Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)

Convexity Hierarchies in Grid Networks

Johannes Blum, Ruoying Li, Sabine Storandt

University of Konstanz
{johannes.blum,ruoying.li,sabine.storandt}@uni-konstanz.de

Abstract

Several algorithms for path planning in grid networks rely on graph decomposition to reduce the search space size; either by constructing a search data structure on the components, or by using component information for A* guidance. The focus is usually on obtaining components of roughly equal size with few boundary nodes each. In this paper, we consider the problem of splitting a graph into convex components. A convex component is characterized by the property that for all pairs of its members, the shortest path between them is also contained in it. Thus, given a source node, a target node, and a (small) convex component that contains both of them, path planning can be restricted to this component without compromising optimality. We prove that it is NP-hard to find a balanced node separator that splits a given graph into convex components. However, we also present and evaluate heuristics for (hierarchical) convex decomposition of grid networks that perform well across various benchmarks. Moreover, we describe how existing path planning methods can benefit from the computation of convex components. As one main outcome, we show that contraction hierarchies become up to an order of magnitude faster on large grids when the contraction order is derived from a convex graph decomposition.

Introduction

Fast methods for path planning in large networks usually rely on some sort of preprocessing in which network information is gathered and stored. The main goal is to get a significantly reduced search space size when computing the shortest path between any source and target vertex. This is either achieved by computing information that lead to better heuristics to guide A* search or to restrict Dijkstra's algorithm to certain subgraphs, see e.g. (Rabin and Sturtevant 2016; Harabor and Stuckey 2018), or by encoding shortest path information in such a way that search on the input network is not needed anymore on query time at all. Examples for the latter paradigm are compressed path data bases (Shen et al. 2021) and labeling techniques (Delling et al. 2014).

One recurring ingredient in the preprocessing phase of various path planning algorithms is graph decomposition. Splitting the graph into subgraphs may allow to completely disregard certain parts of the graph. This idea is reflected e.g. in the *dead-end* heuristic (Björnsson and Halldórsson 2006) and in the notion of *swamps* (Pochter et al. 2010), which both are capable of reducing the number of nodes expanded by A* significantly. Another way to exploit subgraphs is to precompute certain shortest path information for each and then combine those to accelerate path planning. One way to do so is to precompute the shortest paths between all border nodes of each subgraph and then build an overlay graph connecting those (Delling et al. 2017), or to globally precompute shortest path distances between all border nodes and use those to get a highly informed *gate-way* heuristic (Björnsson and Halldórsson 2006). Furthermore, hierarchical techniques as *contraction hierarchies* (Geisberger et al. 2012) may be constructed based on recursive graph decomposition, which results in provable search space size guarantees (Bauer et al. 2016) as well as practical benefits (Dibbelt, Strasser, and Wagner 2016).

While different decomposition techniques are used in these preprocessing methods, the goals often align: Subgraphs (on one hierarchical level) should be of roughly the same size and the number of their border nodes should be small. In this paper, we enforce an additional subgraph property, namely *convexity*. It implies that for all node pairs in a subgraph, the shortest path between them never leaves said subgraph. Björnsson and Halldórsson (2006) claimed that a convex decomposition would benefit the gate-way heuristic. However, their proposed decomposition algorithm does not guarantee subgraph convexity. We will introduce several heuristics that are capable of separating a given grid network into convex components and discuss how existing path planning techniques can benefit from convex decompositions.

Related Work

Graph partitioning has a wide range of applications, including scientific computing (George 1973), VLSI design (Bhatt and Leighton 1984), task scheduling (Ababei et al. 2002), and communication networks (Lee et al. 2018). The idea is to split the graph into several pieces that are not connected to each other. This can be achieved by removing a set of vertices (a *vertex separator*, or simply *separator*) or edges (an *edge cut*). In most cases it is desirable that the remaining parts are of (roughly) the same size and that the number of removed vertices or edges is small. In this work, we focus on vertex separators. Note that for every edge cut, the set

of vertices incident to a cut edge yields a vertex separator. In general however, vertex separators can be significantly smaller than edge cuts. Formally, a vertex separator or edge cut of a n-vertex graph is α-balanced for $\alpha \in (0, 1)$, if after its removal there remain only connected components of size at most $\alpha \cdot n$. Computing an α-balanced separator of minimum size is NP-hard (Bui and Jones 1992). Therefore a lot of work has been carried out to develop approximation algorithms. In their seminal work, Leighton and Rao (1988) showed that a $\mathcal{O}(\log n)$-approximation can be computed in polynomial time, which comes with an arbitrary small loss in the balance factor. Later on, Feige and Mahdian (2006) showed how to compute an α-balanced separator of size k for any $\alpha \geq 2/3$ (if it exists), unless there is a $(\alpha + \epsilon)$-balanced separator of size $k' < k$, in which case the latter is found. Their algorithm has a runtime of $n^{\mathcal{O}(1)} \cdot 2^{\mathcal{O}(k)}$, which is polynomial for $k \in \mathcal{O}(\log n)$.

For planar graphs, the planar separator theorem of Lipton and Tarjan (1979) states that there is always a $2/3$-balanced separator of size $\mathcal{O}(\sqrt{n})$, which can be found in linear time. However, the constructed separator does not need to be of minimum size. Amir, Krauthgamer, and Rao (2003) developed a constant factor approximation for the minimum balanced separator of planar graphs, which comes with a small loss in the balance factor. Interestingly, it is not known whether computing a minimum balanced separator is NP-hard on planar graphs (Fukuyama 2006).

There exists a multitude of heuristic approaches, which produce good balanced separators in practice. Many of them were designed to find small separators in road networks, which we also expect to work well on grid graphs. Examples are FlowCutter (Hamann and Strasser 2018), which solves a sequence of incremental flow problems, and Inertial Flow (Schild and Sommer 2015), which computes a flow between the "first" and "last" vertices, based on a given embedidng. InertialFlowCutter (Gottesbüren et al. 2019) combines both of these ideas. PUNCH (Delling et al. 2011) and Buffoon (Sanders and Schulz 2012) are multi-step heuristic designed to find natural cuts in road networks. Moreover, there exist also several partitioning heuristics for general graphs, such as METIS (Karypis and Kumar 1998), SCOTCH (Pellegrini and Roman 1996), or KaHIP (Sanders and Schulz 2013).

