# Memory-Based Heuristics for Explicit State Spaces

**Nathan R. Sturtevant**
Computing Science
University of Alberta
Edmonton, AB, Canada
nathanst@cs.ualberta.ca

**Ariel Felner** and **Max Barrer**
Information Systems Engineering
Deutsche Telekom Labs
Ben-Gurion University
Be'er-Sheva, Israel
felner,maxb@bgu.ac.il

**Jonathan Schaeffer** and **Neil Burch**
Computing Science
University of Alberta
Edmonton, AB, Canada
jonathan,burch@cs.ualberta.ca

## Abstract

In many scenarios, quickly solving a relatively small search problem with an arbitrary start and arbitrary goal state is important (e.g., GPS navigation). In order to speed this process, we introduce a new class of memory-based heuristics, called *true distance heuristics*, that store true distances between some pairs of states in the original state space can be used for a heuristic between any pair of states. We provide a number of techniques for using and improving true distance heuristics such that most of the benefits of the all-pairs shortest-path computation can be gained with less than 1% of the memory. Experimental results on a number of domains show a 6-14 fold improvement in search speed compared to traditional heuristics.

## 1 Introduction and Overview

A common direction in heuristic search is to develop techniques which allow larger problems to be solved. However, there are many domains, such as map-based searches (common in GPS navigation, computer games, and robotics), where quickly solving a 'small' problem is most important. Optimal paths for such domains can be found relatively quickly with simple heuristics, especially compared to the exponential size of combinatorial domains. But relative quickness might still not be enough in such real-time domains as the search should be extremely fast. A significant body of work has been performed to speed up such searches by compromising on the solution quality (e.g., [Bulitko *et al.*, 2008]). We address how available memory can be used to improve search performance while still returning optimal solutions.

Over the last decade pattern databases (PDBs) have been explored as a powerful method for automatically building admissible memory-based heuristics [Culberson and Schaeffer, 1998]. A range of enhancements have been proposed that have been useful for many domains (e.g., [Felner *et al.*, 2007]), although some of these take advantage of unique properties which are not generally available. Therefore, PDBs have some limitations.

First, PDBs are goal-specific; they only provide heuristics for a single goal state. While some domains have properties (e.g., duality [Felner *et al.*, 2005]) that allow a given PDB to be used for many goal states, this is not a general property.

Second, PDBs store abstract distances between states. This guarantees that the distances are lower bounds on distances in the original domain, but if good abstractions are not available, then the estimates will be poor. PDBs work very well for domains where a state can be described by assigning values to a set of variables (e.g., locations to tiles, as in the sliding-tile puzzle). Replacing some of the assignments with a *don't care* value can yield an effective abstraction. But, in map-based pathfinding problems a state is just an $x/y$ coordinate. Replacing the $x$ or $y$ coordinate by a *don't care* yields an abstraction that is too general to be effective.

We introduce a new class of memory-based heuristics called *true distance heuristics* (TDHs). They are useful in any undirected graph, even for applications where traditional PDBs are not applicable. True distance heuristics aim to provide heuristic information between *any* pair of start and goal states and as a consequence they may have significant memory needs. Thus, they are most applicable in what would ordinarily be considered 'small' domains such as maps, where the entire search space fits into memory.

Unlike traditional PDBs, which store distances in an abstract state space, TDHs store information about true distances in the original state space. A perfect heuristic for any pair of start and goal states on any graph could be achieved by computing and storing *all-pairs-shortest-path* distances (e.g., using the Floyd-Warshall algorithm). However, due to time and memory limitations this is not practical. TDHs compute and store only a small part of this information. We present two ideas for reducing the time and memory overhead of the full all-pairs-shortest-path data. First, a *differential heuristic* stores distances from all states to one or more *canonical states*. A second approach is to increase the number of canonical states but only store distances between canonical states. We unify these two ideas into a general framework and provide experimental results to show the benefits of true distance heuristics on a number of different domains.

