Graph Fill-In, Elimination Ordering, Nested Dissection and Contraction Hierarchies

Ben Strasser and Dorothea Wagner

Abstract Graph fill-in, elimination ordering, separators, nested dissection orders and tree-width are only some examples of classical graph concepts that are related in manifold ways. This essay shows how contraction hierarchies, a successful approach to speed up Dijkstra's algorithm for shortest paths, fits into this series of graph concepts. A theoretical consequence of this insight is a guarantee for the size of the search space required by Dijkstra's algorithm combined with contraction hierarchies. On the other hand, the use of nested dissection leads to a very practicable variant of contraction hierarchies that can be applied in scenarios where edge lengths often change.

1 Introduction

We begin by reviewing some relevant basic graph-theoretic concepts before we discuss contraction hierarchies in detail.

1.1 Graph Fill-In and Elimination Ordering

Let $G = (V, E)$ be a connected, undirected simple graph with n vertices and m edges. A graph is *chordal* if it does not contain a chordless cycle of length at least four. Chordal graphs can be characterized by the concept of a perfect elimination ordering. A vertex v is called *simplicial* in G if the neighbors of v form a clique in G. It is known that every chordal graph contains a simplicial vertex and that removing (or eliminating) a simplicial vertex and its incident edges from a chordal graph yields

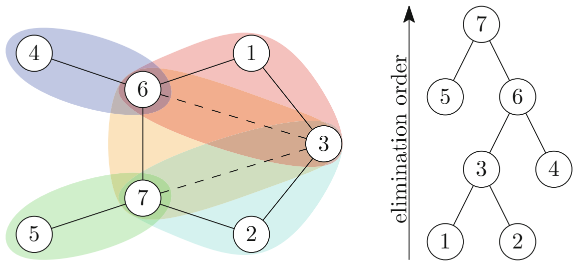
B. Strasser · D. Wagner (✉)
Institut für Theoretische Informatik, Karlsruher Institut für Technologie, Am Fasanengarten 5, 76131 Karlsruhe, Germany
e-mail: dorothea.wagner@kit.edu

B. Strasser
e-mail: strasser@kit.edu

a chordal graph. A *perfect elimination ordering* of G is a vertex ordering r, i.e., a bijective function $r : V \to \{1, \dots, n\}$ where each vertex v is simplicial in the graph induced by the set of vertices $u \in V$ with $r(u) \geq r(v)$. Accordingly, chordal graphs can be characterized as those graphs that have a perfect elimination ordering.

The *elimination game* on G considers a vertex ordering r and eliminates the vertices and all incident edges according to r. When a vertex v is eliminated, all neighbors of v are connected by additional edges to form a clique. The ordering r is called an *elimination ordering*. Let F denote the set of those additional edges. The *filled* graph G^+ is the supergraph of G with edge set $E \cup F$. Obviously, G^+ is a chordal supergraph of G, and the ordering r is a perfect elimination ordering of G^+. Each vertex v together with its neighbors u with $r(u) > r(v)$ form a maximal clique in G^+. The related minimization problem of finding a vertex ordering r such that the size of F is minimum is known as the *minimum fill in* or the *chordal graph completion problem*. Figure 1a depicts a graph G with vertices labeled according to r, the corresponding chordal supergraph G^+ and the induced cliques. The *elimination tree* T for an elimination ordering r of G is a rooted tree with vertex set V and defined by assigning to each vertex v as parent its neighbor u in G^+ with $r(u) > r(v)$ and $r(u)$ minimum. The unique vertex v, which does not posses such a neighbor, is the root of T. The *tree-depth* $\text{td}(G)$ of G is defined as the minimum elimination tree height over all elimination orderings r. Figure 1b depicts the elimination tree related to the elimination order in Fig. 1a.

Chordal graphs are tightly coupled with the concepts of *tree-decomposition* and *tree-width*. Indeed it is common to characterize the tree-width $\text{tw}(G)$ of a graph G using its chordal supergraphs. That is, $\text{tw}(G)$ is the largest number such that, for all chordal supergraphs G^+ of G, $\text{tw}(G) < k$ holds, where k is the maximum clique size of G^+. A common technique to compute some tree-decomposition of G consists in choosing a vertex ordering r of G and constructing the chordal supergraph G^+ induced by the elimination game on G with ordering r. Then the maximal cliques



(a) Graph G and chordal supergraph G^+ (b) Elimination tree T

Fig. 1 The left figure depicts the graph G and its chordal supergraph G^+. The right figure depicts the corresponding elimination tree. The vertices are numbered according to r. In the left figure the solid lines denote the edges of G. When constructing the corresponding chordal supergraph G^+ the dashed edges are added. The colored regions are the maximal cliques in the chordal graph G^+ and the respective vertex sets of a tree decomposition of G.

of G^+ form the sets of a tree-decomposition of G. The size of the maximum set in this tree-decomposition is called its width. So finding a tree-decomposition of G with minimum width $\mathrm{tw}(G)$ is equivalent to finding a chordal supergraph G^+ with smallest maximum clique size.