Contribution

In this paper, we first introduce the notion of balanced convexity separators and convex graph decompositions. We show that computing a balanced convexity separator of minimum size is NP-hard. Then we propose different heuristics to efficiently find small balanced convexity separators in a given grid network. As one application of convex graph decomposition, we propose a variant of the contraction hierarchies shortest path algorithm, which allows to prune the search space during a query by exploiting convexity of induced subgraphs. We evaluate our algorithms on different 2D pathfinding benchmark sets and show that on instances that have small balanced convexity separators, our convex CH variant outperforms classic CH by up to an order of magnitude.

Preliminaries

We now formally define the notions of convexity and separators that will be used in the remainder of the paper. Throughout the paper, we assume the input is given as a grid domain, that is, a map discretized into grid cells where some cells are traversable (or free) and others may be not (obstacles). From that we construct a 4-connected grid network that reflects for each cell which neighboring cells are directly reachable. All edge costs are uniformly set to 1.

Convexity

The central graph property that we will study in the context of grid networks is subgraph convexity.

Definition 1 (Convex Subgraph)**.** *Let $G = (V, E)$ be a graph and $W \subseteq V$ a vertex set. The induced subgraph $G[W]$ is convex if for all $s, t \in W$ the shortest path in $G[W]$ has the same length as in the original graph $G = (V, E)$.*

Note that there also exists a stricter definition of convexity in which the subgraph needs to contain *all* shortest paths between every s-t-pair and not only *some* shortest path. Still, as our focus will be on path planning applications where it usually suffices to detect some shortest paths, we will use this weaker definition of convexity throughout the paper. Moreover, the whole graph $G[V]$ is convex by definition.

Balanced Separators and Graph Decomposition

To decompose a graph into smaller parts, we can either remove edges or vertices. We will focus here on the latter, i.e. on vertex separators. The goal is to remove a small number of vertices such the size of every resulting connected components is below a certain threshold. This goal is captured by the notion of an α-balanced vertex separator.

Definition 2 (Balanced Separator)**.** *Given a graph $G = (V, E)$ and $\alpha \in (0, 1)$, an α-balanced separator of G is a vertex set $S \subseteq V$ such that every connected component of $G[V \setminus S]$ has size at most $\alpha \cdot |V|$.*

In a hierarchical graph decomposition of $G = (V, E)$, we first find a separator S in the original graph G and then recursively do the same for the connected components of $G[V \setminus S]$. Figure 1 shows an example of recursive decomposition of a grid graph. The corresponding structure can be formalized as a decomposition tree.

Definition 3 (Decomposition Tree)**.** *Given a graph $G = (V, E)$ and $\alpha \in (0, 1)$, an α-balanced decomposition tree T is a rooted tree whose nodes are disjoints subsets of V. The root of T is an α-balanced separator of G whose removal results in connected components $G_1, \ldots, G_d$ with $d \geq 2$. The children of the root are the roots of decomposition trees of the individual components. The leaves of T correspond to subgraphs of G of size one (or some other cut-off threshold).*

We now strive to combine the concepts of graph decomposition and convexity by demanding that every connected components resulting from the removal of a separator is a convex subgraphs of G. We call such a separator also a *convexity separator*. Moreover, we will refer to the resulting structures as *convex graph decompositions* and *convex decomposition trees*, respectively.

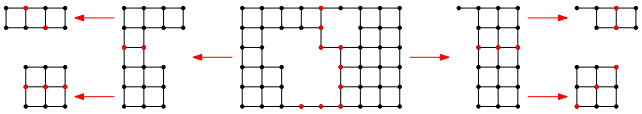

Figure 1: First three levels of a (convex) grid network decomposition. Seperator nodes are marked red.

Balanced Convex Decompositions

In this section, we first study the complexity of computing an optimal α-balanced convex graph decomposition on general graphs. Afterwards, we devise efficient heuristics to construct a convex decomposition tree on grid networks.

NP-Hardness

We now prove that it is NP-hard to compute an optimal convex graph decomposition, in particular to find an α-balanced convexity separator of minimum size. To that end we reduce from BALANCED COMPLETE BIPARTITE SUBGRAPH (BCBS). In this problem, the input is a bipartite graph $G = (V \uplus U, E)$ and a positive integer k, and the question is whether there are two sets $V' \subseteq V$ and $U' \subseteq U$ such that $|V'| = |U'| = k$ and such that their union induces a complete bipartite graph. The proof of our theorem follows the hardness proof for balanced separators of Müller and Wagner (1991), which however is not concerned with convexity and only considers $\alpha = 1/2$.

Theorem 4. *It is NP-hard to decide whether a given graph has an α-balanced vertex separator of size at most b, which splits the graph into a set of convex components.*

Proof. Consider an instance of BALANCED COMPLETE BIPARTITE SUBGRAPH, which is NP-hard (Garey and Johnson 1979). Such an instance is given by a graph $G = (V \uplus U, E)$ and an integer k. Let H be the complement graph of G, let $n = |V| + |U|$, and choose α such that $\alpha n = k$.

We prove that (G, k) is a yes-instance if and only if H has an α-balanced convexity separator of size $n - 2k$. Suppose that there is a set $S \subseteq V \cup U$ of size $|S| \leq n - 2k$, which separates H into a set of convex connected components of size at most αn. By construction, $V' = V \setminus S$ and $U' = U \setminus S$ each induce a clique. This means that removing S from H results in the connected components V' and U', as otherwise there would be one single connected component of size $2k > \alpha n$. As we have $|V'| \leq k, |U'| \leq k, |S| \leq n - 2k$, and $|V'|+|U'|+|S| = n$, it follows that $|V'| = |U'| = k$. This means that $U' \cup V'$ induce a balanced complete bipartite subgraph in G.