It is important to note that while we discuss these ideas under the assumption that the full state space fits in memory, there are a number of abstraction-based algorithms which find solutions in smaller abstract spaces as an intermediate step towards finding optimal [Holte *et al.*, 1996] or suboptimal [Sturtevant and Buro, 2005] solutions in the original space. If the state space does not fit in memory, TDHs can be used to more quickly find solutions in abstracted state spaces. These

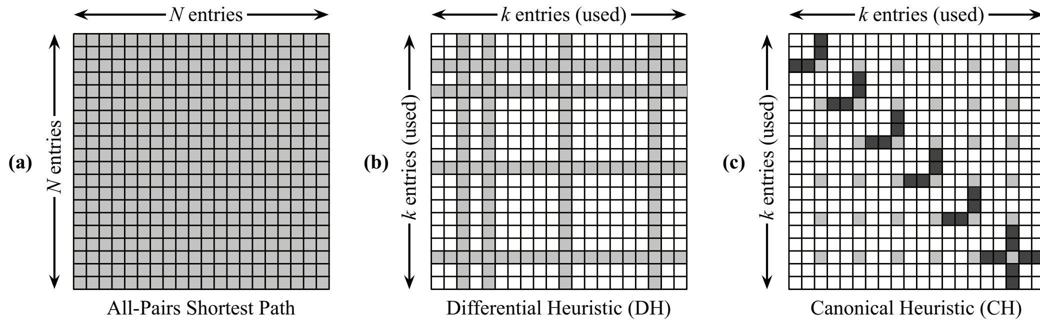



Figure 1: Types of heuristics.

solutions are then used in the original, larger state spaces.

Björnsson and Halldórsson [2006], suggest a similar idea, using the exact distances between some of the states in the domain. However, their work is specific to maps of rooms and passageways as seen in computer games (canonical states were doorways into rooms). Our work is more general and does not exploit application-specific properties.

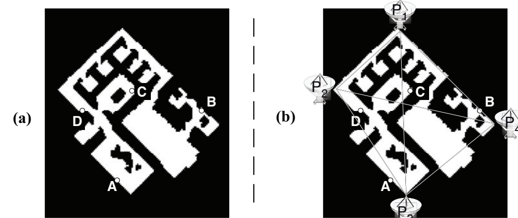



Figure 2: DH and CH examples.

## 2 True Distance Heuristics (TDHs)

Let $N$ denote the number of vertices in a graph. If the full all-pairs shortest-path database is available, then the exact distance between two states, $d(x, y)$, can be used as a perfect heuristic between $x$ and $y$. This situation is illustrated in Figure 1a. Each row and column corresponds to a state in the world, and an entry in the grid is marked if the corresponding distance is stored. Computing such a database will require as much as $O(N^3)$ time which might not be feasible (even in an offline phase). If there are N states, storing this database will require $O(N^2)$ memory – more than is available in many domains. We propose two methods that reduce the memory needs by using a subset of the all-pairs-shortest-path information to compute a heuristic distance between any two states. The two ideas are presented separately, after which a unified general formalization is given. The heuristics are admissible, but a proof is omitted due to space limitations.

### 2.1 Differential Heuristics (DHs)

The first idea is shown Figure 1b. In this case, shortest path data is only stored for $k$ of the $N$ states ($k \ll N$). We denote these $k$ states as *canonical states*. Because the database is symmetric around the main diagonal (the graph is undirected), this is equivalent to retaining only $k$ rows (or columns) out of the full all-pairs database. If $s$ is one of the canonical states then $d(x, s)$ is available for any state $x$. The heuristic between arbitrary states $a$ and $b$ would be the max of $h(a, b) = |d(a, s) - d(b, s)|$ over all $s$.

We call this database a *Differential Heuristic* (DH). Figure 2a illustrates how a DH works (with $k = 1$). The traversable portion of the map is white. $C$ is the canonical state and, hence, the shortest path from state $C$ to all other points is available. If $d(C, D) = 10$, $d(C, B) = 50$ and $d(C, A) = 20$, then this database will give an accurate estimate for $h(A, D) = |10 - 20| = 10$ and $h(D, B) = |10 - 50| = 40$ because $D$ is close to the optimal path between $C$ and $A$ as well as $C$ and $B$. But the estimate between $B$ and $A$ is not very accurate: $h(A, B) = |20 - 50| = 30$ because $B$ and $A$ are opposite directions from $C$.

