Portal-Based True-Distance Heuristics for Path Finding

Meir Goldenberg
Information Systems Engineering
Ben-Gurion University
Be'er-Sheva, Israel 85104
mgoldenbe@yahoo.ca

Ariel Felner
Information Systems Engineering
Deutsche Telekom Labs
Ben-Gurion University
Be'er-Sheva, Israel 85104
felner@bgu.ac.il

Nathan Sturtevant and **Jonathan Schaeffer**
Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2E8
{nathanst, jonathan}@cs.ualberta.ca

Abstract

True distance memory-based heuristics (TDHs) were recently introduced as a way to obtain admissible heuristics for explicit state spaces. In this paper, we introduce a new TDH, the *portal-based heuristic*. The domain is partitioned into regions and portals between regions are identified. True distances between all pairs of portals are stored and used to obtain admissible heuristics throughout the search. We introduce an A*-based algorithm that takes advantage of the special properties of the new heuristic. We study the advantages and limitations of the new heuristic. Our experimental results show large performance improvements over previously-reported TDHs for commonly used classes of maps.

1 Introduction

A common research direction in heuristic search is to develop techniques which allow larger problems to be solved with fixed resources. However, there are domains, such as map-based searches (common in GPS navigation, computer games, and robotics), where solving a "small" problem rapidly is most important. Although optimal paths can be found relatively quickly with simple heuristics, this might still not be enough for some real-time domains. 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)). In this paper, we address using memory to improve performance while still returning optimal solutions.

A class of memory-based heuristics, *true distance heuristics* (TDHs), was introduced in (Sturtevant *et al.* 2009; Felner *et al.* 2009). TDHs store distances between *selected pairs* of states in the *original* state space (hence the term *true distance*). This information is used to compute admissible heuristic values between any pair of states. The different TDHs are distinguished by the pair selection criteria and by the way the heuristic is calculated.

We introduce a new type of TDH, the *portal-based true distance heuristic* (PTDH), or simply the *portal heuristic* (PH). First, the state space is partitioned into disjoint regions as shown in Figure 1 (left). States on the borders between the regions are called *portals* (denoted p in the figure). True distances between all pairs of portals (solid arrows in the figure) are pre-computed and stored. These distances are then used to compute an admissible heuristic for any two states.

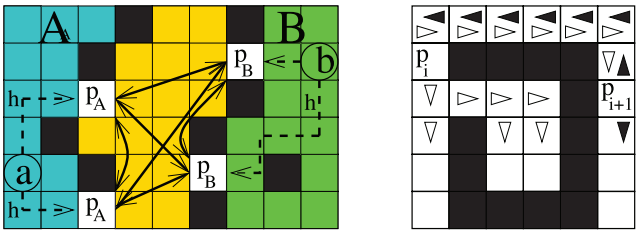

Figure 1: PH (left) : Direction enhancement (right).

Consider the left-most region (A) and right-most region (B) in Figure 1 (left). Every path between states $a \in A$ and $b \in B$ consists of three stages: (1) exiting A via one of the portals (p_A), (2) entering B via one of the portals (p_B), and (3) reaching b. Since the exact distance between all pairs of portals (denoted $d(p_A, p_B)$) is available, the heuristic value $h(a, b)$ is obtained by adding heuristic estimates of $d(a, p_A)$ and $d(b, p_B)$ to $d(p_A, p_B)$. Taking the minimum among all pairs of portals p_A and p_B guarantees admissibility.

PH has a number of interesting properties that can be utilized to speed up the search. In particular, the search graph can be collapsed to only include (1) states in the start and goal regions, and (2) edges connecting portals of these regions. We introduce *Portal-Based Search* (PBS), an algorithm that uses this property to divide the path-finding problem into smaller sub-problems. A number of enhancements that further speed up PBS are presented. In particular, with only a moderate amount memory, a perfect heuristic can be produced for some families of graphs. Experimental results on grid-based maps show that PBS outperforms search with other TDHs, in many cases by an order of magnitude. However, PH also has some limitations and its relative performance for some types of graphs over other TDHs degrades.

PH can be used in conjunction with any graph partitioning algorithm. This paper concentrates on the new heuristic itself while the influence of a particular way of partitioning the graph is not discussed.

2 Related Work

Pattern databases (PDBs) (Culberson & Schaeffer 1998) are a powerful method for automatically building admissible memory-based heuristics based on domain abstractions.

However, (Sturtevant *et al.* 2009; Felner *et al.* 2009) showed that PDBs work well only for implicit exponential domains, such as combinatorial problems, where a single goal state is usually specified and good abstractions of the domain are available. PDBs are not well-suited for explicit domains, such as map-based path finding, where a path between any arbitrary pair of states might be needed.