Suppose now that there are sets $V' \subseteq V$ and $U' \subseteq U$ of size $k = \alpha n$ each, whose union induce a complete bipartite graph in G. It holds that in H, V' and U' induce a clique, i.e., a convex subgraph. This means that $S = (V \cup U) \setminus (V' \cup U')$ is an α-balanced convexity separator of size $n - 2k$. □

Heuristics for Balanced Convex Decompositions

Next, we try to exploit the structure of grid networks to efficiently compute convex decompositions of good quality, i.e., with small balanced convexity separators on each level.

Guess & Check Any efficient method to decide whether a vertex set S is an α-balanced convexity separator for a given graph $G = (V, E)$ and a certain balance-factor $\alpha \in (0, 1)$, allows us to search for feasible convexity separators by exploring vertex sets in a systematic manner. Computing the maximum α for which S is an α-balanced separator can be accomplished in linear time by determining the connected components $C_1, \ldots, C_d$ of $G[V \setminus S]$ and comparing their size to the original graph size, i.e. $\alpha = \max_{i=1}^{d} |C_i|/|V|$. But checking whether all C_i are convex is more demanding, as we have to ensure that the shortest path distance between all pairs $s, t \in C_i$ is the same in $G[C_i]$ and in G. Clearly, it suffices to check this property for all pairs of border nodes of C_i, i.e., the nodes of C_i that are adjacent to S. Still, we need two one-to-many comparative Dijkstra runs for every border node (one in $G[C_i]$ and one in G). Thus the checks become expensive if separator nodes have many neighbors or if the network is big. In the following, we will show that for particular choices of S, faster checks are possible.

Separators from One Shortest Path A candidate for a convexity separator S is the vertex set of a separating shortest path.

Definition 5 (Separating Shortest Path)**.** *A shortest path p in a connected graph $G = (V, E)$ is called separating if $G[V \setminus p]$ consists of more than one connected component.*

Observe that in a planar graph, any shortest path, that does not only contain boundary nodes, is separating. Given a grid network, we can compute separating paths by first extracting the boundary nodes of the grid network and then computing shortest paths between all of them (or a sample thereof). Among the paths whose vertex set is a valid α-balanced convexity separator, we pick one of minimum size.

The idea is that separating shortest paths are likely to produce convex connected components. Still, this is not necessarily the case. Thus, we need to perform a convexity check for each such path as described above. However, the check might be skipped if the path does not go along the borders of any obstacle in the grid domain as shown in the next lemma.

Lemma 6. *If except for start and end node, each node in a separating shortest path p in G has degree 4, then all connected components in $G[V \setminus p]$ are convex.*

Proof. Assume for contradiction that there exists a connected component C in $G[V \setminus p]$ that is not convex and consider vertices $s, t \in C$ such that in G, all shortest s-t-paths traverse vertices outside of C. Consider any such shortest path p' and let v be the first vertex on that path that intersects p, and w the last. Further let u be the direct predecessor of v in p' and x the direct successor of w. As p is a shortest path, the subpath between v and w on S is a shortest path as well. Now we consider the path from u to x that runs parallel to p, i.e. along the border of C with p. This path has to exist, as all nodes in p between v and w have degree 4 which implies that in the grid domain all of their neighboring cells are obstacle-free. This observation also directly entails that p has to be both x- and y-monotone in the grid domain. Finally, we observe that the path from u to x parallel to the path from v to w (which then also has to be monotone) has

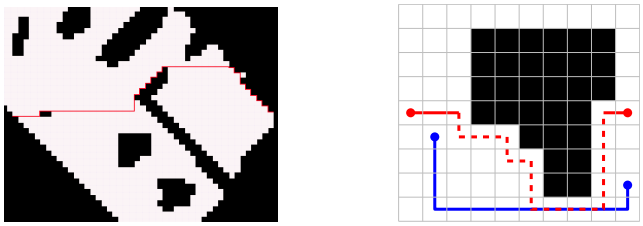

Figure 2: Left: A grid network where convexity checks are not necessary for the upper connected component due to Observation 7. Right: If the red separating path gets reduced (i.e. the dashed part is deleted), then the connected component containing the blue path becomes convex.

cost $c(u, x) \leq c(v, w) + 2$. This applies as in any monotone path in a grid domain the direction changes along the path to left and right alternate and hence the total sum of left and right turns differs by at most one. For the parallel path either the left turns safe a node and the right turns add a node or vice versa. An additional node increases the path cost by 2. Hence the total path costs of two parallel monotone paths in a grid domain cannot differ by more than two. Thus, the path from u to x via v and w has cost $2 + c(v, w) \geq c(u, x)$. Accordingly, the path between u and x inside C is not longer than the one leaving it. This contradicts the assumption that the shortest path from u to x (and therefore the shortest path from s to t) has to go outside of C. Thus, C is convex. □

However, especially in grid domains with many obstacles, separating shortest paths surrounded by only free cells might not occur. Nevertheless, we might still benefit from the insights provided in the lemma above for individual components as described below.

Observation 7. *Let p be a separating shortest path in G and C a component in $G[V \setminus p]$. Let B be the set of vertices in C that are adjacent to p in G. If the nodes in B form a single monotone path, C is convex.*

An example where Observation 7 applies is provided in Figure 2 (left). Reasons why the neighbors of p in component C might not fulfill the property described in the observation are that either there are obstacles such that the nodes in B form multiple paths in C or that p itself is not monotone due to obstacles. However, in the latter case, we can always reduce the path p to a path p^* where we keep only the nodes of degree 4 and for each subsequence of nodes of smaller degree only the first and the last. This reduction removes all parts from p that are parallel to obstacle borders and thus do not contribute to the separation at all (as among their three or less neighbors, two have to be in p and so at most one neighbor is free). Accordingly, p^* is still a separator in G. Note that the start and end node of p might be removed as well in case they do not have neighbors in more than one component. As we aim for small separators, the reduction from p to p^* is clearly beneficial. It furthermore can help to get convex components as shown in Figure 2 (right).