These examples show that a DH provides a more accurate estimate between two points when the optimal path between one of the points and the canonical state goes through the other point. If we use $k > 1$ canonical states, we can take the maximum from each of the independent heuristics. A differential heuristic can be built using $k$ complete single-source searches. The time will be $O(kN)$ and the memory used is also $O(kN)$. The canonical states can be chosen by various methods discussed in Section 3. An idea similar to DH was independently suggested by [Cazenave, 2006].

### 2.2 Canonical Heuristics (CHs)

Our second idea uses canonical states in a different way, illustrated in Figure 1c. Again, we first select $k$ canonical states. Here, the shortest path between *all* pairs of these $k$ states is stored in the database (primary data). Additionally, for each of the $N$ states in the world we store which canonical state is closest as well as the distance to this canonical state (secondary data). The shortest-path data (primary data) is marked in a light-gray in Figure 1 while the secondary data is slightly darker. Note that in a domain with regular structure, it might be possible to avoid storing the secondary data, instead computing it on demand. We call this a *canonical heuristic* (CH). Define $C(x)$ as the closest canonical state to $x$. Then:

$$h(a, b) = d(C(a), C(b)) - d(a, C(a)) - d(b, C(b))$$

This can be less than 0 and can be rounded up but in practice we always take the max of the canonical heuristic and an existing heuristic. In addition, if $C(a) = C(b)$, then the differential heuristic rule can be used instead.

Figure 2b illustrates canonical heuristics. There are four canonical states, $P_1 \ldots P_4$. The distance between $A$ and $B$ will be estimated accurately because these points are both

| Technique | Storage | Time to Build |
|---|---|---|
| All-Pairs Shortest Path | $N^2$ | $O(N^3)$ |
| DH | $kN$ | $O(kN)$ |
| CH(k, 1) | $k^2 + 2N$ | $O(kN)$ |
| CH(k, d) | $k^2 + 2dN$ | $O(kN)$ |

Figure 3: Memory and time complexity.

| Closest States Stored | Num Canonical States | Total Memory ($2dN + k^2$) |
|---|---|---|
| d < k | | |
| d = 1 | k = $\sqrt{8N}$ | 10N |
| d = 2 | k = $\sqrt{6N}$ | 10N |
| d = 3 | k = $\sqrt{4N}$ | 10N |
| d = 4 | k = $\sqrt{2N}$ | 10N |
| d = k | | |
| d = 5 | k = 5 | 10N |
| d = 10 (optimized) | k = 10 | 10N |

Figure 4: Transition between DH and CH.

close to canonical points. But, the distance between $B$ and $C$ will be estimated less accurately because $C$ is far from $P_2$, the closest canonical state. Once $k$ canonical states are chosen, we need to perform $k$ complete single-source searches. The time complexity is again $O(kN)$. The memory needed is $k^2$ for the primary data but additional $2N$ memory might be needed for the secondary data (when applicable).

### 2.3 Unified View

Figure 3 summarizes the time and memory requirements for DH and CH. Building either database requires $k$ single-source searches of the entire state space. However, DHs generally use a much smaller $k$, so they can be built more quickly; they can also be built incrementally.

Differential and canonical heuristics can be viewed as opposite extremes of a general framework. Suppose the available memory is fixed at $10N$. Memory can be filled in one of two ways. First, 10 differential heuristics can be built, each of which takes $N$ memory. Alternately, $k = \sqrt{8N}$ canonical states can be selected for a canonical heuristic which will use $k^2 = 8N$ memory. With the additional $2N$ memory for storing the secondary data, this will also require $10N$ memory.

Consider that instead of keeping the distance to the closest canonical state (and its identity) in the secondary data, we keep the distance to the $d$ closest canonical states among the $k$ canonical states available ($d < k$). We denote this as $CH(d, k)$. The memory required is $2dN + k^2$. Our introductory discussion to CH implicitly used $d = 1$. When $d = k$ then every state maintains the exact distance to all $k$ canonical states – which is actually a differential heuristic.