Unlike PDBs, which store distances in an abstract state space, TDHs store accurate distances between selected pairs of states in the original, *unabstracted* state space. A perfect heuristic could store *all-pairs-shortest-path* distances. Since this is not practical due to time and memory limitations, TDHs pre-compute and store only a small part of this information. The following TDHs were introduced in (Sturtevant *et al.* 2009; Felner *et al.* 2009). Let V be the set of states[1] and $d(x, y)$ be the cost of the shortest path between $x, y \in V$.

(1) Differential heuristic (DH). Choose the set $K \subset V$ ($|K| \ll |V|$) of *canonical states*. For each state x, the distances to all canonical states (i.e. $d(x, s)$ for all $s \in K$) are pre-computed and stored in the database. For arbitrary states a and b, $|d(a, s) - d(b, s)|$ is a lower bound on $d(a, b)$. The *differential heuristic* between arbitrary states a and b is $h(a, b) = \max_{s \in K} |d(a, s) - d(b, s)|$.

(2) Canonical heuristic (CH). Choose the set $K \subset V$ ($|K| \ll |V|$) of *canonical states*. For each canonical state, the distances to all other canonical states are pre-computed and stored in the database (*primary data*). For each non-canonical state x, the distances to the m closest canonical states (denoted $C_1(x), C_2(x), \ldots, C_m(x)$) are stored (*secondary data*). The *canonical heuristic* between arbitrary states a and b is $h(a, b) = \max_{1 \le i,j \le m} [d(C_i(a), C_j(b)) - d(a, C_i(a)) - d(b, C_j(b))]$.

(3) Border heuristic (BH). The domain is split into disjoint regions. The states of a region that have neighbors in other regions are called *border states*. For each pair of regions A and B, the distance $d(A, B)$, defined as the minimal distance between the border states of A and B, is precomputed and stored in the database (*primary data*). In addition, for each state x, the distance to the closest border state of its region (denoted $D_B(x)$ where B stands for "border") is stored (*secondary data*). The *border heuristic* between arbitrary states $a \in A$ and $b \in B$ is $h(a, b) = d(A, B) + D_B(a) + D_B(b)$.[2]

The *gateway heuristic* (GH) (Björnsson & Halldórsson 2006) is an approach that is related to PH. In GH portals are defined by grouping a number of neighboring border states which form a line segment into a single *gate*. A similar composite-portal idea appeared in (Botea, Müller, & Schaeffer 2004) but for non-optimal search. In this sense GH may be seen as a continuum between BH and PH. In PH portals may only have a single gate state while in BH *all* border states of a given region are grouped together.

It is important to note that, in order to obtain an admissible heuristic, GH uses a specialized map partitioning algorithm that looks for regions of certain shape (namely, whose borders consist of horizontal and vertical chains of states; these chains become the gates) and works well for the maps encountered in video games. The algorithm starts forming a new region at the top leftmost state that has not been previously assigned to any region and proceeds to "flood" the map until a certain stopping condition is met. At that point, the region is fixed without an option for further partitioning. Hence, the memory needs of the partitioning algorithm used in conjunction with GH are fixed. This is a very important consideration when comparing GH to PH or any other TDH, that are designed to use any given amount of memory.

The advantage of PH over all these heuristics is due to storing distances between *individual* border states. This results in more accurate heuristic estimates and enables the Portal-Based Search algorithm described below.

The algorithmics community has researched path-finding in explicit graphs (e.g., (Bauer *et al.* 2008)). Their work is usually based on Dijkstra's algorithm and does not use admissible heuristics in the same way that the AI community does. These methods use graph-theory abstractions to reduce the search space (e.g., by eliminating nodes and/or adding short-cut edges). Their research is orthogonal to that reported in the AI literature. Future work may see the AI and algorithmic efforts combined.

3 Portal-Based TDH

Given a graph $G = (V, E)$, a *k-way graph partitioning* of G is defined as partitioning of V into k disjoint subsets, $V_1, V_2, \ldots, V_k$. The subgraphs induced by these subsets are called *partitions*. Optimal or *good* k-way partitioning can be defined in a number of ways. For our purposes, a good partitioning has two properties: (1) the partition sizes are as balanced as possible (ideally, $\forall i \quad |V_i| = |V|/k$), and (2) the number of edges of E whose incident vertices belong to different partitions (*crossing edges*) is minimized.