Separators From Two Shortest Paths While separating the grid network with a single (reduced) separating shortest path is a promising approach to find small convexity separators, convexity checks are still required. Moreover, there could also be no separating shortest path with the desired balance factor α. We hence discuss now how to potentially benefit from computing two separating shortest paths.

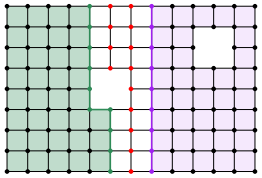

Figure 3: Grid network with two separating shortest paths (green and purple nodes). The red nodes enclosed by both paths form a convexity separator as their removal results in two convex connected components.

Lemma 8. *Let G be a connected graph and let p_1, p_2 be two separating shortest path such that no vertex of p_1 has a neighbor in p_2. Let S be the union of all connected components of $G[V \setminus (p_1 \uplus p_2)]$ that have neighbors in both p_1 and p_2 in G. Then S is a convexity separator of G.*

Proof. Consider any separating shortest path p and a connected component C in $G[V \setminus p]$. Then $C \cup p$ is always convex, as any shortest path that exits and reenters the node set has to do that via two vertices in p; and between those the shortest path is simply the respective subpath of p. Now if we consider any two separating shortest paths p_1, p_2, where there are no vertices in one path with a neighbor in the other, then all components C in $G[V \setminus (p_1 \uplus p_2)]$ that only have neighbors in p_1 become convex if the vertices of p_1 are added to the component and the same applies for components only neighboring p_2 when adding the vertices of p_2. Thus, if we remove all components that have neighbors in both but let the vertices in p_1 and p_2 remain in the graph, then the resulting graph consists of at least two connected components and all connected components are convex. □

The lemma can be applied to any 4-connected grid network, as they are planar by construction. Figure 3 shows an example of a convexity separator enclosed by two shortest paths. The question is now how to compute separating shortest paths with the desired properties efficiently. Of course, we can extent our approach to find a single separating shortest path by testing all pairs of them. However, this might be quite time-consuming. Therefore, we now describe a heuristic that aims at identifying such paths faster.

The underlying idea of our heuristic is that it would be most beneficial in terms of separator size if S is sandwiched by both p_1 and p_2, i.e., each vertex of S has a neighbor in both p_1 and p_2. Hence if we find three neighboring parallel paths where the outer two ones are shortest paths, we would be done. To find three parallel neighboring paths efficiently, we first apply a compression step to our input map. Here we consider disjoint chunks of 3×3 grid cells and convert them into a single grid cell. The new cell is traversable if and only if all nine underlying cells are. Then in the new graph, we simply search for a single shortest separating path that

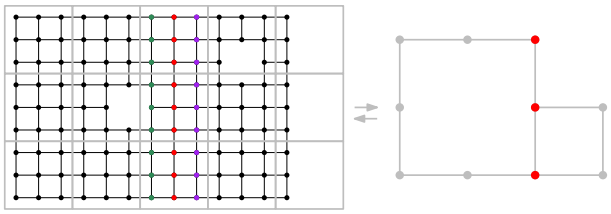

Figure 4: Left: Input grid network with coarser cells indicated by gray lines. Right: Compressed network with separating shortest path. After decompression, the red node set in the left image is identified as suitable convexity separator.

is balanced and convex. If such a path is found, we decompress it into three parallel paths in our original input grid map and recheck whether the two outer paths are also separating shortest paths in there. If that applies, we can output the path inbetween as convexity separator (see Figure 4).

Separators by Hole Cutting We observe that convexity checking often fails when the grid has internal obstacles (holes). Note that the shortest path between two nodes next to a convex internal obstacle usually follows the obstacle's border. Assume the shortest path is forced to go around an internal obstacle and follows its border in such a way that the shared part of the shortest path and the internal obstacle's border is less than half the border length. In this case, removing the shortest path may cause a nonconvex partition, since the shortest path between nodes on the part of the border which is not covered by the separating path may overlap with the removed path. Therefore, we attempt to cut all internal obstacles into four approximately equal parts by removing all nodes on lines parallel to the coordinate axes crossing the hole in the middle of the height or width of the obstacle's rectangular outer frame. Then we apply the balance and convexity check in order to verify the validity of the computed cut.

Algorithm Overview Finally, we present our whole pipeline that aims at identifying an α-balanced convexity separator S in a given grid network G: We start by running the KaFFPa algorithm from KaHIP (Sanders and Schulz 2013) on the input network, which is designed to find small balanced separators, but does not take edge weights or shortest path structures into account. If this does not result in the desired outcome, we use the separating shortest path technique described above with the improved convexity check methods. If this step is not successful either, we search for two shortest separating paths with the 3×3 compression method described above. If not successful, we move on to testing simply each row and column of the grid domain individually as candidate node set S. Except for the first and last row or column with traversable cells, these are clearly all separators, but balance and convexity needs to be checked. In the end, we apply hole cutting. If none of the proposed methods work, we return $S = V$. Note that V indeed always constitutes an α-balanced separator with $\alpha = 0$, but of course this is not in line with our goal of finding a small set S that decomposes the network into multiple components.

Convex Contraction Hierarchies

As one application of convex graph decomposition, we will construct a contraction hierarchy (CH) based on a convex decomposition tree. CH was originally developed to accelerate shortest path planning in road networks (Geisberger et al. 2012), but was also demonstrated to work well on grids (Storandt 2013). We will first briefly recap the general CH concept and then describe how it can be constructed upon and benefit from a given convex graph decomposition.