1.2 Nested Dissection

A *balanced separator* S of size $|S|$ is a subset of V that induces a partition V_1, V_2 of $V \setminus S$ such that $|V_1| \leq \alpha n$, $|V_2| \leq \alpha n$, with $\frac{1}{2} \leq \alpha < 1$ and there exists no edge $\{x, y\} \in E$ with $x \in V_1$ and $y \in V_2$. The subgraphs G_1 and G_2 induced by V_1 and V_2 are the sides of the separator. The objective is to compute a balanced separator of minimum or at least bounded size, i.e., $|S| \in O(n^\beta)$ where $0 \leq \beta < 1$.

A common technique to compute an elimination ordering of a graph G is *nested dissection*. The idea is simple: Recursively partition the graph using small balanced separators. Then consider the elimination ordering of the vertices induced by the recursive structure of the partition where each of the two sides of the separator is eliminated first, and then the separator vertices are eliminated. More precisely, the nested dissection starts by determining a small balanced separator S of G. Then recursively elimination orderings r_1 and r_2 are determined for the two sides of S. The elimination ordering r of G consists of concatenating r_1, followed by r_2, followed by S. The base case of the recursion is reached when the graph has less than some constant number of vertices. In this case the vertices are taken in some arbitrary ordering. Nested dissection is illustrated in Fig. 2. It is known that every graph G with recursive $O(n^\beta)$ balanced separators has a tree-depth in $O(n^\beta)$. Using a nested dissection ordering yields a corresponding elimination tree.

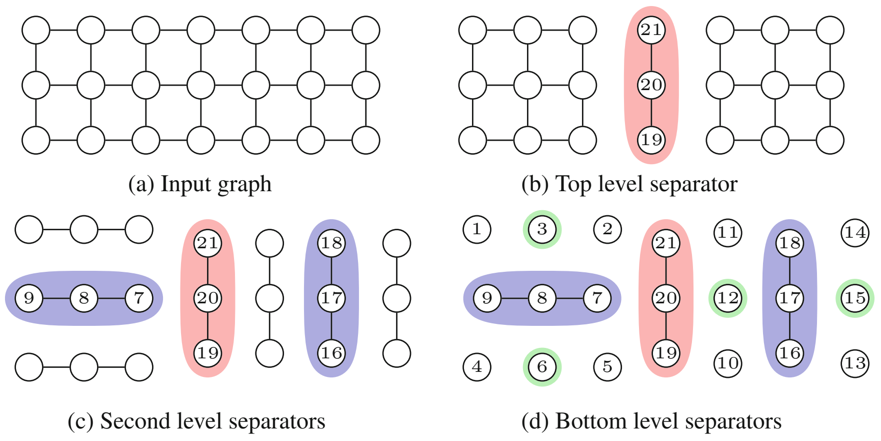


Fig. 2 Steps of the nested dissection procedure. Vertices are labeled according to the induced elimination ordering.

There is a classical approach to use chordal graph completion and elimination orderings to speed up the *Gaussian elimination* of sparse symmetric matrices $M \in \mathbb{R}^{n\times n}$. Given M, one can construct a graph $G = (V, E)$ with $V = \{v_1, \dots, v_n\}$ and $\{v_i, v_j\} \in E$ if and only if $M_{ij} \neq 0$. Eliminating the ith row and ith column in M corresponds to the elimination of v_i in G. Eliminating the rows and columns according to a good elimination ordering of G assures that many zero-entries are conserved in M during the execution of the Gaussian algorithm. This can be exploited to significantly improve the running times.

1.3 Two-Phase and Three-Phase Shortest Path Computation

In a directed graph $G = (V, A)$ with an arc weight function $w : A \to \mathbb{R}_{\geq 0} \cup \{\infty\}$, an st-path P is defined as a sequence of vertices $v_1 \dots v_k$, such that $s = v_1$, $t = v_k$, $(v_i, v_{i+1}) \in A$, and its length is defined as $w(P) := \sum_{i=1}^{k-1} w(v_i, v_{i+1})$. The input of the *classic shortest path problem* consists of a directed graph G, an arc weight function w, and two vertices s and t. The output is an st-path of minimum length. This problem can be solved using Dijkstra's algorithm in near-linear running time. However, for graphs with millions of vertices and edges this is not fast enough. To accelerate the shortest path computation for graphs that do not change often, one can apply a *two-phase shortest path computation*: In a first *preprocessing phase* only G and w are known and some auxiliary data are computed. In a second phase, the *query phase*, s and t become available and a shortest st-path is computed. The second phase can use the auxiliary data. For example, one may apply a variant of Dijkstra's algorithm that makes use of the previously determined auxiliary data to reduce its search space. Note that the auxiliary data are independent of s and t and can therefore be used again and again for many queries. However, a central assumption of this approach is that G and w do not change between queries.