Consider the graph $G = (V, E)$, whose vertices correspond to the states in the search space. Suppose that G has been partitioned and that G' is the subgraph induced by the crossing edges and their incident vertices. Choose a vertex cover of G' (minimal vertex cover is known to be NP-hard, so an approximation is adequate). The states that correspond to the vertices in the vertex cover are called *portals*. We denote the set of all portals by P. The partitions with the portals removed are called *regions*. A portal is therefore a state that has neighbor(s) in two or more regions. Although a portal is not considered part of a region, "portal p of region A" will informally mean that portal p has neighbors among the states of region A.

In the two-way partitioning example in Figure 2a, the graph was split along the dashed line and the bold edges are

[1] Throughout the paper, we use the term *state* to refer to states of the domain. The term *vertex* is used in graph-theoretical contexts. The term *node* refers to the search tree of A*.

[2] Some forms of these TDHs have appeared before. For example, the DH heuristics were independently used by a number of previous researchers (Goldberg & Harrelson 2005; Ng & Zhang 2002). However, the current line of research started by (Sturtevant *et al.* 2009; Felner *et al.* 2009) is the first to deeply explore these heuristics and provide theoretical analysis and thorough experimental results.

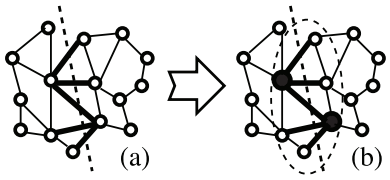

Figure 2: Regions and portals.

the crossing edges. The dashed ellipse in Figure 2b encircles the graph induced by the crossing edges. Choosing the minimal vertex cover results in the two bold states becoming portals.

Assume that the graph has been partitioned into k regions and that a local heuristic h_R (e.g. Manhattan distance) is defined for each region R and that the true distance between every pair of portals is pre-computed and stored in the database. The *portal heuristic* (PH) between two states located in different regions, $a \in A$ and $b \in B$, is defined as:

$$h(a,b) = \min_{p_A \in A, p_B \in B} \left(h_A(a,p_A) + d(p_A,p_B) + h_B(p_B,b) \right),$$

where p_A and p_B are portals of A and B, respectively. When one of the states (say a) is a portal, the heuristic takes the form: $h(p_A,b) = \min_{p_B} \left(d(p_A,p_B) + h_B(p_B,b) \right)$. When both states belong to one region ($a,b \in A$) then $h(a,b) = h_A(a,b)$. It is easy to see that if the local heuristics are admissible, then the portal heuristic is admissible.

Assume there are $|P|$ portals. A straightforward implementation of PH has the following memory requirements:

(1) Portals: Since portals usually belong to two regions, $2|P|$ memory is needed to list the portals of each region.[3] $|P|$ memory is needed to store the portals' locations.

(2) Inter-portal distances: $|P|(|P|-1)/2$ memory to store true distances between portals (*primary data*).

(3) Region identification: $|V|$ memory to assign each state to a region (*secondary data*).

This adds up to a total of $|V| + 3|P| + |P|(|P|-1)/2$ memory.

4 Partitioning the State Space

PH's performance depends on partitioning the state space into regions, ideally of similar size with few border states. The benefits of finer partitioning (i.e. more regions) are three-fold. First, it puts more start-goal pairs in different regions, thus enabling PH. Second, in many cases, smaller regions possess a smaller number of portals resulting in a less expensive computation of the portal heuristic. Lastly, we will see that finer partitioning provides an additional large benefit for the Portal-Based Search algorithm defined below. Of course, finer partitioning requires more memory.

The emphasis of the current paper is on the performance of PH, while the influence of a particular partitioning algorithm on the effectiveness of the heuristic is a research question that we leave for future work. In the following, we describe a partitioning algorithm that has the desirable properties of minimizing the number of crossing edges and balancing the sizes of the regions.

Our algorithm, called *counter-based graph-partitioning*, partitions a graph into two regions based on an approximation of the *betweenness centrality* (BC) (Holzer *et al.* 2004; Geisberger, Sanders, & Schultes 2008) metric of the edges.[4] BC is a measure of how popular an edge is with respect to using it as a member of shortest paths between vertices of the graph. Partitioning using this metric is based on the observation that edges connecting components of a well-partitioned (i.e., balanced sizes of components and few crossing edges) graph possess a large BC value.

The algorithm's initialization step assigns a counter to each edge and initializes all counters to 0. Each subsequent step consists of randomly choosing two vertices and finding a shortest path between them. The counters of the edges along the shortest path are increased. Edges whose counters reach a pre-defined limit are deleted from the graph and cannot be used again for the shortest path calculation.