Classic CH

Given a weighted graph $G = (V, E)$ with edge costs $c\colon E \to \mathbb{R}^+$, a CH data structure is supergraph of G characterized as follows. Let $\pi : V \to \{1, \dots, n\}$ denote a permutation of the vertices, also referred to as contraction order. In the preprocessing phase, the vertices are contracted one-by-one in the given order. The contraction of a node v consists of the removal of v and its incident edges from the graph, and inserting so called shortcut edges between neighbors u, w of v in case the shortest path distance between them would otherwise increase after the contraction of v. The cost of the shortcut equals the shortest path distance between u and w. After all vertices have been contracted, the CH-graph $G^+ = (V, E^+)$ is constructed by augmenting G with all shortcuts that were created during the contraction phase. A shortcut (or original edge) is said to go upward from v if $\pi(v) < \pi(w)$ and downward from v otherwise.

In a CH graph G^+, for any pair of vertices $s, t \in V$ there exists a shortest path $s = v_0, v_1, \dots, v_\ell = t$ with the same cost as a shortest s-t-path in G, such that all edges $\{v_i, v_{i+1}\}$ are upward from v_i up to a certain value of i and thereafter all downward from v_i. This allows for fast shortest path computation with a bidirectional Dijkstra run from s and t, which only relaxes upward edges from the current vertex.

The search space size and thus the query time crucially depend on the chosen contraction order. Identifying an optimal contraction order poses an NP-hard problem (Milosavljević 2012). The most successful heuristic for determining a good contraction order is based on the notion of *edge difference* (ED). The ED of a vertex v is the number of edges that will be inserted upon contraction of v minus the number of currently incident edges. Contracting vertices with small ED first is supposed to keep the number of shortcuts in G^+ low, which is beneficial for both space consumption and query times. Note that the ED values of the neighbors of the contracted vertex have to be updated to stay truthful. While ED-based contraction orders work remarkably well in practice, a contraction order based on balanced separator decompositions was proposed by Bauer et al. (2016), originally to study theoretical properties of CH. It was later confirmed that this also leads to competitive results in practice (Dibbelt, Strasser, and Wagner 2016). In the following, we will discuss decomposition-based contraction in more detail and in particular the impact of a convex decomposition.

CH on Convex Separator Decompositions

Given a (convex) decomposition tree of the input graph $G = (V, E)$, the CH-graph $G^+ = (V, E^+)$ is constructed

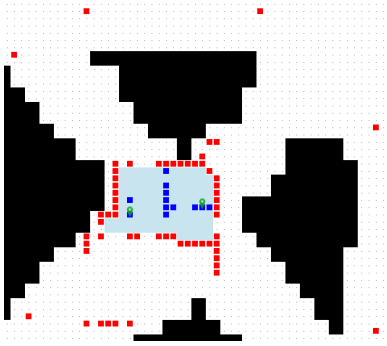

Figure 5: Search space of an s-t-query with restriction to the convex subgraph G_S (blue nodes; size 12) and without restriction (blue and red nodes; size 72). The subgraph G_S is indicated by the light blue area.

by contracting vertices based on a bottom-up traversal of the tree. This means that the vertices contained in the top-level separator are contracted last, the vertices in the second-level separators second to last, etc. Within every separator, vertices might be contracted in arbitrary order. In our implementation, we order the vertices within one separator by ED.

Consider now a convex graph decomposition T. For clarity, we call a separator of T also a *bag*. We can associate every bag S with the subgraph G_S induced by S and its descendants in T. It follows immediately that for every bag S of T, the subgraph G_S is convex.

Given two vertices s and t, we can use the standard CH query algorithm in order to compute a shortest s-t-path in G^+ (and thus in G), i.e., we perform a bidirectional run of Dijkstra's algorithm, which only follows edges pointing upwards in the vertex hierarchy. However, we can observe that it suffices to restrict the query to the smallest convex subgraph G_S that contains both s and t. As Figure 5 shows, this restriction to G_S can lead to a substantial reduction of the search space, if s and t are close to each other in the decomposition tree and G_S is small. It is easy to see that the separator S associated with G_S is the lowest common ancestor of the bags S_s and S_t containing s and t, respectively, in the convex separator decomposition. Therefore, when performing an s-t-query, we first identify the bags S_s and S_t and compute their lowest common ancestor S. In the subsequent execution of Dijkstra's algorithm we have to efficiently check for every considered edge whether it leaves the subgraph G_S. To that end we proceed as follows. In the beginning of the preprocessing we rename the vertices with numbers from 1 to n such that for every bag S, the vertices of the associated induced subgraph G_S form a contiguous interval. This can be achieved through a post-order traversal of the convex separator decomposition and visiting the vertices within every bag in arbitrary order. Then we store for every bag S the interval the corresponds to the vertices of G_S. Throughout the execution of Dijkstra's algorithm we can now simply check in constant time whether a considered edge leaves G_S or whether it needs to be considered.

Further Applications

There are further planning technqiues that might benefit from a convex separator decomposition. While for CH we explicitly used the hierarchical decomposition to guide the preprocessing, in other approaches we might only need a single separator that splits the graph into a set of convex connected components. Given some upper bound U on the component size, we can easily extract a suitable separator from a given convex separator decomposition by cutting off the lower levels of the decomposition tree that separate components which are already smaller than U, and then taking the union of all nodes in the levels above. Note that cutting at any level still yields a set of convex connected components.

For the implementation of the gateway heuristic (Björnsson and Halldórsson 2006), a decomposition method based on edge cuts was described. The boundaries between different areas are called gateways. In the preprocessing phase, distances between gateways are precomputed. In fact, four different distances are stored, depending on whether the gateway will be used to enter or exit the area and whether passing through the area is allowed or not. That information is used on query time to guide A* search. Now, instead of determining the areas based on edge cuts in the corresponding grid network, one could also use a trimmed convex tree decomposition as discussed above. Then, the gateways would be formed by the separator vertices between two areas and would hence serve both of them at the same time, potentially decreasing the overall space consumption. Furthermore, if we decompose a grid network with a separating shortest path, the resulting components are not necessarily convex (as discussed above) but each component *combined* with the separating shortest path is indeed convex. Thus, if we split the network recursively by separating shortest paths and use them also to define the gateways, we expect to get quite accurate distance lower bounds. This observation could also be beneficial for other methods that use shortest path distances to separators to inform A* search, e.g., the differential separator heuristic described by Chen and Gotsman (2021).