A prominent application of the two-phase shortest path computation is route planning in transportation networks where many queries need to be answered quickly. The key observation is that it is enough to make the query phase fast, while the preprocessing phase can be slow. However, the assumption that G and w do not change between queries is not always true. In road graphs, G is indeed mostly constant but w may change due to the current traffic situation or individual restrictions by the users. The corresponding shortest path problem is called *customizable route planning*. For such scenarios it is more adequate to apply a *three-phase shortest path computation*. Here the preprocessing phase is split into two phases: A weight-independent preprocessing phase that computes weight-independent auxiliary data based only on G, and a *customization phase* where the auxiliary data are augmented depending on w. The weight-independent preprocessing phase may still be slow, but the customization phase should be reasonably fast.

2 Contraction Hierarchies

In order to simplify the presentation, only undirected graphs are considered in the following sections. Moreover, for technical reasons we assume that for each pair of vertices s, t the shortest st-path is unique. However, we want to point out that all these results are developed and presented for directed graphs in the original papers and are applied in shortest path computations and its applications in route planning as stated in Sect. 1.3.

An ingredient of many shortest path acceleration techniques are *shortcuts*, i.e., additional edges computed in the preprocessing phase to build the auxiliary data or at least a part of it. The idea consists of selecting an important path $v_1 \dots v_k$ and adding an additional edge to the graph from v_1 directly to v_k with the weight $\sum_{i=2}^{k} w(v_{i-1}, v_i)$. Acceleration techniques use shortcuts to prevent Dijkstra's algorithm from visiting all intermediate vertices. The speed-up achieved depends on the choice of paths bypassed by shortcuts, and it is crucial that the number of shortcuts is small. *Contraction Hierarchies (CH)* is an elegant and effective two-phase shortest path computation based primarily on shortcuts.

2.1 Two-Phase Contraction Hierarchies

The two phases of CH consist of a systematic preprocessing approach to add shortcuts and a sophisticated way to perform the query phase. The name giving operation *contraction* selects a vertex x from G, removes x from G and adds, if necessary, the edge $\{y, z\}$ if and only if yxz is a shortest yz-path in the current graph with respect to w. The edge $\{y, z\}$ is assigned the weight $w(y, z) = w(y, x) + w(x, z)$. In the CH preprocessing phase, contraction is applied iteratively according to some previously chosen *contraction ordering* $r : V \to \{1, \dots, n\}$ of the vertices in G. Note that this process bears some similarity to the elimination game introduced in Sect. 1.1. However, not all neighbors of the contracted vertex v are connected by a new edge like in the elimination game. In the following, we denote the set of all original edges and added shortcuts by E^+, and let $G^+ = (V, E^+)$. See Fig. 3. Commonly, the graph G^+ together with the corresponding edge weight w is called a *contraction hierarchy*. For achieving a significant speed-up the choice of the contraction ordering is crucial.

The CH query phase works similar to the bidirectional variant of Dijkstra's algorithm. The shortcut augmented graph $G^+ = (V, E^+)$ together with r induces a directed graph $G^\uparrow = (V, E^\uparrow)$, where $E^\uparrow$ contains all edges (y, x) of E^+ such that y comes before x in the contraction ordering. The subgraph of $E^\uparrow$ reachable from a vertex x is called the *upward search space* of x. To determine a shortest st-path a bidirectional variant of Dijkstra's algorithm is run. The forward search is restricted to the search space of s, whereas the backward search is restricted to the search space of t. As G is assumed to be connected, these two search spaces overlap. See Fig. 4.

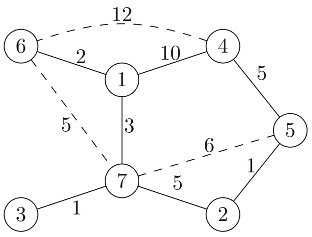


Fig. 3 Graph and CH shortcuts. The vertices are numbered according to the contraction ordering. The *solid lines* are edges in G and the *dashed lines* are inserted shortcuts. Edges are annotated with their weights. When contracting vertex 1 shortcuts from 6 to 4 and from 6 to 7 are added. However, no shortcut from 4 to 7 is added because the path $4 \to 1 \to 7$ has length 13, which is longer than the path $4 \to 5 \to 2 \to 7$ with length 11. For the same reason no edge is added between vertices 5 and 6 when contracting vertex 4.

