defined as the *regular* DH and illustrate it in Figure 1(left). Assume the search has reached state a and that a heuristic estimate to g is required. Let p_1 and p_2 be pivots, so that the DH stores $d(a,p_1) = 4$, $d(g,p_1) = 9$, $d(a,p_2) = 3$ and $d(g,p_2) = 14$. The resulting heuristic estimates with respect to the two pivots are $dh_{p_1}(a,g) = |4-9| = 5$ and $dh_{p_2}(a,g) = |3-14| = 11$. Therefore, $dh(a,g) = \max\{5,11\} = 11$. One can easily verify that the regular DH is admissible and consistent.

2.1 Compressed DH

A CDH stores part of the information contained in the regular DH. For each state $a \in V$, only distances to pivots from a subset $P_a \subset P$ are stored. In Figure 1(right), $P_a = \{p_1,p_2\}$ but $P_g = \{p_1\}$. Solid edges represent distances that are stored in the CDH. The methods for selecting the pivots P and for choosing the subset P_a for each a are discussed later.

Assume that the A^* search reached state a and that an admissible estimate to the goal, $h(a,g)$, is needed. Let p be a pivot to which the distance from a, $d(a,p)$, is stored in the CDH (i.e. $p \in P_a$). There are now two cases.

Case 1: $d(g,p)$ is also stored. In this case, the CDH with respect to p is calculated as in the regular DH: $cdh_p(a,g) = dh_p(a,g)$. The pivot p_1 in Figure 1(right) demonstrates this case, since distance to p_1 is stored both from a and from g. Therefore, $cdh_{p_1}(a,g) = dh_{p_1}(a,g) = |4-9| = 5$.

Case 2: $p \notin P_g$ (p_2 in our example). In this case, $cdh_p(a,g)$ is defined as follows. Equation 1 is based on the inequalities $d(a,g) \geq d(a,p) - d(g,p)$ and $d(a,g) \geq d(g,p) - d(a,p)$. Therefore, equation 1 can be re-written as

$$dh_p(a,g) = \max\{d(a,p) - d(g,p), d(g,p) - d(a,p)\}. \tag{2}$$

Let $\overline{d}(g,p)$ and $\underline{d}(g,p)$ denote an upper and lower bound on the missing distance $d(g,p)$. Using them with equation 2 gives the following admissible heuristic with respect to p:

$$cdh_p(a,g) = \max\left\{d(a,p) - \overline{d}(g,p), \underline{d}(g,p) - d(a,p)\right\}.$$

Maximizing over all pivots yields the *compressed differential heuristic* (CDH):

$$cdh(a,g) = \max\left\{h_{base}(a,g), \max_{p\in P_a} cdh_p(a,g)\right\}.$$

2.2 Computing the bounds

Let $p \in P$ be a pivot to which the distance from the goal state g, $d(g,p)$, is not stored as in Case 2) above. Let x be an arbitrary state such that the distance from x to p is stored. More formally, $p \notin P_g$ and $p \in P_x$. The triangle inequality for the vertices $\{g,x,p\}$ gives the following bounds on $d(g,p)$ with respect to x:

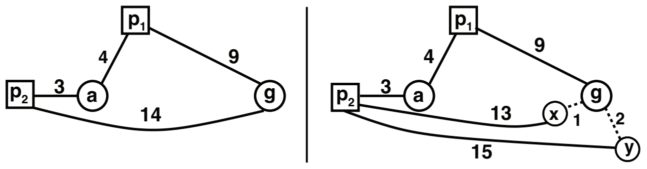


Figure 1: Regular DH (left) and compressed DH (right).

$$\overline{d}_x(g,p) = d(g,x) + d(x,p), \text{and}$$

$$\underline{d}_x(g,p) = |d(g,x) - d(x,p)| \tag{3}$$

For a given problem instance, the goal state remains unchanged for the duration of the search. Therefore, for each pivot p, the bounds $\overline{d}(g,p)$ and $\underline{d}(g,p)$ can be precomputed before the main search begins. These bounds are stored in a table called the *bounds table*, to be used throughout the search for this particular problem instance.