Experimental Results

We implemented the previously described decomposition and planning algorithms in C++ and performed experiments on a single core of an AMD Ryzen 7 3700X processor (clocked at 3.6 GHz) with 128 GB main memory. KaHIP 2.10 was compiled using Clang 11.0.0 with OpenMPI 1.4.1.

Data Sets

We evaluated our algorithms on different 2D pathfinding benchmark sets from the Moving AI map repository (Sturtevant 2012). Experiments were conducted on both artificial and non-artificial maps, in particular on the benchmark sets 'Baldurs Gate II' (scaled to 512×512), 'street maps', 'mazes', 'random maps', and 'room maps'. Furthermore, we generated a benchmark set 'pixel'[1] by taking the pixels of six images as grid points. The number of instances and the number of nodes of each benchmark set is shown in Table 1.

[1] available at https://doi.org/10.5281/zenodo.7733466

Benchmark	#instances	min. $\|V\|$	max. $\|V\|$
Baldurs G. II	75	17 587	231 469
mazes	60	131 071	232 929
room maps	40	206 527	254 122
random maps	70	53 642	235 910
street maps	90	47 331	798 264
pixel	6	26 507	4 347 591

Table 1: Sizes of the used benchmark sets.

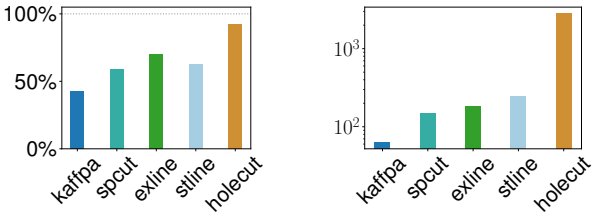

Figure 6: Left: Success rates of heuristic algorithms in finding a $2/3$-balanced convexity separator for the 75 instances of 'Baldurs Gate II'. Right: Average separator sizes in the 18 instances of 'Baldurs Gate II', on which all algorithms found a valid convexity separator. The used heuristics were KaFFPa (algorithm for balanced min cut node partition from KaHIP), spcut (separators from single shortest path), exline (separators from two shortest paths), stline (separators from a row or column), and holecut (separators by hole cutting).

In every instance we have $|E|/|V| \approx 2$, except for a few 'mazes' instances where $|E|/|V| \approx 1$.

Evaluation of Decomposition Methods

We first evaluated the performance of each heuristic algorithm for determining the first layer α-balanced convexity separators, where $\alpha = 2/3$. In the implementation of separators from a single shortest path, we compare the differences in the 'Baldurs Gate II' instances with and without pruning the nodes on the shortest path that do not contribute to the partition graph. The experimental results show that the method of the trimmed shortest path finds a balanced convexity separator on the first level on four more instances than the untrimmed shortest path within a time out of 60 seconds. Additionally, the separator sizes are smaller on almost all instances. Figure 6 shows that most of the separators found by KaFFPa in the 'Baldurs Gate II' instances do not guarantee convexity, but have the smallest size among the qualified convexity separators compared to other algorithms. The first four algorithms are able to find much smaller separators than hole cutting. However, hole cutting finds valid convexity separators in almost all instances. It only fails in six out of 75 instances due to the absence of internal obstacles in these six instances. The other four algorithms found valid convex separators in these six instances easily. We design our decomposition pipeline based on the size of the separator, and

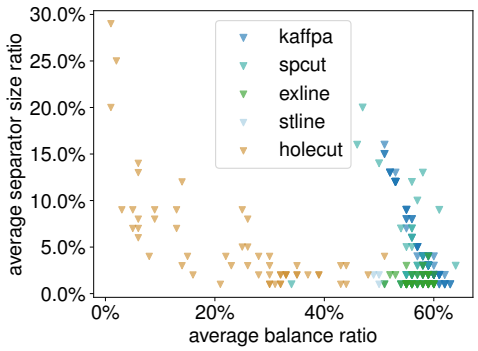

Figure 7: Average balance ratio and relative separator size for the top five levels of each convex separator decomposition for Baldurs Gate II.

the algorithm runs in the order KaFFPa, spcut, exline, stline and holecut. Each method has a timeout of 60 seconds. It stops once a balanced convexity separator is found.

The pipeline is able to find a valid $2/3$-balanced convex decomposition, whose leaves have size at most 10, for every instance in our benchmark sets. On average it takes three minutes to construct the convex decomposition for 'Baldurs Gate II' and 'maze', five minutes for 'room maps', eight minutes for 'random maps', and more than 20 minutes for 'street maps' and 'pixel'. KaFFPa found valid balanced convexity separators in an average of over 90% of the decomposition subgraphs per instance.

Figure 7 shows the average ratio of the size of every separator and the size of the subgraph it separates, the average balance factor, both for the top five levels of each convex separator decomposition for 'Baldurs Gate 2'. We can observe that the size of the separators found by hole cutting is relatively large and splits the graph into smaller partitions. In contrast, the other methods find relatively small separators.

On the pixel instances, the pipeline could, despite comparably long running times, always find a small balanced convexity separator. In over 97%, this separator could already be found by KaFFPa. In Figure 8, separators of the top five levels on the world map is illustrated. Since the world map only contains a small amount of holes of small sizes, the separator can in almost all cases be chosen as a straight line cutting the current component through its center.

Evaluation of Convex Contraction Hierarchies

We evaluated the described convex CH approach on the five different Moving AI benchmarks. For every instance, we constructed a classic CH with an ED-based contraction order, and a convex CH based on a convex separator decomposition computed with the previously described pipeline. Within every bag of the decomposition, we used ED to determine the contraction order. We first describe our results for 'Baldurs Gate II'. Given the separator decomposition as an input, we could compute the CH graph of every instance in at most 31 seconds. Our convex CH approach introduced more shortcuts than classic CH: On average, the ratio E^+/E be-

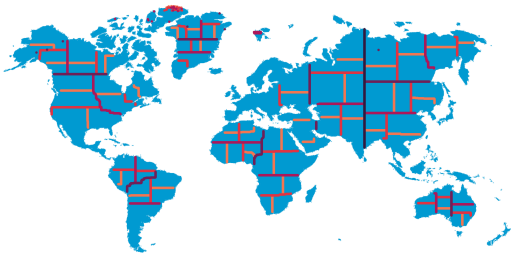

Figure 8: $^2/_3$-balanced convexity separators on the top five levels in a pixel instance.