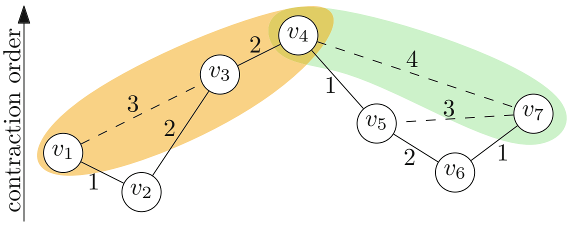


Fig. 4 An illustrated CH query. G is the line graph with vertices $v_1 \dots v_7$. The vertical position of a vertex corresponds to its position in the contraction order. The *solid lines* represent the edges in G with their weights. The *dashed lines* are the added shortcuts in G^+. The *orange area* is the search space of v_1 and the *green area* is the search space of v_7. Both search spaces meet in v_4. The CH finds the path v_1, v_3, v_4, v_7 of length 9 in G^+, which has the same length as the shortest path $v_1 \dots v_7$ in G.

Lemma 1 *The CH query phase is correct, i.e., it computes a shortest st-path.*

Proof The CH query induces an st-path $su_1 \dots u_k q d_1 \dots d_l t$ in G^+ such that the vertices $s, u_1 \dots u_k, q$ appear in increasing contraction ordering, i.e., $r(s) < r(u_1) < \cdots < r(u_k) < r(q)$, and the vertices $q, d_1 \dots d_l, t$ appear in decreasing contraction ordering, i.e., $r(q) > r(d_1) > \cdots > r(d_l) > r(t)$. Such an st-path is called *up-down* st-path. Note that one of the two subpaths might be empty or even both if $s = t$.

It remains to prove that for each pair of vertices s and t there exists a shortest up-down st-path in G^+. Consider an arbitrary shortest path $P = v_1 \dots v_h$ in G and iteratively construct an up-down path of equal length in G^+ as follows. Either P already is an up-down path in G^+, or there exists a vertex v_i in P such that $r(v_{i-1}) > r(v_i)$ and $r(v_i) < r(v_{i+1})$. That is, in the CH preprocessing phase v_i was contracted before v_{i-1} and v_{i+1}. Further, $v_{i-1}v_iv_{i+1}$ is a shortest path as it is a subpath of P. Therefore there exists a shortcut from v_{i-1} to v_{i+1} in G^+. Remove v_i from the path, insert this shortcut instead and apply this operation iteratively for the resulting path. As the number of vertices in the path shrinks in each iteration, it is guaranteed that the construction eventually ends with an up-down path that is also a shortest path. □

Remarkably, the correctness of the CH query is still guaranteed if additional "unnecessary" edges are added during the preprocessing phase, as long as the weight of such an edge (y, z) is assigned the length of a shortest yz-path.

2.2 Implementing Contraction Hierarchies

Determining whether yxz is a shortest yz-path is called *witness search*. The straightforward approach consists of running Dijkstra's algorithm on $G\backslash\{x\}$ to find a shortest yz-path in $G\backslash\{x\}$. However, this can be too slow to work on huge graphs. One idea is to abort the witness search at some point and insert the shortcut (y, z) anyway, independently of whether yxz is a shortest yz-path or not. This might result in adding unnecessary shortcuts which make the query phase slower, but fortunately not incorrect.

A heuristic approach to come up with a "good" contraction hierarchy consists in ordering the vertices by ascending "importance." Unfortunately, there is no universal definition for importance of a vertex in a graph. In road graphs "importance" is usually based on the intuition that a vertex on a highway is important whereas a dead-end of a street in a rural area is unimportant.

A contraction ordering may be computed by exploring all possible vertex contractions and greedily picking the vertex that results in the fewest shortcuts and assigning it a low "importance," i.e., we iteratively try to identify dead-end-like structures. This strategy can be refined with further heuristics that try to assure that G is contracted uniformly. A different top-down heuristic consists in iteratively assigning a high "importance" to a vertex that covers many shortest paths, i.e., vertex x that covers as many paths as possible gets highest importance and next the vertex y that covers as many paths as possible not already covered by x is determined, and so on. The idea is that many shortest paths traverse highways. For references, see Sect. 5. Note that these strategies and the quality of the resulting CH depend on the weight function of G. An ordering that leads to a "good" contraction hierarchy for a weight function w_1 can lead to a huge number of shortcuts when used with another weight function w_2 on the same graph G. Even correlated weights such as travel-time and travel-distance in road graphs are not interchangeable in practice.

The next two sections are devoted to these two aspects: How can we determine a contraction ordering for which we can give a guarantee for the size of the search space? How can we construct a weight-independent contraction ordering and design a three-phase shortest path algorithm?

