Radboud University Nijmegen

Faculty of Science

The Influence of Graph Metrics on the Performance of Pathfinding Algorithms

Thesis BSc Computing Science

Author:
Hans Lous

Supervisor:
dr. Bernard Van Gastel

Second reader:
dr. Jurriaan Rot

August 2022

Abstract

Pathfinding algorithms have found many different applications. Route-planning and pathfinding in video games are two such applications with similar requirements but different strategies for achieving good performance. Speedup techniques for route-planning make use of a certain property of road networks: small highway dimension. This paper investigates whether other properties of graphs can help explain the variance in effectiveness of a number of pathfinding algorithms from route-planning and video game literature when they are used on different kinds of maps. We also investigate whether looking at these graph properties can help make a choice between two algorithms with similar performance. We find that graph properties have a significant influence on algorithm performance, but that the graph properties we chose cannot help choose between the similarly performing HPA* and contraction hierarchies.

Contents

1 Introduction

Context

There are many different applications for pathfinding algorithms, some examples being simulation (particularly video games), route-planning, robotics (also related are self-driving cars), and problem solving (problems may be reduced to finding the shortest path in a (weighted) graph). In many of these applications, pathfinding can become a major bottleneck for performance as the input gets larger. This has led to a large number of speedup techniques being developed for different applications, often making use of characteristics that are specific to the input of that particular field. For instance, many pathfinding algorithms from the route-planning field take advantage of the fact that - when travelling a large enough distance - you are likely to have to take a central highway. This specific observation has been formalized by [11], leading to the concept of highway dimension: a way to characterize a graph as 'easy' for route-planning algorithms. However, less is known about how to characterize a graph as 'easy' for algorithms from other fields. There is indication that this is possible, as [7] finds that there are significant differences between 'easy' and 'hard' maps for different algorithms.

Hence, there is clear indication that the 'shape' of the input graph affects the running time of pathfinding algorithms. This 'shape' can be described by a set of graph metrics: (numerical) properties of the graph that can be computed. One well-known graph metric is planarity (i.e. can the graph be drawn in such a way that no two edges cross?), however a graph metric can be as simple as the number of nodes in the graph. For a graph we can get an idea of its 'shape' by calculating the values of such metrics.

Goal

The goal of this paper is as follows: Explore to what extent the 'shape' of the graph, expressed though graph metrics, has a meaningful impact on the performance of pathfinding algorithms and find the best algorithm for a map based on the metrics. By investigating this question we want to provide an overview of the performance of a number of algorithms and see to what degree the map metrics affect the choice of algorithm.

Scope

We restrict this study to the following map metrics: the number of states, the longest path, the average degree, the difficulty, the dimension, and the highway dimension. Precise definitions of these metrics can be found in section 5. We then compare how these metrics affect the performance of the chosen algorithms. What the performance of an algorithm means can be unclear, so we restrict ourselves to the following measures of performance:

1. Running time, measured as the time it takes for a query to be answered.
2. Memory use, measured as total memory allocated during the algorithm's run.
3. Space use, measured as the amount of memory it takes to store the data required to answer queries, divided by the amount of memory it takes to store the graph. The precise definition varies based on the algorithm, section 6 explains how this metric is calculated for each of the algorithms.
4. Preprocessing time, measured as the time it takes for preprocessing to finish on a given graph.

Additionally, we restrict ourselves to CPU-based pathfinding algorithms, since massively parallel pathfinding algorithms are still in their infancy and have not seen any sort of widespread adoption yet. Finally, we also consider only informed search algorithms, as this is the most active area of research and most pathfinding algorithms are informed search algorithms. Specifically, we have compared HPA* [1] from the simulation literature and contraction hierarchies [2] and hub labels [3] from the route-planning literature. In addition, Dijkstra's algorithm and A* were also used to produce baseline measures. Since HPA* is made specifically for grid maps, the benchmarks

were performed on a set of grid maps taken from [4], which was chosen because it has a variety of different map types.

Contributions

We provide an overview of how the performance of the chosen pathfinding algorithms varies based on the chosen map metrics, in addition, we also show how these algorithms compare to each other. We also attempt to use the metrics to characterize maps on which one algorithm will outperform another and provide guidance on when to choose a certain algorithm, based on the metrics.

Approach

First, we established a set of benchmark problems, based on the benchmark problems from [5] and computed or collected the metric values for each of these problems. These should cover a wide range of map metrics. The algorithms were implemented in a custom graph framework in the C# language to allow them to be compared to each other. Benchmarks were then performed using the BenchmarkDotNet library [13]. The benchmarks were run on a home computer with hardware suited for gaming, see section 6 for specifications. Finally, the results were examined for any relationship between the map metrics and algorithm performance.

2 Background

Formally, the shortest path problem is defined as follows: given a weighted (di)graph $G = (V, E)$ and two nodes s and t find either the length of the shortest path between s and t (known as a distance query) or the shortest path between between s and t (known as a path query). This problem statement remains largely consistent between fields, but there are many auxiliary requirements that vary between fields. For instance, while the base search is one-to-one, searches can also be one-to-many, many-to-one, or many-to-many. In modern route-planning solutions, edge weights often change based on real time traffic data. In modern simulation applications, agents may need to avoid moving obstacles, coordinate with other agents to avoid collisions, or chase another agent instead of moving to a static goal. In robotics, agents may have a limited range of movement (for instance, the turn radius of wheeled vehicles) and factors such as acceleration and braking determine movement, turning the search space into a continuous 'field' as opposed to the discrete graph space. The following is a non-exhaustive list of some concrete requirements:

- Informed vs. uninformed: In an informed search, the entire search space and the location of the goal node are available from the start of the search. This contrasts with an uninformed search, where the algorithm must (blindly) 'explore' the search space to look for the goal.
- Heuristics: Certain algorithms (like the A* algorithm) use heuristics to speed up the search. It may not always be possible to define a proper heuristic for an application, and it is usually impossible for an uninformed search.
- Hardware: The restrictions and opportunities provided by hardware vary between applications: do we have just one processing core or multiple? Are we working with a massively parallel (GPU) computing system? How much memory do we have available? Do we want the lowest memory use or the fastest running time (or a bit of both)? Mobile systems tend to impose tight restrictions on hardware.
- Optimality: Path optimality (defined informally as the deviation of a path from the optimal path) may or may not be desired in certain applications. Academic route-planning applications tend to see 100% optimal paths as required (however, commercial route-planning applications often don't use 100% optimal paths [9]), whilst simulation applications tolerate a degree of suboptimality.
- Dynamism: how much the search space changes over time depends on the application. In video games the map might change when structures are built or destroyed, in route-planning

the travelling times change depending on traffic data and road construction may block certain nodes altogether. However, certain applications may feature more static search spaces that never change at all or only infrequently.

- Representation: certain algorithms may take advantage of underlying map representations (e.g. grid, polygons) beyond the abstract graph, or properties of the graph (e.g. planarity). This limits in what situations these algorithms can be applied. HPA*, for instance, relies on the map representation being a grid.

Beyond the criteria mentioned above, the 'shape' of the search space might vary greatly between applications as discussed earlier. For instance, the requirements for simulation and route-planning algorithms are similar: both are informed one-to-one searches for which a heuristic can be formulated, both have a degree of dynamism (e.g. traffic and road obstructions for route-planning, and mutable environments for video games), both allow for a degree of suboptimality, and both don't impose strict hardware requirements but do require that the system be able to answer queries in real-time (i.e. less than a 100 milliseconds, preferably even faster) on substantially large inputs. However, there are differences: In simulation applications, the input is often a grid, whilst in route-planning applications the input is a road network. These two graph types are different from each other in the following ways:

- Grids are typically planar, whilst road networks are often nonplanar.
- Grids have uniform edge weights - depending on the application - whilst road networks have varying edge weights: some edges (like those representing long stretches of highway or ferry rides) are long, whilst others (like those in the middle of a town) are shorter.
- Road networks are known to have a sort of 'hierarchical structure'. Informally, this means that a small number of 'highway nodes' are almost certainly going to be present on a shortest path of sufficient length. Contrast with grids, where shortest paths may or may not have significant overlap. This observation is formalized in the notion of 'highway dimension' by [11], see section 5.6 for a more detailed explanation.

These differences have had steered the development of algorithms for both applications: many route-planning algorithms make use (either directly or indirectly) of the concept of highway dimension, i.e. these algorithms are designed for graphs with a low highway dimension. Algorithms from the simulation literature tend to make use of graph abstraction [1, 12, 15] - constructing a simplified, abstract representation of the search space - or geometric [16] techniques.

3 Algorithms

3.1 HPA*

Hierarchical pathfinding A* (HPA*) [1] limits the search space by generating an abstract graph on top of the original graph and searching in the smaller abstract space for a shortest path. The preprocessing phase starts by dividing the grid into rectangles of $n \times m$ cells. It then scans the edges of these rectangles for sections where walkable cells belonging to different rectangles lie next to each other (these are known as 'doors'). Based on the length of the door, we choose either the middle cell or the two edge cells to represent the door in the abstract graph. Once the abstract graph has been populated with all the door nodes, two kinds of edges are added to the abstract graph:

1. Edges with weight 1 between adjacent doors belonging to different rectangles.
2. Edges between different doors of the same rectangle. The edge weight is calculated by performing a shortest path query between the two doors.

Once the first abstract graph is fully constructed, we can add additional layers of abstract graphs on top of it by 'merging' adjacent rectangles. This removes internal doors and keeps doors on the

edges of the new, larger rectangle. Note that the distances between doors in the larger rectangle need to be calculated again, but this can be done in the abstract graph that already exists.

Making a distance query then requires inserting the source node v and destination node u into the abstract graph(s). This is done by running a one-to-all distance query from the v to all door nodes in its rectangle to generate all the edge weights. Once the v and u have been inserted, we simply use A* to make a distance query between them in the highest layer abstract graph. Do note that this distance query will not always return the true distance between v and u. This is because the search will only move through the door nodes, which may not actually be on the shortest path between v and u. The size of the error thus depends strongly on the amount and placement of the door nodes. HPA* has quite a few hyperparameters to configure:

1. The width of the initial rectangles: w_a
2. The height of the initial rectangles: h_a
3. The number of abstract layers: n
4. The amount of rectangles to merge when moving up an abstract layer: m, representing that a $m \times m$ square of rectangles is to be merged. We only use 2 as the value for this hyperparameter.

Through some trial and error, reasonable formulas for these hyperparameters have been found. These formulas have been constructed such that the query time is low, whilst offering at least 90% accuracy for distance queries. The formulas are as follows, where w is the grid width and h is the grid height:

$$r = \min(w, h)$$

$$n = \begin{cases} 1 & r < 200 \\ 2 & 200 \leq r < 500 \\ 3 & \text{else} \end{cases}$$

$$w_a = \max\left(4, \left\lfloor \frac{w}{2\left\lfloor \frac{\sqrt{r}}{1.65} \right\rfloor} \right\rfloor\right)$$

$$h_a = \max\left(4, \left\lfloor \frac{h}{2\left\lfloor \frac{\sqrt{r}}{1.65} \right\rfloor} \right\rfloor\right)$$

3.2 Contraction hierarchies

The contraction hierarchies algorithm [2] (sometimes abbreviated as CH within this text) limits the search space by defining a total order on all the nodes in a graph. Shortcut edges might also have to be added to preserve shortest paths. Once the total order is established and the shortcut edges have been added, the query is simple: run a forward Dijkstra search from the source node and a backward Dijkstra search from the destination node. These Dijkstra searches only go 'up' in the order, i.e. the forward search only considers an edge (u, v) if $u < v$ in the total order, and the backward search only considers (u, v) if $u > v$.

Computing an optimal total order is an NP-complete problem, so we use heuristic techniques to compute a total order that performs adequately (even if it is non-optimal). This results in a procedure where we contract nodes one-by-one, with the total order defined as the order in which we contracted the nodes. Node contraction consists of removing a node u from the graph while preserving shortest paths that include u. Hence, for each path $P = \langle v, u, w \rangle$, we must check if it is the shortest path between v and w. If P is such a shortest path, we add a shortcut edge (v, w), with weight equal to the length of P.

In particular, nodes are assigned a priority value based on the following equation:

$$\texttt{priority}\,(v) = 2 \cdot ED(v) + CB(v) + 5 \cdot \texttt{level}\,(v)$$

And then contracted in order of increasing priority. Here $ED(v)$ is the edge difference of v, defined as the amount of shortcut edges that need to be added when v is contracted minus the degree of v (i.e. the net amount of edges added when v is contracted). Nodes that are part of a lot of shortest paths are considered more 'important', and should end up higher in the hierarchy. Consequently, these nodes will have a high edge difference and end up being contracted later. $CB(v)$ is defined as the amount of neighbors of v that have already been contracted (called the contracted neighbors heuristic). This term increases the priority of nodes in 'hot zones' where many other nodes are also contracted. The goal of including the contracted neighbors is to make the hierarchy more 'flat' (that is to say, evenly distributed over the graph). The level of a node v (denoted $\texttt{level}\,(v)$) is defined as follows:

$$\texttt{level}\,(v) = \begin{cases} 0 & \text{if } u > v \text{ for all } (u,v) \text{ and } (v,u) \text{ in } E \\ \texttt{level}\,(u) + 1 & \text{else, where } u = \max\{u \mid (v,u),(u,v) \in E\} \end{cases}$$

Note that the priority of a node might change after contraction of another node. Hence, before extracting the node with the lowest priority to be contracted next, we recalculate its priority first. If it is still the lowest, it will be contracted, otherwise we repeat this procedure for the new bottom node (this is known as lazy updating).

3.2.1 Contraction hierarchies with A*

The contraction hierarchies query algorithm can easily be altered to use the A* algorithm instead of Dijkstra's algorithm for the forward and reverse searches. This offers a significant speedup in certain situations. If contraction hierarchies is used in this way, it will be referred to as "contraction hierarchies with A*" or abbreviated as 'CHH' (contraction hierarchies heuristic) within this text.

3.3 Hub labels

Hub labelling [3] is based on assigning a forward and reverse label to each node v. The forward label $LB_f(v)$ is a set of nodes along with all the distances from v to those nodes. The reverse label $LB_r(v)$ is a set of nodes along with all the distances from those nodes to v. The labels of a graph must have the cover property: for every two nodes v and u, $LB_f(v) \cap LB_r(u)$ contains a node on the shortest path between v and u. A labelling with the cover property is called valid. We generate a labelling based on a node ordering generated by CH. The forward label of a node v is all nodes visited by a forward search with source v, and the reverse label is all those visited in a backwards search with source v. It is easy to see that this labelling is valid, after all if it wasn't a valid labelling, there would be nodes v and u for which $(LB_f(v) \cap LB_r(u)) \cap P(v,u) = \emptyset$ (here $P(v,u)$ denotes the shortest path between u and v). However, this would mean that a contraction hierarchies query from v to u wouldn't find the shortest path. Hence, if the underlying contraction hierarchies implementation is correct, the generated labelling will be valid. Note however, that not all nodes contained in the labels generated this way are necessary and some nodes could be removed from the label. In particular, if for a node u the distance d' found during the forward search is greater than the true distance $d(v,u)$, we can exclude u from the forward label of v (idem dito for backward labels). Computing the true distance checking the labels for redundant nodes is called a 'check query'. In our case, contraction hierarchies was used for the check queries but other choices are also possible.

Once the labels are generated, making a distance query is simple: for source node v and destination node u simply return:

$$\min\{d(v,w) + d(w,u) \mid w \in LB_f(v) \cap LB_r(u)\}$$

4 Literature

4.1 Simulation

Zarembo et al [6] compare BFS, Dijkstra's algorithm, A*, HPA* and LPA*. Their grids differ from ours, since they don't allow for diagonal motion between grid cells. This permits the use of the Manhattan distance as a heuristic, which has the advantage of being substantially cheaper to calculate than the Euclidean distance, since the Euclidean distance requires computing a square root. We did not use the Manhattan distance since we allow diagonal motion through the grid - this causes the Manhattan distance to overestimate true distances, making it an inadmissible heuristic. The hyperparameters for HPA* were chosen as follows:

$$w_a = h_a = 4 \qquad n = 3 \qquad m = 2$$

The benchmarks were conducted on randomly generated grids, where 20% of the cells were randomly filled in with walls. This corresponds to the random-20 category of maps in our benchmark set. The sizes of the randomly generated maps in their benchmark set ranged from 64×64 to 1024×1024.

