Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

Complexity Results for Compressing Optimal Paths

Adi Botea
IBM Research
Ireland

Ben Strasser
Karlsruhe Institute of Technology
Germany

Daniel Harabor
NICTA
Australia

Abstract

In this work we give a first tractability analysis of Compressed Path Databases, space efficient oracles used to very quickly identify the first arc on a shortest path. We study the complexity of computing an optimal compressed path database for general directed and undirected graphs. We find that in both cases the problem is NP-complete. We also show that, for graphs which can be decomposed along articulation points, the problem can be decomposed into independent parts, with a corresponding reduction in its level of difficulty. In particular, this leads to simple and tractable algorithms which yield optimal compression results for trees.

Introduction

A Compressed Path Database (CPD) is an index-based data-structure for graphs that is used to very quickly answer first-move queries. Each such query takes as input a pair of nodes from the graph, and asks for the first arc on a shortest path from s to t. CPDs have successfully been applied in a number of contexts important to AI. For instance, Copa, a CPD-based pathfinding algorithm, was one of the joint winners at the 2012 Grid-based Path Planning Competition (Botea 2012). A related algorithm, MtsCopa, is a fast method for moving target search over known and partially known terrain (Botea et al. 2013; Baier et al. 2014).

A trivial CPD consists of a square matrix $\mathbf{m}$ with dimensions $|V| \times |V|$. The matrix $\mathbf{m}$, constructed during a precomputation step, stores in each cell, e.g. $\mathbf{m}[s, t]$, the identity of a first arc on a shortest st-path. By convention we say that rows of $\mathbf{m}$ correspond to fixed source nodes and the columns to fixed target nodes. This is optimal in terms of query time but the $O(|V|^2)$ space consumption quickly becomes prohibitive for larger graphs. The challenge is to design a compact representation of $\mathbf{m}$ that trades a small increase in query times for a large decrease in space consumption.

A number of different matrix compression techniques have been suggested for this purpose (Sankaranarayanan, Alborzi, and Samet 2005; Botea 2011; Botea and Harabor 2013). In each case the objective is to conserve space by grouping together entries of $\mathbf{m}$ which all share a common source node and which all store the same first-arc information. The latest work of this type is SRC (Strasser, Harabor, and Botea 2014); a compression scheme based on Run-Length Encoding which is not only highly space competitive but also the current state of the art in terms of query running time. The idea behind SRC is simple: compute the first-move matrix $\mathbf{m}$, reorder its columns in a "good" way and then apply run-length encoding to each row. An intuitive definition of "good" in this case is a column ordering that assigns adjacent IDs to elements having the same first-move data – so they can be encoded together within the same run.

Experiments by Strasser, Harabor, and Botea (2014) employ heuristic methods for column ordering. They have shown that "good" orderings can be computed efficiently in practice. The ideal case for SRC is to create an RLE-compressed path database using an optimal column ordering. Several optimality criteria can be considered, such as minimizing: the size of the compressed matrix; the expected response time per query, when answering a query involves a binary search through a compressed matrix row; or the maximum response time per query. Toy examples demonstrate that these types of optimization are not equivalent to each other. In this work we focus on minimizing the size of the compressed matrix, since experiments have shown that the size is a substantially more important bottleneck than the response time in practice (Strasser, Harabor, and Botea 2014).

Related (Kou 1977; Oswald and Reinelt 2009) and weaker, less specific (Mohapatra 2009) results on RLE-based matrix compression are available in the literature. However, as known, the NP-hardness of a class of problems does not necessarily imply the NP-hardness of a subset of the class. Thus, despite previous related results (Mohapatra 2009), it has been an open question whether the optimal RLE-compression of a *first-move matrix computed from an input graph* is tractable. Focusing on this, we prove general intractability results, and also identify islands of tractability.