The *Bounding* procedure calculates the bounds by applying equation 3 to states that are in close proximity to g as follows. Initially, for each pivot $p \notin P_g$, we set $\overline{d}(g,p) = \infty$ and $\underline{d}(g,p) = h_{base}(g,p)$. Perform a Dijkstra search from g. For each expanded state x and for each $p \in P_x$, we use the distance $d(x,p)$ to compute the bounds $\overline{d}_x(g,p)$ and $\underline{d}_x(g,p)$. These bounds are then used to improve the global upper and lower bounds on $d(g,p)$. This process is continued until at least r distances $d(x,p)$ have been seen for each pivot $p \notin P_g$, where r is a parameter.

Let us see how the Bounding procedure with $r = 2$ works for the example in Figure 1(right). Recall that $p_2 \in P_a$ but $p_2 \notin P_g$. Since $d(g,p_2)$ is missing from the table, the Bounding procedure will compute bounds on the missing distance. States around g are expanded in Dijkstra's order. State x will be expanded first. Since $p_2 \in P_x$, the bounds with respect to x will be computed according to equation 3: $\overline{d}_x(g,p_2) = 13+1 = 14$ and $\underline{d}_x(g,p_2) = |13-1| = 12$. Had we set $r = 1$, the Bounding procedure would stop here. Since $r = 2$, the bounding procedure continues and expands y. Since $p_2 \in P_y$, the bounds with respect to y are computed: $\overline{d}_y(g,p_2) = 15+2 = 17$ and $\underline{d}_y(g,p_2) = |15-2| = 13$. The global lower bound is updated to $\underline{d}(g,p_2) = 13$, while the global upper bound remains unchanged: $\overline{d}(g,p_2) = 14$. After computing $cdh_{p_2}(a,g) = \max\{3-14, 13-3\} = 10$, we obtain $cdh(a,g) = \max\{cdh_{p_1}(a,g), cdh_{p_2}(a,g)\} = \max\{5,10\} = 10$. Note that the regular DH in the above example achieved a larger heuristic value of $dh(a,g) = 11$ by using more memory.

Given a problem instance $s,g \in V$, the search driven by the CDH consists of two stages: **(1)** the Bounding procedure and **(2)** the main search. Both stages must be included in the cost of finding a solution. Note that the Bounding procedure expands at least $r|P|/m$ nodes. Our experiments show that often choosing relatively large values of r justifies itself by improving the quality of CDH.

It is important to distinguish the Bounding procedure (which is executed for each problem instance) from the main preprocessing phase, which computes the CDH's look-up table once for a given input graph to be used across multiple problem instances.

3 Management of pivots

Given $m|V|$ memory, a regular DH is forced to use exactly $|P| = m$ pivots as the distance from each state to *all* pivots is stored. The effectiveness of the regular DH will be determined by where the m pivots are placed. A CDH is more flexible. It can use any number of pivots and then decide which and how many distances to store for each state based on the available memory m. We discuss these issues in turn.

3.1 Placement strategy

Ideally, the strategy for selecting the pivot locations should ensure the highest heuristic values possible. We follow (Goldberg and Werneck 2005) and refer to the quality of placement of a set of pivots by the term "coverage" without regards to any particular measure of this quality. A good placement strategy covers the state space well.

(Goldberg, Kaplan, and Werneck 2009) (GKW) state that the best performing of the known placement strategies is *Avoid*. We therefore use Avoid in all of our experiments.[1] The main idea is to "avoid" areas that are well covered by the existing pivots. Given that $k \geq 0$ pivots have been chosen, pivot $k + 1$ is chosen as follows:

(1) Choose a root state t. GKW give a rule that biases the choice of the root towards states located far away from the existing pivots. In our implementation, the farthest state from the existing pivots was chosen.