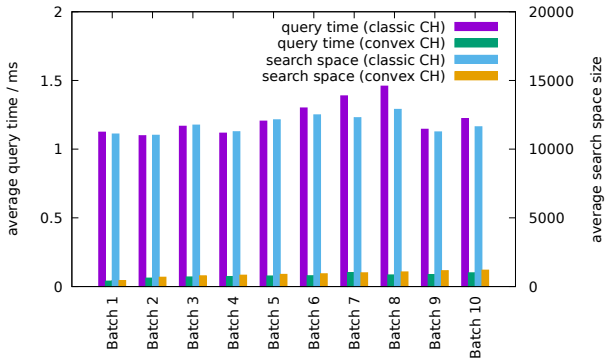

Figure 9: Query times and search space sizes for classic and convex CH on an instance of Baldurs Gate II.

tween the number of CH edges and original edges was about 1.5 for classic CH, whereas it was about 1.9 for convex CH. Still, despite introducing more shortcuts, the convex CH outperforms the classic approach w.r.t. query performance. For every instance, we performed 10 000 shortest path queries between randomly chosen vertex pairs. We grouped these queries into 10 batches, where the first batch contains the 1000 queries with the smallest distance, and the last batch contains the 1000 queries between the furthest query pairs. By doing so, we aim to measure the impact of restricting every query to a convex subgraph, as the shortest path distance between two vertices correlates with their distance in the convex separator decomposition. On average the convex CH queries were a factor of 8.6 faster than then classic CH queries. With increasing query distance, the speedup dropped: For the first batch, the average speedup was 14.8, whereas for the last batch it was 6.6. This is also illustrated in Figure 9, which contains a plot of search space sizes and query times of an instance with a speedup of 68.7 for the first batch.

Note that in our experimental setup, the CH query algorithm always explores the full search space, and does not stop once the shortest path is found, in order to measure the pruning effect of our approach. When we added such a stopping criterion, convex CH still outperformed classic CH, but the average speedup dropped to 6.2.

Table 2 also shows the preprocessing times and query

Benchmark	preprocessing		query speedup		
	conv.	trad.	avg.	min.	max.
Baldurs G. II	31 s	2520 s	8.6	1.9	68.7
mazes	9 s	406 s	3.3	1.0	12.7
room maps	13 s	293 s	1.6	0.6	4.3
random maps	187 s	276 s	0.9	0.6	1.2
street maps	387 s	15 483 s	23.3	0.7	149.2

Table 2: Maximum preprocessing time and query speedup of convex CH and classic CH for different benchmark sets. Note that the preprocessing time for convex CH does not include the time to compute the separator decomposition.

speedups on the remaining benchmark sets. On every benchmark set, our approach significantly outperformed the classic approach w.r.t. preprocessing time. From the street maps benchmark set we only evaluated 51 of the 90 instances, as the classic preprocessing took up to several hours for larger instances. On every benchmark, the convex CH approach introduced more shortcuts than classic CH, comparable to the results on 'Baldurs Gate II'. On 'mazes', 'room maps', and 'street maps', our approach lead to average query speedups between 1.6 and 23.3, the best speedup with a value of 149.2 was achieved on an instance of 'street maps' for batch 1. On some instances of 'mazes', convex CH did not provide any speedup over classic CH. However, in the respective instances, the ratio $|E|/|V|$ was almost 1, so both approaches exhibited very small search spaces and the search space pruning of convex CH had no big impact. On some instances of 'room maps' and 'random maps' as well as one instance of 'street maps', our pipeline only found relatively large convexity separators and as a consequence the queries of convex CH were actually slightly slower than classic CH. But on average across grid domains, we observe that exploiting a convex decomposition is clearly worthwhile in terms of preprocessing and query time.

Conclusions and Future Work

We have demonstrated that grid networks can be recursively decomposed into convex components via carefully chosen balanced separators. Furthermore, we have shown that contraction hierarchies benefit from a given convex decomposition and the explicit knowledge that the search space can be restricted to the smallest convex component that contains both source and target. The proposed methods can also be applied to planarized octile grids (note that planarization does not change the shortest path structures in such networks at all). A more challenging task would be to transfer the proposed concepts to road networks, though. Here, the techniques custom-tailored to grids (as e.g. the straight line method) are not applicable and separating shortest paths are more difficult to find as road networks are not necessarily planar (and planarization here indeed changes shortest paths). However, it would be very interesting to see whether the speed-ups achieved for contraction hierarchies in grids would also manifest in road networks.

References

Ababei, C.; Selvakkumaran, N.; Bazargan, K.; and Karypis, G. 2002. Multi-objective circuit partitioning for cutsize and path-based delay minimization. In Pileggi, L. T.; and Kuehlmann, A., eds., *Proc. ICCAD 2002*, 181–185. ACM / IEEE Computer Society.

Amir, E.; Krauthgamer, R.; and Rao, S. 2003. Constant factor approximation of vertex-cuts in planar graphs. In Larmore, L. L.; and Goemans, M. X., eds., *Proc. STOC 2013*, 90–99. ACM.

Bauer, R.; Columbus, T.; Rutter, I.; and Wagner, D. 2016. Search-space size in contraction hierarchies. *Theoretical Computer Science*, 645: 112–127.

Bhatt, S. N.; and Leighton, F. T. 1984. A Framework for Solving VLSI Graph Layout Problems. *J. Comput. Syst. Sci.*, 28(2): 300–343.

Björnsson, Y.; and Halldórsson, K. 2006. Improved heuristics for optimal pathfinding on game maps. In Laird, J.; and Schaeffer, J., eds., *Proc. AIIDE 2006*, volume 2, 9–14. AAAI Press.

