Improving Heuristics for Pathfinding in Games

Stefan Janssen
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
s.c.janssen@student.utwente.nl

ABSTRACT

Pathfinding is used extensively in modern computer games: more efficient pathfinding allows for more units and bigger games. Several heuristics already exist to improve the performance beyond that of heuristics as the Manhattan distance when used in the A* search algorithm. In this paper, the new Boundary Heuristic (along with several variants) is introduced in order to attempt to increase the performance even further. The Boundary Heuristic attempts to identify regions in which the more simple heuristics, such as the Manhattan distance, provide correct results. The distance between the edges of these areas are calculated prior to the pathfinding. All heuristics are evaluated by performing a large number of pathfinding searches on different map types. The experimental results show performance improvements of the Boundary Heuristic on three of the four different map types.

Keywords

pathfinding, game, heuristic, search.

1. INTRODUCTION

Modern computer games make extensive use of pathfinding. Units cannot move to their destinations without a path to take them there. These paths are getting more complex as game worlds are getting more crowded. Artificial intelligence in games benefits from accurate distances (for example to resources or to enemy camps) in their decision making. Using efficient pathfinding algorithms saves critical resources that can be spent elsewhere. In large scale games (such as real-time strategy games) there are several thousands of units, each with its own destination. An efficient pathfinding algorithm is of even larger importance to these games.

Most game pathfinding implementations use the A* algorithm. This algorithm uses heuristics such as the Euclidean distance, Manhattan distance or others depending on the allowed movement. These approaches[1] tend to explore a large number of cells in maps with obstacles (see Figure 1). A number of heuristics have been proposed that use map specific knowledge to drastically decrease the number of explored cells [2, 5]. While all perform better than naive heuristics such as the Euclidean distance, each still has its own strengths and weaknesses. A heuristic combining aspects of the several approaches seems likely to further improve pathfinding in games.

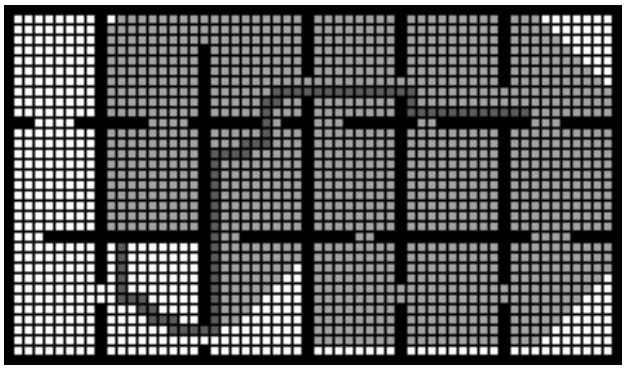

Figure 1. Unnecessary explored cells (grey, optimal path in dark grey) by the A* algorithm with the Manhattan distance.

Optimal informed search algorithms (such as A*) can be improved in several ways: new or optimized search algorithms [1], algorithms to reduce the number of cells [6] and better heuristics [2, 5] can be found in literature. Developments in each part show clear improvements over the naive approach. This paper will focus on improving the heuristics.

The goal of this paper is to investigate how heuristics used in game pathfinding can be further improved. In particular, the following questions will be investigated:

- What are (several of) the best performing (exploring the least amount of cells) heuristics that have been proposed in literature?
- What are the strengths and weaknesses of these heuristics?
- What combined heuristic uses these strengths while avoiding the weaknesses?
- How does the new heuristic perform compared to the previous heuristics?

In the remaining part of the paper, the new Boundary Heuristic is introduced. The Boundary Heuristic attempts to further improve the performance of the Portal Heuristic by changing the partitioning algorithm. In the evalua-

[1] In practice, game developers also use several non-optimal techniques such as 'Hierarchical Pathfinding' to improve performance.

14th Twente Student Conference on IT January 21st, 2011, Enschede, The Netherlands.

tion, the Boundary Heuristic is compared to several other heuristics on four different map types.

2. RELATED WORK

Better heuristics are not only needed in pathfinding, searches in other problems benefit from good heuristics as well. Several heuristics have been made for problems in general which can also be applied to pathfinding. Some of these heuristics store their pre-calculations in memory, and are therefore called the 'memory-based' heuristics. For example, Pattern Databases [3] provide a method to build admissible heuristics, but are less suited for pathfinding [7, 4], because of the variation of the goal state. Another group of memory-based heuristics, the true distance heuristics, performs better at pathfinding and similar domains. Examples of true distance heuristics are the Differential Heuristic [7, 4], the Canonical Heuristic [7, 4] and the Border Heuristic [4].

With the Differential Heuristic [7, 4] a small number of special states (the canonical states) are selected. For each of these states, the cost to all other states is pre-calculated. Because $|d(s,a) - d(s,b)|$ places a lower bound on the distance between the start state a and goal state b, $d(a,b)$, for each state s, a heuristic value can be determined for each of the canonical states. All these values are combined to a final heuristic value, which is the maximum of the value for each canonical state. For even better results, the canonical states are placed with the 'advanced placement', which places the states as far apart as possible.

The Canonical Heuristic [7, 4] also selects a small number of canonical states, but contrary to the Differential Heuristic, the Canonical Heuristic provide a heuristic value for each pairs of canonical states. For each pair of canonical states, the heuristic value is $d(C_i, C_j) - d(a, C_i) - d(b, C_j))$ where a and b are the start and goal state and C_x are the canonical states. This value might be negative in some cases, so the heuristic also uses the Differential Heuristic, $|d(C_i, a) - d(C_i, b)|$, for canonical state. The final heuristic value is the maximum of the heuristic value for each pair and for each canonical state alone.