Noori et al [8] compare Dijkstra, A*, HPA* and a number of other algorithms that aren't relevant for us. The benchmarks were performed on 64×64 grids with either 10% or 50% of the grid randomly filled in with walls. Diagonal motion was allowed, but Manhattan distance was used as a heuristic, which casts doubt on the validity of the results. In addition, the hyperparameters used for HPA* are not specified, which makes it much harder to interpret the results found by this paper.

Sturtevant et al [7] compare A*, A* with improved heuristics, sector abstraction and contraction hierarchies. Sector abstraction is a pathfinding algorithm that was developed in a company setting for a video game, which is similar to HPA*. The sector sizes vary between 8×8 and 16×16 based on the map size (however, only 8×8 sectors were used in the benchmarks) and two layers of sectors where used (a 2×2, 3×3 or 4×4 square of sectors is merged). Two versions of of the contraction hierarchies algorithm were used: a faster version that used more memory, and a slower version that used less memory. The slower version uses a simple contraction heuristic: it is based on a single term - called σ - defined as the amount of 'real' edges that will be merged into shortcut edges when a node is contracted. The faster version uses the following heuristic:

$$25 \cdot ED(v) + 60 \cdot \sigma(v) + 20 \cdot \texttt{level}\,(v) + 2 \cdot CB(v)$$

Benchmarks were only performed using the slower version, as the full version used too much memory. In addition, a distinction is made between the 'full' and 'mobile' versions of the algorithm: the mobile version uses a compression scheme to significantly reduce memory usage but which degrades runtime performance. The benchmarks were performed on maps from the game Baldur's Gate, scaled to size 512×512.

Table 1 gives an overview of the runtime results of each paper. The HPA* listing for [7] gives the runtimes for 16×16 sector abstraction, since that represents the closest analog for the HPA* hyperparameters used in [6]. Results from [6] are as expected: A* is significantly faster than Dijkstra, and HPA* is significantly faster than both Dijkstra and A*. Note that the speedup for HPA* over Dijkstra is about 2 for the small 64×64 map listed in the table. This speedup improves monotonically to about 3 for the largest 1024×1024 map. The speedup for A* remains at about 2 for all map sizes. There aren't many interesting results to report from [8], other than the fact that A* had higher reported runtimes than Dijkstra for 10% filled random maps, which is a surprising result: A* tends to perform better on open maps, since the heuristic gives a better estimation of the true distance in those cases. The degradation in performance observed may be connected to the erroneous use of the Manhattan distance as a heuristic, since it gives incorrect estimates of the distance when diagonal motion is allowed. Besides that, [8] finds about a 2 times speedup for HPA*.

Paper	Algorithm	Runtime (ms)
[6]	Dijkstra (64×64)	6.00
	A* (64×64)	4.00
	HPA* (64×64)	3.00
	A* (512×512)	265.00
	HPA* (512×512)	190.00
[7]	A* (512×512)	0.23
	HPA* (512×512)	0.08
	CH (high memory)	0.04
	CH (low memory)	0.06
[8]	Dijkstra (64×64)	1.89
	A* (64×64)	1.96
	HPA* (64×64)	1.11

Table 1: Runtime found for Dijkstra, A*, HPA* (or HPA* analog) and CH by various other papers

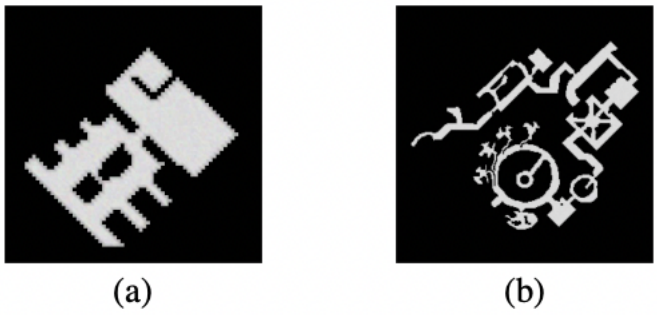


Figure 8: Map (a) is easy for abstraction but hard for CHs.
Map (b) is hard for abstraction but easier for CHs.

Figure 1: (taken directly from [7]) Hard and easy maps for sector abstraction and contraction hierarchies

The most interesting results come from [7]: it finds that, on average, contraction hierarchies is slower than sector abstraction. However, when looking at longer queries (i.e. queries where the true distance is larger), contraction hierarchies is faster than sector abstraction. This difference is illustrated by what maps are more difficult for each algorithm: maps where more paths are long are more difficult for sector abstraction, while maps where more paths are short are more difficult for contraction hierarchies (figure 1 gives two example maps). This finding is what inspired this study: among other things, we want quantify this notion that there are certain situations in which contraction hierarchies (or some other algorithm) performs better.

4.2 Route planning

Citations [9] and [10] are two meta-analyses of papers from the route-planning literature. The most important conclusion to draw from these papers is the sheer efficiency of contraction hierarchies and hub labels: [9] reports a speedup of 41 thousands times (relative to Dijkstra's algorithm) for contraction hierarchies and [10] reports a speedup of anywhere from 14 thousand to 22 thousands times for contraction hierarchies and 1.5 to 7.3 million times for hub labels. This is countered by the preprocessing times of these algorithms: [9] gives a preprocessing time of 32 minutes for contraction hierarches, and [10] anywhere from 2 to 10 minutes for contraction hierarchies and anywhere from 10.5 to 1200 minutes (20 hours) for hub labels.

Abraham et al [11] - in an attempt to explain why route-planning algorithms are so effective on road networks - introduce the concept of highway dimension and provide bounds on the amount of work done by theoretical versions of several route-planning algorithms based on the highway dimension: for contraction hierarchies queries take $O((h \log D)^2)$ time and hub label queries take $O(h \log D)$ time. Here, h is the highway dimension and D is the diameter of the input graph, i.e. $\max\{d(v,u) \mid v,u \in V\}$. These bounds are based on undirected graphs, but [11] mentions that the results could be extended to work for directed graphs with some extra work. An explanation is also provided for why road networks tend to have relatively low highway dimension: a model that is meant to approximate the formation of a road network is defined, and it is shown that graphs produced under this model have constant highway dimension.

Storandt [12] examined the performance of contraction hierarchies in a completely different setting: grid maps. The preprocessing phase of the algorithm was modified to work better on grid maps, both in terms of computation time and the quality of the node order. During preprocessing, a modified version of the edge difference was used as the heuristic value. Benchmarks where performed on the maps in [5], both for query time and preprocessing time. For mazes, a speedup of 198 was observed and a speedup of 67 was observed for room maps (see section 6 for an explanation of these map types). While a smaller speedup than for road networks, this is still significant. However, speedups observed for random and game maps were only 10 and 7, respectively. Averaged over all map types, the speedup comes out to about 50. This suggests that, while contraction hierarchies is still effective on grid maps, it offers a smaller advantage for this map type. As for preprocessing, the observed time was around 10 seconds for all map types - again less than on road networks.

5 Metrics

5.1 Number of states

This is a simple metric, defined as the number of nodes in the graph. Thus, this metric indicates the size of the search space the algorithm must comb through to find the destination node, but doesn't give any indication about the shape of that search space. Nonetheless, this metric could have a significant impact on the performance of an algorithm. A search by Dijkstra's algorithm, for example, will expand in a sort of bubble until the bubble contains the destination node. In this case, more room for the bubble to grow (because the graph has more nodes) will mean a longer running time. The A* algorithm should be affected less, since it actively directs the search towards the goal. Do note however that A* has worst case graphs where it expands the same amount of nodes as Dijkstra's algorithm.

5.2 Longest path

The longest path (also known as the diameter D of a graph) is defined as the greatest distance (i.e. the longest shortest path) between any pair of nodes in the graph, and it is estimated by running n Dijkstra searches and taking the maximum across all distances found. In our case, n was 1 million. On grid maps, the longest path metric tells us something about the 'windiness' of the map. Here, by windiness, we mean the average 'directness' by which a path leads from a given source node to a destination node. For instance, paths in an open map tend to be more direct than those in a maze.

5.3 Average degree

The average degree indicates the density of the graph. On grid maps, the degree of nodes is any value from 0 to 8, but on arbitrary graphs the degree can range from 0 to $|V|$. On grid maps, this metric says something about the 'openness' of the map: open areas contain many nodes of

degree eight, as no neighbors are walls, hence the greater the average degree, the larger and more open areas the map contains.

5.4 Dimension

The dimension metric is another way to indicate the 'openness' of a map and was taken from [4]. We estimate it as follows: run n Dijkstra searches from random locations on the map. During these searches we keep track of the depth of each node, i.e. its distance (in number of nodes) from the source. We then fit the number of nodes at each depth to the polynomial $a + bx + cx^2$. We estimate the dimension as the value of c. Since the amount of nodes at each depth tends to decrease as the search comes closer to its end, we only consider the first half of the search.

5.5 Difficulty

This metric gives some indication as to how well 'basic' searches perform, and how much of a help the Euclidean distance heuristic is. It is based on the 'nodes expanded' and 'heuristic accuracy' metrics from [4]. It is defined as the number of nodes expanded by A* during a search divided by the number of nodes in the path. We average this over different path lengths by going through all the buckets in a scenario. Note that this metric is only defined for maps where the Euclidean distance heuristic can be applied.

5.6 Highway dimension

We draw our definition of highway dimension from [11]:

> The highway dimension h of a graph $G = (V, E)$ is the smallest integer such that:
> for all $v \in V$ and $r > 0$ there exists a set $H \subseteq V$ such that:
> $|\, H \,| \leq h$ and $H \cap P \neq \emptyset$ for all paths P that are $(r, 2r)$-close to v

Here, a path P being (r, d)-close to a node v means two things:

1. P is r-significant, which is to say that it has an r-witness path P'. An r-witness path P' of a path P is P extended with at most one node at each end such that the length of P' is greater than r.
2. The r-witness path P' has distance at most d from v. The distance between a path P and a node v is defined as $\min\{d(v, w) \mid w \in P\}$.

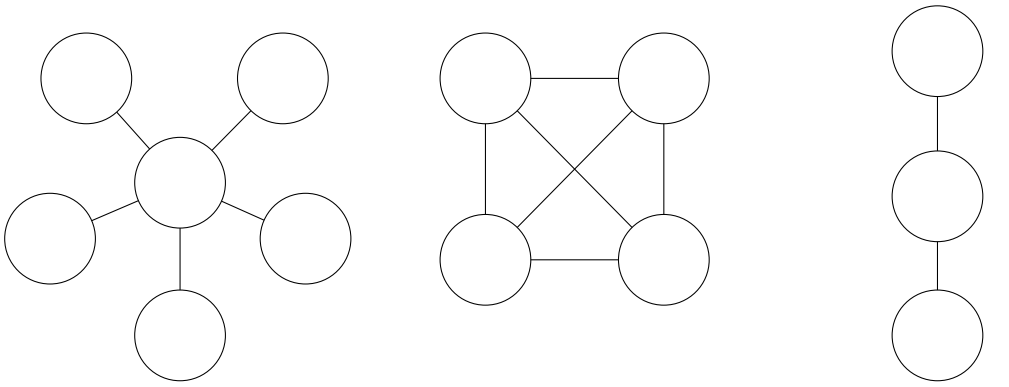
In this definition of highway dimension, H represents a 'hub set': a set of 'important' nodes that are on all shortest paths for a node v, at scale r. The scale component is important, as v might have a small H at small scales, but large H at larger scale, or vice versa. The maximum size of such a hub set is taken as the highway dimension of the graph. The highway dimension of a graph, then, gives some indication of how 'hub-based' the graph is, i.e. a low highway dimension indicates that a small group of nodes are on many of the shortest paths at each scale.

5.6.1 Theoretical examples

The definition of highway dimension can be hard to parse without concrete examples. This section will provide values for the highway dimension of a number of graph types to clarify the concept. Figure 2 gives some examples of these graph types.

The star graph S_n is defined as a center node connected with $n - 1$ leaf nodes. It can also be seen as a tree graph with branching factor $n - 1$ and depth 1. The highway dimension of S_n is always equal to n, as follows: first, note that all paths fall into one of three categories:

1. The path consists of a single node from the graph, call these paths $\mathcal{P}_0$. These paths all have length 0.



Star graph S_6 | Complete graph on 4 nodes | Line graph L_3

Figure 2: Several different types of graph

2. The path consists of the center node and a leaf node, call these paths $\mathcal{P}_1$. These paths all have length 1.
3. The path consists of a leaf node, the center node and another leaf node, call these paths $\mathcal{P}_2$. These are the longest paths possible and all have length 2.

If we now take the center node as v and $r = 0.9$, the set of all paths that are $(0.9, 1.8)$-close to v is all paths: all paths $P \in \mathcal{P}_0$ are 0.9-significant, since they can be extended with the center node to make a witness path P' of length 1 - since this witness path includes the center node v $d(v, P') = 0$ it is $(0.9, 1.8)$-close to v. For $P \in \mathcal{P}_1 \cup \mathcal{P}_2$ we have that the length of P is ≥ 1 and that they include v, hence they are $(0.9, 1.8)$-close to v. Now, we have to construct the set H that 'hits' all of these paths. Since H needs to hit all $P \in \mathcal{P}_0$, we have $H = \mathcal{P}_0 = V$. Hence, $|\, H \,| = n$. We know that this must be the highway dimension, since the highway dimension cannot be greater than the number of nodes in the graph. Note that the same holds for the complete graph on n nodes, since the star graph S_n is embedded in this graph.

Next we will examine the line graph L_n, defined simply as a 'line' of n nodes, each connected to the previous and next in the line. A closely related graph is the cycle graph C_n, in which the 'head' and 'tail' nodes of the line graph are connected, such that the graph forms a closed circle of n nodes. We will see that for $n \leq 5$ the highway dimension of L_n and C_n is n, while for $n > 5$ it is always 5. Take the line graph L_5, with v the middle node of the line, and $r = 0.9$. Once again, all paths are $(0.9, 1.8)$-close to v. This means that $H = V$ and thus that the highway dimension of L_5 is 5. Note that if we make the line shorter, we're simply removing one of the singleton paths from the set of paths that need to be hit, hence $H = V$ will also hold for $n < 5$. If we make the line longer by adding a node w, we are adding a singleton path, but it will not be $(0.9, 1.8)$-close to the chosen v (see figure 3). Thus, w will not be added to H, so $|\, H \,| = 5$ still holds. Note that this H is indeed the largest H possible - if we take $r = 1.1$, the set of all paths that are $(1.1, 2.2)$-close to v becomes all paths except the singleton paths. We could make an H that hits all of these easily: for instance, take H to be the set containing w and the neighbors of v. Now we have $|\, H \,| = 2$. Hence, adding a node does not increase the highway dimension of a line graph. Note that the same holds for cycle graphs, since any node we pick will be essentially the 'middle' of a line.

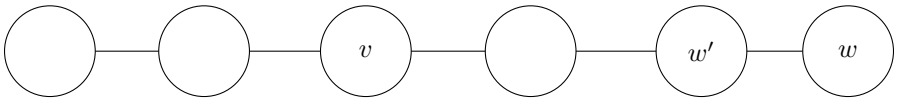


Figure 3: $d(v, w') = 2 > 1.8$, hence $P = \langle w \rangle$ with 0.9-witness $P' = \langle w, w' \rangle$ is not $(0.9, 1.8)$-close to v

Graph	F_2	F_3	F_4	F_5	F_6	F_7
HD	4	9	11	13	18	24

Table 2: Highway dimension for some field graphs

The closest analog to the maps used for benchmarking is the field graph F_n, consisting of a $n \times n$ square 'net' of nodes. See figure 4(a) for an example. The highway dimension of these graphs is harder to precisely determine theoretically, especially as n grows larger. One way to determine the highway dimension of field graphs is to look at the $(r, 2r)$-neighborhoods for low values of r. We define the $(r, 2r)$-neighborhood of a node v as the set of all paths P that are $(r, 2r)$-close to v. See figure 4(b), 4(c) and 4(d) for some 'maximal' neighborhoods. We can construct a hitting set H such that $H \cap P \neq \emptyset$ for all P in a neighborhood, such that $| H |$ gives the 'highway dimension' of that neighborhood. In the figure, red nodes represent nodes that are in such an H. Overlaying these neighbors on a field graph F_n and counting the maximum number of red nodes possible will give some idea of the highway dimension (see table 2).