Bui, T. N.; and Jones, C. 1992. Finding Good Approximate Vertex and Edge Partitions is NP-Hard. *Inf. Process. Lett.*, 42(3): 153–159.

Chen, R.; and Gotsman, C. 2021. Efficient fastest-path computations for road maps. *Comput. Vis. Media*, 7(2): 267–281.

Delling, D.; Goldberg, A. V.; Pajor, T.; and Werneck, R. F. 2017. Customizable Route Planning in Road Networks. *Transp. Sci.*, 51(2): 566–591.

Delling, D.; Goldberg, A. V.; Razenshteyn, I. P.; and Werneck, R. F. F. 2011. Graph Partitioning with Natural Cuts. In *Proc. IPDPS 2011*, 1135–1146. IEEE.

Delling, D.; Goldberg, A. V.; Savchenko, R.; and Werneck, R. F. 2014. Hub Labels: Theory and Practice. In Gudmundsson, J.; and Katajainen, J., eds., *Proc. SEA 2014*, volume 8504 of *LNCS*, 259–270. Springer, Springer.

Dibbelt, J.; Strasser, B.; and Wagner, D. 2016. Customizable contraction hierarchies. *ACM J. Exp. Algorithmics*, 21: 1–49.

Feige, U.; and Mahdian, M. 2006. Finding small balanced separators. In Kleinberg, J. M., ed., *Proc. STOC 2006*, 375–384. ACM.

Fukuyama, J. 2006. NP-completeness of the Planar Separator Problems. *J. Graph Algorithms Appl.*, 10(2): 317–328.

Garey, M. R.; and Johnson, D. S. 1979. *Computers and intractability:: A Guide to the Theory of NP-Completeness*.

Geisberger, R.; Sanders, P.; Schultes, D.; and Vetter, C. 2012. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transp. Sci.*, 46(3): 388–404.

George, A. 1973. Nested Dissection of a Regular Finite Element Mesh. *SIAM J. Numer. Anal.*, 10(2): 345–363.

Gottesbüren, L.; Hamann, M.; Uhl, T. N.; and Wagner, D. 2019. Faster and Better Nested Dissection Orders for Customizable Contraction Hierarchies. *Algorithms*, 12(9): 196.

Hamann, M.; and Strasser, B. 2018. Graph Bisection with Pareto Optimization. *ACM J. Exp. Algorithmics*, 23.

Harabor, D.; and Stuckey, P. 2018. Forward Search in Contraction Hierarchies. In Bulitko, V.; and Storandt, S., eds., *Proc. SOCS 2018*, 1, 55–62. AAAI Press.

Karypis, G.; and Kumar, V. 1998. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.*, 20(1): 359–392.

Lee, J.; Kwak, J.; Lee, H.; and Shroff, N. B. 2018. Finding minimum node separators: A Markov chain Monte Carlo method. *Reliab. Eng. Syst. Saf.*, 178: 225–235.

Leighton, F. T.; and Rao, S. 1988. An Approximate Max-Flow Min-Cut Theorem for Uniform Multicommodity Flow Problems with Applications to Approximation Algorithms. In *Proc. FOCS 1988*, 422–431. IEEE Computer Society.

Lipton, R. J.; and Tarjan, R. E. 1979. A Separator Theorem for Planar Graphs. *SIAM J. Applied Mathematics*, 36(2): 177–189.

Milosavljević, N. 2012. On optimal preprocessing for contraction hierarchies. In Winter, S.; and Müller-Hannemann, M., eds., *Proc. IWCTS 2012*, 33–38. ACM.

Müller, R.; and Wagner, D. 1991. α-Vertex separator is NP-hard even for 3-regular graphs. *Computing*, 46(4): 343–353.

Pellegrini, F.; and Roman, J. 1996. SCOTCH: A Software Package for Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs. In Liddell, H. M.; Colbrook, A.; Hertzberger, L. O.; and Sloot, P. M. A., eds., *Proc. HPCN 1996*, volume 1067 of *LNCS*, 493–498. Springer.

Pochter, N.; Zohar, A.; Rosenschein, J. S.; and Felner, A. 2010. Search space reduction using swamp hierarchies. In *Proc. AAAI 2010*.

Rabin, S.; and Sturtevant, N. R. 2016. Combining Bounding Boxes and JPS to Prune Grid Pathfinding. In Schuurmans, D.; and Wellman, M. P., eds., *Proc. AAAI 2016*, 746–752. AAAI Press.

Sanders, P.; and Schulz, C. 2012. Distributed Evolutionary Graph Partitioning. In Bader, D. A.; and Mutzel, P., eds., *Proc. ALENEX 2012*, 16–29. SIAM / Omnipress.

Sanders, P.; and Schulz, C. 2013. Think Locally, Act Globally: Highly Balanced Graph Partitioning. In Bonifaci, V.; Demetrescu, C.; and Marchetti-Spaccamela, A., eds., *Proc. SEA 2013*, volume 7933 of *LNCS*, 164–175. Springer.

Schild, A.; and Sommer, C. 2015. On Balanced Separators in Road Networks. In Bampis, E., ed., *Proc. SEA 2015*, volume 9125 of *LNCS*, 286–297. Springer.

Shen, B.; Cheema, M. A.; Harabor, D. D.; and Stuckey, P. J. 2021. Contracting and compressing shortest path databases. In Biundo, S.; Do, M.; Goldman, R.; Katz, M.; Yang, Q.; and Zhuo, H. H., eds., *Proc. ICAPS 2021*, 322–330. AAAI Press.

Storandt, S. 2013. Contraction Hierarchies on Grid Graphs. In Timm, I. J.; and Thimm, M., eds., *Proc. KI 2013*, volume 8077 of *LNCS*, 236–247. Springer, Springer.

Sturtevant, N. 2012. Benchmarks for Grid-Based Pathfinding. *IEEE Trans. Comput. Intell. AI Games*, 4(2): 144 – 148.