The Border Heuristic [4] works slightly differently by defining a number of neighbourhoods, instead of canonical states. The minimal distance is pre-calculated between the border states of each neighbourhood. For each node a the heuristic value to the goal state b is $d(C(a), C(b)) + D_B(a) + D_B(b)$, where $d(C(a), C(b))$ is the distance between the two neighbourhoods and $D_B(x)$ is the distance to the nearest border state.

An alternative approach specific to pathfinding is to attempt to identify the natural areas on maps. Paths will be guided through the small gates between these areas, avoiding any dead-ends and therefore reducing the unnecessary exploration of cells. The Gateway Heuristic [2] is an example of this. This heuristic uses a specialized flooding algorithm to find more-or-less rectangular areas and the 'gates' between them. The distances from each of these gates to all other gates are calculated and stored in memory. When the heuristic has to estimate the distance to the target, this extra information is used to provide a more accurate guess. The Portal Heuristic [5] works on the same principle, but uses a different algorithm to partition the map in multiple zones. Another differences is that the Portal Heuristic does not combine nodes to form gates: the distances are stored between each two individual nodes. Similar to the true distance heuristics, the amount of available memory for the Portal Heuristic can be specified.

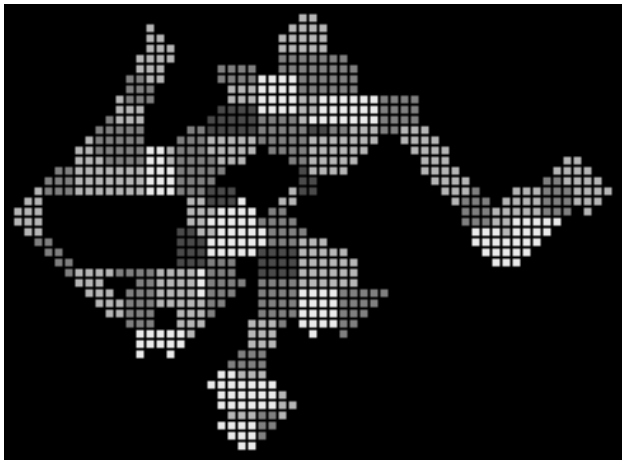

Figure 2. The eager partitioning of the Portal Heuristic with only $8|V|$ memory in a game map, resulting in 58 partitions.

3. BOUNDARY HEURISTIC

The Portal Heuristic, along with the Differential Heuristic, outperforms most other heuristics. While the Differential Heuristic tends to perform well on all kinds of maps, the Portal Heuristic is particularly good in room maps.

A major weakness of the Portal Heuristic however, lies with its inability to cope with open areas, such as *open with hills maps* (see Table 3 from the evaluation at the end). The partition algorithm attempts to eagerly create, depending on the memory allowance, too many partitions (compare Figure 2 and 3). These partitions are a waste, because a more simple, naive heuristic, the Manhattan distance in this case, works perfectly in completely open areas. Ideally, in a map without any obstacles at all the partition algorithm should create only one big partition.

A possibly better heuristic can be created by looking for areas where the naive heuristic does provide correct distances. Conceptually, this is not much different from the Gateway Heuristic, which also looks for areas in which the naive heuristic works well. After the partitions are formed, portals are created between them, and the distances between all portals are stored, much like the Portal Heuristic. In the remaining of this paper, the new Boundary Heuristic will be introduced and evaluated.

3.1 Partitioning algorithm

The partitioning algorithm has to look for areas in which the naive heuristic (such as the Manhattan distance) provides correct distances. In order to find the borders of an additional area the algorithm first picks a random unoccupied node which is not part of any area yet. The new area consists of the nodes next to the selected node for which the naive heuristic gives the correct distance. These nodes can be found efficiently by performing a breadth-first search. In addition to blocked cells, the search does not continue into any cells which have a faulty naive heuristic value. All nodes explored by the search are assigned to the new area, with the exception of the node directly next to the nodes with the faulty heuristic values. These searches are preferably started from corners and other dead-ends, otherwise the search might start at the border between two areas and cover them both. Like the Differential Heuristic, where canonical states are also placed in dead-ends of a map, dead-ends can be found by conducting a breadth-first search from a random, unassigned

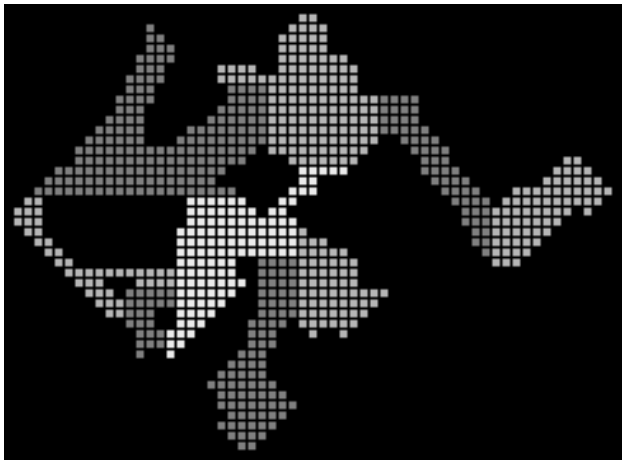

Figure 3. The more minimalistic partitioning of the Boundary Heuristic without any memory limitations with only 9 partitions in the same game map.

node. The unoccupied node which is the farthest away from the random node is used as the base node for the new area. Notice that, to prevent zones from 'recapturing' nodes from other zones, nodes that are already assigned to an area are considered blocked. This process is repeated until there are no more unassigned, non-blocked nodes.