5.6.2 Estimating highway dimension

Algorithm 1 describes a procedure for estimating the highway dimension. This procedure is based on the definitions and theory of sparse hitting sets in [21], specifically definition 4.1 and theorem 4.2. Note that it is possible for this procedure to both over- and underestimate the true highway dimension of the given graph. The highway dimension can be underestimated in two ways: either the procedure doesn't pick enough paths based on the nodes, which causes an undersized hitting set, or the procedure doesn't investigate the right scale, as it only looks at scales m_a, m_i and m_v. Overestimation is possible due to the greedy generation of minimal hitting sets, which might cause an oversized hitting set. Despite these limitations, the procedure seems to give reasonable estimates of the highway dimension (see table 3). As can be seen in the table, the procedure tends to underestimate, and not overestimate, the highway dimension on field graphs.

Graph	Theoretical HD	Estimated HD
S_{10}	10	10
S_{100}	100	100
L_2	2	2
L_4	4	4
L_5	5	5
L_6	5	5
L_{10}	5	5
L_{100}	5	5
F_2	4	4
F_3	9	9
F_4	11	11
F_5	13	13
F_6	18	13
F_7	24	14

Table 3: Theoretical and estimated (with $n = 100$) values of highway dimension for a variety of graphs

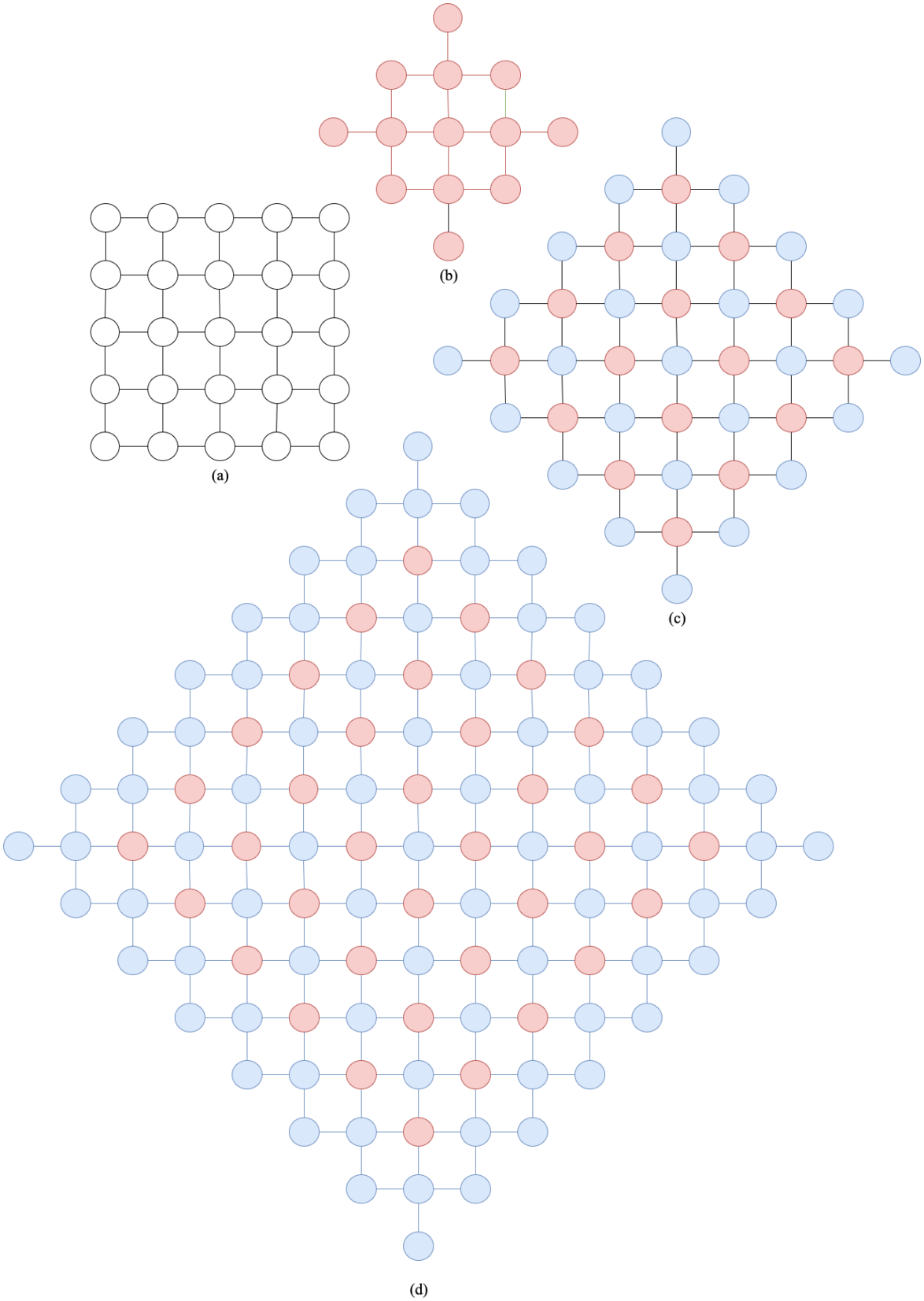


Figure 4: (a): field graph F_5. (b): $(0.9, 1.8)$-neighborhood. (c): $(1.1, 2.2)$-neighborhood. (d): $(2.1, 4.2)$-neighborhood.

Algorithm 1 Estimate Highway Dimension

procedure HIGHWAYDIMENSION(V, E, n)
  $D \leftarrow$ array of distance arrays of length n
  $\Pi \leftarrow$ array of shortest path trees of length n
  for i **from** 1 **to** n **do** ▷ generate n distance arrays and shortest path trees
    $v \xleftarrow{\$} V$ ▷ chosen without replacement
    $D[i], \Pi[i] \leftarrow$ DIJKSTRA(V, E, v)
  end for
  $m_a \leftarrow \max\{\max\{d[v] \mid v \in V\} \mid d \in D\}$
  $m_i \leftarrow \min\{\min\{d[v] \mid v \in V\} \mid d \in D\}$
  $m_v \leftarrow \frac{m_a + m_i}{2}$
  for i **from** 1 **to** n **do**
    Trace 100 randomly chosen paths from $\Pi[i]$ ▷ chosen without replacement
    Of those, add m_a-significant paths to $\mathcal{P}_a$
    Of those, add m_v-significant paths to $\mathcal{P}_v$
    Of those, add m_i-significant paths to $\mathcal{P}_i$
  end for
  $S_a \leftarrow$ MINHITTINGSET$(\mathcal{P}_a)$ ▷ generated by a basic greedy algorithm
  $S_v \leftarrow$ MINHITTINGSET$(\mathcal{P}_v)$
  $S_i \leftarrow$ MINHITTINGSET$(\mathcal{P}_i)$
  HD $\leftarrow 0$
  for each chosen v **do**
    HD $\leftarrow \max(\text{HD}, |\, B_{m_a}(v) \cap S_a \,|)$
    HD $\leftarrow \max(\text{HD}, |\, B_{m_v}(v) \cap S_v \,|)$
    HD $\leftarrow \max(\text{HD}, |\, B_{m_i}(v) \cap S_i \,|)$
  end for
  return HD
end procedure

6 Methods

The benchmark maps were taken from [5]. These maps are divided into categories: there are 17 types of 'artificial' map and a number of maps that are taken from different video games. The artificial maps are all 512×512 maps. This was too big to allow for timely preprocessing for contraction hierarchies and hub labels, so we reduced their size to 80×80 by taking the top-left 80×80 square as the new map. This leaves the following different map types:

1. 8rooms and 16rooms: 8×8 or 16×16 square rooms divided by walls. Holes are randomly placed in the walls between rooms to act as doors.
2. maze-1, maze-2, maze-4 and maze-8: mazes with corridors of width 1, 2, 4 or 8.
3. random-10, random-15, random-20, random-25, random-30, random-35: empty maps where 10% to 35% (in increments of 5%) of the map is randomly turned into a wall.
4. game: maps that where taken from video games (referred to as 'misc' in some figures)

Where possible, map metrics where taken from [5], otherwise they where calculated according to the definitions and methods given in section 5. In order to be able to compare the runtimes of the algorithms to each other, custom implementations of the algorithms were produced, rather than using pre-made implementations. The implementation was done in C# and benchmarks were performed using the BenchmarkDotNet library [13]. For non-preprocessing benchmarks, the preprocessing was done beforehand and the result was serialized to a file. Before running the benchmarks, the data was deserialized again. The benchmarks were run on a machine with an AMD Ryzen 5 5600x hexacore processor running at 3.7 GHz, 32 GB of DDR4 memory running at 2133 MHz, on the Windows 10 version 20H2 operating system. The query benchmarks consisted of 10 queries with a true length between 100 and 104. Benchmarks were run on all maps for Dijkstra, A* and HPA*, on maps with grid size at most 420×420 for contraction hierarchies and on maps with grid size at most 140×140 for hub labels. Preprocessing was also benchmarked for HPA*, contraction hierarchies and hub labels. Preprocessing for contraction hierarchies and especially hub labels took too long on game maps, hence it was only benchmarked on the non-game maps. Preprocessing was benchmarked on all maps for HPA*.

The method for calculating the space use varied per algorithm. For HPA*, the space use is measured as the sum of the amount of space taken up by all the abstract graphs and the input graph, divided by the size of the input graph. We calculate the space taken up by a graph as follows: for each edge, 1 unit of memory is required to store the source node, 1 unit for the destination node and 2 units for the weight of the edge. This corresponds to using 32-bit integers for the node IDs and a double precision (64-bit) floating point number for the edge weight. The same method can be applied to contraction hierarchies: in our case, we discarded the input graph and stored the results of the preprocessing as two graphs: one up-graph where the edges only go up in the node order, and one down-graph where the edges only go down in the node order. The space use of contraction hierarchies is then the space required to store these graphs divided by the space required for the input graph. For hub labels, there is no graph required to make queries, only the labels. The size of a label is calculated as the amount of label entries times three: one unit to store the node ID (a 32-bit integer) and two units to store the distance to that node (a double precision floating point number). The space use is then defined as the sum of the sizes of all the labels, divided by the space use for the input graph.

7 Results

7.1 Overview

The average values for the performance metrics for each of the algorithms can be found in table 4. The preprocessing for contraction hierarchies and hub labels took too long for them to be

Algorithm	Time (μs)	Memory (B)	Space	Preprocessing (ms)	Speedup
Dijkstra	15,629.46	15,054,816.49	1.00	-	1.00
A*	9,258.79	10,003,550.52	1.00	-	1.69
HPA*	1,743.04	719,147.32	1.15	10.20	8.97
CH	1,369.59	3,081,200.25	1.62	2,649.60	11.41
CH with A*	1,079.76	2,992,474.58	1.62	2,649.60	14.47
HL	8.43	56.00	11.04	103,537.52	1,853.22

Table 4: Average running time, memory use, speedup and preprocessing time with respect to Dijkstra for each algorithm

Algorithm	Time (μs)	Memory (B)	Space	Preprocessing (ms)	Speedup
Dijkstra	27,100.80	35,367,098.18	1.00	-	1.00
A*	16,600.77	30,727,261.09	1.00	-	1.63
HPA*	3,487.06	1,019,904.00	1.21	10.91	7.77
CH	3,481.26	4,575,604.36	1.94	2,873.40	7.78
CH with A*	1,969.49	3,431,703.27	1.94	2,873.40	13.76
HL	20.94	56.00	21.62	112,909.29	1,294.51

Table 5: Average running time, memory use and speedup with respect to Dijkstra for each algorithm for the top 10 most difficult maps

run on some larger maps, hence these results are averages over the maps on which all of the algorithms were run. A straightforward summary of the results would be thus: hub labels has the lowest running time and memory use of all the algorithms, but also the highest space use and preprocessing time. Contraction hierarchies (with and without A*) and HPA* exhibit lower running times and memory use than both Dijkstra and A*. Finally, contraction hierarchies is about 1.27 times faster than HPA*, but uses more than four times the memory and 1.4 times the space and has higher preprocessing time. The same holds for contraction hierarchies with A*, except that it is 1.62 times faster. Looking at the hardest (table 5) and easiest (table 6) top 10's, we see that as the maps get harder, the difference in running time between HPA* and contraction hierarchies shrinks, although the same does not hold for HPA* and contraction hierarchies with A*: the latter is about 1.77 times faster on the most difficult maps. We can also see that the effectiveness of using A* with contraction hierarchies shrinks on the easier maps, where using A* makes no significant difference. For hub labels, the space use is lower for easier maps and higher for harder maps, however this does not hold for contraction hierarchies or HPA*.

7.2 Map metrics

The number of states, longest path and dimension metrics are taken from [4], the difficulty metric is meant to combine the 'nodes expanded' and 'heuristic accuracy' metrics from this paper. The highway dimension is defined in [11]. Table 7 shows the median values for each of the metrics per

Algorithm	Time (μs)	Memory (B)	Space	Preprocessing (ms)	Speedup
Dijkstra	1,230.18	2,708,107.64	1.00	-	1.00
A*	1,011.70	2,542,871.27	1.00	-	1.22
HPA*	222.48	341,085.09	1.25	2.72	5.53
CH	71.08	2,097,152.00	1.76	200.60	17.31
CH with A*	74.86	2,097,152.00	1.76	200.60	16.43
HL	1.65	56.00	5.80	1,802.60	744.62

Table 6: Average running time, memory use and speedup with respect to Dijkstra for each algorithm for the top 10 easiest maps

Map type	States	L. path	Avg. degree	Difficulty	Dimension	Highway dim.
8room	5065.00	136.20	6.28	23.61	0.66	17.00
16room	5665.50	139.14	7.19	29.97	0.62	10.50
maze-1	3179.00	663.50	1.99	2.78	0.00	5.00
maze-2	4214.00	499.34	4.92	6.76	0.00	9.50
maze-4	5084.00	287.70	6.46	13.35	0.00	11.50
maze-8	5646.50	224.56	7.21	22.24	0.01	11.50
random-10	5742.00	119.10	6.38	20.13	0.97	94.00
random-15	5448.50	123.23	5.79	21.04	0.87	71.00
random-20	5105.00	127.62	5.18	20.42	0.78	47.00
random-25	4742.50	134.19	4.60	20.33	0.66	35.00
random-30	4393.00	144.96	4.16	18.84	0.51	22.00
random-35	3903.50	166.13	3.76	16.35	0.35	12.50
game	64426.00	619.55	7.65	59.32	0.31	-

Table 7: Median values for each of the metrics, categorized by map type

map type. The highway dimension was not feasible to estimate on large maps due to memory use, hence it has only been calculated for the non-game and some of the smaller game maps on which contraction hierarchies could be run. For the 'longest path' and 'dimension' metrics we find similar results to those found in [4]: mazes have long paths and dimension 0, and the longest paths for rooms and random maps are similar. However, the values for the 'dimension' metric we found are significantly lower than those mentioned in [4]. This might be a result of the maps being scaled down. For the 'difficulty' metric, we see that mazes with low corridor size are easy, while random maps and larger corridor mazes have middling difficulty, with rooms being the hardest. We find that mazes have low highway dimension, and that it increases with corridor size. Rooms also have low highway dimension and the highway dimension decreases as the rooms get bigger. This makes sense, since bigger rooms means less rooms and thus less doors. Since the doors act as choke points, with many shortest paths going through them, less doors will mean a lower highway dimension. Random maps have high highway dimension, which decreases sharply as the obstruction ratio increases.

In the following sections, we will illustrate how the different metrics relate to the running time, memory use, space use and preprocessing time of each algorithm. The values for the metrics have been normalized to make it easier to compare and combine them.

7.2.1 Dijkstra

Dijkstra shows the strongest correlation overall with difficulty (Pearson correlation coefficient is 0.78). We can combine different metrics to get a cleaner relationship. The strongest overall correlation is with the following combination of metrics:

$$\text{difficulty} + 0.11 \cdot \text{dimension} - 0.6 \cdot \text{states} + 0.2 \cdot \text{average degree}$$

This gives a correlation of 0.87 overall, and 0.92 for non-game maps. Figure 5 shows a scatter plot comparing the runtime to this combination of metrics. For game maps, this relation is still messy, with a correlation coefficient of 0.78. A different combination of metrics gives a cleaner relation for game maps:

$$\text{difficulty} + \text{dimension} + 15 \cdot \text{average degree}$$

Figure 6 shows a scatter plot illustrating this relation. This gives a good correlation for game maps of 0.92, but only a 0.71 correlation overall.