This process is continued until the deletion of an edge puts the two vertices of the current step in different connected components. Vertices that may have been disconnected from the graph by the earlier steps are assigned to either of the two components. Portals are identified by an approximation of the minimal vertex cover. The components with the portals removed become the new regions.

This algorithm produces a two-way graph partitioning. To obtain a finer partitioning, we repeatedly apply the algorithm to the region with the largest number of states (allowing for roughly balanced region sizes) until the memory requirements of PH reach a pre-set limit.

5 Enhancements to PH

PH can be enhanced in a number of ways. The most important is the Portal-Based Search (PBS) algorithm which exploits the fact that the graph can be collapsed.

5.1 The Portal-Based Search (PBS) Algorithm

Recall the example in Figure 1 (left). Every path between states $a \in A$ and $b \in B$ has to exit A through one of the portals p_A and enter B through one of the portals p_B. Since the exact distances between portals are known, the search graph can be collapsed as follows. First, all regions except A and B are removed from consideration. Second, the regions A and B are connected by *macro-edges* p_A-p_B with the appropriate weights retrieved from the TDH database. The corresponding *collapsed search graph* of Figure 1 includes the left region (A), the right region (B), and the four macro-edges shown as straight solid arrows.

The following 3-step algorithm, Portal-Based Search (PBS), exploits the collapsed graph structure to find an optimal path by solving a series of sub-problems.

[3] This assumes that the number of portals which belong to more than two regions is negligible.

[4] The classical betweenness centrality index is defined for a vertex v as $\sum_{s,t \in V} |SP_{st}(v)|/|SP_{st}|$, where SP_{st} is the set of all shortest paths between s and t and $SP_{st}(v)$ is the set of such paths passing through v. We used a similar definition applied to edges.

Step 1: Finding the portals p_A and p_B on the optimal path. This is solved by applying an A^*-search powered with PH to the *collapsed search graph*. After this step, we have a partial optimal path that contains a single macro-edge p_A-p_B. To construct the complete path, the macro-edge needs to be refined. This is accomplished by the next two steps.

Step 2: Finding the remaining portals on the optimal path. We need to replace the macro-edge (p_A, p_B) of the partial optimal path by a sequence of portals. Suppose that the portal p_A is on the boundary of regions A and C. The second portal on the optimal path is a portal p_C that minimizes $d(p_A, p_C) + d(p_C, p_B)$.[5] Whenever there is a tie, we choose the portal closest to p_A. The remaining portals are found similarly.

Step 3: Finding an optimal path between each pair of successive portals. Note that each pair of successive portals belong to the same region. Furthermore, the way of breaking ties in Step 2 above guarantees that we do not need to consider any state outside of that region (even for concave-shaped regions). Thus, each of these searches constitutes a sub-problem that can be solved independently by searching within one region using the local heuristic of that region.

5.2 Choice of Search Direction

The third step of PBS finds the shortest path between successor portals, say p_i and p_{i+1}. This search could be conducted starting from p_i towards p_{i+1} or vice versa. Figure 1 (right) illustrates this idea. Searching from p_i with the Manhattan distance heuristic will result in exploring all of the cells inside the room, while searching from p_{i+1} will save this effort. For each pair of portals that share the same region a bit is stored in the PH's database to indicate which search direction leads to the least-effort search.

5.3 Run-time Optimizations

The following optimizations can reduce the time cost of computing the value of PH:

(1) Some portals are irrelevant for a given pair of start/goal regions. For example, assume a search from region A to region B. Let p_A be a portal in A such that, for every portal p_B in B, there exists another portal in A, p'_A, for which there is a shortest path from p_A to p_B that passes through p'_A. In such a case p_A can be removed from any heuristic calculation between regions A and B. This can be calculated online or done off-line (saved in a table). In our experiments we adopted the online option.

(2) When searching from $a \in A$ to $b \in B$, b as the goal state is fixed throughout the search. As long as the search remains in A, the heuristic estimate from each p_A to b is fixed and needs to be computed only once.

These run-time optimizations were enabled in all of the reported experiments.

[5] Alternatively, for each macro-edge, the next portal can be stored, but this adds memory to every entry in the primary data of the TDH database.

5.4 Achieving Perfect Portal Heuristics

Recall that a state $a \in A$ has the identity of its region stored in the *secondary data.* Suppose that, in addition to this, we store distances $d(a, p_A)$ from this state to its region's portals. Now, finding the optimal path between states $a \in A$ and $b \in B$ ($A \neq B$) can be done in time that is linear in the length of the optimal path by modifying the PBS algorithm:

(1) Choose the pair of portals p_A and p_B that minimizes the expression $d(a, p_A) + d(p_A, p_B) + d(p_B, b)$. This is a perfect heuristic.