3.2 Reducing the number of portals

If enough memory is available for all portals in the complete partitioning, then we are done. If this is not the case, we need a way to reduce the number of portals. This can be done in two different manners: as in the Portal Heuristic and the Border Heuristic, different zones can be merged into larger zones, or, as in the Gateway Heuristic, several portals can be grouped together into one larger portal (similar to a gateway).

When reducing the number of different zones, two neighbouring zones are selected with the largest number of shared portals between them, relative to the size of the largest zone. These two zones are merged into one larger zone. This process is repeated until the memory usage is acceptable. Notice that this will always complete: eventually the entire map will be covered in one zone and use practically no memory.

Alternatively, portals can be grouped together to reduce memory requirements. This reduction strategy is similar to the method used in the Gateway Heuristic, in which several portals are more or less combined in a single gateway. When multiple portals are combined, the new, bigger portal is ideally placed at the position of the portal closest to the centre of the original group of portals. The distance from the combined portal to the other portals is the minimum of the distances from each original portal to the other portal. Alternatively, the distances can also be corrected for a possible detour: the distance to the farthest combined portal is subtracted twice (twice for the portal on each end, so up to four times) from all the distances to all other portals. This detour occurs when the optimal path goes through the portal farthest from the combined portal on both sides. Either picking the minimum or applying the correction keeps the heuristic admissible and thus A* optimal. Notice that this method can not make the memory usage arbitrary small: each gateway requires at least one portal.

In addition, both methods could also be combined. First the number of areas can be reduced to a certain point, after which portals are combined until the memory requirements are met. Which method of reducing the number of portals creates better heuristics is subject of further research.

3.3 Distance estimation during runtime

The distance estimation is identical to the estimation in the Portal Heuristic. If the start and goal are in the same region, the naive heuristic is used. If the two states are in different regions, the Boundary Heuristic is defined as: $h(a,b) = \min(h_{naive}(a,P_a) + d(P_a,P_b) + h_{naive}(P_b,b))$ for all portals in both areas. Here $h_{naive}(a,P_a)$ indicates the value of the naive heuristic, for example the Manhattan distance, between the two positions. The start and goal state are encoded as a and b, their portals as P_a and P_b. The stored distance between two portals is represented with $d(P_a,P_b)$.

3.4 Possible optimisations

In order to further improve performance, several optimisations can be made. Similar to the Portal Heuristic, the distance from each portal pA to the end state, b can be cached. Other optimisations from the Portal Heuristic are also applicable to the Boundary Heuristic, but the resulting heuristic would not be a 'pure' heuristic. For example, the local search between two portals could be reversed to go from goal to start if this would be more efficient. For each pair of portals in the same zone, a bit would have to be stored to indicate this.

4. EMPIRICAL EVALUATION

An experiment is performed to evaluate the new Boundary Heuristic in several variants. During the evaluation a large number of paths are calculated using A* with each of the heuristics. During the calculations aspects such as the number of explored cells, time taken to compute the path, memory usage and the initial estimation of path cost are measured. The start and goal position of these paths are random, but only reachable destinations are considered. Because different heuristics might perform better on different maps, the heuristics are tested in randomly generated rooms, randomly generated mazes, open space with randomly placed U-shaped obstacles and real game maps. The game of choice is *Baldur's Gate II*, as it has already been used by others [2, 5]. The results of the new heuristics will be compared to those of the existing heuristics.

The A* search algorithm is used because it is the most mainstream search algorithm, and therefore most likely to

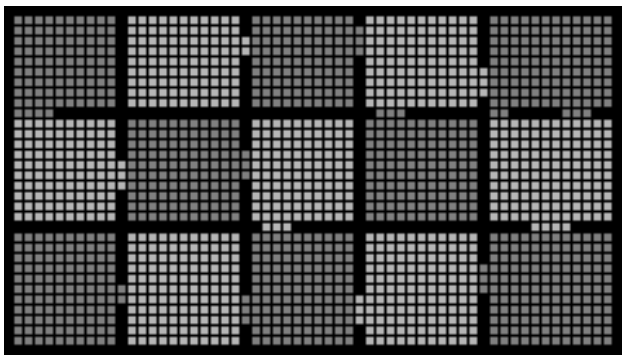

Figure 4. A sample of a randomly generated room map with Boundary (∞) partitioning.

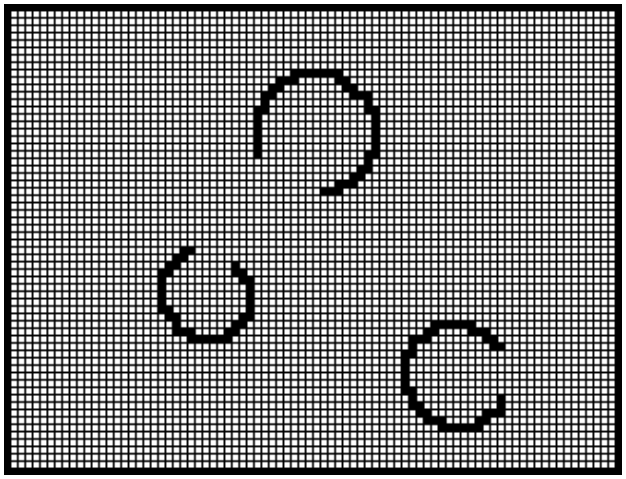

Figure 5. A sample of a randomly generated *open with hills map*.

