Contraction Hierarchies on Grid Graphs

Sabine Storandt

Albert-Ludwigs-Universität Freiburg
79110 Freiburg, Germany
storandt@informatik.uni-freiburg.de

Abstract. Many speed-up techniques developed for accelerating the computation of shortest paths in road networks, like reach or contraction hierarchies, are based on the property that some streets are 'more important' than others, e.g. on long routes the usage of an interstate is almost inevitable. In grids there is no obvious hierarchy among the edges, especially if the costs are uniform. Nevertheless we will show that contraction hierarchies can be applied to grid graphs as well. We will point out interesting connections to speed-up techniques shaped for routing on grids, like swamp hierarchies and jump points, and provide experimental results for game maps, mazes, random grids and rooms.

1 Introduction

Efficient route planning in grid graphs is important in a wide range of application domains, e.g. robot path planning and in-game navigation. While many search algorithms like $A*$ provide relatively fast solutions on the fly, it might be worthwhile to allow some preprocessing to speed up query answering if the same grid map is used multiple times. For road networks state-of-the-art preprocessing techniques like *contraction hierarchies* [1] and *transit nodes* [2] enable shortest path computation in a few milliseconds or even microseconds on graphs with millions of nodes and edges. But the structure of street graphs differs clearly from grids: Shortest paths are almost always unique and some edges (e.g. corresponding to highways and interstates) occur in significantly more optimal paths than others. Therefore it is not obvious that such speed-up techniques carry over to grid graphs. Nevertheless, it was shown that the idea of transit node routing can be adapted to grid graphs [3], leading to a significantly improved performance on video game maps. Contraction hierarchies were also tested on grid graphs [4], but these grids had non-uniform costs and moreover the construction algorithm for road networks was applied without any adaption. In this paper, we will show how to modify contraction hierarchies to take care of the special structure of grid graphs.

1.1 Contribution

We will first describe in detail how contraction hierarchies can be modified to work on grid graphs, introducing some simple rules to speed up the preprocessing phase. Moreover, we especially focus on how to be able to compute canonical

optimal paths, i.e. paths with a minimal number of direction changes which are desirable in many applications. In our experimental evaluation, we give empirical evidence for the small amount of auxiliary data created by constructing a contraction hierarchy upon a grid. We compare query answering with our approach to the A*-baseline for several input categories, like random graphs and mazes. Finally, we also point out connections to other speed-up techniques developed for path planning on grids.

1.2 Related Work

Because of its significance in many applications, speed-up techniques for routing on grids are described in numerous papers. We distinguish between online approaches, where no preprocessing is applied – like the standard *A** algorithm (and variants thereof) or *jump points* [5] – and offline algorithms, which allow a preprocessing phase, like *swamps* [6]. Moreover, there are optimal and suboptimal search algorithms, with *HPA** [7] being an example for the latter. Finally, some speed-up techniques are very specific for certain instance classes (like only for 4-connected grids [8], or preferably for game maps as described e.g. by Bj önsson et al. [9]), while others are beneficial in several application domains. In this paper, we describe an offline, optimal and unspecific technique to efficiently retrieve shortest paths in grid graphs. We will come back to similarities and compatibility with other methods towards the end of the paper.

2 Contraction Hierarchies (CH) on Grids

In this section, we want to review the standard contraction hierarchy approach [1] and give some intuition why this method will work on grids (with uniform costs) besides no obvious hierarchy among the edges. Subsequently, we will describe how to use CH-search to retrieve canonical paths via an edge classifier approach.