(2) The macro-edge (p_A, p_B) is refined as in Step 2 of PBS.

(3) Compute the path from a to p_A. The second state on the path (call it a') is chosen among the neighbors of a such that $d(a, p_A) - cost(a, a') = d(a', p_A)$. The rest of the path is computed similarly.

We call this option the *perfect portal heuristic* (PPH). PPH is only available if a and b are in different regions. If they are in the same region, then the differential heuristic $h_d(a, b) = |d(a, p) - d(b, p)|$, where p is a portal in the region of a and b, can be used.

If each region has m portals on average, then additional $m|V|$ memory is required in the secondary data to store all state-portal distances. For a given amount of memory, PPH allows for the partitioning of the domain into a smaller number of regions as compared to the basic PH. This tradeoff will be studied below.

6 Experimental Results

Experiments were performed using three types of grid-based maps: rooms, mazes and maps taken from the popular game BALDUR'S GATE. An example for each type of map, along with its partitioning is shown in Figure 3. The white cells are portals. We used the same five 256×256 rooms maps (rooms are 8×8 with randomly opened doors between rooms) and 256×256 mazes that were used in (Sturtevant *et al.* 2009; Felner *et al.* 2009).[6] We handled these maps as four-connected grids and used Manhattan distance (MD) as the benchmark heuristic. We scaled the mazes to the size of 512×512 to increase the branching factor, the same way it was done in (Sturtevant *et al.* 2009; Felner *et al.* 2009). The BALDUR'S GATE maps differ considerably from each other. Hence, the averaging of results is not meaningful. We only report results for one BALDUR'S GATE map of size 148×139, but similar trends were obtained for other maps. Since diagonal moves are frequently allowed in computer games, we handled the BALDUR'S GATE maps as eight-connected and used the *octile distance* as the benchmark heuristic. In all cases the results were averaged over $1,000$ randomly chosen state-goal pairs.

6.1 Portal Heuristic Analysis

Two important attributes that influence the efficiency of PH are: **(1)** The number of portals per region. More portals per

[6] We have conducted experiments with rooms and mazes of different sizes as well. Although the exact numbers were different, the trends were similar to the ones reported below.

	Number of portals per region.						#Regions (Probability of start-goal in diff. regions)				
Mem	$\|V\|/64$	$\|V\|/16$	$\|V\|/4$	$\|V\|$	$4\|V\|$	$16\|V\|$	$\|V\|/64$	$\|V\|/16$	$\|V\|/4$	$\|V\|$	$4\|V\|$
Rooms	11.9	9.1	6.9	5.1	3.9	2.8	99 (0.83)	111 (0.93)	142 (0.98)	228 (0.99)	441(0.99)
Mazes	3.9	3.9	4.0	3.9	4.0	4.0	32 (0.96)	64 (0.98)	130 (0.99)	260 (0.99)	520 (0.99)
B. Gate	8.4	12.8	14.6	17.5	18.6	19.9	5 (0.55)	10 (0.68)	16 (0.85)	19 (0.92)	27 (0.96)

Table 1: Properties of environments w.r.t. partitioning. Memory does not include the $|V|$ needed to identify states with regions.

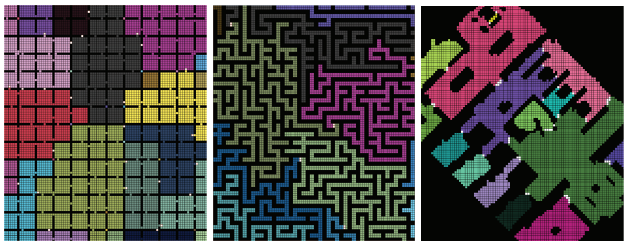

Figure 3: Rooms (left), mazes (center), *Baldur's Gate* maps (right) and their partitioning using $|V|/4$ memory.

region will make the calculation of PH harder and increase the memory requirements of PPH. Also, with more portals per region, a smaller number of regions is available for a given amount of memory. **(2)** The percentage of start-goal pairs that fall into different regions. For such pairs PH is available. For pairs within the same region, only the local heuristic is available.

Table 1 shows how these attributes are affected by the available memory. Our three domains cover a broad range of possibilities. Room maps represent environments where further partitioning the graph results in a decrease in the number of portals per region. Mazes represent environments where adding memory does not affect the number of portals per region. Finally, BALDUR'S GATE maps represent environments where further partitioning the graph results in an increase of the number of portals per region.