(2) Construct a spanning tree rooted at t. Two rules are applied in sequence to choose a leaf of that tree to be the next pivot. The first rule chooses a subtree whose states have low heuristic values to t given the current selected pivots. The second rule chooses a particular leaf of that subtree to be the next pivot (see pages 22-23 of GKW for more details).

3.2 Pivots in CDH

Two phases precede the CDH look-up table computation:

(1) Determining the number of pivots $|P|$. Large number of pivots provide a diversity of heuristic values. However, due to the compression, many of the state-to-pivot distances will not be stored in the CDH's look-up table. Therefore, the CDH will rely more on the bounds produced by the Bounding procedure. With a large number of pivots, these bounds are less accurate as pivots might be located far from the goal state. Furthermore, the cost of running the Bounding procedure increases, since more states have to be expanded before r distances to each pivot are seen. We do not have a general solution to find the right balance for this trade-off. Therefore, we determine $|P|$ by performing a tuning phase as described in the experimental section.

(2) Constructing the set P_a for each state a. The computation of CDH, $cdh(a, g)$, requires that the identities of the pivots in P_a be known at the time of heuristic computation. Therefore, in general, it may be required that the sets P_a be stored explicitly in the CDH's look-up table, along with the distances from a to the pivots in P_a. To avoid this extra storage, implicit rules for choosing the size of P_a and the identities of pivots in P_a need to be formulated. This is achieved by defining two functions: 1) An ordering $index(a, p) : V \times P_a \rightarrow \{0, 1, \ldots, |P_a| - 1\}$ on the elements of P_a that has an inverse which can be computed efficiently. 2) The inverse $identity(a, i) : V \times \{0, 1, \ldots, |P_a| - 1\} \rightarrow P_a$. This function determines which pivot the i^{th} distance stored for a refers to. With this information, $cdh(a, g)$ is computed using Procedure 1.

Procedure 1 Computing heuristics from a CDH.

```
Output: $cdh(a, g)$
1: Compute $|P_a|$
2: $cdh(a, g) \leftarrow h_{base}(a, g)$
3: for $i = 0$ to $|P_a|$ do
4:     $p \leftarrow identity(a, i)$
5:     Compute $cdh_p(a, g)$ // Consult the bounds table
6:     $cdh(a, g) \leftarrow \max\{cdh(a, g), cdh_p(a, g)\}$
7: end for
```

We realize this idea by choosing pivots for P_a using a round-robin distribution as follows. Suppose that $m|V|$ memory is given (m may be non-integer). The idea is to distribute distances to pivots among the states in such a way that, for each state a, the immediate neighborhood of a should contain distances to as many pivots as possible. A simple way to approximate this behavior is to distribute these distances in a round-robin fashion[2]. We show this with an example. Assume $2|V|$ memory (i.e. $|P_a| = 2$ for all a) and eight pivots (i.e. $|P| = 8$). The pivots are arranged around a circle. For state a with index fifteen, we start by assigning to a the distance to pivot $15 \bmod 8 = 7$ and continue to go around the circle stepping $|P|/|P_a| = 4$ each time. One can easily extend this example to a generic formula for any a, $|P|$ and integer m. With some more mathematical effort, this idea can be extended to non-integer values of m as well.

The round-robin distribution does not use any graph-specific information when assigning a pivot to a state. However, it enables cutting the memory by half as identities of pivots need not be stored. This is very important as one can now store distances to twice as many pivots for each state.

4 Building the compressed DH

A CDH can be built directly, without the need to first build and store the much larger uncompressed regular DH. This is done in two phases:

(1) For each pivot $p \in P$, form a list of states S_p which will store a distance to p. With each state $a \in S_p$, store the index of p in P_a, i.e. $index(a, p)$.

(2) For each pivot p, perform a breadth-first search to determine all distances $d(a, p)$ for $a \in S_p$. For each $a \in S_p$, the distance $d(a, p)$ is stored at position $index(a, p)$, computed in step (1), in the array of distances stored with a.

[1]MaxCover, based on Avoid, results in slightly faster searches, but requires much more time for computing the pivot's locations.