The possible heuristics using $10N$ memory are shown in Figure 4. When $d < k$, both the optimal distance to the closest canonical states and the identity of these canonical states must be stored in the secondary data. When $d = k$ (logically a *differential heuristic*) the distance to all canonical states is stored. Thus, the identity of the canonical state is not needed, allowing twice as many canonical states to be used. Additionally, when $d = k$ the primary data of all-pairs-shortest-path distance between the $k$ canonical states is redundant here, as it is already stored in the secondary data. This allows us to reuse the space by doubling $d$ ('optimized' in Figure 4).

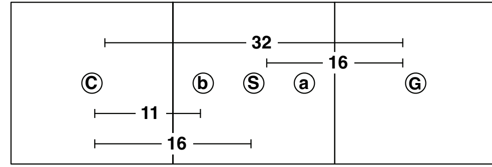



Figure 5: Two heuristic lookups are better than one.

In this unified scheme, there are many possible heuristic lookups. For any two states $a$ and $b$ we need to choose two out of $d$ different canonical states as reference points for the canonical heuristic; a total of $d^2$ possible lookups. As well, there could be as many as $d$ valid differential lookups. Clearly there is a tradeoff; the maximum over multiple heuristic values yields a better heuristic but at the cost of increased execution time. In addition, larger $d$ means fewer canonical states.

The advantage of using $d > 1$ is shown in Figure 5. The search is between the start ($S$) and the goal ($G$), both of which are canonical states. While the canonical heuristic will store the exact distance between these states (16), it will give no guidance to an A* search as to which nodes are on the optimal path to $G$. Consider states $a$ and $b$. They are equally far from $S$ (say, 5) so their heuristic value from $G$ will be the same ($16 - 5 = 11$), they will have the same $f$-cost (16), and will both be expanded. But, we can use canonical state $C$ to improve the heuristic estimate for $b$. In particular, $h(b, G) = d(C, G) - d(b, C) - d(G, G) = 32 - 11 = 21$. With a $g$-cost of 5, $b$ will have an $f$-cost of 26 and will not be expanded ($26 > 16$). The second lookup can be seen as triangulating the position of state $b$ to improve its heuristic value.

## 3 Canonical State Placement

Several optimizations can be made in placing canonical states. In general, we can do so randomly, with an advanced placement algorithm, or by domain specific methods.

### 3.1 Canonical Heuristics – Advanced Placement

For each canonical state $s$, define the *canonical neighborhood* of that state as all nodes $x$ for which $C(x) = s$. Consider the example in Figure 6. There are two canonical states in this example, $A$ and $B$. $A$'s canonical neighborhood is a circle. This means that, when subtracting the distance from some state $a$ to the canonical state $A$, the maximum value subtracted is the radius of the canonical neighborhood. $B$'s canonical neighborhood is asymmetric, meaning that it is possible for a state in the neighborhood to be much further from the canonical state, resulting in lower heuristic estimates.

Thus, it would be wise to place the canonical states as uniformly as possible throughout a (multi-dimensional) state space, minimizing the radius (and possible penalty) of each canonical neighborhood. We approximate this approach with the following. First, choose an unmarked node at random, and then do a breadth-first search of $N/k$ states, marking each of them. Second, repeat the first step until all nodes have been marked. Canonical states are then placed at the middle of

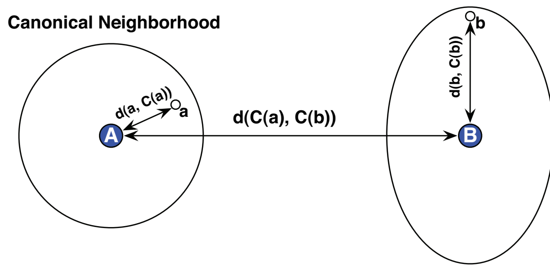
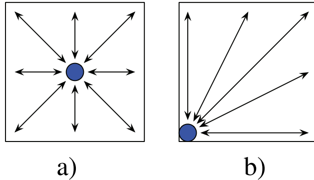



Figure 6: Placement of canonical heuristics.



a) b)

Figure 7: Placement of differential heuristics.

each marked region (seeds of the BFS). As the process involves some randomness, we cannot exactly control the number of canonical states, but sample until we are within 10% of our intended size of $k$ states. We show the effectiveness of this in the experimental results.