Table 2 shows the performance of different variants of PH on room maps (similar trends were obtained for mazes). Time and Nodes are the ratios of those obtained with the MD heuristic (i.e., 0.5 means half that required by MD). The rows in Table 2 represent the amount of memory allowed (beyond the $|V|$ needed to identify each state with a region). The columns correspond to different versions of PH. The best variant is highlighted in bold.

The first column (PH) corresponds to regular A* with the PH heuristic (without any enhancements). This simple version of PH reduces the search effort (compared to MD) by a large factor when sufficient memory is available. With $8|V|$ memory, the speedup is an order of magnitude. Since constant time per node is used for database lookups (and finding the minimum among these lookups), the improvement in the actual time is somewhat smaller than the reduction in the number of expanded nodes. The second column (PH+ME) corresponds to adding the macro-edges enhancement, i.e. searching with PBS. This reduces the search effort by up to an additional factor of two. Adding the correct directions enhancement to PBS yields a small improvement. This version is denoted in the reminder of the paper as PHe (for *PH-enhanced*). Note that as larger amounts of memory become available, the enhancements become less effective. The reason is that the regions are already small and a further increase in memory does not decrease the amount of the local search portion (Step 1 of PBS) by much.

PPH needs substantially more memory to store the distances from each state to the portals of its region. Out of the five room maps that we considered, this becomes available with $6|V|$ memory for three maps and with $7|V|$ for the other two maps. For mazes (not shown in Table 2), PPH becomes available for even a smaller amount of memory.

It is reasonable to expect that availability of a perfect heuristic would result in great performance improvements. Our results show that, at least for the rooms maps, this is not always the case. We attribute this to two artifacts of our off-line partitioning stage. First, the secondary data is stored as soon as enough memory is available. Second, partitioning is stopped as soon as the memory limit is attained. This may result in a small number of regions (each of large size) and many start-goal pairs being located in the same region – in which case the perfect heuristic is not available. Continuation of partitioning into more regions after reaching the memory limit may result in less secondary data if the number of portals per region decreases. This is a domain-dependent attribute of room maps which affected the results of PPH. For mazes, this stopping criterion is more effective because the number of portals per region does not decrease with more regions. Thus, PPH outperforms PHe for mazes, as detailed below. Improving on the off-line stage scheme for PPH based on domain-dependent attributes is a subject of future research.

6.2 Comparison With Previous Work

(Sturtevant *et al.* 2009) compared canonical heuristics (CH) and differential heuristics (DH) on rooms and mazes and concluded that DH provide better results on both domains. In addition, (Felner *et al.* 2009) defined the border heuristic (BH). However, BH was never compared to the other TDHs for grid-based maps. We compare all these TDHs here. For DH we used the *advanced placement* algorithm to place canonical states. CH is parameterized by the number of canonical states m to which distances from each state are stored. Results are provided for the best m value.

The relative performance compared to MD for a variety of TDHs is shown in Figure 4 for time (rooms, left; mazes, right) and Table 3 for nodes. For $|V|$ memory, DH already achieves a significant improvement over MD. PH and BH need $|V|$ memory to assign each cell of the grid to a region. Another $|V|$ memory is used by BH to store the distances

	Avg path length: 217; MD time: 0.00768sec; MD nodes: 5340							
	Nodes				**Time**			
	PH	**PH+ME**	**PHe**	**PPH**	**PH**	**PH+ME**	**PHe**	**PPH**
$\|V\|/64$	0.794	0.597	**0.584**	N/A	1.633	0.679	0.667	N/A
$\|V\|/16$	0.564	0.313	**0.294**	N/A	1.043	0.339	**0.320**	N/A
$\|V\|/4$	0.353	0.163	**0.147**	N/A	0.561	0.173	**0.157**	N/A
$\|V\|$	0.199	0.095	**0.084**	N/A	0.289	0.103	**0.094**	N/A
$4\|V\|$	0.111	0.067	**0.061**	N/A	0.160	0.079	**0.074**	N/A
$6\|V\|$	0.093	0.062	**0.058**	0.147	0.137	**0.067**	0.073	0.140
$7\|V\|$	0.086	0.061	**0.057**	0.143	0.129	**0.066**	0.073	0.119
$8\|V\|$	0.080	0.060	**0.056**	0.079	0.121	0.066	0.073	**0.051**

Table 2: Performance of PH (rooms maps). Memory does not include the $|V|$ needed to identify states with regions.