2.1 Conventional Contraction Hierarchy

Given a (di)graph $G(V, E)$, the basic idea behind CH is augmenting the graph with shortcuts that allow to save a lot of edge relaxations at query time. To that end, in a preprocessing phase nodes are sorted according to some notion of importance. Afterwards the nodes get contracted one by one in that order while preserving all shortest path distances in the remaining graph by inserting additional edges (so called shortcuts). More precisely, after removing a node v the distance between any pair of neighbours u, w of v has to stay unchanged. Therefore an edge (u, w) with proper costs is inserted if the only shortest path from u to w is uvw. Hence if there exists a so called *witness path* with lower cost than u, v, w or with equal cost but not visiting v (typically found via a Dijkstra run from u) the shortcut can be omitted. If the graph is undirected, we assume for clarity of definitions that an edge is represented by its two directed versions. Note, that it is not necessary to use this transformation in the

actual implementation. After all nodes have been removed, a new graph G' is created by adding all shortcuts to the original graph. An edge (v, w) in G' is called upward if the importance l of v is smaller than that of w ($l(v) < l(w)$) and downward otherwise. A path is called upward/downward, if it consists of upward/downward edges only. By construction, for every pair of vertices $s, t \in V$ it exists a shortest path in G', which can be subdivided into an upward and a downward path. Therefore s-t-queries can be answered bidirectionally, with the forward run (starting at s) considering only upward edges and the backward run (starting at t) considering exclusively downward edges. We call the respective subgraphs containing all upward paths starting at s or all downward paths ending in t respectively as $G^{\uparrow}(s)/G^{\downarrow}(t)$ and the highest node wrt to l on an s-t-path the peak.
To get the path in the original graph and not in G', we have to expand contained shortcuts back to paths. For that purpose, we store during the CH-construction the IDs of the two skipped edges for every shortcut. A recursive unpacking procedure allows then to retrieve the original edges.

On the first sight, the construction of a CH upon a grid seems to be a bad idea. Consider a 4-connected grid, the contraction of a node would remove four edges; but any two of these edges might form a shortest path, hence up to six shortcuts must be inserted. Contracting the neighbouring nodes this effect amplifies, giving the impression that we might end up with a quadratic number of edges in G' (see Figure 1, left). But there are two characteristics of grids preventing this: Optimal paths in grids with uniform costs are ambiguous, but only one optimal solution needs to be preserved. In fact, contracting the first node in a complete 4-connected grid, only two shortcuts instead of six have to be inserted because of ambiguity, see Figure 1 (right) for an illustration. Unfortunately, the sparser the grid, the more the ambiguity of shortest paths subsides. But if a grid is far from being complete, the holes introduce a certain kind of hierarchy as well because now shortest paths tend to use their borders. Hence CH construction upon a grid might work even if it seems counter-intuitive at the first glance.

2.2 Accelerating the CH Construction

There are actually no modifications necessary to run CH on a grid as the basic framework at least in theory works on any graph. But we can speed up the preprocessing using the characteristics of a grid with uniform costs.

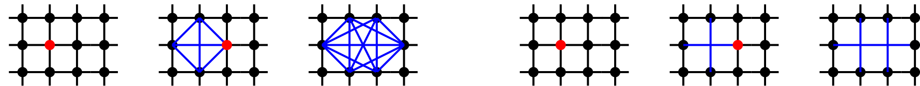

Fig. 1. Illustration of two contraction steps (removal of the red node): On the left without considering ambiguity (or assuming non-uniform costs), on the right with inserting only shortcuts between neighboring nodes if the shortest path via the red node is unique.

Knowing the positions of the nodes in the grid, we do not have to start a witness search for u, v, w if the nodes are all on a straight line and the summed costs of (u, v) and (v, w) comply with the interval between u and w. Also if $c(u, v) + c(v, w)$ equals the absolute positional difference between u and w, we can restrict the witness search to the rectangle spanned by u and w. Moreover we can plug-in A* or any other search algorithm to accelerate the witness search in any case. Note, that even a suboptimal algorithm would be alright because not detecting an existing witness might lead to the insertion of a superfluous shortcut, but this will not compromise optimality. In fact, we could insert all shortcuts right away and the quality of the queries would be unaffected. But as additional shortcuts increase the graph size and therefore the runtime of the preprocessing as well as the query answering, we aim for keeping G' sparse.

2.3 Maintaining Canonical Paths

To save energy (especially concerning robot navigation) and to enable a natural way of moving, we aim for an optimal path with a minimal number of turns, i.e. a canonical path. But neither Dijkstra nor plain A* can guarantee to find such a solution if the optimal path is ambiguous.