[2]It seems that any hashing function that distributes distances to any given pivot somewhat uniformly among the states will closely reproduce our results.

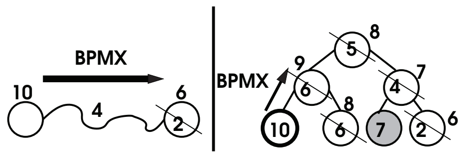


Figure 2: BPMX examples.

5 BPMX used for CDH

A heuristic h is *inconsistent* if there exists a pair of states a, b, such that $|h(a,g) - h(b,g)| > d(a,b)$. CDHs are inconsistent, since two neighboring states a and b may use two different sets of pivots to compute their heuristic estimates.

Bidirectional pathmax (BPMX) (Zahavi et al. 2007) is a method that propagates admissible, inconsistent heuristics values as follows. For two arbitrary states $a, b \in V$, the heuristic estimate $h(a,g)$ can be updated to be $\max\{h(a,g), h(b,g) - d(a,b)\}$. The technique is illustrated in Figure 2(left). BPMX uses this rule in both directions of a search tree. In Figure 2(right), assuming unit edge costs, the h-value of the left grandchild (10) is propagated up and then down the tree, increasing heuristic estimates of all states in its neighborhood except for the gray node.

Using inconsistent heuristics can cause A* to re-expand nodes many times. (Zhang et al. 2009) suggests that implementing BPMX on top of A* significantly reduces the number of node re-expansions. They concluded that only propagation between an expanded node v and its immediate children, BPMX(1), is worthwhile. We have confirmed this result in our work, and all our experiments use BPMX(1).

6 Summary of the approach

Our approach can be summarized as follows. Given an input graph and the amount of memory, the following preprocessing phase is performed:

(1) Determine the number of pivots $|P|$. This is done empirically as discussed below.

(2) Place the pivots using the Avoid placement strategy.

(3) Distribute distances to pivots among states as described in Section 3.2.

The above preprocessing phase is performed once for a given input graph. By contrast, the following two phases are performed to solve a given problem instance:

(1) Compute the bounds on the distances from the goal state to the pivots using the Bounding procedure.

(2) Perform the main A^*-based search driven by the CDH, which is computed using Procedure 1. Since the CDH is admissible, this search produces an optimal path. BPMX(1) is used to propagate heuristic values.

7 Experimental results

We performed experiments to show the key properties of CDHs and compare their performance with regular DHs. Three real-world domains were used, as shown in Figure 3.

(1) Dragon Age maps. These computer game maps, taken from Bioware's *Dragon Age: Origins* (DAO),[3] are

[3]http://movingai.com/benchmarks.

Figure 3: The environments.

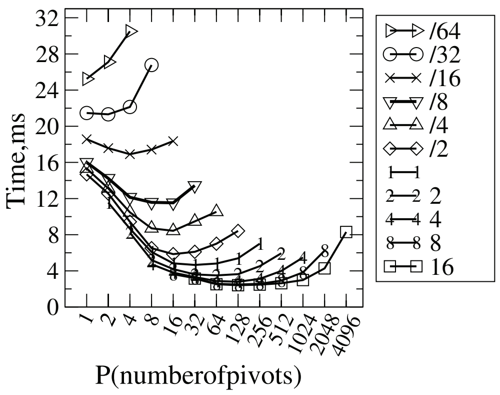


Figure 4: Setting the parameters of CDH (DAO map).

eight-connected grids. Octile distance was used as the base heuristic. The reported results are for the map `ost100d` (shown in the figure), the second largest map in the game (the largest map had few obstacles). Our experiments confirmed the reported trends using other Bioware maps.

(2) Road maps. The road maps of North American cities and states were used in these experiments.[4] Air distance was used as the base heuristic.