We formally define and study optimal RLE-compression of first-move matrices produced from input graphs. We consider the case of directed input graphs and the case of undirected weighted input graphs. We show that both versions are NP-complete. Focusing on such distinct types of graphs, each result brings something new compared to the other. We also show that, for graphs which can be decomposed along articulation points, the problem can be decomposed into independent subproblems. If optimal orderings are available

Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

for the subproblems, a global optimal ordering can easily be obtained. In particular, a depth-first preorder is optimal on trees, and the general ordering problem is fixed-parameter tractable in the size of the largest 2-connected component.

Preliminaries

Run length encoding (RLE) compresses a string of symbols by representing more compactly substrings, called *runs*, consisting of repetitions of the same sysmbol. For instance, string $aabbbaaa$ has three runs, namely aa, bbb, and aaa. A run is replaced with two numbers, called the *start* and the *value* of the run. The start is the index of the first element in the substring, whereas the value is the symbol contained in the substring. In our example, the first run aa has the start 0 and the value a. Run bbb has the start 2 and the value b, whereas the last run has the start 5 and the value a.[1]

When the first and the last run have the same value, there is no need to encode both. The first run can easily be reconstructed in constant time. First, decide whether the first run has been removed or not, checking if the first run among the preserved ones has the start equal to 0. Secondly, if needed, reconstruct the first run, using 0 as a start position and a value equal to the value of the last encoded run.

Definition 1. *Given an ordered sequence of elements (string), we say that two positions are:* adjacent *if they are next to each other;* cyclic-adjacent *if they are adjacent or one is the first and the other is the last position in the ordering;* separated *otherwise.*

Let σ be an ordered sequence of elements (string) over a dictionary Σ. Given a symbol $\alpha \in \Sigma$, let an *α-run* be an RLE run containing symbol α. $N_\alpha(\sigma)$ is the total number of occurrences of symbol α in σ. $R^s_\alpha(\sigma)$, the *number of sequential α-runs*, is the total of number of α-runs under the condition that, if σ has α-runs both at the beginning and at the end, these are counted separately. We call these *sequential α-runs*. $R^c_\alpha(\sigma)$, the *number of cyclic α-runs*, is similar except that, if σ has α-runs both at the beginning and at the end, these are counted as one. We call these *cyclic α-runs*. $R^c_*(\sigma)$ is the total number of cyclic RLE runs, and $R^s_*(\sigma)$ is the total number of sequential RLE runs. In this paper, we assume that first-move compression uses cyclic runs.

First-Move Compression for Directed Graphs

Recall that the ordering of the columns of a first-move matrix $\mathbf{m}$ affects the number of RLE runs in the matrix. In this section we show that obtaining an optimal ordering is intractable in general when the input graph is directed.

Definition 2. *The* FMComp-d *(First Move Compression–Directed) problem:*
Input: A directed graph $G = (V, A)$; a matrix $\mathbf{m}$ of size $|V| \times |V|$ where each cell $\mathbf{m}[i, j]$ encodes the first move on an optimal path from node i to node j; an integer k.
Question: Is there an ordering of the columns of $\mathbf{m}$ such that, if we apply RLE on each row, the total number of cyclic RLE runs summed up for all rows is k?

[1] Alternative encodings exist, such as the value followed by the run length. E.g., $a, 2; b, 3; a, 3$ in the example.

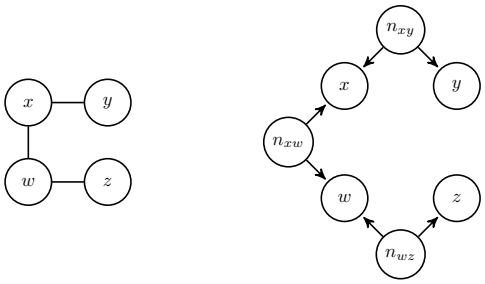

Figure 1: Left: sample graph G_H. Right: G_F built from G_H. In G_F, x, y, w, z are type-n nodes; n_{ij} have type e.

Theorem 1. *The FMComp-d problem is NP-complete.*

Proof. It is easy to see that the problem belongs to NP, as a solution can be guessed and verified in polynomial time.