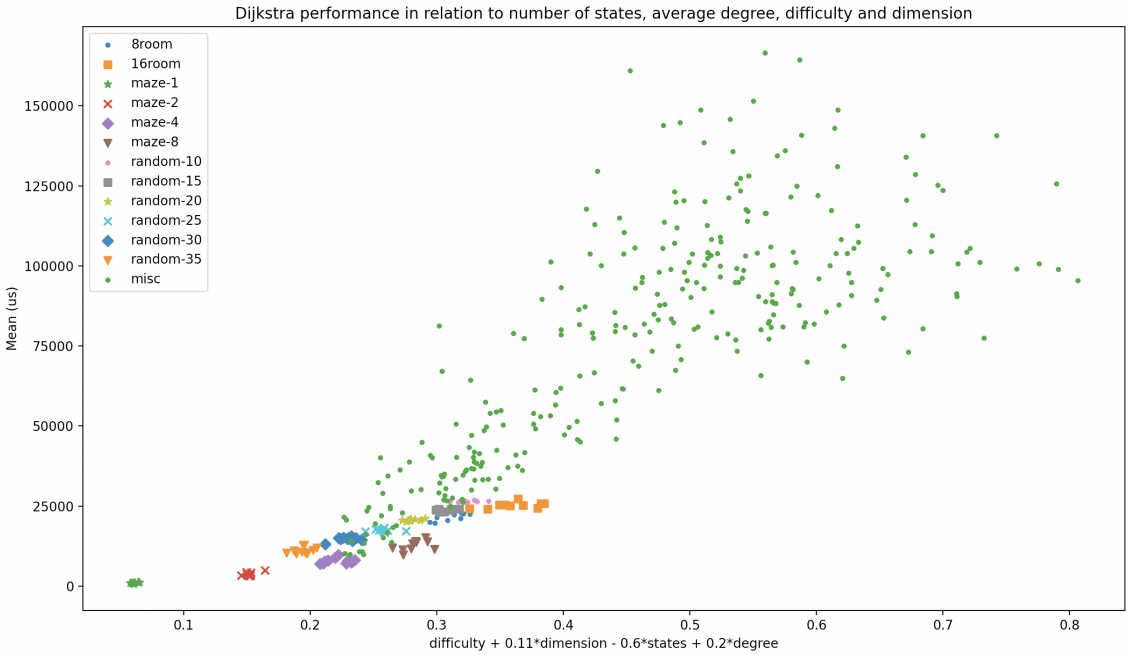


Figure 5: Dijkstra performance in relation to difficulty + 0.11 · dimension − 0.6 · states + 0.2 · average degree

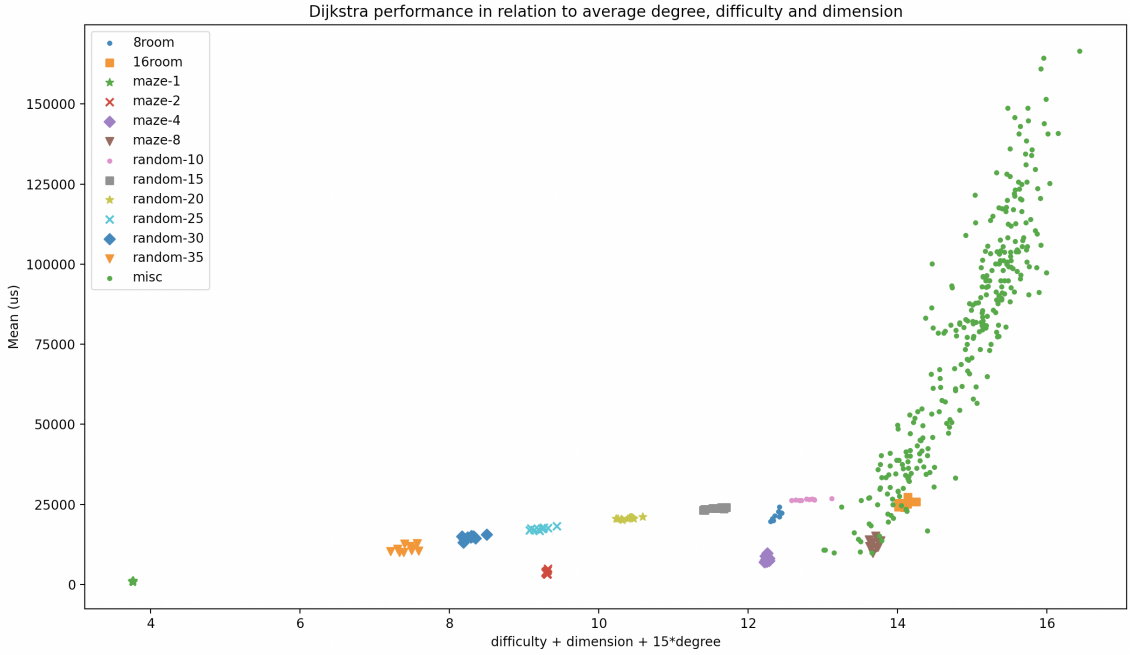


Figure 6: Dijkstra performance in relation to difficulty + dimension + 15 · average degree

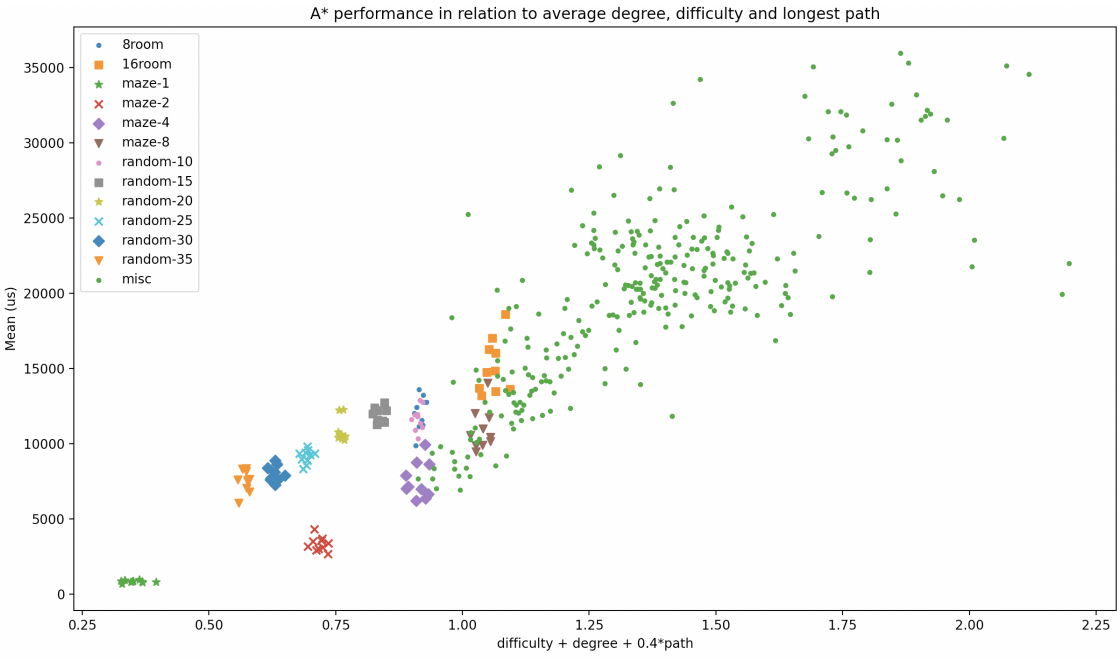


Figure 7: A* performance in relation to difficulty + average degree + $0.4 \cdot$ longest path

7.2.2 A*

A* shows the strongest correlation with difficulty: a Pearson correlation coefficient of 0.81 overall, and 0.94 for non-game maps. This is to be expected, since the difficulty metric is based directly on A*. An overall correlation of 0.88 is achieved by the following combination of metrics:

$$\text{difficulty} + \text{average degree} + 0.4 \cdot \text{longest path}$$

Figure 7 shows a scatter plot illustrating this relationship. Adding these extra terms, however, does reduce the correlation coefficient for non-game maps to 0.79. The main reason for this is the maze maps: these have long paths and A* performs well on them. Hence, the longest path term inflates their metric-value to higher than it should be.

7.2.3 HPA*

HPA* has the strongest correlation with difficulty (correlation coefficient 0.89), but the cleanest relationship is with the following combination of difficulty and dimension (correlation coefficient 0.91):

$$\text{difficulty} + 0.17 \cdot \text{dimension}$$

Figure 8 shows this relationship. The major outliers are the 'rooms' maps, on which HPA* has a lower runtime than the formula would suggest. Remember that the searching algorithm for HPA* consists of two stages: insertion of the source and goal nodes followed by an A* search through the abstract graph. Table 8 gives an overview of insertion and searching times by map type. Insertion times are higher for 'rooms' maps compared to the other map types. However, this does not explain the discrepancy. A possible explaination for the low running time is the specific layout of the rooms maps and choice of hyperparameters: 8×8 or 16×16 square rooms with single-cell doors. Since the size of the initial HPA* cells is 8×8, this aligns perfectly with the walls of the rooms, possibly explaining the low running time.

Finally, there is evidence to suggest that the best choice of hyperparameters for HPA* is influenced by map metrics. Since the non-game maps are all 80×80 grids, the current method for determining

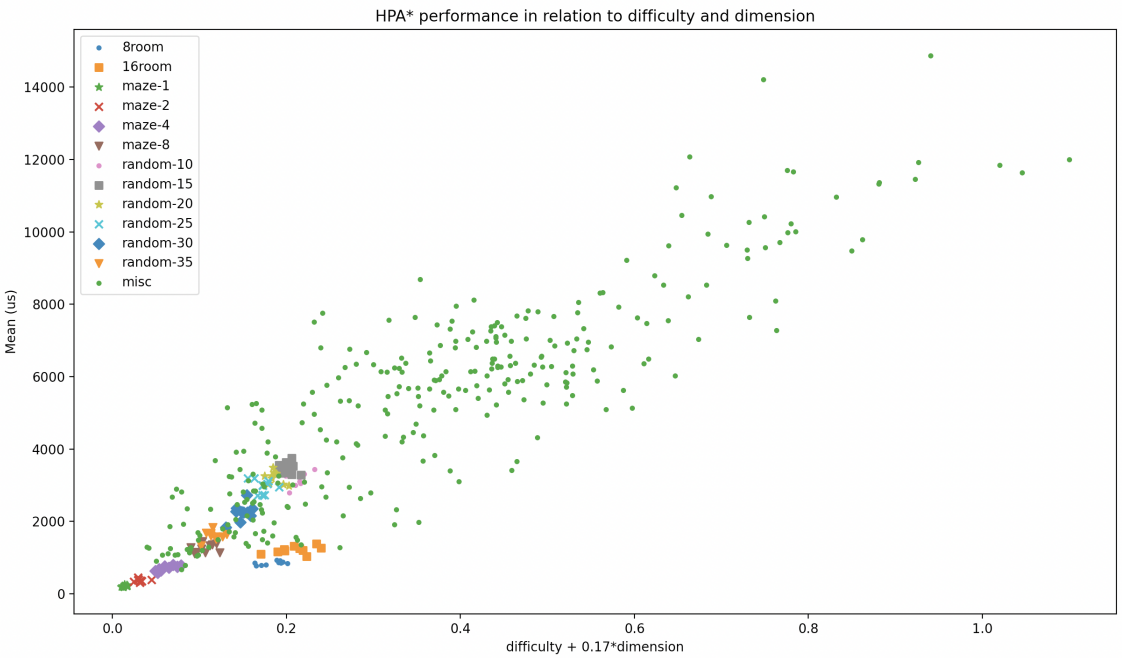


Figure 8: HPA* performance in relation to difficulty $+ 0.17 \cdot$ dimension

Map type	Insertion	Searching	Proportion spent searching
8rooms	4.10	4.40	52%
16rooms	4.77	7.32	61%
maze-1	0.62	1.57	72%
maze-2	1.46	2.27	61%
maze-4	2.66	4.48	63%
maze-8	3.86	8.73	69%
random-10	4.45	27.45	86%
random-15	4.02	30.66	88%
random-20	3.45	28.77	89%
random-25	2.94	26.76	90%
random-30	2.40	20.10	89%
random-35	1.79	14.17	89%

Table 8: Insertion and searching time averages by map type

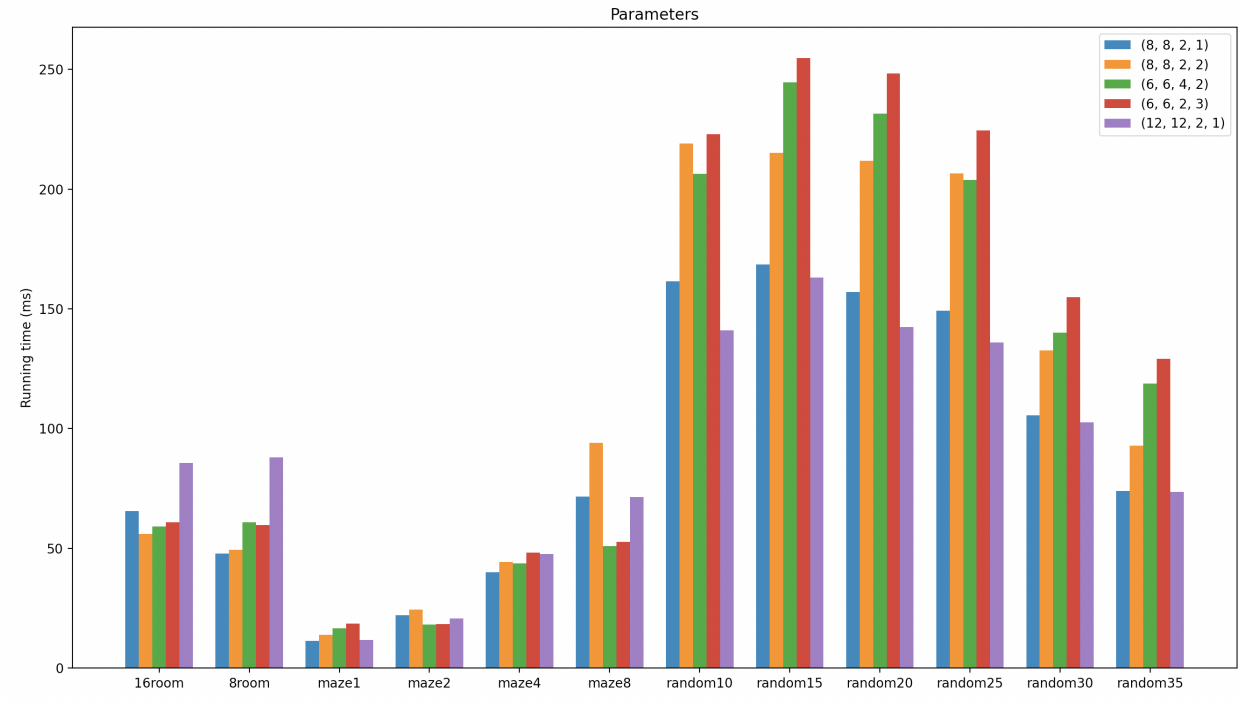


Figure 9: Performance of HPA* on non-game maps for a variety of values for the hyperparameters

the values for the hyperparameters will produce the same result for all of them, namely:

$$w_a = h_a = 8 \qquad n = 1 \qquad m = 2$$

In order to investigate whether this was the best choice, benchmarks were run for a variety of different choices for the hyperparameters. See figure 9 for the results (in the figure, the hyperparameter values are presented as (w_a, h_a, m, n)). We found that different map types generally have different best choices for the hyperparameters, with no one choice being the best for all map types.