### 3.2 Differential Heuristics - Advanced Placement

A differential heuristic will give a perfect heuristic value between two points if one of them is on an optimal path between the other point and the canonical state. Additionally, the maximum value returned by a differential heuristic is bounded by the maximum distance from a canonical state to any other state in the world. Thus, we want to maximize the length of these paths. This suggests putting the canonical states as far as possible from each other and from other states in the world.

In Figure 7a the canonical state $S$ is in the middle. It will only provide good heuristic values between two states if they both fall on the same optimal path line originating from $S$. In this case, the lines are short, so heuristic values returned must be small. But, when $S$ is placed in the corner, as in Figure 7b, much larger values can be returned.

In practice, we first perform a breadth-first search from a random state. The furthest state from the random point is our first canonical state. The second canonical state is chosen by finding the state that is farthest away from the first canonical state. Subsequent states can be chosen by finding the node for which the minimum distance to all existing canonical states is maximized.

### 3.3 Domain-Specific Methods for Placement

Consider GPS navigation. The North American map easily fits into memory on many portable devices. Instead of just using one method, regular differential heuristics might be used for intra-city search, while population centers would be effective canonical states for inter-city search. Thus, we expect that many domains have specific properties for which these approaches could be adapted (e.g., our 8-puzzle experiments below).

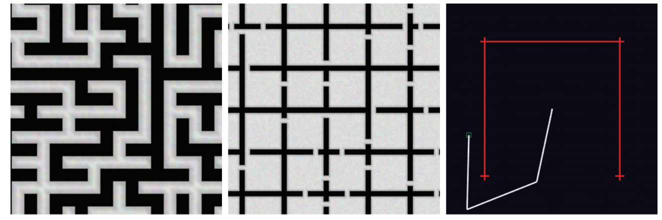


Figure 8: Example mazes/rooms (left/center) and robotic arm.

## 4 Experimental Results

We provide detailed experimental results for the pathfinding domain and an actuated robotic arm, as well as a short summary of results on the 8-puzzle. In all experiments we use the max of the existing heuristic with the canonical heuristics. Search is performed with A* in all domains except the 8-puzzle. All times are reported in seconds.

### 4.1 Pathfinding: Differential Heuristics

Pathfinding is an example of a domain where the entire state space is usually kept in memory. The real-time nature of this domain requires finding a path as quickly as possible.

Two types of maps were used which are illustrated in Figure 8: mazes (left) and rooms (center). In a simple maze there is only one path between any two points, however we use corridors with width two, which increases the average branching factor from two to five. The octile-distance heuristic, which is similar to Manhattan distance except that it allows for diagonal moves, can be very inaccurate on mazes. Room maps are composed of small (16×16) rooms with randomly opened doors between rooms. Octile-distance is more accurate on these maps. All maps used here are publicly available.

To begin, we compare the search effort required with the full all-pairs-shortest-path data to the differential heuristics. The results in Figure 9 use a 512×512 room map with 206,720 states. The number of canonical states varied from 0 to 125 by intervals of 5. Because we cannot plot '0' canonical states on a log-plot, the first point denotes the result with the default octile heuristic. The results are averaged over 640 problems with solution lengths between 256 and 512.

The all-pairs data was estimated by assuming that with a perfect heuristic only the optimal path would be explored. We drew a line from the 125 data point to the 206,720 data point (all pairs) to approximate data in between.

The top of the graph shows how the average $h$-value grows as the number of canonical states is increased. Note that the $x$-axis is logarithmic. The optimal heuristic value is 384.59 and would require the full all-pairs data and 21 billion heuristic entries. With 125 canonical states (26 million entries) we get an average heuristic value of 382.04. With just 10 canonical states (2 million entries) the heuristic value is 370.34. In contrast, the average octile-distance heuristic is 306.83.