To gain this ability in G' we proceed for a 4-connected grid as follows: We assign to an edge/shortcut $e = (u, v)$ in the graph a classifier $[a(u), t, a(v)]$, with $a(u), a(v)$ implicating with which kind of an edge the path $p(e)$ spanned by e starts/ends. Here, we use h if it is a horizontal edge and v if it is a vertical one. Moreover we assign the number of turns t on $p(e)$ to e as well. So every vertical edge in the original graph G has the classifier $[v, 0, v]$ in the beginning, every horizontal one $[h, 0, h]$. Bridging two edges via a new shortcut, the classifier can easily be determined: Let the node skipped by the shortcut be v and the two bridged edges $e_1 = (u, v)$ and $e_2 = (v, w)$ with classifiers $[a(u), t_1, a(v)]$ and $[a'(v), t_2, a'(w)]$. If $a(v) = a'(v)$ the shortcut (u, w) receives the classifier $[a(u), t_1 + t_2, a'(w)]$, otherwise $[a(u), t_1 + t_2 + 1, a'(w)]$. To maintain canonical solutions – without inserting too many shortcuts – we adapt the CH-construction slightly: Whenever the resulting classifier for a potential shortcut reveals $t = 0$ or $t = 1$, we insert the shortcut right away (despite the possible existence of a witness), because the spanned paths are trivially canonical (at most one turn) and therefore optimal for sure. But as soon as $t \geq 2$, we have to apply witness search again. If a witness is found with lower cost or equal cost and fewer turns, we omit the shortcut, otherwise we insert it. If at some point a potential shortcut (u, w) exhibits the same costs but fewer turns than an already existing edge (u, w), we update the respective classifier (and do not need to start a witness search).

We will now prove by induction over the number of turns that this approach maintains for every pair of vertices $s, t \in V$ a canonical shortest path between them in the CH. For the base clause, we first verify the claim for paths with at most one turn.

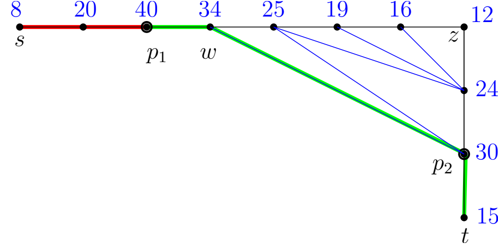

Fig. 2. Illustration of the proof for Lemma 1: Blue numbers imply the contraction order of the nodes, original edges are black, shortcuts are coloured blue. The red background colour marks an upwards path, the green colour indicates a downward path, showing that the canonical path from s to t via z is contained in $G^{\uparrow}(s) \cup G^{\downarrow}(t)$.

Lemma 1. *Every optimal trivially canonicals-t-path (i.e. exhibiting at most one turn) can be reconstructed considering only $G^{\uparrow}(s) \cup G^{\downarrow}(t)$.*

Proof. If the shortest path between s and t is a straight line, the optimal path is unique and therefore contained in $G^{\uparrow}(s) \cup G^{\downarrow}(t)$ for sure. So let now z be the turning point on the path as depicted in Figure 2. As s-z and t-z are unique optimal paths, they can be given in CH-description. So let p_1, p_2 be the peak nodes on those subpaths, w.l.o.g. $l(p_1) > l(p_2)$ and w the lowest node on the path p_1-z with $l(w) > l(p_2)$. As w-z goes downwards and z-p_2 upwards, the shortcut (w, p_2) will be considered and inserted because it represents a trivial canonical path. Hence s-p_1 is in $G^{\uparrow}(s)$ and p_1-w-p_2-t in $G^{\downarrow}(t)$. ∎

Theorem 1. *For every pair of vertices $s, t \in V$ all optimal canonical shortest paths between them are contained in $G^{\uparrow}(s) \cup G^{\downarrow}(t)$.*