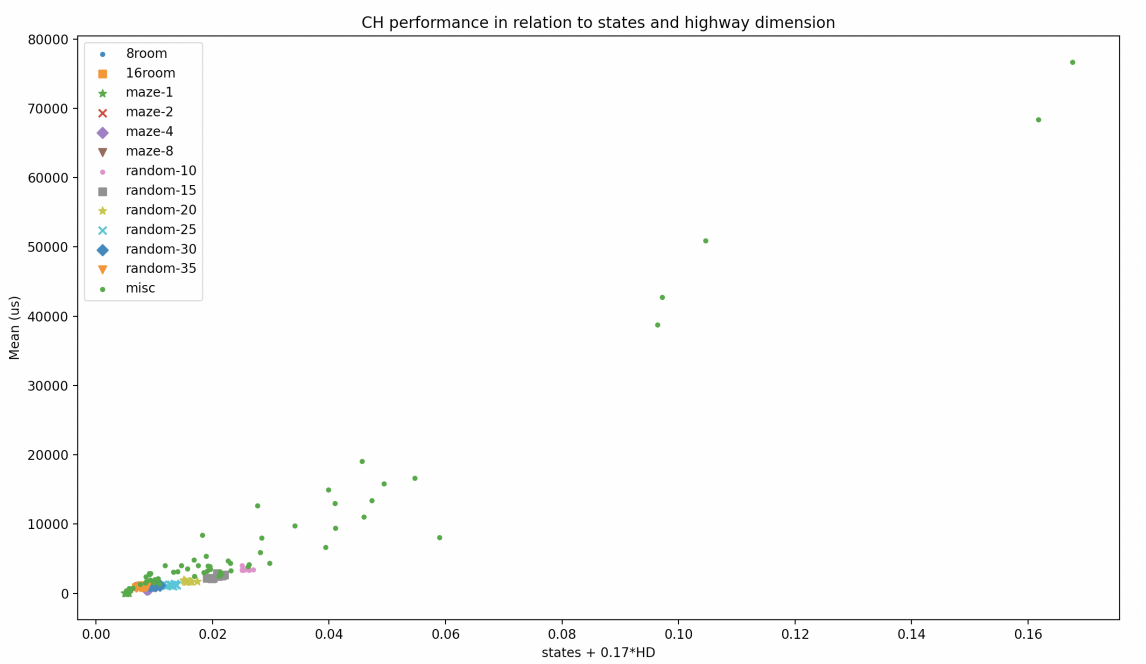


Figure 10: Contraction hierarchies performance in relation to highway dimension $+ 144.4 \cdot$ states

7.2.4 Contraction hierarchies

The runtime performance of contraction hierarchies shows the strongest correlation with the number of states (correlation coefficient 0.89). Combining this with the highway dimension as follows yields a clean relationship (correlation coefficient 0.97, see also figure 10):

$$\text{states} + 0.17 \cdot \text{highway dimension}$$

7.2.5 Contraction hierarchies with A*

Using A* with contraction hierarchies has a significant impact on the performance characteristics. The number of states becomes more strongly correlated with the running time (from 0.84 to 0.94) and the highway dimension becomes more weakly correlated (from 0.84 to 0.72). Once again, combining these two metrics yields the best relationship (correlation coefficient 0.96, see figure 11):

$$\text{states} + 0.031 \cdot \text{highway dimension}$$

7.2.6 Hub labels

Hub labels, like contraction hierarchies, shows a strong correlation with highway dimension (correlation coefficient 0.86). Unlike contraction hierarchies, there is also a moderate correlation with dimension (correlation coefficient 0.66, versus 0.32 for contraction hierarchies). A strong relationship (correlation coefficient 0.92, see also figure 12) is found with the following combination of metrics:

$$\text{highway dimension} + 0.15 \cdot \text{degree}$$

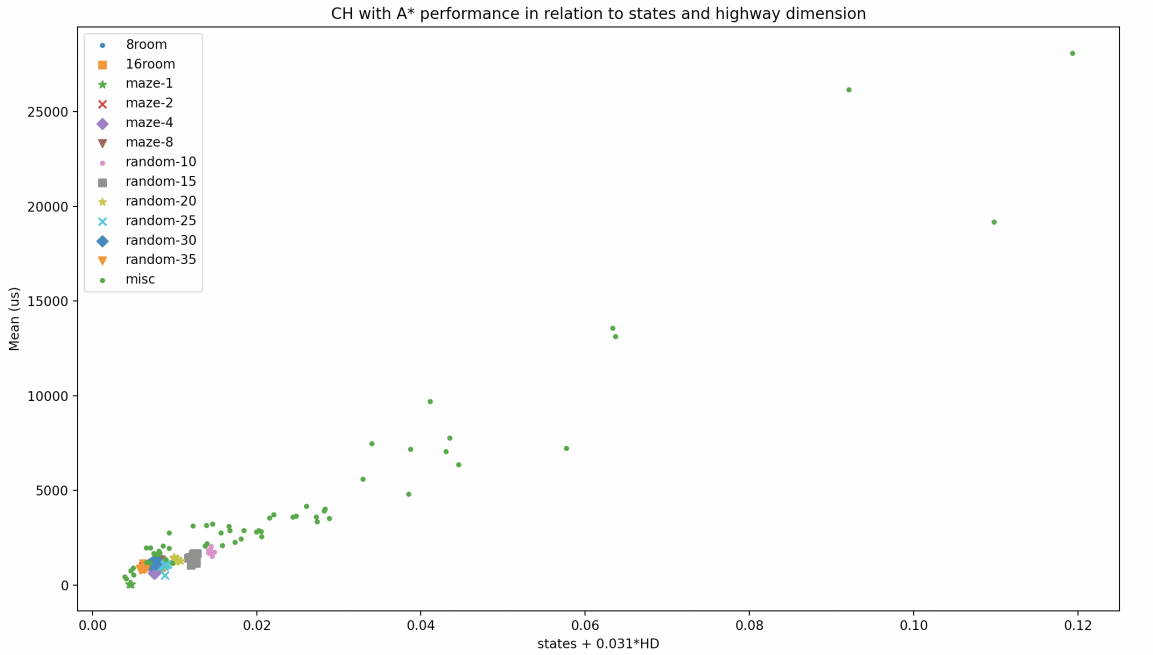


Figure 11: Contraction hierarchies with A* performance in relation to states + 0.031 · highway dimension

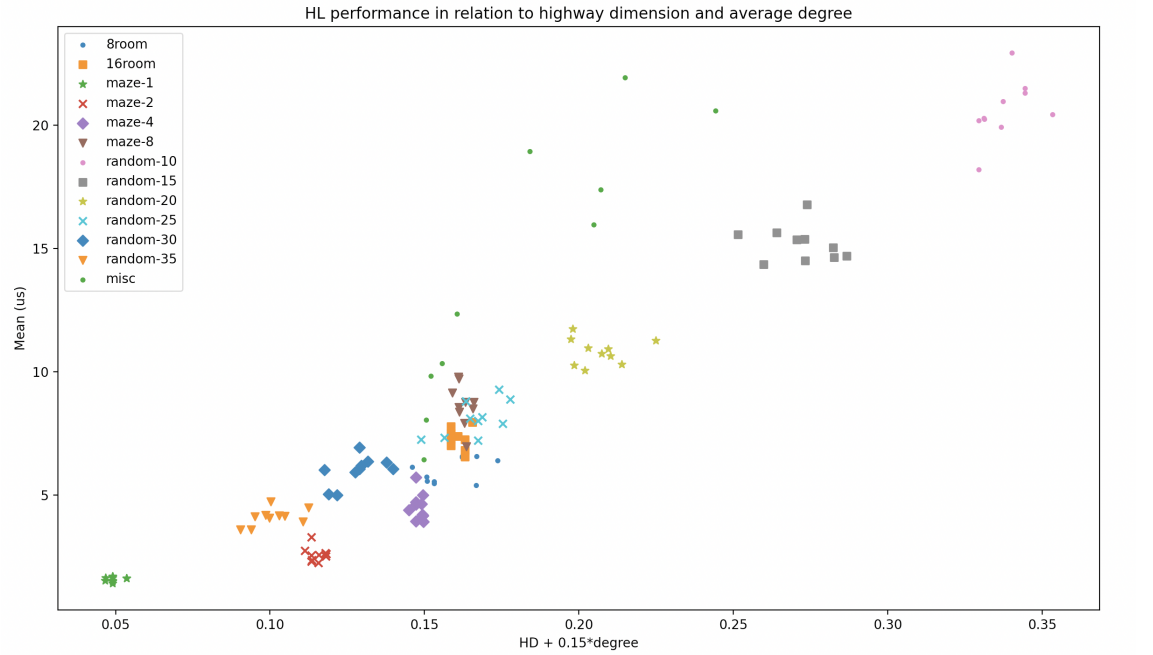


Figure 12: Hub labels performance in relation to highway dimension + 0.15 · degree

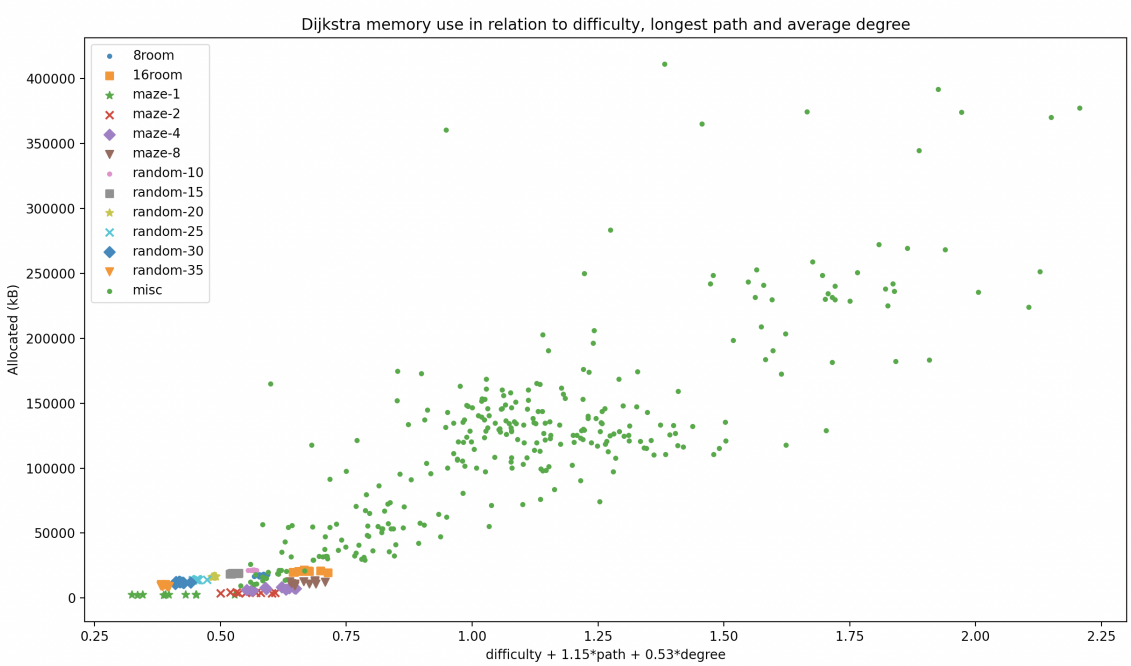


Figure 13: Dijkstra memory use in relation to difficulty $+ 1.15 \cdot$ longest path $+ 0.53 \cdot$ average degree

7.3 Memory use

The memory use data is similar to the running time data, indicating that the amount of allocated memory is directly related to the running time of each algorithm. However the values are rounded and displayed as, for instance, '3 MB', instead of a more exact number. The result is that the memory use data is a sort of 'banded' version of the running time data, as if the running times were rounded. As a result of this, there was no analysis to be performed on the memory use data for hub labels, since the only unique value was 56 bytes.

7.3.1 Dijkstra

Memory use for Dijkstra was most strongly correlated with difficulty (correlation coefficient 0.82), but the cleanest relationship was with the following combination of metrics, see also figure 13 (correlation coefficient 0.88):

$$\text{difficulty} + 1.15 \cdot \text{longest path} + 0.53 \cdot \text{average degree}$$

7.3.2 A*

Memory use for A* was most strongly correlated with the longest path (correlation coefficient 0.82), but the cleanest relationship was with the following combination of metrics, see also figure 14 (correlation coefficient 0.88):

$$\text{longest path} + 0.73 \cdot \text{states} + 0.3 \cdot \text{average degree}$$

Figure 14: A* memory use in relation to longest path $+ 0.73 \cdot$ states $+ 0.3 \cdot$ average degree

7.3.3 HPA*

Memory use for HPA* was most strongly correlated with the difficulty (correlation coefficient 0.95), but the cleanest relationship was with the following combination of metrics, see also figure 15 (correlation coefficient 0.96):

$$\text{difficulty} + 0.54 \cdot \text{states} + 0.41 \cdot \text{longest path}$$

7.3.4 Contraction hierarchies

Memory use for contraction hierarchies was strongly correlated with the number of states (correlation coefficient 0.997), but the cleanest relationship could be achieved by adding the highway dimension as follows, see also figure 17:

$$\text{states} + 0.01 \cdot \text{highway dimension}$$

7.3.5 Contraction hierarchies with A*

Memory use for contraction hierarchies with A* was strongly correlated with the number of states (correlation coefficient 0.997). The relationship could not be significantly improved by adding another metric. See figure 17 for a plot.

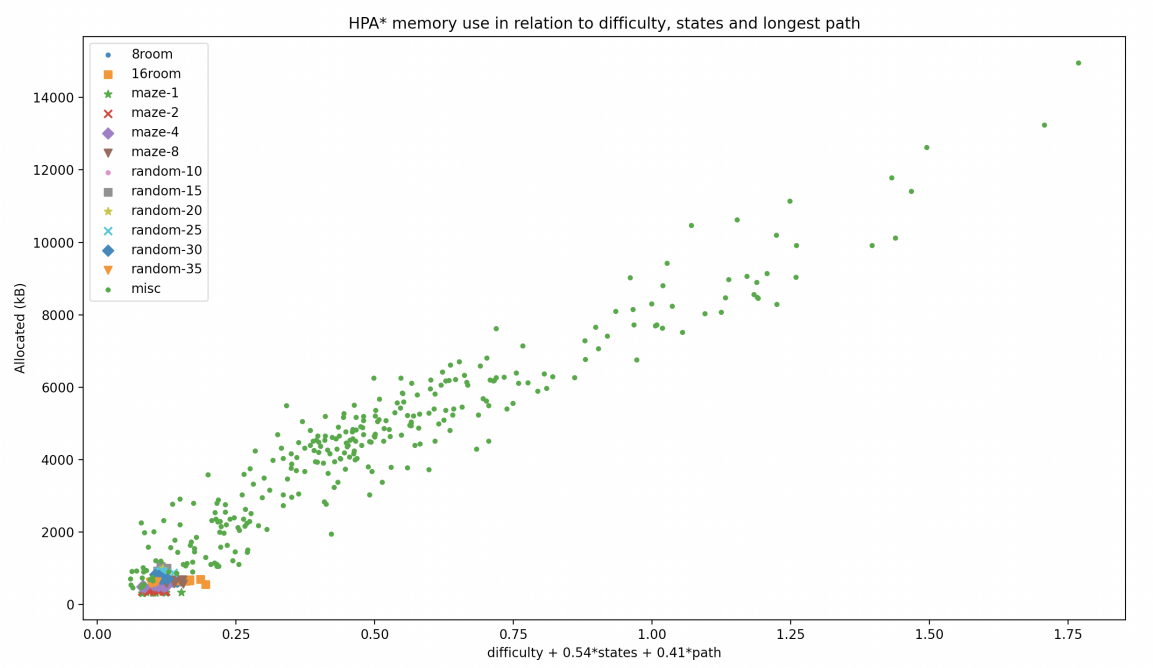


Figure 15: HPA* memory use in relation to difficulty $+ 0.54 \cdot$ states $+ 0.41 \cdot$ longest path

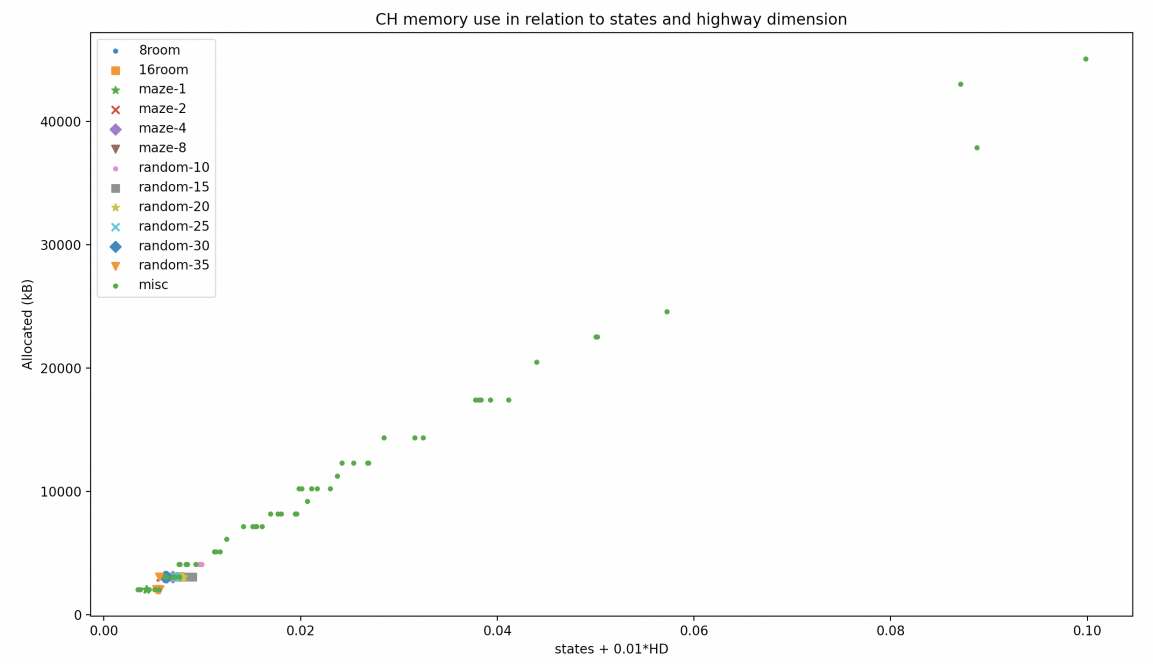


Figure 16: Contraction hierarchies memory use in relation to states $+ 0.01 \cdot$ highway dimension