The number of node expansions are shown in the bottom of Figure 9. This is a log-log graph, so the line looks much shallower than it actually is. With octile-distance, 21,686 nodes are expanded on average. 10 canonical states reduces this to 3,440 nodes. 125 canonical states reduces this further to 760 nodes (29× reduction). The absolute minimum, assuming a

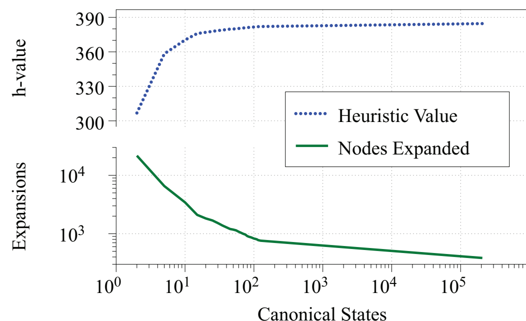



Figure 9: DH performance as memory usage grows.

| h | | Random Placement | | | Advanced Placement | | |
|---|---|---|---|---|---|---|---|
| d | k | nodes | $h$-val | time | nodes | $h$-val | time |
| Octile | | 7792 | 151 | 0.068 | | | |
| 1 | 1448 | 2621 | 596 | 0.026 | 2377 | 611 | 0.026 |
| 2 | 1254 | 2221 | 610 | 0.023 | 1845 | 626 | 0.022 |
| 3 | 1042 | 1992 | 613 | 0.022 | 1729 | 627 | 0.021 |
| 4 | 724 | 1955 | 605 | 0.023 | 1776 | 619 | 0.023 |
| 5 | 5 | 1749 | 610 | 0.022 | 1793 | 610 | 0.026 |
| 10 | 10 | 811 | 629 | 0.011 | 707 | 636 | 0.010 |

Figure 10: Results on maze maps. Memory = 10N.

perfect heuristic would be 384.6 nodes. A 29× reduction in nodes expanded can be achieved with only $1/1000$ of the total memory needed for the full all-pairs information (which will only achieve an additional reduction of a factor of two.)

Time, not plotted here, tracks closely with the nodes expanded, however it peaks with a 13× reduction when there are 80 canonical states. With 125 canonical states there is still a 12× speedup. In a DH the number of differential lookups grows linearly with the number of canonical states. It is likely that most of the gain for a particular problem comes from a few canonical states. If these were identified, the overhead could be reduced (See Holte [2006] for a deeper discussion of reducing the time overhead of heuristic lookups).

## 4.2 Pathfinding: Canonical Heuristics

Next, we look at the transition between canonical heuristic parameters. We begin with the default heuristic, octile distance. Then, fixing the total memory at $10N$ we build a CH$(d, k)$ (where $d = \{1 \ldots 5\}$ and $k = \sqrt{(10N - 2Nd)}$) and a DH$(k = 10)$. The canonical and differential heuristics were built twice: once with random and once with advanced placement of canonical states.

We present the average number of nodes expanded by A*, starting $h$-cost, and average time for the search. The results are averaged over 640 problem instances on each of 5 maze and room maps (3,200 total instances for each map type). All maps are $512 \times 512$. Paths were evenly distributed between lengths 256 and 512 on the room maps and between lengths 512 and 768 on the maze maps. Paths are longer on maze maps, but fewer nodes are expanded because the search is more restricted in the maze corridors.

The results for mazes are in Figure 10. The average heuristic between start and goal points is 151 with octile distance, while the optimized DH (last line) has an average heuristic value of 636. The DH expands over 11× fewer nodes than the octile heuristic but is only 6.8× faster due to the overhead of the heuristic lookups.

Results for room maps are in Figure 11. The octile heuristic is more accurate in these maps with an average value between start and goal pairs of 309, compared to 370 with the best canonical heuristic. There is a saddle point in the canonical results, where the best results are with $d = 3$ (for mazes too). The best time performance is with the optimized DH, 5.5× faster than the octile heuristic with 6× fewer nodes.

The average DH value between the start to the goal is 370, lower than the CH$(d = 3)$ (376), but fewer nodes are expanded with the DH. To investigate this, we recorded the $h$-value of every node expanded over 640 problems on a single room map. A histogram of values is in Figure 12. Searching with either heuristic expands the same number of nodes with high heuristic values. But, the DH results in far fewer nodes expansions with low heuristic values. CHs are inaccurate near the borders of the canonical neighborhoods which suggests there are enough nodes along these borders to significantly increase the cost of search.