Proof. Our induction hypothesis is, that for every pair of vertices s, t with an optimal canonical path between them exhibiting $\leq k$ turns, the path can be found in $G^{\uparrow}(s) \cup G^{\downarrow}(t)$. For $k \leq 1$ we proved correctness in Lemma 1. Now, for the induction step, let z be the last turning point on a canonical s-t-path. By induction hypothesis the path from s to z is contained in $G^{\uparrow}(s) \cup G^{\downarrow}(z)$. Also let z' be the last turning point on the path s-z (if s-z is a straight line, set $z' = s$), p'' the peak node on this path, and further p' the peak node on z'-z and p the peak node on z-t (which both must be well defined as the respective paths are unique shortest paths). Following the argumentation in the proof of Lemma 1, the shortcut (p', p) will be inserted for sure. If $l(p') > l(p)$, we are done, because s-$p'' \in G^{\uparrow}(s)$ and p''-p'-p-$t \in G^{\downarrow}(t)$. Otherwise assume $l(p'') > l(p)$. Then let w be the node on the path p''-p with the smallest label exceeding $l(p)$. Hence at some point, we have to decide whether to insert the shortcut (w, p). As there can not exist a witness with lower costs or one with equal costs and fewer turns (otherwise the considered s-t-path would not be canonical), the (w, p) will be inserted. Therefore s-$p'' \in G^{\uparrow}(s)$ and p''-w-p-$t \in G^{\downarrow}(t)$. If $l(p'') < l(p)$ the argumentation works exactly the same, now choosing w to be the node on p'-p with the smallest label exceeding $l(p'')$, resulting in s-p''-w-$p \in G^{\uparrow}(s)$ and p-$t \in G^{\downarrow}(t)$. ∎

On the basis of Theorem 1, we now want to describe how to extract a respective canonical path for given s, t, considering only nodes and edges in $G^{\uparrow}(s) \cup G^{\downarrow}(t)$. We still use two Dijkstra runs, one in $G^{\uparrow}(s)$ and the other one in $G^{\downarrow}(t)$. The crucial difference is now that during edge relaxation we do not only update the tail node v of the edge if we can reduce the costs, but also if the costs stay the same and the number of turns can be decreased. This number can be easily computed along, as the summed number of turns of the classifiers along the path from s/t to v plus the number of nodes on the path at which we change from horizontal to vertical or vice versa (with the information being contained in the classifiers as well). But now we have to be careful, because it makes a difference with which kind of edge (h or v) a path ends as it influences the number of turns on superpaths. Therefore we allow now the assignment of two labels per node if both exhibit the same costs and number of turns, but the first one corresponds to a path ending with a vertical edge and the other one to a path with the final edge being horizontal. After termination of the two Dijkstra computations, we iterate over all expanded nodes in both runs and keep track of the node which minimizes the summed costs and also the summed turns (incremented by one if a turn occurs at this node, too). For the resulting node, we backtrack the two subpaths and unpack them to get the final path in the original graph.

The whole argumentation carries over to 8-connected grids with uniform costs (i.e. diagonal edges cost $\sqrt{2}$). Here, we introduce two new direction parameters in the edge classifier, d_1 and d_2, indicating diagonal movement (from the lower left corner to the upper right or from the lower right corner to the upper left). Turning points are now also nodes where movement is changed from straight to diagonal or vice versa. One could argue, that the change of direction is less significant here, as the turning angle is now only $45°$ and not $90°$. Hence we could also increase our turn counter by only 0.5 instead of 1 for each such direction change. In the extraction phase the Dijkstra algorithm can now assign four different labels to each node, but as with the edge classifiers this only results in a constant overhead.

3 Experimental Results

Now we want to evaluate the impact of our approach on real-world and synthetic instances[1]. We implemented CH-search in C++ and performed experiments on an Intel i5-3360M CPU with 2.80GHz and 16GB RAM. We start by providing some implementation details for the CH-construction and measure the amount of auxiliary data created in the preprocessing phase. Upon that, we analyse the number of expanded nodes for our approach in comparison to A*. We also describe, how to combine CH- and A*-search and provide experimental results for this scenario as well. Then, for several input categories, we will individually give some intuition why CH-search accelerates the query answering. Finally, we will draw some connections to other speed-up techniques.