(3) Robotic Arm. A segmented arm moves in two-dimensional space and can have any number of segments. We say n-arm to refer to the robotic domain with an arm possessing n segments. The $n-1$ angles between the adjacent segments of the arm define the state. An action changes one of these angles by $360°/512 \approx 0.7°$ in any direction. The other $n-2$ angles remain intact. Actions that result in intersecting the arm's segments or in colliding of any segment into an obstacle are forbidden. The 3-arm and the 4-arm domains have about 134 million and 69 billion states, respectively. The base heuristic is the sum of the differences in angles between the arm segments (taken by the absolute value) between the start and goal state.

In addition to the results presented here, we confirmed the trends reported in this section by experimenting with two synthetically generated environments: rooms maps and Delaunay graphs, but these results are not included.

7.1 Parameter tuning

Assuming that the amount of available memory is given as input, two parameters must be set: the number of pivots $|P|$ and the r-parameter of the Bounding procedure. Figure 4 shows how the choice of $|P|$ affects the time of searches

[4]http://www.dis.uniroma1.it/∼challenge9/download.shtml.

Memory	$\lvert V\rvert/16$	$\lvert V\rvert/4$	$\lvert V\rvert$	$4\lvert V\rvert$	$16\lvert V\rvert$
r	16	8	8	8	4
Overhead	3.7%	4.2%	4.1%	9.7%	1.5%
Time vs. $r = 1$	12.4%	13.6%	19.5%	16.2%	16.6%

Table 1: Summary of the Bounding procedure.

Graph	$\lvert V\rvert$	Nodes	Time
`ost001d`	10,557	1,743	1.69
`brc501d`	57,719	8,840	10.18
`ost100d`	137,375	20,389	26.73
NY	264,346	59,821	68.17
SF Bay	321,270	82,848	97.65
Colorado	435,666	110,293	136.89

Table 2: The baseline.

Memory	Heuristic	$\lvert P\rvert$	Avg. Nodes	Avg. Time
3-arm				
0	Base		11,673,442	47,079
$\lvert V\rvert/64$	CDH	1	3,938,965	17,563
$\lvert V\rvert/16$	CDH	1	2,903,863	13,316
$\lvert V\rvert/4$	CDH	2	83,498	737
$1\lvert V\rvert$	CDH	2	5,951	24
$1.5\lvert V\rvert$	CDH	3	5,518	23
$1\lvert V\rvert$	Regular DH	1	481,289	1,770
$2\lvert V\rvert$	Regular DH	2	828	10
4-arm				
0	Base	0	2,772,943	34,040
$\lvert V\rvert/64$	CDH	1	2,482,112	23,742
$\lvert V\rvert/32$	CDH	1	2,115,652	21,801
$\lvert V\rvert/16$	CDH	2	1,942,464	16,918

Table 3: The robotic arm.

for the `ost100d` DAO map. Each curve corresponds to a different amount of memory available for the CDH. For each data point, we show results with the best choice of r associated with this $|P|$. We see that the trade-off described in Section 3.2 expresses itself in a clear saddle point in the curves. We exploited this by designing a parameter tuning procedure (called also the tuning phase), which ran searches for one hundred problem instances with $|P| = m$, $|P| = 2m$, $|P| = 4m$, etc. until the saddle point was detected.

A similar saddle point, though a bit less expressed, exists for the r parameter as well. Note that trying the different values of r for the tuning phase is inexpensive (relative to trying the different values of $|P|$), since the look-up table of CDH does not need to be rebuilt when r changes.

Table 1 shows the overhead incurred by the Bounding procedure versus the performance gains of choosing $r > 1$ for several amounts of memory. For example, the left-most data column shows that, for $|V|/16$ memory, the Bounding procedure with $r = 16$: 1) expands 3.7% nodes compared to the number of nodes expanded by the main search and 2) results in 12.4% overall search speed-up compared to the search with $r = 1$. We see that the application of the Bounding procedure with $r > 1$ is well justified.

The tuning phase was performed for all input graphs. For each memory size, the values of $|P|$ and r resulting in the fastest searches were used in our experiments.