3 Weak Contraction Hierarchies

In the following a generalization of CH is presented for which, depending on the properties of the graphs considered, a guarantee for the size of the search space can be given. Consider a graph G with edge weights w and an arbitrary contraction ordering r. Let $G^+ = (V, E^+)$ denote the contraction hierarchy induced by r with corresponding edge weight w as defined in Sect. 2.1. A graph $H = (V, E_H)$ with $E^+ \subseteq E_H$ and for which the additional edges $\{y, x\}$ are shortcuts is called a *weak contraction hierarchy*. A CH making use of a weak contraction hierarchy is called *weak CH*. Note that a contraction hierarchy determined by CH as originally defined is a *minimum weak contraction hierarchy*, while a *maximum weak contraction hierarchy* contains all possible shortcuts. Moreover, the search space size induced by a maximum weak contraction hierarchy is an upper bound for the search space size induced by any weak contraction hierarchy that is contained therein. In the following, a contraction hierarchy (be it minimum, maximum or any weak contraction hierarchy in between) is denoted by $G^+ = (V, E^+)$.

3.1 A Crucial Observation

The first insight is that the maximum weak contraction hierarchy G^+ for G and w induced by some contraction ordering r, is the chordal supergraph of G with corresponding edge weights w that is obtained by using r as elimination ordering. The next observation is that the vertices in the search space of a vertex x of a weak CH are the ancestors of x in the corresponding elimination tree. Then, instead of applying Dijkstra's algorithm, the query can make use of the elimination tree. Just follow the paths from s and t, respectively, up to the root of the elimination tree. Consider a graph that admits recursive graph separators of size $O(n^\beta)$. Choosing r as a nested dissection ordering yields an elimination tree of depth $O(n^\beta)$. It follows that for every vertex x the number of vertices in the search space of the weak CH using r is in $O(n^\beta)$.

It is well known that planar graphs admit recursive $O(\sqrt{n})$ balanced separators. This induces a weak CH for planar graphs with a search space size guarantee of at most $O(\sqrt{n})$ vertices. The search spaces are not necessarily planar and therefore we can have up to $O((\sqrt{n})^2) = O(n)$ edges in the search space. This results in a running time of $O(n \log n)$ using Dijkstra's algorithm. However, the alternative query procedure that makes use of the elimination tree yields a better worst-case running time of $O(n)$.

3.2 Notes on Some Practical Implications

The theoretical insights discussed above have various practical implications. Most importantly, the bounds on the size of the search space of weak CH based on a nested dissection ordering are completely independent of the edge weights. This is surprising since the performance of CH induced by a weight-dependent ordering, as originally designed, very much depends on the weights, as already mentioned in Sect. 2.2. It is interesting to have a closer look at CH search spaces based on different, i.e., weight-dependent versus weight-independent orderings on one hand, and on minimum versus maximum contraction hierarchies on the other hand. Weak CH using a nested dissection ordering turns out to be also useful in practice. One reason is that road graphs are somehow close or at least similar to planar graphs. Therefore one can expect road graphs to have recursive balanced separators of small size. Indeed, experiments show that road graphs have small separators. Therefore one can expect that CH or weak CH using nested dissection orderings work well on road graphs. Experiments show that this is indeed the case. At least, experimental evaluations show that maximum weak contraction hierarchies based on nested dissection orderings are small enough to be useful.

While CH search spaces for (at least almost) minimum contraction hierarchies induced by weight-dependent orderings are small in practice, the search space of CH based on minimum contraction hierarchies obtained using nested dissection orderings are larger. Experiments have further shown that the search spaces of weak CH using maximum contraction hierarchies resulting from weight-dependent orderings are too huge to be computable in practice. The experiments clearly show that contraction hierarchies resulting from the weight-dependent orderings (considered so far) differ from (weight-independent) contraction hierarchies obtained by nested dissection. But how these two worlds relate is not yet fully understood.

4 Customizable Contraction Hierarchies

In a departure from the problem setting considered in the previous two sections, we now consider situations in which the arc weight function w may change between queries due to the current traffic situation or individual restrictions by users. Recall from Sect. 1.3 that an adequate approach in such scenarios consists of a three-phase shortest path computation where the preprocessing is split into two phases, a weight-independent preprocessing phase and a customization phase.