[1] extracted from http://movingai.com/

3.1 Preprocessing

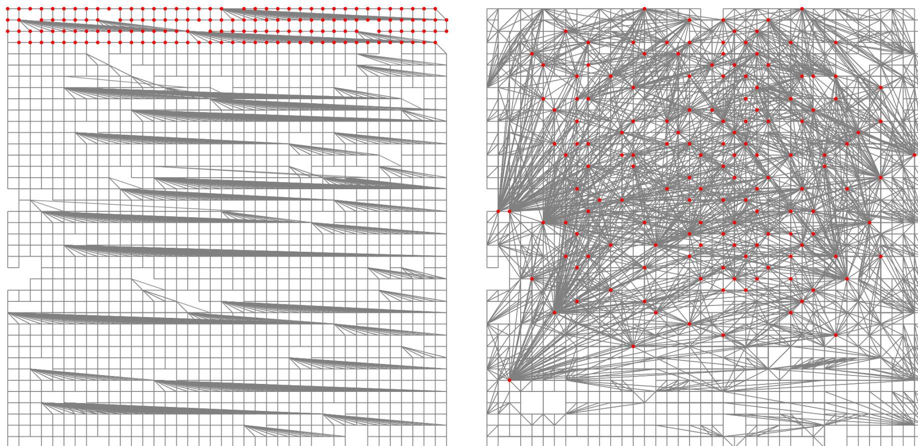

Fig. 3. Two different CH-construction schemes tested on a 40x40 grid graph with 5% randomly deleted vertices and uniform costs, exhibiting 2867 edges. The 10% of the nodes which were contracted last are coloured red. Left: Contraction order is the enumeration order of the nodes, resulting in a CH-graph with only 3490 edges but no measurable speed-up. Right: Contraction order based on weighted edge-difference and consecutively chosen independent set of nodes. The number of edges here is 6771, in a random query in this graph the number of expanded nodes is only halve the number of expanded nodes for plain A*.

To construct a CH, we have to define the order in which the nodes should be contracted. As one goal is to keep the resulting graph as sparse as possible, the classical indicator for road networks is the so called edge-difference (ED) [1]. The ED is the number of edges we have to insert when removing the node minus the number of adjacent edges. So normally the lower the ED the better. Therefore one always contracts the node with the current lowest ED next. In our application this could lead to an undesired effect: We might add no shortcuts at all. Consider a complete 4-connected finite grid with uniform costs; we could just enumerate the nodes row-by-row from left to right and use these numbers as the importance l. Contracting the nodes in this order, a node has at most two adjacent edges (removing nodes with less than two edges does not lead to shortcut insertion anyway). For nodes with two edges, the path over these two is indeed optimal, but also ambiguous at the moment of contraction, so no shortcut has to be inserted. Of course, as an artefact of our canonical path maintenance strategy, we would insert shortcuts here as well, but even this would be superfluous as also all canonical paths are preserved automatically by the contraction order. What seems to be a nice feature at the first glance – the CH-construction not increasing the graph size at all – is unfortunately not an advantage, be-

cause using this contraction order the search space does not diminish at all as the subgraph of G in the spanned rectangle of source and target is completely contained in $G^{\uparrow}(s) \cup G^{\downarrow}(t)$. So to force shortcut insertion, we have to make sure that not always nodes at the border of the actual graph are contracted. One way to achieve this, is to use a weighted version of the ED where removing edges from the graph gets more rewarded. A second approach is to contract always an independent set of nodes in the graph, which is found in a greedily manner considering the nodes sorted increasingly by their actual ED. As independent nodes do not influence the shortcut insertion of each other, this approach does not compromise optimality but induces the contraction of 'inner' nodes as an early stage of the CH-construction process. In Figure 3 the resulting CH-graph for this construction scheme is compared to the one for the enumeration approach.

For the witness search we used A* with the straightforward distance estimation for 4- and 8-connected grids. Of course, for certain instance classes better heuristics are at hand, and using them might speed up the CH-construction. But here we want to emphasize that CH can be used without prior knowledge of the kind of input.