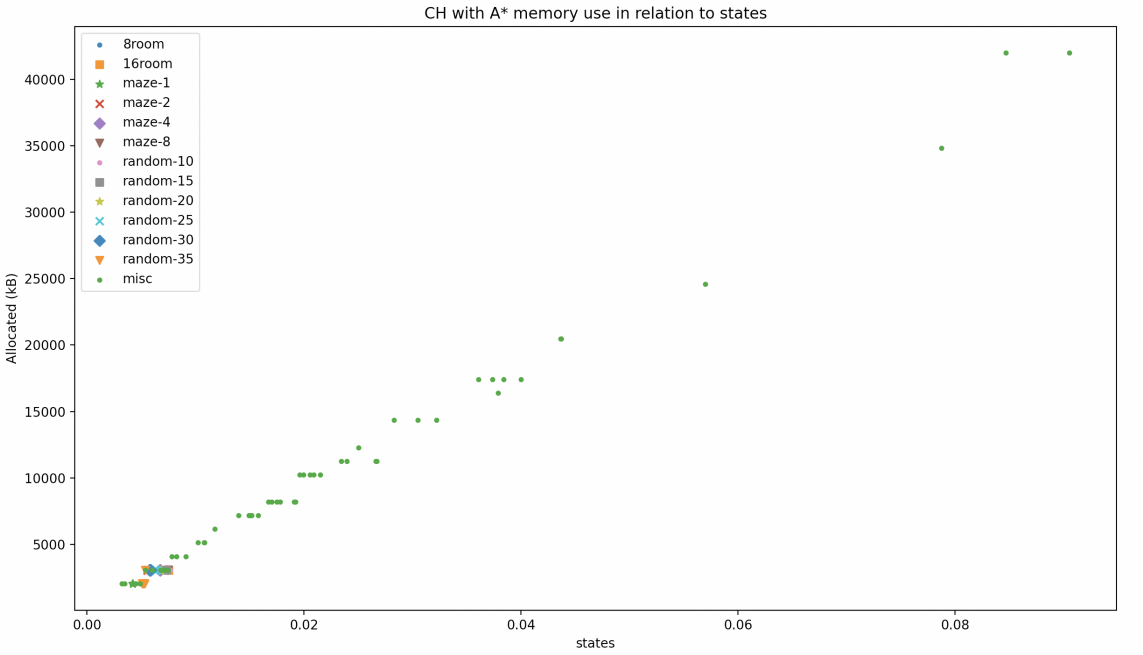


Figure 17: Contraction hierarchies with A* memory use in relation to number of states

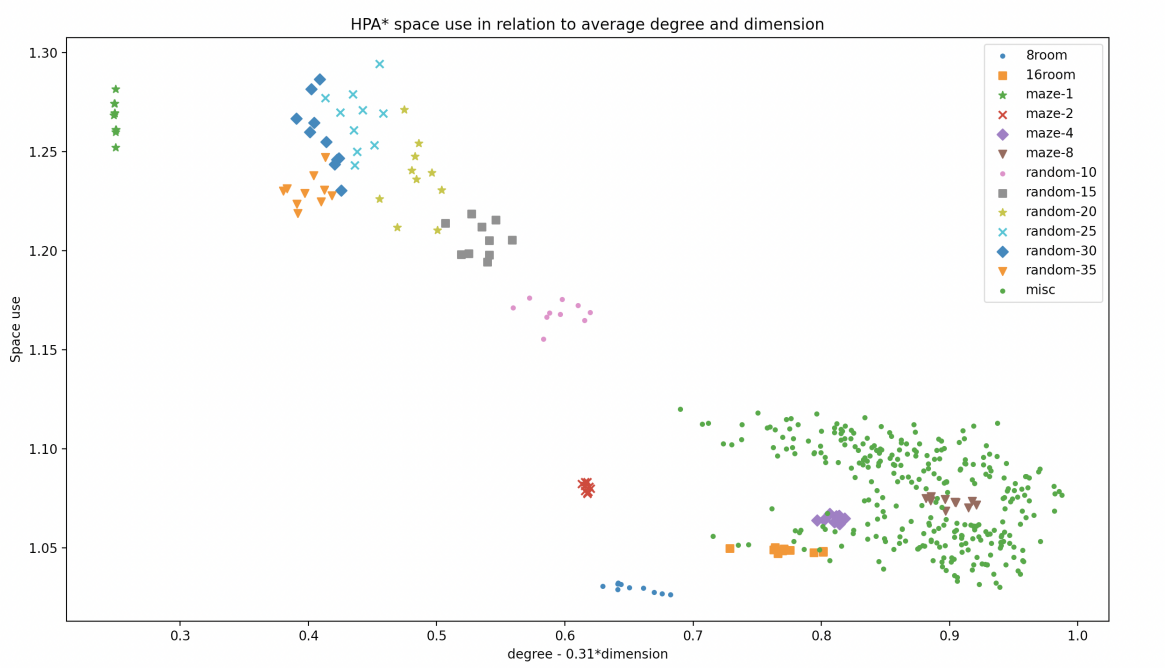


Figure 18: HPA* space use in relation to average degree $-\ 0.31 \cdot$ dimension

7.4 Space use

7.4.1 HPA*

The space use of HPA* shows a moderate correlation with the average degree (correlation coefficient -0.78). We can add the dimension to make it slightly stronger - correlation coefficient -0.8 (see figure 18), but we nonetheless don't get a clean relationship.

7.4.2 Contraction hierarchies

The space use of contraction hierarchies shows no strong correlation with any one metric, but we can combine the number of states and the highway dimension as follows:

$$\text{states} + 0.09 \cdot \text{highway dimension}$$

To get the relationship seen in figure 19. Again, we don't see a clean linear relationship, but there is a hint of a square root or logarithmic relationship. The latter is supported by [11], theorem 6.2, which states that nodes in a shortcut graph have degree $O(h \log D)$, since the space use is based on the number of shortcuts.

7.4.3 Hub labels

Space use for hub labels shows a strong correlation with highway dimension (correlation coefficient 0.87). We can combine this with the longest path to get a cleaner relationship (correlation coefficient 0.9), see figure 20. Note however, that the arrangement is similar to the one in figure 19, but since we lack data for hub labels on some of the larger maps we don't get the same picture. Since the space use of hub labels is based on the label size and [11] mentions that the maximum size of a label is $O(h \log D)$, it is plausible that the relationship is similar to the one observed for contraction hierarchies.

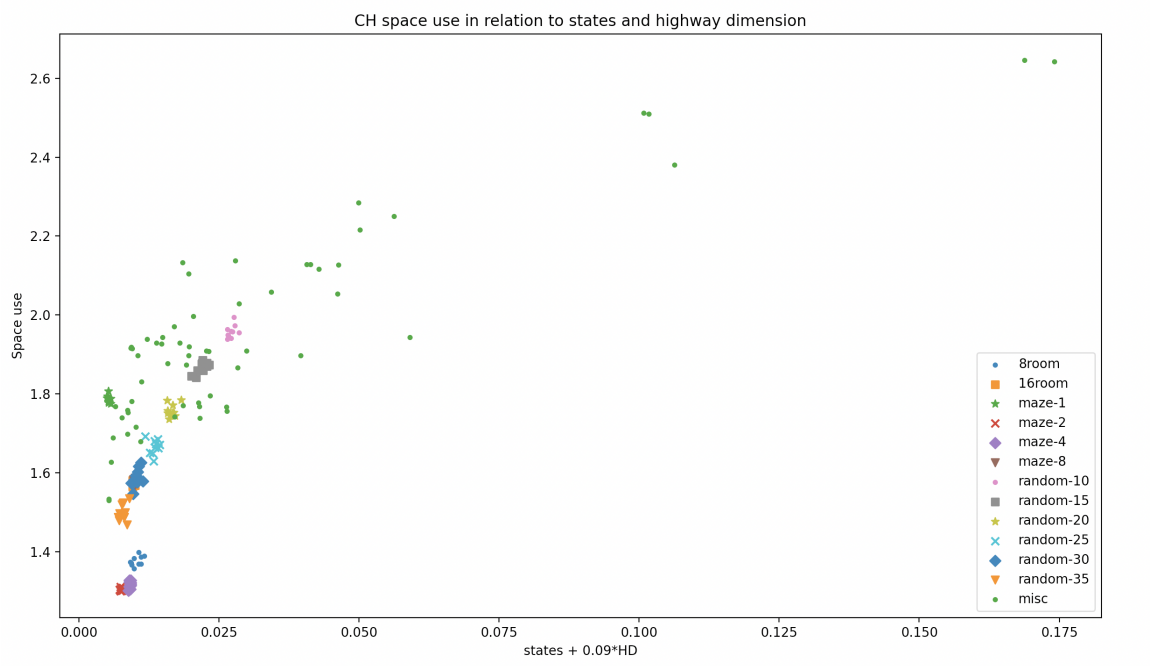


Figure 19: Contraction hierarchies space use in relation to states $+ 0.09 \cdot$ highway dimension

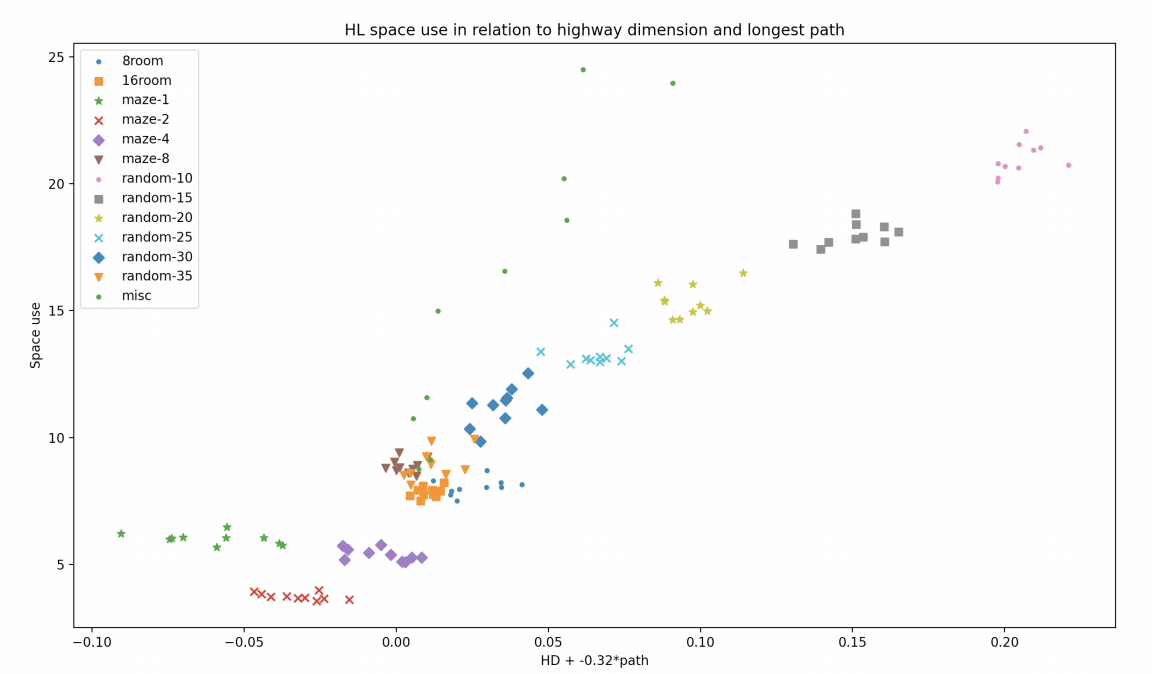


Figure 20: Hub labels space use in relation to highway dimension $- 0.32 \cdot$ longest path

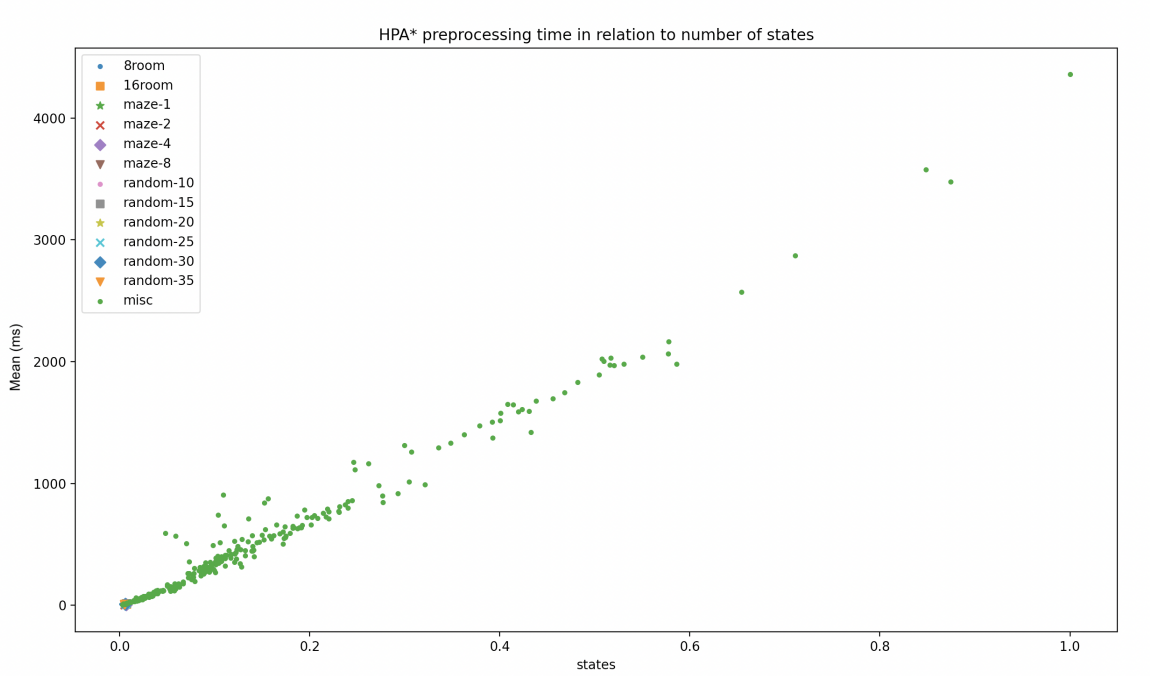


Figure 21: HPA* preprocessing time in relation to number of states

7.5 Preprocessing

7.5.1 HPA*

HPA* preprocessing time shows a near-perfect correlation with the number of states (correlation coefficient 0.99, see figure 21). This is to be expected, since the preprocessing time is directly based on the number of doors, which will increase with the number of states. The non-game maps all have a similar number of states, and hence show a weaker - but still strong - correlation with the number of states (correlation coefficient 0.79). By combining the number of states with the highway dimension and longest path, we get a solid relationship with both the game (correlation coefficient 0.99) and non-game maps (correlation coefficient 0.95), see also figure 22:

$$\text{highway dimension} + 50 \cdot \text{states} - 0.3 \cdot \text{longest path}$$

7.5.2 Contraction hierarchies

Preprocessing time for contraction hierarchies doesn't appear to show any strong correlation (correlation coefficient greater than 0.8) to a single one of the metrics, nor can we combine them to get a stronger relationship. Note however that the Pearson correlation coefficient only indicates the strength of a linear relation. It is plausible that there is a superlinear relation between the preprocessing time of contraction hierarchies and one of the metrics. For instance, figure 23 appears to indicate an exponential relation between the preprocessing time and average degree.