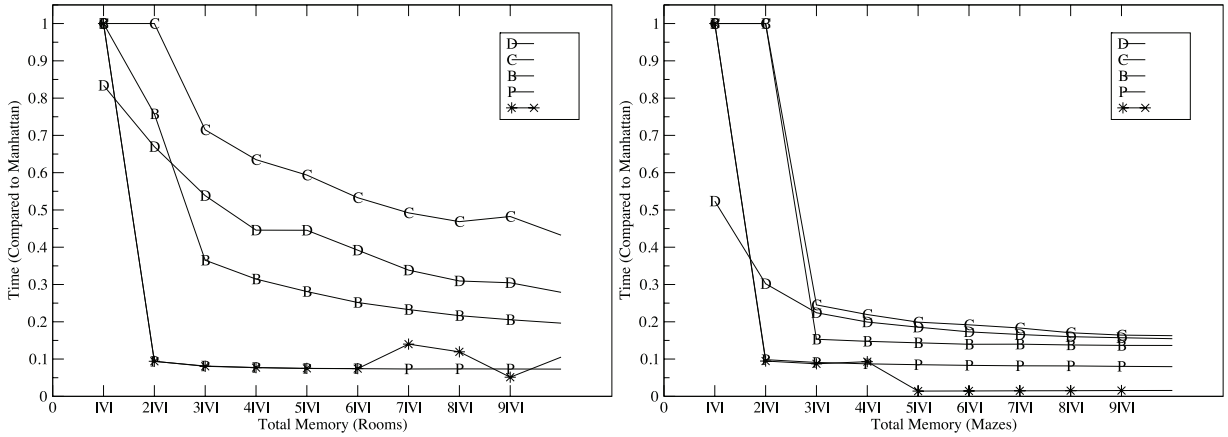

Figure 4: Time performance of TDHs.

	DH	CH	BH	PHe	PPH
	Rooms				
$\|V\|$	0.796	N/A	N/A	N/A	N/A
$2\|V\|$	0.639	N/A	0.714	**0.084**	N/A
$4\|V\|$	0.418	0.688	0.293	**0.064**	N/A
$8\|V\|$	0.275	0.484	0.196	**0.057**	0.143
	Mazes				
$\|V\|$	0.480	N/A	N/A	N/A	N/A
$2\|V\|$	0.270	N/A	N/A	**0.104**	N/A
$4\|V\|$	0.170	0.236	0.113	**0.089**	0.123
$8\|V\|$	0.123	0.160	0.103	0.079	**0.043**

Table 3: Nodes (compared to MD) on rooms and mazes.

	DH	CH	BH	PHe
	Nodes			
$2\|V\|$	0.600	N/A	N/A	**0.163**
$3\|V\|$	0.427	0.733	0.773	**0.152**
$8\|V\|$	0.208	0.532	0.690	**0.131**
	Time			
$2\|V\|$	0.674	N/A	N/A	**0.316**
$3\|V\|$	0.489	0.792	0.930	**0.333**
$8\|V\|$	**0.278**	0.612	0.858	0.325

Table 4: Results for 8-connected BALDUR'S GATE maps.

from each cell to its region's border. For larger amounts of memory, the border-based heuristics (BH and PH) outperform the other TDHs by a large margin. All versions of PH outperform BH. Unlike rooms, for mazes the number of portals per region stays constant. Therefore, for mazes, whenever $5|V|$ or more memory is available, PPH significantly outperforms PHe.

Baldur's gate maps. We conducted experiments with 8-connected BALDUR'S GATE maps. Since the number of portals per region gets larger with the increased amount of memory, these maps are a showcase of a limitation of the PH. The results are shown in Table 4. It is interesting to note that PH still outperforms other TDHs when not too much memory is available. When the amount of memory increases (see the $8|V|$ row), the number of portals per region increases too and the relative performance of PH degrades as expected and it is no longer the best TDH. PPH is not available here, since the high number of portals per region precludes storing the secondary data.

6.3 Comparison With the Gateway Heuristic

We now compare PH to GH. Recall from the introduction that GH uses a specialized map partitioning algorithm that incurs fixed memory requirements. There we explained that this artifact of GH is needed to guarantee admissibility. In contrast, PH uses an iterative partitioning algorithm that can fit the available amount of memory. It is because of this distinction between GH and PH (and all other TDHs) that we

cannot compare the two heuristics for a large range of memory amounts as was done in our comparison of the different versions of PH and the other TDHs. Rather, the amount of memory will be dictated by the GH's partitioning algorithm and the maps being used.