7.2 Pathfinding experiments

For each input graph, the parameters obtained by the tuning phase were used to solve one thousand problem instances (independent of those used in the tuning phase). Table 2 establishes the baseline for our results by showing how the base heuristic performs on each of the input graphs. All times that we report are measured in milliseconds.

The results of the experiments on the maps are shown in Figure 5. Charts (a) and (b) show the time performance of both the regular and the compressed DH for the `ost100d` DAO map and the SF Bay Area road map, respectively. For the memory amounts between $1|V|$ and $8|V|$, CDH significantly outperforms the regular DH. When larger amounts of memory are available, regular DHs achieve good coverage of the state space, leaving little space for improvement.

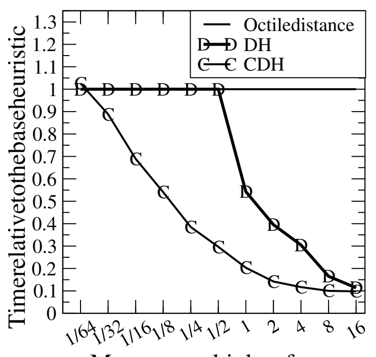
In industries such as video games and GPS navigation, although the speed of searches is very important, little memory can be spared to store an accurate memory-based heuristic, such as the regular DH, which requires at least the memory equal to the size of the domain. CDHs are applicable for any size of memory, even less than $1|V|$, and can outperform the base heuristic by using as little as $|V|/32$ memory. The improvement factors achieved by the CDH over the base heuristic for $|V|/2$ memory are shown in table (c) of Figure 5. Furthermore, chart (a) shows that, for the DAO map, CDH that uses $|V|/2$ memory outperforms the regular DH that uses as much as $4|V|$ memory.

7.3 Robotic arm experiments

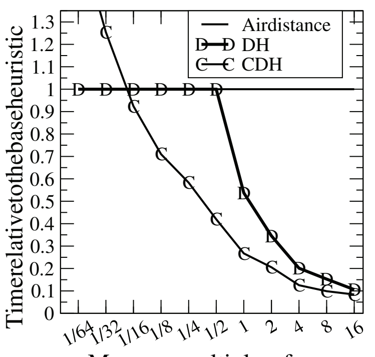
For this experiment, the obstacles consisted of three walls as shown in Figure 3. The results are averaged over 100 problem instances with the start/goal states located on the outside/inside of the walls. For the 3-arm domain, the tuning phase was limited to finding the best number of pivots. For the 4-arm domain, no tuning was performed. The r-parameter was irrelevant as a slightly different bounding procedure was used, as follows. A breadth-first search from the goal state was performed until a distance to a pivot was found. All distances that could be discovered at the same depth were used to form the bounds. Distances to the remaining pivots were not used in the heuristic calculation. Also, the pivots were placed as far as possible from each other instead of using the Avoid placement.

Table 3 shows the results. The upper half of the table corresponds to the 3-arm domain. The base heuristic (the first data row) expanded 11.7 million nodes on average. The geometry computations to determine which actions are permitted from a given state are expensive, hence the relatively long running times. With as little as $|V|/64$ memory, CDH achieves a 2.6 times improvement over the base heuristic. With $1|V|$ memory, CDH shows a more than 70-fold improvement in time performance over the regular DH.

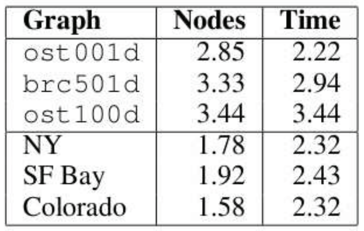
The lower half of the table shows the 4-arm results. This domain is too large to afford the regular DH, even with one pivot. We allowed $|V|/64 - |V|/16$ memory for the CDH's look-up table. Also, five minutes were allotted for solving each problem instance. The results in the table are for the 79 (out of 100) problem instances that were solved in the allotted time by both the CDH and the base heuristic. With $|V|/16$ the CDH was two times faster than the base heuristic



(a) `ost100d`



(b) SF Bay Area map



Graph	Nodes	Time
`ost001d`	2.85	2.22
`brc501d`	3.33	2.94
`ost100d`	3.44	3.44
NY	1.78	2.32
SF Bay	1.92	2.43
Colorado	1.58	2.32

(c) Best CDH improvement factor

Figure 5: Pathfinding experiments.