7.5.3 Hub labels

Preprocessing time for hub labels shows a strong correlation to highway dimension (correlation coefficient 0.96). This makes sense, since - because we use contraction hierarchies for the check queries - preprocessing may involve thousands of distance queries for contraction hierarchies. Hence, the preprocessing time for hub labels will be mostly based on the runtime performance of contraction hierarchies. Combining the highway dimension with the number of states, as we did

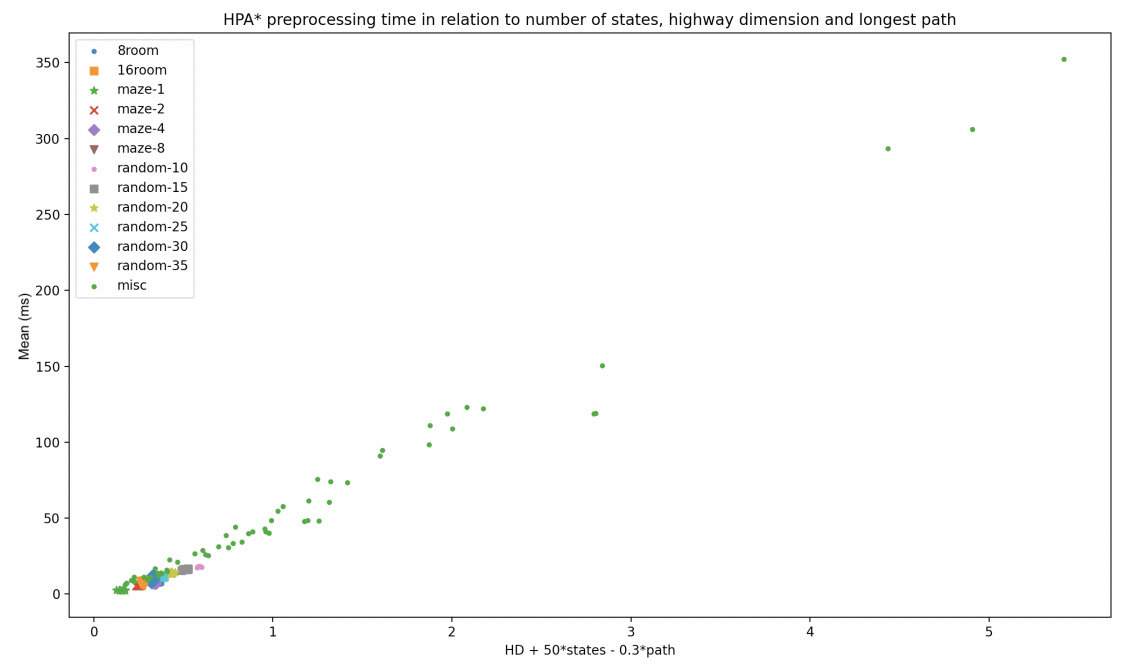


Figure 22: HPA* preprocessing time in relation to highway dimension+50·states−0.3·longest path

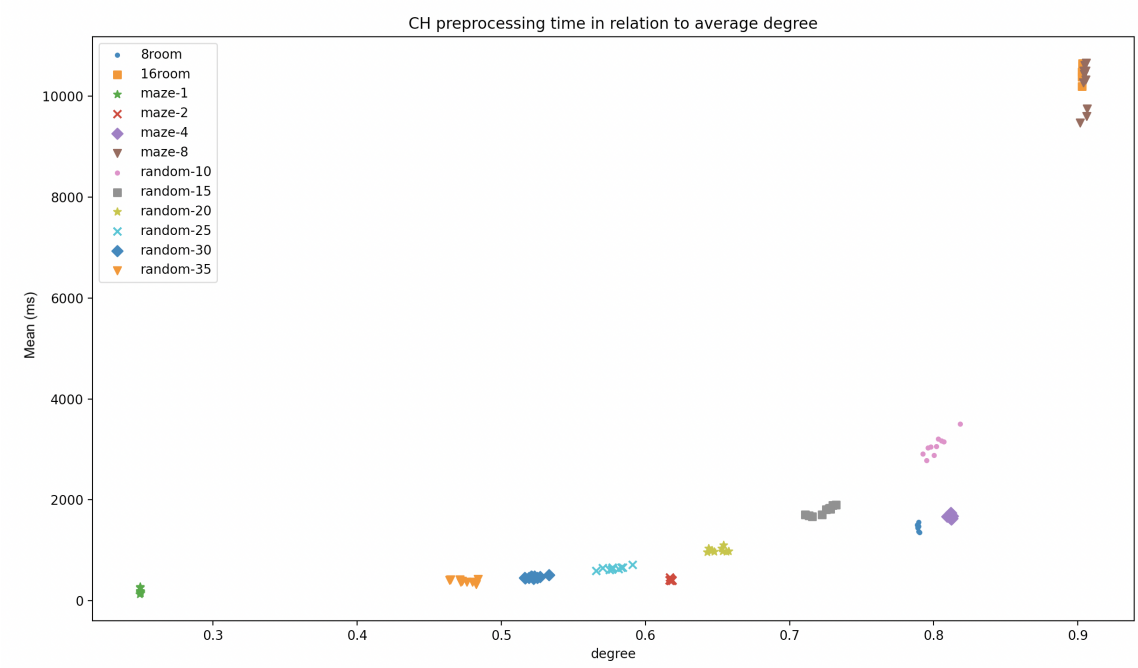


Figure 23: Contraction hierarchies preprocessing time in relation to average degree

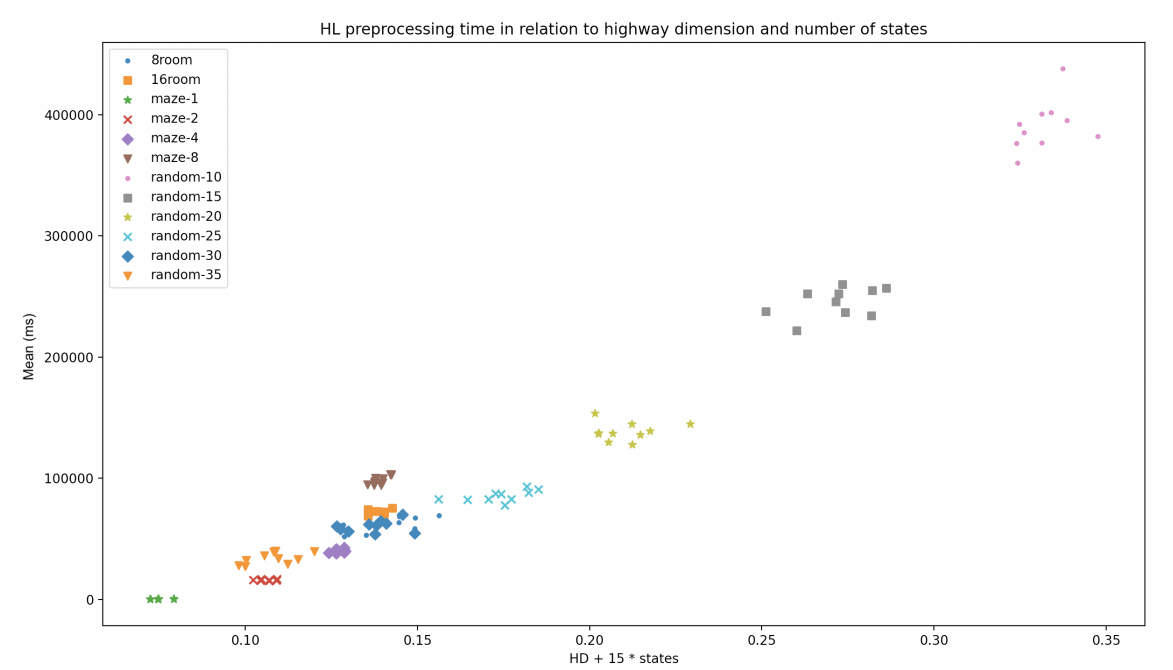


Figure 24: Hub labels preprocessing in relation to highway dimension $+ 15 \cdot$ states

for the runtime of contraction hierarchies, yields a clean relationship (correlation coefficient 0.97, see also figure 24):

$$\text{highway dimension} + 15 \cdot \text{states}$$

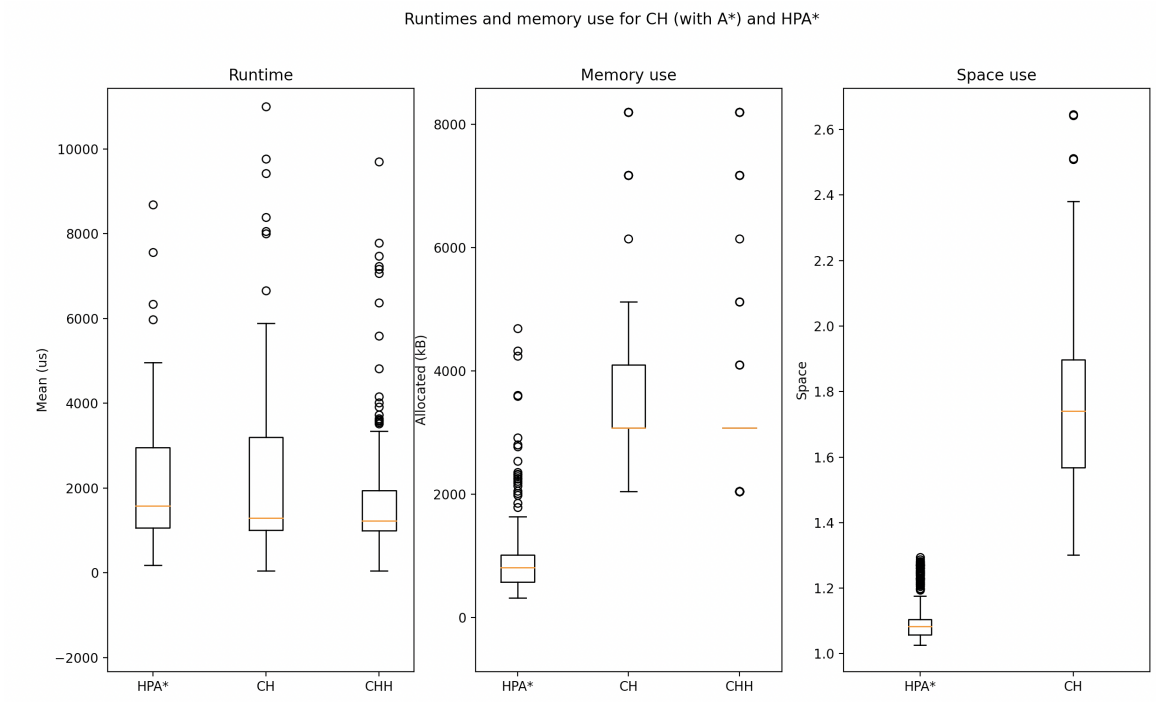


Figure 25: Comparison of runtime, memory allocation and space use for contraction hierarchies (with A*) and HPA*

7.6 HPA* vs. contraction hierarchies

In section 7.1 it was mentioned that contraction hierarchies, both with and without A*, is faster than HPA*, but at the cost of having higher memory and space use. As can be seen in figure 25 the difference in running time is not significant, whilst the differences in memory and space use are.

This does not mean that the difference in runtime is always insignificant. Figure 26 shows the running time of contraction hierarchies plotted against the running time of HPA*. The figure shows that contraction hierarchies is significantly faster on a number of map types, namely random-10 to random-35 and maze-1. Figure 27 shows the same plot for contraction hierarchies with A*, which adds random-10 to the list of map types on which contraction hierarchies with A* is significantly faster. From these figures we can also see that HPA* is significantly faster than both contraction hierarchies with and without A* on game and maze-2 maps.

Despite this, there seems to be no way to predict, based on the map metrics, whether or not contraction hierarchies will be faster than HPA*, as is illustrated in figures 28 and 29.

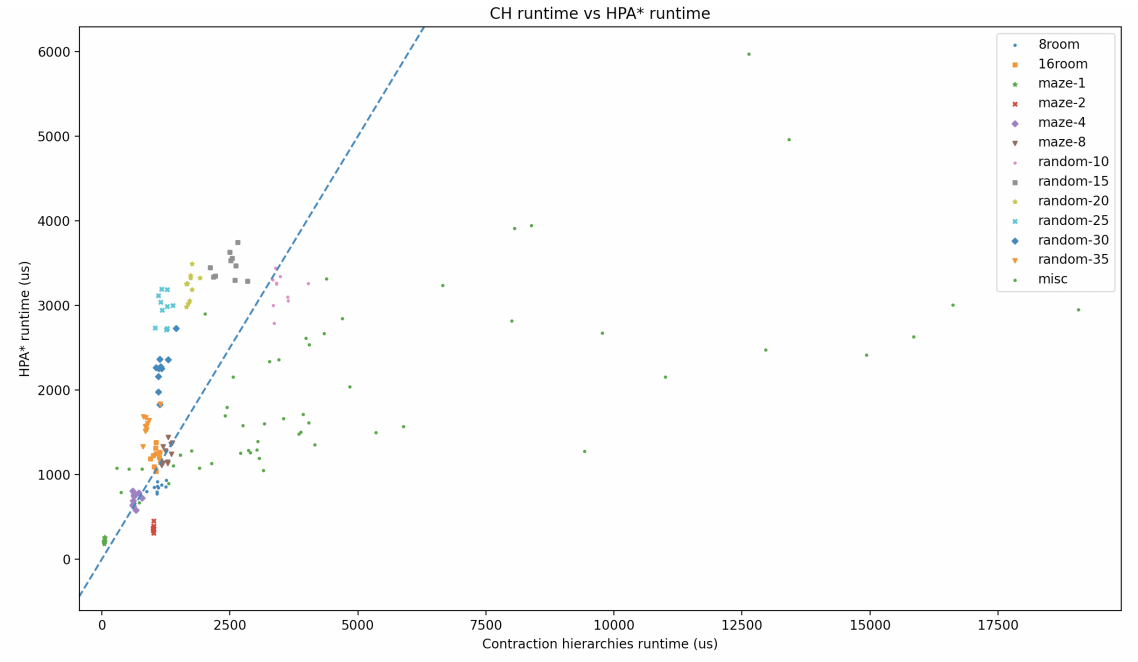


Figure 26: Contraction hierarchies running time vs HPA* running time, with dotted line indicating $x = y$

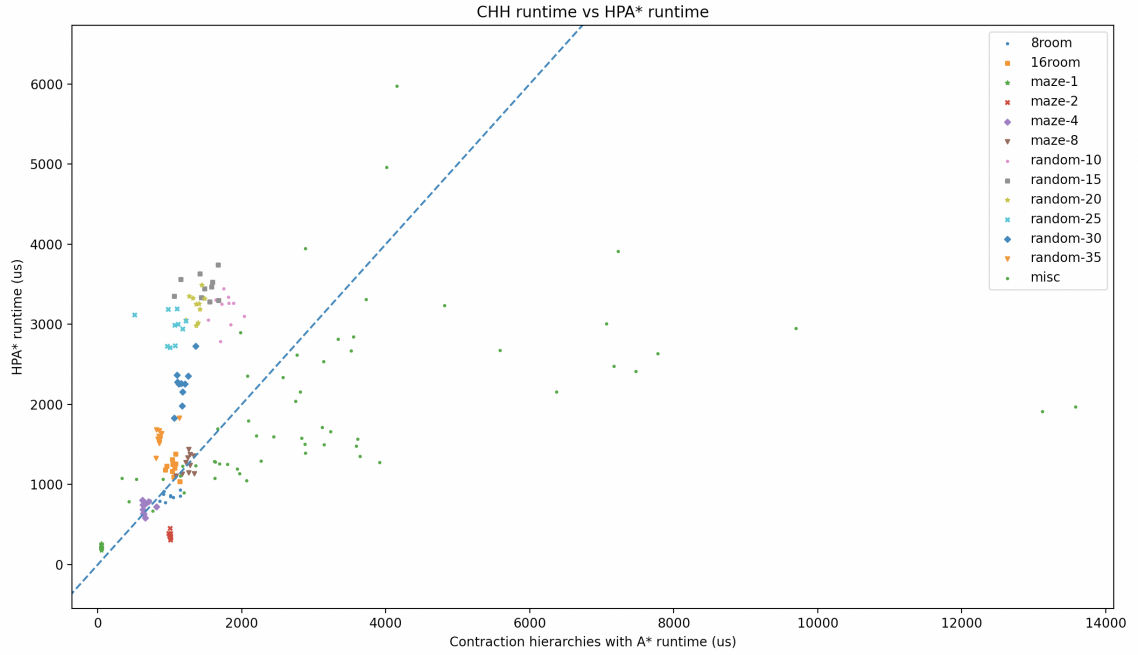


Figure 27: Contraction hierarchies with A* running time vs HPA* running time, with dotted line indicating $x = y$