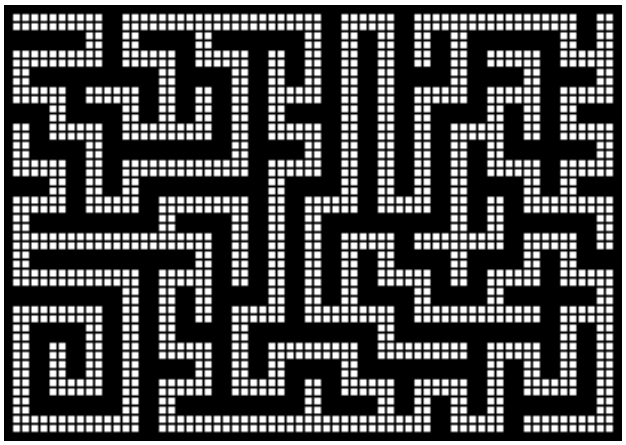

Figure 6. A sample of a randomly generated maze map.

be used in actual games. Notice that other algorithms exist, such as Beam Search and even newer developments, like Fringe Search [1]. Search algorithm that solve a collection of search problems, like D* search, also exist. In the implemented A* search, the priority list is implemented as a tree, using an additional hash table for reverse lookups. The priority list breaks ties in a LIFO fashion, so the search tends to act like a depth-first search. Notice that neighbours of a single cell are added or updated in a random order. Overall, this makes the A* implementation quite similar to the optimized implementations that are expected to be found in real games.

4.1 Maps

The heuristics are evaluated on different types of maps. The room, *open with hills* and mazes maps are randomly generated, while the game maps are randomly picked from a provided collection. The sizes of the generated maps are distributed uniformly between 19 by 22 cells and 316 by 318 cells, which are the sizes of respectively the smallest and largest game maps (see Figure 7).

The randomly generated room maps (see Figure 4 and Figure 1) consist of several rooms. At least one path is guaranteed to exist between two rooms, though a number of additional doors are added. The doors are of varying width. Unintentionally the additional doors may also overlap or join with the already existing doors to create even

Figure 7. One of the larger game maps.

larger doors. These room maps are intended to simulate the kind of map typically found in Role-Playing Games.

The open maps consist of open space, filled with some hill shaped obstacles (see Figure 5). The hills are of random size and placed at random locations, although hills cannot overlap with other hills or the map border. Each hill has a non-blocked ramp on one side. The size of the ramp varies between hills. These maps bear similarities with the larger, more open maps found in Real-Time Strategy games. These maps are largely open space, but feature several hills. These hills have a slope on one side, and steep, impassable cliffs at the other sides. Naive heuristics such as the Manhattan Distance are expected to perform better on open maps due to the lack of obstacles.

The maze maps are, as the name suggests, mazes. All the corridors in the maze are two cells wide (see Figure 6), so even within corridors there is some potential to unnecessarily explore cells. The mazes are generated with a randomized version of the depth-first algorithm. This mazes created by this algorithm have only one route between each two points and no loops, so-called 'perfect mazes'. The maze maps make for an interesting case because naive heuristics like the Manhattan Distance are expected to perform badly on these maps. The path to the destination on maze maps usually involves a detour, whereas the direct route (which is favoured by naive heuristics) tends to be a dead-end.

The game maps are picked from a collection of real game maps and provide a real-life evaluation (see Figure 2 and Figure 3). The game maps are from the game *Baldur's Gate II*. Others have already used these maps for the same purpose [2, 5].

4.2 Heuristics

In the evaluation the new heuristic, the Boundary Heuristic, will be compared to older heuristics. The Differential Heuristic, the Canonical Heuristic, the Border Heuristic, the Gateway Heuristic and the Portal Heuristic are used

as comparison. For each of these heuristics, the heuristic value used by the search algorithm is the maximum of the naive heuristic and the heuristic itself.

Being the least complex heuristics, both the implementations of the Differential Heuristic and the Canonical Heuristic are identical to their original specification [7, 4]. For the Differential Heuristic the 'advanced placement' as described in [7] is used. This places the canonical states in the corners or dead-ends of the map. The Border Heuristic [4] does not provide any 'advanced placement', so neighbourhoods are placed randomly, with cells belonging to their closest neighbourhood. Both the primary and the secondary data are pre-calculated and stored in memory.

The implementation of the Gateway Heuristic differs from the description in [2]. The partition algorithm is identical to the one provided. However, because a 'pure' heuristic is not provided with the direction of the search, gateways do not have specific directions. Therefore, for each pair of gateways only one distance is calculated. Likewise, the implementation does not prevent the search from entering possible dead-ends by returning a infinite value, because an admissible heuristic never overestimates the cost. In addition the distances between gateways are not calculated using multiple A* searches as suggested in [2], but with a significantly smaller amount of breadth-first searches. This will not affect the found distances however.