Bucket	`ost100d`		SF Bay	
#	Hops	Time	Hops	Time
1	58.69	0.66	134.46	1.14
4	233.24	1.36	412.05	1.85
7	408.31	2.17	551.68	2.32
10	745.01	4.34	806.06	2.50

Table 4: Improvement of CDH for different path lengths.

for this problem set. Furthermore, CDH solved 86 problem instances in total, while the base heuristic solved only 80.

7.4 CDH for problems of varying difficulty

To explore how the length of the optimal solution affects CDH's performance, we fix the available memory and divide the problem set into buckets according to the cost of the optimal solution. We chose $|V|/2$ memory and used ten buckets, each consisting of 100 problem instances. In Table 4, the average number of hops in the optimal solution and the improvement factor in time of CDH over the base heuristic are shown for the `ost100d` DAO map and the SF Bay Area road map. We observe that the improvement factor of CDHs over the octile/air distance heuristic increases as the length of the optimal solution increases. Some deviations from this trend were observed in our experiments. We attribute these deviations to cache effects.

8 Conclusions and future work

We introduce a compression method for DHs which stores only a portion of the original uncompressed DH, while preserving enough information to enable efficient search. Experimental results for three real-world domains show that the resulting heuristic achieves several-fold improvement in time performance over the regular DH when a moderate (i.e. not enough to make the regular DH almost perfect) amount of memory is given. When the amount of memory is less than the memory needed to store the input graph (as is the case for many practical applications such as video games), DH is not applicable, but CDH achieves a several-fold improvement over the base heuristic of the domain.

We would like to continue this work by exploring more sophisticated ways of distributing distances to pivots among the states, for both implicit and explicit storage of pivot identities. A starting point in this direction is to prioritize the pivots, so that distances to some pivots can be stored with greater frequency than to the other pivots.

9 Acknowledgements

This research was supported by the Israeli Science Foundation (ISF) under grant no. 305/09 to Ariel Felner. Financial support of iCORE and NSERC to Jonathan Schaeffer is greatly appreciated.

References

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Felner, A.; Korf, R. E.; Meshulam, R.; and Holte, R. C. 2007. Compressed pattern databases. *JAIR* 30:213–247.

Felner, A.; Berrer, M.; Sturtevant, N.; and Schaeffer, J. 2009. Abstraction-based heuristics with true distance computations. In *Proceedings of SARA*.

Goldberg, A., and Harrelson, C. 2005. Computing the shortest path: A* search meets graph theory. In *SODA*, 156–165.

Goldberg, A., and Werneck, R. 2005. Computing point-to-point shortest paths from external memory. In *ALENEX*, 26–40.

Goldberg, A.; Kaplan, H.; and Werneck, R. 2009. Reach for A*: Shortest path algorithms with preprocessing. In *Ninth DIMACS Implementation Challenge*, 93–140.

Goldenberg, M.; Felner, A.; Sturtevant, N.; and Schaeffer, J. 2010. Portal-based true-distance heuristics for path finding. In *Proceedings of SoCS*.

Ng, T., and Zhang, H. 2002. Predicting internet network distance with coordinates-based approaches. In *INFOCOM*, 170–179.

Sturtevant, N.; Felner, A.; Barer, M.; Schaeffer, J.; and Burch, N. 2009. Memory-based heuristics for explicit state spaces. *IJCAI* 609–614.

Zahavi, U.; Felner, A.; Schaeffer, J.; and Sturtevant, N. R. 2007. Inconsistent heuristics. In *AAAI*, 1211–1216.

Zhang, Z.; Sturtevant, N.; Holte, R.; Schaeffer, J.; and Felner, A. 2009. A* search with inconsistent heuristics. In *Proceedings of IJCAI*, 634–639.