We implemented GH and experimented with the same maps and test cases that were used above. For both room and maze maps, the partitioning algorithm of GH tends to form small regions with the result of using a large amount of memory. For the rooms maps, 1,027 regions with 2,719 gates[7] were formed on average, using $72|V|$ memory. The number of expanded nodes compared to the MD heuristic is 0.062, which is comparable to the number of nodes expanded by PHe with $5|V|$ memory. For the maze maps, 4,185 regions with 8,443 gates were formed on average, using $272|V|$ memory. The number of expanded nodes compared to the MD heuristic is 0.057, which is better than PHe, but worse than PPH which uses $5|V|$ memory.

The BALDUR'S GATE maps is the kind of environment that GH were designed for. A total of 91 regions with 255 gates were formed, which corresponds to using about $6.36|V|$ memory. The number of expanded nodes compared to the octile distance heuristic is 0.685 and the time is 0.982. These results are comparable to the results reported in (Björnsson & Halldórsson 2006) (although their time results are somewhat better, which must be due to using less generic code with specific optimizations). As reported in Table 4 PH expanded about a fourth of this number of nodes and used only about a third of this time.

7 Conclusions and Future Work

We introduced the portal heuristic, a new TDH which in most cases outperforms previous TDHs. The two most important advantages of PH over previous TDHs are the more accurate heuristic values and the ability to sub-divide the path-finding problem into a number of smaller problems by applying the Portal-Based Search algorithm. In addition, a perfect heuristic (PPH) can be produced with a moderate amount of memory for states located in different regions.

The main limitation of PH is that its performance depends on whether the underlying graph can be partitioned into regions that are connected with each other by a small number of edges. This is true, for example, when corridors or areas surrounded by obstacles exist in the graph. This is the case with the room and maze maps. In these cases, the number of portals per region is rather low when we further continue to partition the graph. Thus, PH substantially outperformed other TDHs. PH is not that effective when there are too many portals per region. Both the memory needs and the computation time are increased in this case and the relative performance of PH compared to other TDHs decreases. The BALDUR'S GATE maps is a showcase of this limitation.

This research can be extended in several interesting ways.

(1) In this paper we have only experimented with grid-based graphs. In order to better demonstrate the advantages and limitations of PH it should be also implemented on other graphs such as real road maps, robot's visibility graphs etc.

(2) A graph partitioning algorithm was proposed that appeared to have the desirable properties of balancing the sizes of the regions and minimizing the number of edges between the regions. It remains to be seen how sensitive PH's performance is to the partitioning, and whether there exists a more PH-oriented partitioning algorithm.

(3) We have shown that run-time optimizations can be applied to improve the efficiency of PH despite the presence of a high number of portals per region. Continuing this line of research may make PH applicable to graphs that do not possess good partitioning properties.

(4) The gateway heuristic (GH) proposed by (Björnsson & Halldórsson 2006) offers an attractively low computational cost. However, in its current form, GH is limited to grids, incurs a fixed amount of memory needs and, most importantly, does not produce as high heuristic values as PH. However, one could conceive of a generalization that would combine the generic partitioning scheme of PH with the computational efficiency of GH through using a more flexible mechanism of grouping portals into gates.

8 Acknowledgments

This research was supported by the Israeli Science Foundation (ISF) grants No. 728/06 and 305/09, and by iCORE.

References

Bauer, R.; Delling, D.; Sanders, P.; Schieferdecker, D.; Schultes, D.; and Wagner, D. 2008. Combining hierarchical and goal-directed speed-up techniques for dijkstras algorithm. In *7th Workshop on Experimental Algorithms*, 303–318. LNCS #5038.

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

Botea, A.; Müller, M.; and Schaeffer, J. 2004. Near Optimal Hierarchical Path-Finding. *J. of Game Development* 1(1):7–28.

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

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

Felner, A.; Barer, M.; Sturtevant, N.; and Schaeffer, J. 2009. Abstraction-based heuristics with true distance computations. In *Proceedings of SARA-09*.

Geisberger, R.; Sanders, P.; and Schultes, D. 2008. Better approximation of betweenness centrality. In *9th Workshop on Algorithm Engineering and Experiments*.

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

Holzer, M.; Prasinos, G.; Schulz1, F.; Wagner1, D.; and Zaroliagis, C. 2004. Engineering planar separator algorithms. In *The 3rd Int. Workshop on Experimental and Efficient Algorithms*.

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

Sturtevant, N.; Felner, A.; Barer, M.; Schaeffer, J.; and Burch, N. 2009. Memory-based heuristics for explicit state spaces. *IJCAI-09* 609–614.

[7]Note that GH uses two-sided gates and stores four distances for each pair of gates. In our implementation, we created one-sided gates that belong to only one region with a nearly equivalent resulting heuristic. In fact, our results are comparable to those reported in (Björnsson & Halldórsson 2006).