Similar to the Gateway Heuristic the Portal Heuristic also applies a number of tricks that go outside the scope of a 'pure' heuristic. In [5] the described search method is to first construct a higher level path using only portals, and then find the local paths (possibly in a parallel fashion) from portal to portal with a naive heuristic such as the Manhattan Heuristic. Another optimization is to store the search direction that explores the least amount of nodes between each two portals, and if faster, search from goal to start. Not only require these additions the modification of the search algorithm, they could also be applied to other heuristics that use areas, such as the Border Heuristic and Gateway Heuristic. In the 'pure' heuristic implementation the returned value is $d(a, p_A) + d(p_A, p_B) + d(p_B, b)$ for the pair of portals that minimizes this value. Like in the original implementation, the (minimal) value of $d(p_A, p_B) + d(p_B, b)$ is cached for each portal p_A . These caches are cleared if the destination changes.

The Boundary Heuristic is implemented as described earlier. The implementation caches the distances from each portal to the end position. Other optimizations, which would not result in a 'pure' heuristic, are not applied. Several versions of the Boundary Heuristic will be evaluated: one without any reduction (in other words, an infinite memory allowance), and several variants with different memory allowances and different reduction methods.

4.3 Results

The results for the different map types can be found in Table 1 up to and including Table 4. For each map type, the results are based on 25 random maps with a 1000 random paths each. Non-existing paths are excluded because these paths would force A* to explore the entire map.

The heuristics are (if applicable) parametrized with their memory setting, so Differential ($2|V|$) means the Differential Heuristic set to use $2|V|$ of memory ($1|V|$ is the map size, including both blocked and non-blocked nodes). The memory parameter is only a guideline for the heuristic, the actual amount may vary. The memory column does indicate the actual amount of memory used relative to the number of cells in the map. Notice that some heuristics have a fixed memory constraint, and that the naive heuristic does not use any memory. The time column represents the amount of wall clock time taken for each path. This includes the time in both the heuristic code and A* code. The relative number of cells explored is the number of explored cells relative to the path length. The explored cells include cells which are part of resulting path, so the best possible relative score is 1. Like the time, it is important to avoid the unnecessary exploration of cells for performance reasons. The relative estimate is the estimate made by the heuristic before the actual search has been conducted, again relative to the path size. A high estimate means that, if only the rough distance is desired, the entire search can be omitted. Because all heuristics are admissible, the relative estimate can never be higher than 1. The average path length is indicated for each map. The lengths of the found paths will be the same for each heuristic: the exact same paths are executed with all heuristics.

5. DISCUSSION

On the game maps (see Table 1) the Boundary Heuristic gives the best time performance. The Portal Heuristic explores slightly less nodes, but takes more time per node. The Portal Heuristic also has a slightly larger initial estimation. The Boundary Heuristic has the second least amount of explored nodes and does so at a lower memory usage. Notice that the different memory variants with area reduction perform nearly identical to the infinite memory variant, which already uses a low amount of memory.

On the maze and room maps (see Table 2 and Table 4), the Boundary Heuristic performs best in all aspects (except for memory). The infinite memory variant takes the least amount of time, explores the least amount of nodes and gives the best initial estimation. In both map types, the infinite memory variant is followed up by the $8|V|$ area reduction variant, which ends second.

On the *open with hills* maps, the results are quite different however (see Table 3). While the Boundary Heuristic still explores a small amount of cells (a near tie with Differential $8|V|$), relative time performance is a lot worse. While performing slightly better than the Portal Heuristic, time performance is roughly tied with the Manhattan distance. Contrary to the expectations earlier, the Boundary Heuristic fails to significantly improve on the Portal Heuristic on these more open maps.

It seems that partitions with a lot of portals decrease performance. A possible explanation could be the even larger number of portals in such case: each one has to be evaluated in each heuristic call. The Boundary Heuristic explores the least amount cells, but the cost per heuristic call is too large. This is not only problematic for the Boundary Heuristic, but other heuristics that create partitions, such as the Portal Heuristic, also perform poorly in these maps.

The results make a good case for the new Boundary Heuristic, which performs better than the other heuristics on the game, maze and room maps. The version without any memory limitations is the best performing variant, closely followed by the $8|V|$ variant with area reduction on most map types. Area reduction is better than portal reduction: on the room and game maps the $8|V|$ portal reduction variant is even outperformed by the $2|V|$ area reduction variant. This has likely to do with the fact that the portal reduction significantly reduces the heuristic values, which in turn leads to more explored cells and a worse time per-

Table 1. Empirical performance of different heuristics for game maps. The average path size on these maps is 60.72 nodes. Memory is relative to map size, the number of nodes are explored and initial estimate are both relative to the path size. For each column, both the average ($\bar{x}$) and the standard deviation (s_N) are listed. Results are sorted ascending on average time, best values are in bold.