For road networks, the number of shortcuts in G' equals approximately the number of original edges, i.e. the CH doubles the graph size. In Table 1 (left) we collected the main parameters describing the CH-construction for a 512x512 4-connected grid with uniform costs and varying percentage of randomly deleted nodes. We observe that the augmentation factor (AF) of the CH-graph depends strongly on the number of deleted vertices, but even for a complete grid (0% deletion) the graph size increases only by a factor of seven, which is a tolerable space overhead. In the right table, the CH-construction is summarized for several input categories, all being based on 8-connected grids. here, the augmentation factor is always below three and for mazes and rooms even below the typical value of two for street graphs. The preprocessing time is about 10 seconds for all inputs, corresponding to the time to answer approximately 1700 queries on average over all instances. Hence constructing the CH-graph on the fly does only make sense if the number of queries exceeds this bound. But for many applications, like in-game navigation, the preprocessing time does not play a major role as the CH-graph can simple be provided as the map itself.

Table 1. CH-construction: Number of original edges and edges in the CH-graph (original + shortcuts) for several input categories. The augmentation factor (AF) describes the ratio between those two. 'p' in the left table described the percentage of randomly deleted nodes in a 512x512 grid. The column 'time' in the right table gives the preprocessing time in seconds.

p	# edges	# CH-edges	AF
0	523,264	3,590,007	6.8
10	423,229	1,387,549	3.2
25	293,081	604,939	2.0
50	130,664	137,388	1.0

input type	# edges	# CH-edges	AF	time (secs)
mazes	682,922	1,064,079	1.6	10
rooms	847,871	1,242,509	1.5	9
game maps	271,532	768,002	2.8	10
random	492,857	1,319,287	2.7	12

3.2 Query Answering

In the augmented graphs, we applied CH-search as described in Section 2.3 between randomly chosen source-target pairs $s, t \in V$. Our baseline is the number of expanded nodes using A* (with Manhattan-metric for 4-connected grids and the obvious octile metric for 8-connected ones). Hence we evaluated the performance of our approach as the ratio of this value and the number of expanded nodes by CH-search (i.e. the higher the better). Moreover the CH-approach reveals the advantage of being easily combinable with other speed-up techniques as it allows to extract a small subgraph $(G^{\uparrow}(s) \cup G^{\downarrow}(t))$ in which optimal query answering is guaranteed. Therefore we also implemented a combination of CH-search and A*-search. Because A* is known to work better embedded in an unidirectional computation, we modified the approach by marking first all edges in $G^{\downarrow}(t)$ and then run CH-A*-search from s using edges in $G^{\uparrow}(s)$ and marked ones. This unidirectional variation of CH-search was used before on street graphs, e.g. for one-to-many queries [10] or when edge costs were given as functions complicating the backward search (see e.g. Batz et al. [11]). The results for our two search approaches are collected in Table 2, subdivided by input category. We observe a reduction for all inputs, but the speed-up is most significant for mazes and rooms. For these two, the number of expanded nodes by CH-A* is even below the optimal path size on average. This means that any Dijkstra-based path finding approach, which does not use a compressed path description, cannot expand fewer nodes than our method. For game maps we observed mixed results, some inputs responded very well (speed-up by two orders of magnitude) while the structure of other maps led to a large set of long disjoint optimal paths which is not beneficial for our approach. In Figure 5 positive and negative examples are shown. We expect better improvements for game maps when the CH-search is combined with other techniques developed for road networks, like e.g. partitioning [12]. For random maps, the speed-up increases with the sparseness of the grid as the A* baseline gets worse but our approach expands almost the same number of nodes for varying deletion ratios. The reduction of expanded nodes does not fully transfer into run time decrease, as we have some static overhead

Table 2. Experimental results for finding optimal paths in 8-connected grids with the basic map size being 512x512. The speed-up describes the ratio of expanded nodes by A* and CH-A*, the value in brackets equals the ratio of the respective runtimes. All values are averaged of 1000 random queries (10 maps for every category with 100 queries on each).