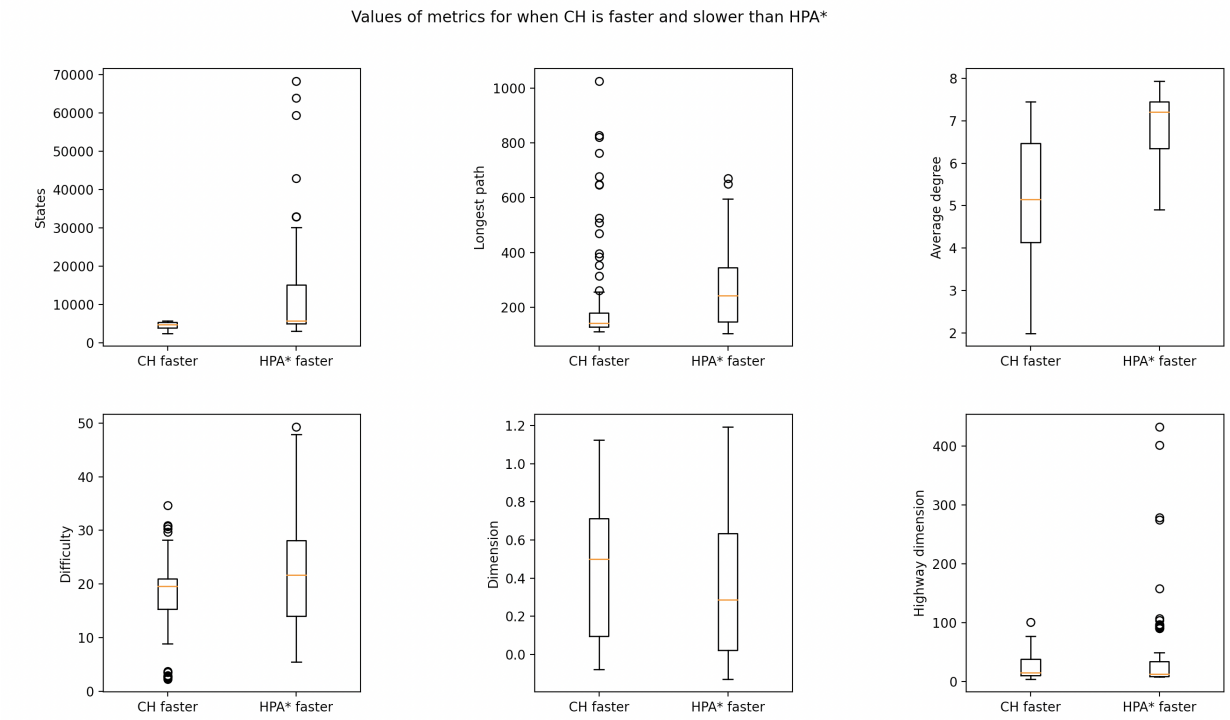


Figure 28: Boxplots for all metrics, both when contraction hierarchies is faster and slower than HPA*

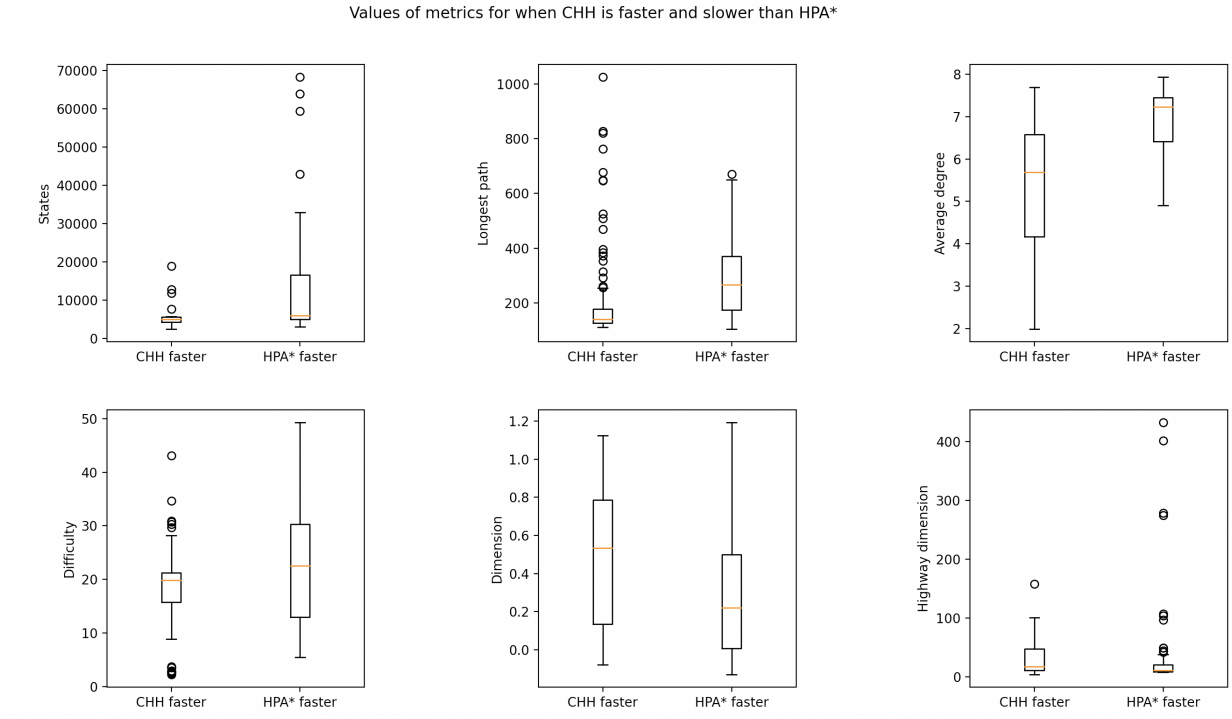


Figure 29: Boxplots for all metrics, both when contraction hierarchies with A* is faster and slower than HPA*

8 Discussion

There are a number of limitations of this study to mention and discuss. Firstly, there are some practical factors which may have influenced the results: the benchmarks were not run in a vacuum, but on a machine that, besides running the benchmarks, was also running the Windows operating system and several background processes and daemons. Any spikes in running time caused by another process running on the machine may have influenced the results, however this effect is alleviated by the fact that the machine has six logical cores among which to distribute work and the BenchmarkDotNet library, which detects and removes outliers from the performance measurements. One process in particular that could have had an impact on the results is the garbage collector. In order to test the extend to which garbage collection influenced performance measurements, we compared a benchmark with garbage collection to one without garbage collection (the garbage collector was temporarily turned off using the built-in GC.TryStartNoGCRegion method) and found that turning off the garbage collector actually increased running time by a factor of about 1.06. This suggests that the garbage collector does not unfairly inflate the running times of algorithms. Another factor is the hardware: it is possible that a specific, unlucky interaction with the used hardware might have increased the runtime of an algorithm. While this is does not necessarily taint the results, re-running the benchmarks on a different machine would improve confidence in the results.

It is possible that a programming error may have unnecessarily slowed down one or more of the algorithms. For the runtime of the algorithms, this does not appear to be an issue since speedups found for HPA* and contraction hierarchies align with those found in the literature: around a 10 times speedup for HPA* is reported in [1] and speedups of 7, 10 and 20 are reported for contraction hierarchies in [12]. However, preprocessing for contraction hierarchies and hub labels took more than an hour on larger maps, whilst this did not seem to be the case in the literature. For instance, [12] runs the preprocessing phase of contraction hierarchies on the same benchmark problems we used and reports times of between 9 and 12 seconds for 512×512 grids. It should be noted that [12] uses a modified version of the contraction hierarchies algorithm with a significantly optimized preprocessing phase, so some difference is to be expected. However, the magnitude of the difference found suggests something may have gone wrong. Note that [7] also ran contraction hierarchies on large grid maps, but did not report preprocessing times. It is also possible that the running time of the standard preprocessing algorithm for contraction hierarchies is not suited for grid maps, causing the slowdown on larger maps.

Finally, there are some methodical limitations to the findings of this study. For one, the measurements were performed only on problems where the true distance was between 100 and 104. We know from [7] that the running time of contraction hierarchies is low on queries with longer true distance: [7] reports a running time of 36.4 μs averaged over true distances, and 49.4 μs on the longest 1% of true distances. The same is not necessarily true for HPA*, hence the comparison between these two algorithms may have turned out differently if problems with a greater true distance had been chosen. Some other limitations come up with respect to the metrics, which may have been altered by scaling the non-game maps down from 512×512 to 80×80. In particular, the amount of states, longest path, average degree and difficulty all tend to decrease when a grid map gets smaller. Whilst this does not necessarily spoil the results, also examining different map types at their full size may improve confidence in the relations between map metrics and performance given. Lastly, computing (or even estimating) the highway dimension of a graph remains a hard problem, as mentioned in [11]. The method provided by this paper appears to provide reasonable values but it is hard to be sure of the validity of the computed values, especially since they cannot be easily verified. The general trend appears to be underestimation of the highway dimension, as suggested by table 3.

9 Conclusions and suggestions for future work

We were able to find strong relations between the map metrics and the performance of the algorithms. Table 9 gives an overview of these correlations.

Algorithm	States	Longest path	Average degree	Difficulty	Dimension	HD
Dijkstra (general)	Strong	-	Weak	Strong	Weak	-
Dijkstra (game)	-	-	Strong	Strong	Strong	-
A*	-	Weak	Strong	Strong	-	-
HPA*	-	-	-	Strong	Weak	-
CH	Strong	-	-	-	-	Strong
CHH	Strong	-	-	-	-	Weak
HL	-	-	Strong	-	-	Strong
Memory use	States	Longest path	Average degree	Difficulty	Dimension	HD
Dijkstra	-	Strong	Weak	Strong	-	-
A*	Strong	Strong	Weak	-	-	-
HPA*	Weak	Weak	-	Strong	-	-
CH	Strong	-	-	-	-	Weak
CHH	Strong	-	-	-	-	-
Preprocessing	States	Longest path	Average degree	Difficulty	Dimension	HD
HPA*	Strong	Weak	-	-	-	Strong
CH	-	-	Strong	-	-	-
HL	Strong	-	-	-	-	Strong
Space use	States	Longest path	Average degree	Difficulty	Dimension	HD
HPA*	-	-	Strong	-	Weak	-
CH	Strong	-	-	-	-	Strong
HL	-	Weak	-	-	-	Strong

Table 9: Overview of how strongly a metric correlates with the performance of a pathfinding algorithm

The claim that the running time of hub labels and contraction hierarchies increases with the highway dimension from [11] is reproduced here. Note that we found the running time to also be dependent on the number of states. This likely corresponds to the diameter component in the bounds provided by [11] (see section 4.2). It is somewhat strange that the longest path did not show a strong relation with the running time of contraction hierarchies, since it is equivalent to the diameter. Although we did find some relation between the running time of HPA* and the difficulty and dimension, there are some findings which suggest another, unexplained influence. Namely, the fact that the performance on rooms maps is not well explained by the given metrics, and the fact that the choice of hyperparameters seems to change how the performance relates to the map metrics. We have also found a contraction hierarchies-like relation to highway dimension and number of states for the preprocessing time of HPA*. We suggest further research into how the choice of hyperparameters relates to the map metrics and preprocessing time. We do not feel confident in making any strong conclusions about the preprocessing time for contraction hierarchies and hub labels, as we found preprocessing times that were unexpectedly high and did not align with those reported in the literature for both algorithms. This suggests that either something has gone wrong or that significant alteration to the algorithms (as was done in [12]) is required in order to achieve lower preprocessing times on larger grid maps. We seem to have found a nonlinear relationship between the space use of contraction hierarchies and the combination of highway dimension and number of states, although we suggest collecting more data on larger game maps to improve confidence in this conclusion. A similar relationship might exist for the space use of hub labels, however more data on game maps is needed in order to verify or falsify this hypothesis. From our data, it appears as though the space use of hub labels is most strongly related to the highway dimension, which is supported theoretically by [11]. We were not able

to establish a clean relationship between the metrics and the space use of HPA*, which suggests that the chosen metrics are unable to predict the space use of this algorithm. The memory use results for Dijkstra and A* seem unreliable due to the banding effect described in section 7.3, so we do not feel confident in those obtained relationships. Finally, note that we found a difference in behaviour between non-game and game maps for the running time of Dijkstra, which suggests there might be some difference between the 'artificial' and game maps that is not captured in the chosen metrics.

Finally, some notes on when each of the algorithms are most useful. If the lowest runtime is all that matters, hub labels is the clear choice. It has the lowest running time of all the chosen algorithms even on large maps with high highway dimension. However, it also has the highest space use of all algorithms treated in this paper, using an average of 11 times the amount of space required for the input graph. This means that memory limitations might restrict when it can be effectively used. In these situations, contraction hierarchies and HPA* offer low running times with smaller space requirements. The running times of these two algorithms are comparable, although there can be significant differences: From the results it is clear that HPA* is faster than contraction hierarchies on video game maps, however it is unclear in what situations contraction hierarchies is faster. We found that contraction hierarchies is faster than HPA* on random maps, but could not explain this result using the map metrics. We suggest expanding the repertoire of map metrics and benchmark problems in future research to try to explain this result. What is clear, however, is that HPA* has lower space and memory use than contraction hierarchies. No meaningful comparison can be made between the observed preprocessing time of contraction hierarchies and HPA*, because of the limitations discussed earlier. We also found that using A* with contraction hierarchies offers better runtime performance with no significant drawback. While it may seem that HPA* is the better choice, the algorithm does have the major drawbacks of being limited specifically to grid maps and returning suboptimal results. In addition, the values chosen for the hyperparameters can have a significant, unpredictable effect on the performance. Here, we suggest further research into the related HNA* algorithm [14], which can be used on any graph, not just grids, to see if it can also achieve performance that is competitive with that of contraction hierarchies.

In summary, we have shown that in a large number of cases the chosen map metrics have a significant impact on the performance of the algorithms. However, we were unable to use the map metrics to decide between two algorithms with comparable running time (contraction hierarchies and HPA*).

References

[1] Near Optimal Hierarchical Pathfinding, 2004, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.479.4675rep

[2] Contraction hierarchies: Faster and Simpler Hierarchical Routing in Road Networks, 2008, http://algo2.iti.kit.edu/download/diploma$_t hesis_g eisberger.pdf$

[3] Hub Label Compression, 2013, hub-labeling algorithm, https://www.microsoft.com/en-us/research/wp-content/uploads/2010/12/HL-TR.pdf

[4] Benchmarks for Grid-Based Pathfinding, 2012, https://webdocs.cs.ualberta.ca/ nathanst/papers/benchmarks.pdf

[5] https://movingai.com/benchmarks/, repository of benchmark pathfinding problems (Benchmarks for Grid-Based Pathfinding, 2012)

[6] Pathfinding Algorithm Efficiency Analysis in 2D Grid, 2013, https://www.researchgate.net/profile/Sergejs-Kodors/publication/282488307$_P athfinding_A lgorithm_E fficiency_A nal$ $Algorithm - Efficiency - Analysis - in - 2D - Grid.pdf$

[7] A Comparison of High-Level Approaches for Speeding Up Pathfinding, 2010, https://www.cs.du.edu/ sturtevant/papers/highlevelpathfinding.pdf

[8] Simulation and Comparison of Efficency in Pathfinding algorithms in Games, 2015, https://www.researchgate.net/publication/315509846$_Simulation_and_Comparison_of_Efficency_in_Pathfinding_algoritl$

[9] Engineering Route Planning Algorithms, 2009, https://i11www.iti.kit.edu/extra/publications/dssw-erpa-09.pdf

[10] Route Planning in Transportation Networks, 2015, https://arxiv.org/pdf/1504.05140.pdf

[11] Highway Dimension and Provably Efficient Shortest Path Algorithms, 2013, https://www.microsoft.com/en-us/research/wp-content/uploads/2013/09/tr-msr-2013-91-rev.pdf

[12] Contraction hierarchies on Grid Graphs, 2015, https://ad-publications.cs.uni-freiburg.de/KI$_gridCH_S2$013.*pdf*

[13] Benchmarkdotnet website, https://benchmarkdotnet.org/

[14] Hierarchical path-finding for Navigation Meshes (HNA*), 2016, https://www.sciencedirect.com/science/article/pii/S0097849316300668

[15] Partial Pathfinding Using Map Abstraction and Refinement, 2005, https://www.aaai.org/Papers/AAAI/2005/AAAI05-221.pdf

[16] Efficient Triangulation-Based Pathfinding, 2006, https://www.aaai.org/Papers/AAAI/2006/AAAI06-148.pdf