The NP-hardness is shown as a reduction from the Hamiltonian Path Problem (HPP) in an undirected graph. Let $G_H = (V_H, A_H)$ be an arbitrary undirected graph, and define $n = |V_H|$ and $e = |A_H|$. Starting from G_H, we build an instance of the FMComp-d problem. According to Definition 2, such an instance includes a directed graph, which we call G_F, the first-move matrix $\mathbf{m}$ of G_F, and a number.

$G_F = (V_F, A_F)$ is defined as follows. For each node $u \in V_H$, define a node in V_F. We call these nodes in V_F type-n nodes, to indicate they are created from original nodes in V_H. For each edge $(u, v) \in A_H$, define a new node $n_{uv} \in V_F$ (type-e nodes). For each new node n_{uv}, define two edges in A_F, one from n_{uv} to u and one from n_{uv} and v. There are no other edges in A_F. See Figure 1 for an example.

Table 1 shows the first-move matrix of the running example. Given a type-n node u, all other nodes are unreachable from u in the graph G_F. Thus, the matrix row corresponding to u has only one symbol, which we chose to be symbol 2, and which denotes that a node is not reachable.[2] Such rows have one RLE run each, regardless of the node ordering.

A matrix row corresponding to a type-e node n_{uv} has three distinct symbols in total: one symbol for the edge to node u, another symbol for the edge to node v, and the "non-reachable" symbol 2 for every other node. Without any generality loss, we use symbol 0 for the edge to u, and symbol 1 for the edge to v. It is easy to see that, when nodes u and v are cyclic-adjacent in a given ordering, the n_{uv}'s row has 3 RLE runs. When u and v are separated, the row will have 4 RLE runs. See Table 1 for a few sample orderings.

We claim that HPP has a solution iff FMComp-d has a solution with $4e + 1$ RLE runs. Let $v_{i_1}, v_{i_2} \ldots, v_{i_n}$ be a solution of HPP (i.e., a Hamiltonian path in G_H), and let $P \subseteq A_H$ be the set of all edges included in this solution. We show that the node ordering in V_F starting with $v_{i_1}, \ldots, v_{i_n}$, followed by the type-e nodes in an arbitrary order, will result

[2] A cell $\mathbf{m}[u, u]$ can encode an arbitrary value, since there is no need to travel from a node to itself. Given a column ordering, $\mathbf{m}[u, u]$ is assigned the same value as (any) one of u's neighbors in the ordering, so that $\mathbf{m}[u, u]$ has no impact on the number of runs. In the example at hand, $\mathbf{m}[u, u] = 2$ for all u.

Row node	String on row	RLE runs
x	2222222	1
y	2222222	1
w	2222222	1
z	2222222	1
n_{xy}	0122222	3
n_{xw}	0212222	4
n_{wz}	2201222	3

Table 1: First-move matrix for the running example. Both rows and columns follow the node ordering $x, y, w, z, n_{xy}, n_{xw}, n_{wz}$.

in $4e+1 = 3(n-1)+4(e-n+1)+n$ runs, with $3n-3$ runs in total for the type-e rows[3] corresponding to edges in P; $4(e-n+1)$ runs in total for the remaining type-e rows; and n runs in total for the type-n rows.

Indeed, for each edge $(u,v) \in P$, the type-e row in $\mathbf{m}$ corresponding to node $n_{uv} \in V_F$ will have 3 RLE runs, since u and v are adjacent in the ordering. There are $n-1$ edges in a Hamiltonian path, with a total number of RLE runs of $3(n-1)$ for all these rows.

For an edge $(u,v) \notin P$, the two nodes are separated and therefore the corresponding matrix row will have 4 runs. This sums up to $4(e-n+1)$ RLE runs for all rows corresponding to edges not included in the Hamiltonian path.

Conversely, consider a node ordering that creates $4e+1 = 3(n-1)+4(e-n+1)+n$ RLE runs in total. We show that the ordering has all type-n nodes as a contiguous block,[4] and that their ordering is a Hamiltonian path in G_H. This is equivalent to saying that there exist $n-1$ pairs of type-n nodes u and v such that u and v are cyclic-adjacent in the ordering, and $(u,v) \in A_H$.