		# of expanded nodes			
input type	avg. path size	A*	CH-Dijkstra	CH-A* (uni)	speed-up
mazes	1,240	104,949	630	499	210 (198)
rooms	282	35,739	625	275	130 (67)
game maps	196	18,477	4,614	937	20 (7)
random	234	16,121	5,158	531	30 (10)

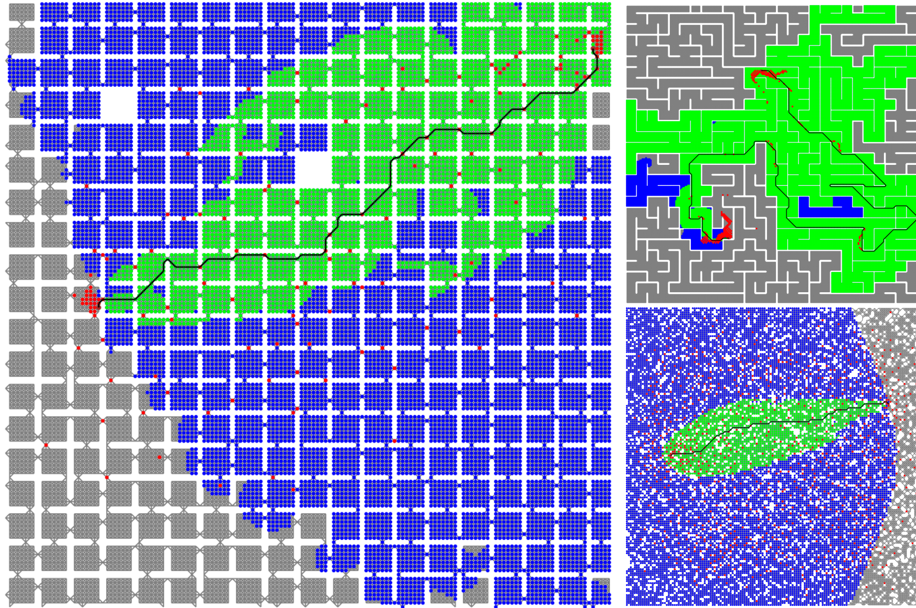

Fig. 4. Examples for rooms, mazes and random graphs. Nodes expanded by Dijkstra blue, by A* green and by the CH-Dijkstra red.

introduced by marking the edges in $G^{\downarrow}(t)$ and also path unpacking is included here. Nevertheless averaged over all instances we achieve a speed-up over 50.

3.3 Connections to other Speed-Up Techniques

Pochter et al. [6] introduced the concept of *swamp hierarchies* with a swamp being a set of nodes that can be excluded a priori from the search space for given s, t. In a CH-graph also the nodes $\notin G^{\uparrow}(s) \cup G^{\downarrow}(t)$ are pruned directly. So both approaches allow for the extraction of a smaller subgraph in which the optimal path must be contained. Nevertheless the kind of blocked nodes differ significantly, hence a combination of both approaches promises further improvement.

Other techniques based on map decomposition (and therefore requiring a preprocessing phase as well) were described by Björnsson et al. [9], in particular the *dead-end heuristic* and the *gateway approach*. In the latter, the graph is divided into subareas with a preferably small number of connections between adjacent ones. For these connections – the gateways – pairwise distances are precomputed. Looking at our results for the rooms instances, we observe that door nodes are naturally considered important in our CH-construction and direct shortcuts exist between almost any two of them. So in some way the *gateway heuristic* is automatically embedded in our approach.