## 4.3 Robotic Arm

The robotic arm domain has often been used as a test-bed for suboptimal search [Likhachev and Stentz, 2008]. A robotic arm can have many degrees of freedom which makes the search space large and optimal solutions infeasible except for small instances of the problem. We experimented on the 3-arm robotic arm in a 2D plane, shown in Figure 8. The base is fixed at the origin and the task is to move its end effector from its initial configuration to an arbitrary $x/y$ location. Arm rotation has been discretized into $0.7°$ increments, resulting in $(360/0.7)^3 = 134$M possible states. Of these, only 24.5M are legal in our configuration space, as the arms are not allowed to cross themselves or other obstacles. An action is moving one joint $0.7°$ and has cost 1.

This search problem is different from previous examples because the goal state is implicit, only specified by a position for the end effector. Thus, there are many possible goal configurations. The default heuristic is the minimum path the arm must travel from the start to the goal divided by the maximum distance moved by the arm in a single step. The cost of moving around obstacles is included. To improve performance, we bucket the possible end effector positions (on a 200×200 grid), storing all possible arm configurations which fall in each cell. This table has 24.5M entries, although a single DH uses 134M entries, which could be optimized. When given a goal location, we search from all arm configurations in that cell towards a single start state. This allows us to use a

| h | | Random Placement | | | Advanced Placement | | |
|---|---|---|---|---|---|---|---|
| d | k | nodes | $h$-val | time | nodes | $h$-val | time |
| Octile | | 21354 | 309 | 0.296 | | | |
| 1 | 1448 | 12269 | 367 | 0.163 | 8698 | 372 | 0.123 |
| 2 | 1254 | 9276 | 370 | 0.128 | 6011 | 375 | 0.091 |
| 3 | 1042 | 7962 | 370 | 0.114 | 5472 | 376 | 0.083 |
| 4 | 724 | 8185 | 368 | 0.120 | 5646 | 373 | 0.092 |
| 5 | 5 | 13742 | 342 | 0.217 | 14473 | 337 | 0.246 |
| 10 | 10 | 6266 | 357 | 0.093 | 3479 | 370 | 0.054 |

Figure 11: Results on room maps. Memory = 10N.

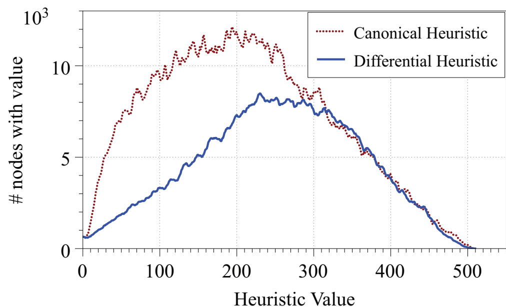



Figure 12: Nodes expanded with CH and DH.

more accurate default heuristic – the number of turns needed to get the arm to the goal configuration (arm-angle heuristic).

| | Random Placement | | | Advanced Placement | | |
|---|---|---|---|---|---|---|
| 0 | nodes | $h$-val | time | nodes | $h$-val | time |
| Dist | 4.04M | 89.99 | 278.3 | | | |
| Arm | 1.98M | 209.89 | 31.5 | | | |
| 1 | 637k | 221.68 | 10.1 | 610k | 223.20 | 10.3 |
| 2 | 334k | 224.52 | 5.2 | 214k | 226.69 | 3.6 |
| 3 | 245k | 225.14 | 3.8 | 162k | 227.65 | 2.9 |
| 4 | 245k | 225.14 | 4.0 | 140k | 228.00 | 2.5 |

Figure 13: Differential heuristic results for robotic arm.