GAME MAP heuristic	memory ($\|V\|$)		time (ms)		rel. explored		rel. estimate	
	$\bar{x}$	s_N	$\bar{x}$	s_N	$\bar{x}$	s_N	$\bar{x}$	s_N
Boundary (∞)	3.04	2.13	**0.38**	**1.48**	1.342	**1.154**	0.992	0.037
Boundary ($8\|V\|$, Area)	2.91	1.86	0.44	1.81	1.385	1.269	0.991	0.040
Boundary ($4\|V\|$, Area)	2.49	0.87	0.46	1.87	1.421	1.341	0.990	0.041
Differential ($8\|V\|$)	8.00	**0.00**	0.47	1.79	1.433	1.544	0.989	0.041
Portal ($8\|V\|$)	8.22	0.15	0.53	2.61	**1.320**	1.364	**0.997**	**0.017**
Boundary ($2\|V\|$, Area)	1.77	0.23	0.60	2.68	1.573	1.830	0.989	0.041
Portal ($4\|V\|$)	4.18	0.13	0.62	2.99	1.441	1.673	0.995	0.022
Differential ($4\|V\|$)	4.00	**0.00**	0.70	2.72	1.805	2.448	0.980	0.060
Portal ($2\|V\|$)	2.15	0.13	0.81	3.71	1.654	2.069	0.991	0.034
Boundary ($8\|V\|$, Portal)	2.44	1.39	0.92	3.77	1.726	2.006	0.989	0.042
Canonical ($8\|V\|$)	8.00	**0.00**	0.97	3.29	1.713	2.055	0.981	0.310
Differential ($2\|V\|$)	2.00	**0.00**	1.00	4.05	2.224	3.521	0.968	0.087
Canonical ($4\|V\|$)	4.00	**0.00**	1.02	3.77	2.148	3.041	0.968	0.207
Border ($8\|V\|$)	8.03	0.02	1.04	3.68	2.262	2.580	0.977	0.045
Boundary ($4\|V\|$, Portal)	2.20	0.63	1.05	4.11	1.857	2.245	0.989	0.038
Border ($4\|V\|$)	4.00	**0.00**	1.14	4.08	2.404	2.820	0.971	0.053
Boundary ($2\|V\|$, Portal)	1.53	0.25	1.25	4.53	2.278	2.622	0.978	0.049
Canonical ($2\|V\|$)	2.00	**0.00**	1.30	4.93	2.704	4.176	0.949	0.129
Differential ($1\|V\|$)	1.00	**0.00**	1.44	5.69	2.839	4.587	0.951	0.108
Gateway	6.66	5.96	2.30	8.52	3.017	4.234	0.948	0.101
Manhattan	**0.00**	**0.00**	2.34	8.03	4.182	6.708	0.908	0.164

Table 2. Empirical performance of different heuristics for maze maps. The average path size on these maps is 1080.05 nodes.

MAZE MAP heuristic	memory ($\|V\|$)		time (ms)		rel. explored		rel. estimate	
	$\bar{x}$	s_N	$\bar{x}$	s_N	$\bar{x}$	s_N	$\bar{x}$	s_N
Boundary (∞)	11.00	7.36	**3.44**	**3.68**	**1.119**	**0.254**	**0.998**	**0.017**
Boundary ($8\|V\|$, Area)	6.50	2.00	4.27	16.24	1.283	0.389	0.996	0.021
Differential ($4\|V\|$)	4.00	**0.00**	4.28	5.04	1.389	0.435	0.987	0.051
Differential ($8\|V\|$)	8.00	**0.00**	4.29	5.08	1.297	0.336	0.994	0.030
Gateway	41.81	30.09	4.64	5.19	1.488	0.380	0.984	0.074
Differential ($2\|V\|$)	2.00	**0.00**	4.74	5.77	1.571	0.587	0.969	0.085
Portal ($8\|V\|$)	8.03	0.03	4.95	5.88	1.526	0.467	0.992	0.027
Boundary ($4\|V\|$, Area)	3.83	0.48	5.08	7.08	1.552	0.508	0.990	0.035
Border ($8\|V\|$)	8.01	0.01	5.60	6.04	1.933	0.349	0.980	0.043
Boundary ($8\|V\|$, Portal)	4.69	1.55	5.63	6.75	1.526	0.457	0.995	0.020
Border ($4\|V\|$)	4.01	**0.00**	5.69	6.03	1.998	0.384	0.971	0.058
Portal ($4\|V\|$)	4.02	0.02	5.78	6.88	1.829	0.508	0.981	0.048
Boundary ($2\|V\|$, Area)	2.00	0.01	6.15	8.53	1.978	0.556	0.969	0.065
Canonical ($4\|V\|$)	4.00	**0.00**	6.57	8.76	1.797	0.734	0.934	0.136
Portal ($2\|V\|$)	2.01	0.01	6.65	7.24	2.209	0.493	0.955	0.080
Boundary ($4\|V\|$, Portal)	3.75	1.67	7.13	8.42	1.865	0.466	0.988	0.029
Differential ($1\|V\|$)	1.00	**0.00**	7.38	9.44	2.379	1.070	0.863	0.225
Boundary ($2\|V\|$, Portal)	3.55	1.81	7.76	8.35	2.101	0.345	0.982	0.036
Canonical ($2\|V\|$)	2.00	**0.00**	8.16	10.25	2.499	1.143	0.813	0.248
Canonical ($8\|V\|$)	8.00	**0.00**	8.32	10.63	1.579	0.605	0.967	0.091
Manhattan	**0.00**	**0.00**	16.11	27.52	5.586	1.754	0.200	0.176

Table 3. Empirical performance of different heuristics for *open with hills* maps. The average path size on these maps is 119.11 nodes.