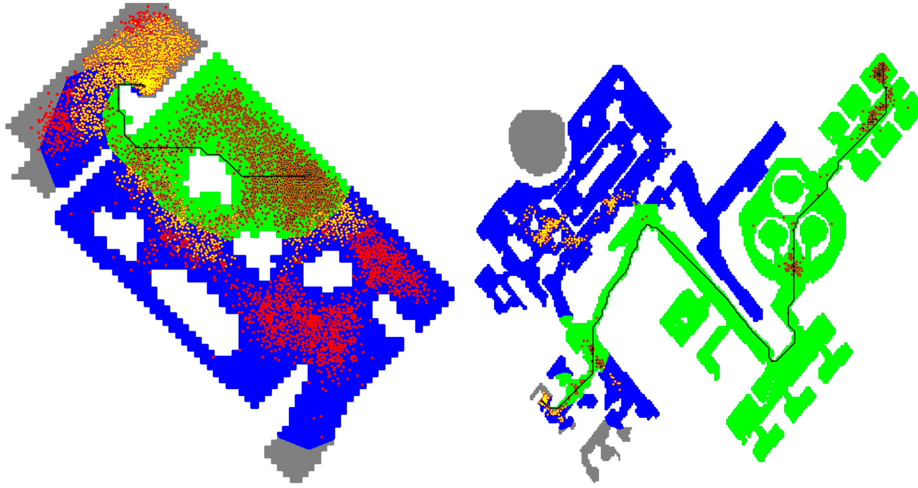

Fig. 5. CH-search on game maps, expanded nodes by Dijkstra (blue), A* (green), CH-Dijkstra (red), bidirectional CH-A* (yellow) and unidirectional CH-A* (black).

The idea of jump points was presented by Harabor et al. [5]. Here no preprocessing is necessary, but sets of nodes between on the fly computed jump points are removed from the search space. The pruning rules applied there have similar effects as a CH-search, namely that many nodes on shortest subpaths can be ignored – because they lie between two consecutive jump points or on the shortest path between two nodes in the CH which are directly connected via a shortcut. Moreover the jump points approach also computes canonical paths, hence it appeals that CH can be seen as kind of an offline jump points approach. But in contrast to their method we are not bound to uniform grid costs. Instead any kind of costs assigned to the edges are allowed, and also directed arcs can be taken into account.

4 Concluding Remarks

In this paper we presented modifications for the speed-up technique contraction hierarchies to work on grid graphs. Despite being developed for road networks, we showed that with minor changes an acceleration of shortest path queries by up to two orders of magnitude can be achieved when applying contraction hierarchies to instances of e.g. rooms or mazes. Moreover we developed an approach based on edge classifiers, which allows to retrieve optimal canonical paths, using only a constant time and space overhead.

Future work includes further reduction of the preprocessing time and evaluating with which other methods contraction hierarchy based search can be combined to accelerate query answering.

References

1. Geisberger, R., Sanders, P., Schultes, D., Delling, D.: Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In: WEA. (2008) 319–333
2. Bast, H., Funke, S., Matijevic, D., Sanders, P., Schultes, D.: In transit to constant time shortest-path queries in road networks. In: ALENEX. (2007)
3. Antsfeld, L., Harabor, D.D., Kilby, P., Walsh, T.: Transit routing on video game maps. In: AIIDE. (2012)
4. Bauer, R., Delling, D., Sanders, P., Schieferdecker, D., Schultes, D., Wagner, D.: Combining hierarchical and goal-directed speed-up techniques for Dijkstra's algorithm. ACM Journal of Experimental Algorithmics **15** (2010)
5. Harabor, D.D., Grastien, A.: Online graph pruning for pathfinding on grid maps. In: AAAI. (2011)
6. Pochter, N., Zohar, A., Rosenschein, J.S., Felner, A.: Search space reduction using swamp hierarchies. In: AAAI. (2010)
7. Botea, A., Müller, M., Schaeffer, J.: Near optimal hierarchical path-finding. Journal of game development **1**(1) (2004) 7–28
8. Harabor, D., Botea, A.: Breaking path symmetries on 4-connected grid maps. In: AIIDE. (2010)
9. Björnsson, Y., Halldórsson, K.: Improved heuristics for optimal path-finding on game maps. In: AIIDE. (2006) 9–14
10. Eisner, J., Funke, S., Herbst, A., Spillner, A., Storandt, S.: Algorithms for matching and predicting trajectories. In: Proc. of the 13th Workshop on Algorithm Engineering and Experiments (ALENEX), Citeseer (2011) 84–95
11. Batz, G.V., Delling, D., Sanders, P., Vetter, C.: Time-dependent contraction hierarchies. In: Proceedings of the 11th Workshop on Algorithm Engineering and Experiments (ALENEX09). (2009) 97–105
12. Delling, D., Goldberg, A.V., Pajor, T., Werneck, R.F.F.: Customizable route planning. In: SEA. (2011) 376–387