It turns out that weight-independent contraction orderings are a very good basis for a three-phase shortest path computation. The key idea is that a maximum weak contraction hierarchy obtained by a weight-independent ordering already contains all necessary shortcuts for each possible weight function. Accordingly, computing a maximum weak contraction hierarchy based on nested dissection in a first pre-processing phase yields a good basis for the subsequent second and third phases. In

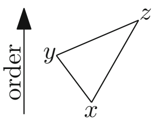


Fig. 5 The triangle $\{x, y, z\}$ is a *lower triangle* of the edge $\{y, z\}$, an *intermediate triangle* of the edge $\{x, z\}$ and an *upper triangle* of the edge $\{x, y\}$.

particular, in the customization phase only the weights of the shortcuts need to be computed. Actually, this second phase plays a decisive role for the design of *Customizable Contraction Hierarchies (CCH)* for three-phase shortest path computation.

4.1 Basic Customization

The first phase of the preprocessing phase just considers the graph G without edge weights and returns an elimination ordering r, the corresponding chordal supergraph $G^+ = (V, E^+)$, the induced upward graph $G^\uparrow = (V, E^\uparrow)$, and the corresponding elimination tree T.

Consider a triangle $\{x, y, z\}$ in G^+ such that $r(x) < r(y) < r(z)$, as illustrated in Fig. 5. The tuple $\{x, y, z\}$ is called a *lower triangle* of $\{y, z\}$, an *intermediate triangle* of $\{x, z\}$, and an *upper triangle* of $\{x, y\}$. The *lower triangle inequality* holds when for every lower triangle $\{x, y, z\}$ the inequality $w(y, z) \leq w(y, x) + w(x, z)$ holds. The proof of Lemma 1, which showed the correctness of the query phase, relies on a weakened variant of the lower triangle inequality; hence, if the lower triangle inequality holds, then the queries are correct. Therefore, the goal of the customization phase is to assign weights to shortcuts with respect to a metric w such that the lower triangle inequality is guaranteed. Initially, each edge of G is assigned its weight according to w, and each shortcut is assigned the weight ∞. Then iterate over all y in the order of increasing $r(y)$. For each y, enumerate for all (y, z) in $E^\uparrow$ the lower triangles $\{x, y, z\}$ by exploring the common neighbors of y and z. If $w(y, z) > w(y, x) + w(x, z)$ for a triangle $\{x, y, z\}$, then $w(y, z)$ is set to $w(y, x) + w(x, z)$. This procedure assigns a weight to every edge that guarantees the lower triangle inequality without modifying the topology of the contraction hierarchy G^+. This procedure is called *basic customization*. Note that the lower triangle inequality does not necessarily imply that the weight of a shortcut $\{y, z\}$ equals the length of a shortest yz-path.

4.2 Perfect Customization

If after the basic customization the weight $w(y, z)$ of a shortcut $\{y, z\}$ is larger than the length of a shortest yz-path, this shortcut is not necessary to ensure correctness of the query phase, as subpaths of a shortest up-down path must be shortest paths.

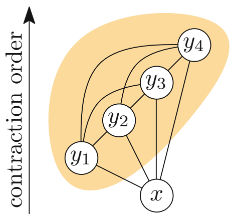


Fig. 6 Upward neighborhood of x. The nodes y_i are all neighbors of x that were contracted after x. By construction, these neighbors must form a clique.

Therefore, the customization phase can be enhanced by a procedure to compute the distances between the endpoints of every edge in G^+ and to subsequently remove all unnecessary shortcuts. This procedure iterates over all x with decreasing $r(x)$ and, for each $\{x, y\}$, enumerates all intermediate and all upper triangles. For each upper and intermediate triangle $\{x, y, z\}$ of $\{x, y\}$ with $w(x, y) > w(x, z) + w(z, y)$, the weight $w(x, y)$ is set to $w(x, z) + w(z, y)$. We will prove that at the end of the iteration, the weight of each edge $\{x, y\}$ corresponds to the length of a shortest xy-path. Such a customization is called *perfect customization*. Although perfect customization costs extra time, it can be beneficial because it results in faster queries as there are less edges in G^+.

Lemma 2 *After perfect customization the weight $w(x, y)$ for each edge $\{x, y\}$ in G^+ equals the length of a shortest xy-path.*

Proof For an inductive proof, consider vertex $x \in V$ and assume that the claim holds for all edges incident to y with $r(y) > r(x)$. Let $y_1, \dots, y_k$ denote the upper neighbors of x, i.e., $r(y_i) > r(x)$, as depicted in Fig. 6. Because all upper neighbors of x appear after x in the order, we know by induction that the weights of all edges between such neighbors $\{y_i, y_j\}$ (the orange area) equal the length of a shortest $y_i y_j$-path. When an upper or an intermediate triangle of the edge $\{x, y_i\}$ is inspected, either the weight of (x, y_i) is equal to the length of a shortest xy_i-path and there is nothing to prove. Or there exists an xy_i-path P of shorter length, which is an up-down path, because of the basic customization, and contains another neighbor y_j. As x is contracted before all its upper neighbors $y_1, \dots, y_k$, the vertices $x, y_1, \dots, y_k$ form a clique in G^+. So there is an edge $\{y_i, y_j\}$, and the length of xy_iy_j is equal to the length of P. Moreover, $\{x, y_i, y_j\}$ is an upper or an intermediate triangle of $\{x, y_i\}$, depending on whether y_i or y_j comes first in the order. $\square$