OPEN WITH HILLS MAP heuristic	memory ($\|V\|$)		time (ms)		rel. explored		rel. estimate	
	$\bar{x}$	s_N	$\bar{x}$	s_N	$\bar{x}$	s_N	$\bar{x}$	s_N
Differential ($8\|V\|$)	8.00	**0.00**	**0.94**	**1.86**	**2.037**	**2.246**	0.992	0.029
Differential ($4\|V\|$)	4.00	**0.00**	1.19	2.84	2.616	3.309	0.988	0.039
Canonical ($4\|V\|$)	4.00	**0.00**	1.51	4.64	2.833	4.263	0.983	0.050
Differential ($2\|V\|$)	2.00	**0.00**	1.60	6.58	3.217	5.027	0.983	0.051
Canonical ($8\|V\|$)	8.00	**0.00**	1.66	4.12	2.243	3.184	0.988	0.040
Canonical ($2\|V\|$)	2.00	**0.00**	1.67	5.80	3.295	5.196	0.980	0.055
Border ($8\|V\|$)	8.01	0.01	1.69	6.79	3.201	5.295	0.984	0.041
Border ($4\|V\|$)	4.01	0.01	1.80	6.86	3.396	5.559	0.982	0.046
Differential ($1\|V\|$)	1.00	**0.00**	1.88	7.85	3.638	5.993	0.979	0.061
Portal ($2\|V\|$)	2.14	0.16	2.01	9.47	3.173	5.987	0.984	0.053
Boundary ($2\|V\|$, Area)	1.67	0.22	2.22	10.36	3.497	5.721	0.982	0.052
Manhattan	**0.00**	**0.00**	2.25	9.75	4.182	7.066	0.974	0.071
Boundary ($8\|V\|$, Portal)	4.88	1.54	2.34	9.65	3.051	5.230	0.987	0.035
Boundary (∞)	27.38	22.79	2.35	6.50	2.038	3.319	**0.993**	**0.027**
Boundary ($4\|V\|$, Portal)	3.01	0.53	2.35	9.63	3.394	5.665	0.984	0.041
Boundary ($2\|V\|$, Portal)	1.79	0.16	2.42	10.03	3.747	6.287	0.980	0.050
Boundary ($4\|V\|$, Area)	3.06	0.72	2.54	12.08	3.264	5.399	0.985	0.045
Portal ($4\|V\|$)	4.31	0.49	2.62	12.42	2.915	5.676	0.986	0.048
Portal ($8\|V\|$)	8.54	0.86	2.86	13.31	2.523	5.213	0.990	0.040
Boundary ($8\|V\|$, Area)	6.49	1.40	2.96	15.98	2.794	4.713	0.988	0.039
Gateway	11.08	9.07	3.51	11.92	3.804	6.299	0.977	0.060

Table 4. Empirical performance of different heuristics for rooms maps. The average path size on these maps is 371.27 nodes.

ROOMS MAP heuristic	memory ($\|V\|$)		time (ms)		rel. explored		rel. estimate	
	$\bar{x}$	s_N	$\bar{x}$	s_N	$\bar{x}$	s_N	$\bar{x}$	s_N
Boundary (∞)	17.62	12.85	**1.23**	**1.05**	**1.031**	**0.242**	**1.000**	**0.004**
Boundary ($8\|V\|$, Area)	6.59	2.24	1.91	2.74	1.519	1.149	0.997	0.014
Boundary ($4\|V\|$, Area)	3.69	0.65	2.83	4.22	2.138	1.726	0.994	0.023
Portal ($8\|V\|$)	8.06	0.08	2.90	4.26	2.116	1.759	0.994	0.023
Differential ($8\|V\|$)	8.00	**0.00**	4.60	6.36	3.289	2.656	0.965	0.073
Boundary ($2\|V\|$, Area)	1.97	0.12	4.79	6.24	3.474	2.592	0.982	0.043
Portal ($4\|V\|$)	4.05	0.05	4.84	6.69	3.337	2.660	0.983	0.041
Gateway	18.09	12.43	5.43	5.76	4.047	1.769	0.978	0.048
Boundary ($8\|V\|$, Portal)	5.22	1.73	6.42	7.55	3.426	2.070	0.989	0.020
Differential ($4\|V\|$)	4.00	**0.00**	6.75	10.10	4.786	4.324	0.933	0.117
Border ($8\|V\|$)	8.01	0.01	8.13	7.39	5.837	1.993	0.961	0.048
Border ($4\|V\|$)	4.00	**0.00**	8.65	7.95	6.247	2.265	0.953	0.058
Portal ($2\|V\|$)	2.02	0.02	8.91	10.98	5.926	4.040	0.947	0.085
Differential ($2\|V\|$)	2.00	**0.00**	11.18	16.73	7.585	7.214	0.874	0.181
Boundary ($4\|V\|$, Portal)	3.72	1.25	11.28	13.29	5.584	3.414	0.966	0.049
Canonical ($8\|V\|$)	8.00	**0.00**	11.63	16.05	5.077	4.063	0.917	0.123
Canonical ($4\|V\|$)	4.00	**0.00**	11.64	15.58	6.950	5.487	0.872	0.159
Boundary ($2\|V\|$, Portal)	3.25	1.58	14.00	14.70	7.235	3.202	0.943	0.061
Canonical ($2\|V\|$)	2.00	**0.00**	18.09	27.25	10.850	9.068	0.778	0.226
Differential ($1\|V\|$)	1.00	**0.00**	19.10	25.70	12.283	9.769	0.771	0.234
Manhattan	**0.00**	**0.00**	36.54	40.38	23.537	13.871	0.451	0.221

formance. Each individual area reduction outperforms the portal reduction variant in the same memory class. The $2|V|$ area reduction variant in particular seems to be a reasonable performing, low memory alternative.