We choose 500 problems at random and report average results over the 101 hardest in Figure 13. These are the problems that take over 1s to solve using the arm-angle heuristic. The first two lines report results with the straight-line (Dist) and arm-angle (Arm) heuristics. The arm-angle heuristic expands 2× fewer nodes, and is much faster, due to a faster heuristic function. (The dist heuristic took 40 hours to solve the *entire* problem set, while the arm heuristic only took 53 minutes.) Subsequent lines show DH (advanced placement) results with 1 to 4 canonical states, which fits in 2GB of RAM. Our implementation is coarse, and could be improved. But, the DHs still provided a 14× improvement in nodes expanded over the arm-angle heuristic, and a 13× improvement in speed. The advanced placement algorithm shows some improvements over the random DH placement.

### 4.4 Experiments on the 8-Puzzle

TDHs will likely work best in domains where paths cover long distances. This means that most puzzles, where the optimal path length tends to grow with the log of the state space size, will not be good candidates for TDHs. We demonstrate this on the 8 puzzle, which has $N = 181{,}440$ states. Figure 14 shows the average results over 31,125 problems where the average optimal path was 21.84 moves. The gains provided by DHs and CHs with 10N are modest. But, we also built a CH where the canonical states are all states with the blanks in the middle (20,160 states). Due to the regular structure of the domain secondary data was not stored and computed on the fly. While the memory requirements are huge (≈200MB), using a CH($d = 3$) results in only 69 nodes expanded on average, 40× better than MD.

| h | d | k | $h$ start | nodes | Mem |
|---|---|---|---|---|---|
| Manhattan | - | - | 14.21 | 3000.26 | - |
| DH | 10 | 10 | 14.41 | 2570.33 | 10N |
| CH | 1 | 1204 | 14.50 | 2376.23 | 10N |
| CH | 3 | 852 | 14.44 | 2422.28 | 10N |
| DH | 200 | 200 | 16.11 | 832.71 | 200N |
| CH | 1 | 20,160 | 18.55 | 345.95 | 1120N |
| CH | 3 | 20,160 | 19.83 | 69.42 | 1120N |

Figure 14: Results on the 8 puzzle.

## 5 Conclusions

In the past decade, pattern databases have received considerable attention in the heuristic search literature. This paper introduces two new table-based heuristics that use memory in a novel way to solve more general problems with arbitrary start and goal states. The result is important performance gains for real-time pathfinding, and a practical demonstration of performance benefits in other domains. Considerable research remains. The best number and location of canonical states is an open problem. Our solution shows the potential of 'smart' placement, but can probably be improved. More insights are also needed into the nature of these heuristics to give guidance to an application developer as to which heuristic (and its parameters) to choose for a given problem.

### Acknowledgments

This research was supported by the ISF under grant number 728/06 to Ariel Felner and by Alberta's iCORE Canada's NSERC.

## References

[Björnsson and Halldórsson, 2006] Y. Björnsson and K. Halldórsson. Improved heuristics for optimal path-finding on game maps. In *AIIDE*, pages 9–14, 2006.

[Bulitko *et al.*, 2008] V. Bulitko, M. Lustrek, J. Schaeffer, Y. Björnsson, and S. Sigmundarson. Dynamic control in real-time heuristic search. *JAIR*, 32:419–452, 2008.

[Cazenave, 2006] Tristan Cazenave. Optimizations of data structures, heuristics and algorithms for path-finding on maps. In *CIG*, pages 27–33, 2006.

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

[Felner *et al.*, 2005] A. Felner, U. Zahavi, R. Holte, and J. Schaeffer. Dual lookups in pattern databases. In *IJCAI*, pages 103–108, 2005.

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

[Holte *et al.*, 1996] R. C. Holte, M. B. Perez, R. M. Zimmer, and A. J. MacDonald. Hierarchical A*: Searching abstraction hierarchies efficiently. *AAAI*, pages 530–535, 1996.

[Holte *et al.*, 2006] R. C. Holte, A. Felner, J. Newton, R. Meshulam, and D. Furcy. Maximizing over multiple pattern databases speeds up heuristic search. *Artificial Intelligence*, 170:1123–1136, 2006.

[Likhachev and Stentz, 2008] M. Likhachev and A. Stentz. R* search. In *AAAI*, pages 344–350, 2008.

[Sturtevant and Buro, 2005] N. Sturtevant and M. Buro. Partial pathfinding using map abstraction and refinement. In *AAAI*, pages 1392–1397, 2005.