4.3 Query Phase

The CCH query phase can be performed analogously to the CH query based on Dijkstra's algorithm. Alternatively, one can also make use of the elimination tree. Recall that every vertex in the search space of a query from s to t must be an ancestor

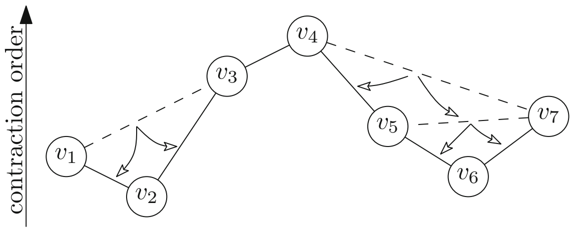


Fig. 7 Path unpacking. The up-down-path $v_1v_3v_4v_7$ in G^+ is unpacked to the path $v_1 \dots v_7$ in G of equal length. The shortcut $\{v_1, v_3\}$ is replaced by $v_1v_2v_3$ by enumerating all lower triangles of the edge $\{v_1, v_3\}$. Similarly $\{v_4, v_7\}$ is unpacked to $v_4v_5v_7$. In a subsequent step $\{v_5, v_7\}$ is unpacked to $v_5v_6v_7$.

of s or t in the elimination tree. Furthermore, shortcuts are always directed upwards. Therefore, the ancestors of s and t can be enumerated simultaneously, ordered by their position in the contraction ordering, until the root is reached.

The result of the query phase is the distance from s to t and an up-down st-path P in G^+. It remains to clarify how the according shortest path in G can be derived. This can be done by a procedure called *unpacking* that iteratively replaces shortcuts. Consider an edge $\{x, y\}$ on P that is a shortcut. To replace $\{x, y\}$ by a shortest subpath in G enumerate all lower triangles $\{x, y, z\}$ of $\{x, y\}$. By construction, at least one triangle with $w(x, z) + w(z, y) = w(x, y)$ must exist. Therefore the edge $\{x, y\}$ can be replaced by the two subsequent edges $\{x, z\}$ and $\{z, y\}$. Note that $\{x, z\}$ or $\{z, y\}$ may again be shortcuts. However, this unpacking step can be performed recursively. The process is illustrated in Fig. 7.

4.4 Note on Directed Graphs

Remember that we restricted the presentation in the previous sections to undirected graphs in order to simplify notation. However, it turns out that CCH as introduced here can be immediately applied to directed graphs. First notice that a directed graph G with weight function w can be identified with the complete directed graph where arcs not present in G have weight ∞. This graph can be also represented by a complete undirected graph with two weights per edge, a forward and a backward weight. Now, remember that a maximum weak contraction hierarchy G^+ contains all possible shortcuts and that in the first phase, CCH just computes G^+ without weights. In the customization phase, instead of one weight per edge two weights need to be considered, one interpreted as upward weight and the other as downward weight. All procedures can be applied to such a scenario without further modification.

5 Notes on the Literature

Algorithms for route planning in transportation networks have recently undergone a rapid development, leading to methods that are up to several million times faster than Dijkstra's algorithm [13]. Prominent examples besides CH [15, 16] are arc-flags [20], multi-level overlays [19, 24], reach [18], or hub-labels [3, 10]. See also [5] for an overview. Only recently, [6] noticed the connection between contraction hierarchies and the classical graph fill-in problem, which led to a theoretical guarantee for the size of the search space required by weak CH. On the other hand, this insight led to a very practicable variant of contraction hierarchies that can be applied in scenarios where edge lengths often change [11]. See also [12], where some of the theoretical results from [6] are even improved. A different approach to obtain theoretical guarantees for the size of the search space required by speed-up techniques is presented in [1, 2, 4]. The resulting bounds depend on a weight-dependent graph measure called highway dimension.

The basic graph concepts discussed here go back to the 1960s and 70s. It was shown in [14] that every chordal graph contains a simplicial node and that removing (or eliminating) a simplicial node and its incident edges from a chordal graph yields a chordal graph. The connection between chordal graph completion and Gauss elimination was noticed in [22, 23], and the concept of nested dissection goes back to [17] and [21]. Chordal graphs are tightly coupled with the concept of tree-width, and we recommend [7–9] for a survey.