The strategy of identifying areas in which the naive heuristic provides correct results does work: the Boundary Heuristic explores the least amount of cells on two of the four map types, ending second by small margins on the other two types.

The results for the Gateway Heuristic slightly differ from their original evaluation [2], which only used game maps. The original implementation reportedly performed better in terms of relative time, explored nodes and initial estimation, but was not implemented as a pure heuristic and used a weighted octile (eight way) movement. The differences are relatively small: the Gateway Heuristic would still end in the lower end of the results.

The Differential Heuristic and Canonical Heuristic are first evaluated in [7], both with $10|V|$ memory and octile movement. For maze maps, the Differential Heuristic performs better in their evaluation, but this seems mostly due to the higher memory setting. Their Canonical Heuristic performs better in terms of time used, but explores more nodes and has a lower estimation. This is most likely because of implementation differences: the original implementation did not go as far as taking the maximum of each pair of canonical states, but only took one. Taking the maximum results in a higher heuristic value and fewer explored cells, but makes the heuristic itself slower. However, even with in the faster variant, the Canonical Heuristic is still one of the slower heuristics on maze maps. On room maps, both the Differential Heuristic and the Canonical Heuristic perform worse by varying amounts in their original evaluation. For both map types, the same goes for [4], which provides the same results. The Border Heuristic is not evaluated for pathfinding by either.

The results are also compared to the results found in [5]. On room maps, their implementations of the Differential Heuristic and the Canonical Heuristic perform worse. The Canonical Heuristic is on the bottom end either way, but the Differential Heuristic appears to be one of the better heuristics (see Table 4). The Border Heuristic performs quite identical, both in terms of time and explored nodes. The Portal Heuristic also performs similar, despite the addition of several features and the resulting non pure heuristic. On maze map, all their heuristics have a better time performance (or their Manhattan distance performs worse). When comparing the number of explored nodes the results are similar: their Differential Heuristic performs slightly worse, the Border Heuristic slightly better. The Canonical Heuristic and the Portal Heuristic performs the same in both cases. The Gateway Heuristic explores less nodes on maze and room maps in their results, but comes with really high memory requirements. No time results are reported for the Gateway Heuristic on these maps. On game maps the performance is similar for both time and explored nodes.

6. CONCLUSIONS

In this paper the Boundary Heuristic was introduced to improve upon the currently existing heuristics. Of all the existing heuristics, the Portal Heuristic and Differential Heuristic perform best. The Portal Heuristic main weakness was assumed to be eager partitioning on open areas, which would explain the slower time performance on the *open with hills* maps. The Boundary Heuristic attempts to avoid this pitfall by identifying these areas.

In the evaluation, the Boundary Heuristic performed best on the maze and room maps. On these maps, the heuristic took the least amount of time and explored the least amount of nodes. The Boundary Heuristic has the best time performance on game maps, but explored slightly more nodes than the Portal Heuristic. While this heuristic outperforms other heuristics on most types of maps, it fails to avoid the weaknesses of the Portal Heuristic (which the Boundary Heuristic is based on) on the *open with hills* maps.

Identifying areas in which the naive heuristic, the Manhattan distance, performs correct results does result in the exploration of less nodes on most of the evaluated map types. Therefore, although not flawless, the Boundary Heuristic can be considered an improvement over the existing heuristics on the game, maze and room maps.

6.1 Future Work

The new Boundary Heuristic has only been tested on four-connected maps (allowing movement in four directions). To give better insights of the heuristic's strengths and weaknesses, tests should also be performed on eight-connected maps and more graph-like maps, such as a real road map.

All current approaches (including the Boundary Heuristic) seem to assume maps in games are static. This is not always the case, since some games feature terrain deformation. The current heuristics require to be rebuilt completely when the map changes: a heuristic that can be updated with map mutations is likely to be more efficient in such games.

A possible refinement of the area reduction algorithm could be to increase the maximum error of the naive heuristic. Currently, when partitioning, the partition stops when the naive Manhattan distance makes even a minimal error (maximum error of 0). Increasing the maximum error would likely result in larger areas and fewer portals.

The Differential Heuristic provides good results on the *open with hills* maps. Using the Differential Heuristic as the 'naive' heuristic in the Boundary Heuristic might give the best of both worlds. This approach might need to be combined with an increased maximum error, since the Differential Heuristic might not always behave like a perfect heuristic in open areas (though it performs very well).

7. REFERENCES

[1] Y. Björnsson, M. Enzenberger, R. Holte, and J. Schaeffer. Fringe search: beating A* at pathfinding on game maps. *IEEE CIG*, pages 125–132, 2005.

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

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

[4] A. Felner, N. Sturtevant, and J. Schaeffer. Abstraction-based heuristics with true distance computations. *SARA-09*, 2009.

[5] M. Goldenberg, A. Felner, N. Sturtevant, and J. Schaeffer. Portal-Based True-Distance Heuristics for Path Finding. *SOCS*, 2010.

[6] N. Pochter, A. Zohar, J. Rosenschein, and A. Felner. Search Space Reduction Using Swamp Hierarchies. *AAAI*, page 292, 2010.

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