References

1. Abraham, I., Delling, D., Fiat, A., Goldberg, A.V., Werneck, R.F.: VC-dimension and shortest path algorithms. In: Proceedings of the 38th International Colloquium on Automata, Languages, and Programming (ICALP'11). Lecture Notes in Computer Science, vol. 6755, pp. 690–699. Springer (2011)
2. Abraham, I., Delling, D., Fiat, A., Goldberg, A.V., Werneck, R.F.: Highway dimension and provably efficient shortest path algorithms. Technical report MSR-TR-2013-91, Microsoft Research (2013)
3. Abraham, I., Delling, D., Goldberg, A.V., Werneck, R.F.: A hub-based labeling algorithm for shortest paths on road networks. In: Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11). Lecture Notes in Computer Science, vol. 6630, pp. 230–241. Springer (2011)
4. Abraham, I., Fiat, A., Goldberg, A.V., Werneck, R.F.: Highway dimension, shortest paths, and provably efficient algorithms. In: Proceedings of the 21st Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'10), pp 782–793. SIAM (2010)
5. Bast, H., Delling, D., Goldberg, A.V., Müller–Hannemann, M., Pajor, T., Sanders, P., Wagner, D., Werneck, R.F.: Route planning in transportation networks. Technical report , ArXiv e-prints, (2015). arXiv:1504.05140
6. Bauer, R., Columbus, T., Rutter, I., Wagner, D.: Search-space size in contraction hierarchies. In: Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP'13). Lecture Notes in Computer Science, vol. 7965, pp. 93–104. Springer (2013)
7. Bodlaender, Hans L.: A tourist guide through treewidth. Acta Cybern. **11**, 1–23 (1993)

8. Bodlaender, Hans L.: Tutorial: A partial k-arboretum of graphs with bounded treewidth. Theor. Comput. Sci. **209**, 1–45 (1998)
9. Bodlaender, H.L.: Treewidth: Structure and algorithms. In: Proceedings of the 14th International Colloquium on Structural Information and Communication Complexity. Lecture Notes in Computer Science, vol. 4474, pp. 11–25. Springer (2007)
10. Cohen, E., Halperin, E., Kaplan, H., Zwick, U.: Reachability and distance queries via 2-hop labels. SIAM J. Comput. **32**(5), 1338–1355 (2003)
11. Dibbelt, J., Strasser, B., Wagner, D.: Customizable contraction hierarchies. In: Proceedings of the 13th International Symposium on Experimental Algorithms (SEA'14). Lecture Notes in Computer Science, vol. 8504, pp. 271–282. Springer (2014)
12. Dibbelt, J., Strasser, B., Wagner, D.: Customizable contraction hierarchies. Technical report, ITI Wagner, Department of Informatics, Karlsruhe Institute of Technology (KIT) (2014). arXiv:1402.0402
13. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numer. Math. **1**, 269–271 (1959)
14. Fulkerson, D.R., Gross, O.A.: Incidence matrices and interval graphs. Pac. J. Math. **15**(3), 835–855 (1965)
15. Geisberger, R., Sanders, P., Schultes, D., Delling, D.: Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In: Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08). Lecture Notes in Computer Science, vol. 5038, pp. 319–333. Springer (2008)
16. Geisberger, R., Sanders, P., Schultes, D., Vetter, C.: Exact routing in large road networks using contraction hierarchies. Transp. Sci. **46**(3), 388–404 (2012)
17. George, A.: Nested dissection of a regular finite element mesh. SIAM J. Numer. Anal. **10**(2), 345–363 (1973)
18. Gutman, R.J.: Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In: Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX'04), pp. 100–111. SIAM (2004)
19. Holzer, M., Schulz, F., Wagner, D.: Engineering multilevel overlay graphs for shortest-path queries. ACM J. Exp. Algorithmics **13**(2.5):1–26 (2008)
20. Köhler, E., Möhring, R.H., Schilling, H.: Acceleration of shortest path and constrained shortest path computation. In: Proceedings of the 4th Workshop on Experimental Algorithms (WEA'05). Lecture Notes in Computer Science, vol. 3503, pp. 126–138. Springer (2005)
21. Lipton, R.J., Rose, D.J., Tarjan, R.: Generalized nested dissection. SIAM J. Numer. Anal. **16**(2), 346–358 (1979)
22. Parter, S.V.: The use of linear graphs in Gauss elimination. SIAM Rev. **3**(2), 119–130 (1961)
23. Rose, D.J.: Triangulated graphs and the elimination process. J. Math. Anal. Appl. **32**(3), 597–609 (1970)
24. Schulz, F., Wagner, D., Zaroliagis, C.: Using multi-level graphs for timetable information in railway systems. In: Proceedings of the 4th Workshop on Algorithm Engineering and Experiments (ALENEX'02). Lecture Notes in Computer Science, vol. 2409, pp. 43–59. Springer, (2002)