Here is a proof by contradiction. Assume there are only $p < n-1$ pairs of type-n nodes u and v such that u and v are cyclic-adjacent in the ordering, and (u,v) is an edge in A_H. For each of these p pairs, the row corresponding to the type-e node n_{uv} will have 3 RLE runs. The remaining $e-p$ type-e rows will be 4 RLE runs each. As mentioned earlier, the type-n rows have n runs in total, regardless of the ordering. Thus, the total number of RLE runs is $3p+4(e-p)+n = 4e-p+n > 4e-(n-1)+n = 4e+1$. Contradiction. □

Compression for Undirected Weighted Graphs

We turn our attention to undirected weighted graphs, showing that computing an optimal ordering is NP-complete.

Definition 3. *The* FMComp-uw *problem (First Move Compression–Undirected, Weighted) is defined as follows.*
Input: An undirected weighted graph $G = (V, A)$; a matrix $\mathbf{m}$ of size $|V| \times |V|$ where a cell $\mathbf{m}[i,j]$ stores the first move on an optimal path from node i to node j; an integer k.
Question: Is there an ordering of $\mathbf{m}$'s columns such that, if we apply run length encoding (RLE) on each row, the total number of cyclic RLE runs in the matrix is at most k?

[3]We say that a row has a type-n (or type-e) iff its associated node has that type.

[4]Here, the notion of a contiguous block allows the case when part of the block is at the end of the sequence, and the other part is at the beginning, as if the sequence were cyclic.

Relation between u and v	String on row r	Nr. 1-runs
Adjacent	00110000	1
Non-adjacent	10000001	2
Non-adjacent	01001000	2

Table 2: Sample orderings and their sequential 1-runs.

As a stepping stone in proving the NP-hardness of FMComp-uw, we introduce a problem that we call SimMini1Runs (Definition 4), and prove its NP-completeness. SimMini1Runs is inspired by the work of Oswald and Reinelt (2009), who have studied the complexity of a problem involving the so-called *k-augmented simultaneous consecutive ones* property ($C1S_k$) for a 0/1 matrix (i.e., a matrix with only two symbols, 0 and 1). By definition, a 0/1 matrix has the $C1S_k$ property if, after replacing at most k 1s with 0s, the columns and the rows of the matrix can be ordered so that, for each row and for each column, all 1s on that row or column come as one contiguous block. Oswald and Reinelt (2009) have proven that checking whether a 0/1 matrix has the $C1S_k$ property is NP-complete. Our proof for SimMini1Runs is related, as we point out later in the proof.

Given a 0/1 matrix $\mathbf{o}$, an ordering of its columns, and an ordering of its rows, let the *global sequential 1-runs count* $G_1^s(\mathbf{o})$ be the number of sequential 1-runs summed over all rows and all columns. That is, $G_1^s(\mathbf{o}) = \sum_\sigma R_1^s(\sigma)$, where σ is iterated through $\mathbf{o}$'s rows and columns. For instance, $G_1^s(\mathbf{o}) = 6$ for the matrix $\mathbf{o}$ shown in Figure 2.

Definition 4. *The* Simultaneous Mini 1-Runs (SimMini1Runs) *problem is defined as follows.*
Input: A 0/1 matrix $\mathbf{o}$ so that every row and column contain at least one value of 1; an integer k.
Question: Is there an ordering of the columns, and an ordering of the rows, so that $G_1^s(\mathbf{o}) \leq k$?

Theorem 2. *SimiMini1Runs is NP-complete.*

Proof. The membership to NP is straightforward. The hardness proof uses a reduction from the Hamiltonian Path Problem (HPP) in an undirected graph. Let $G = (V, A)$ be an undirected graph, without duplicate edges, and define $n = |V|$ and $e = |A|$. We define a 0/1 matrix $\mathbf{m}$ with e rows and n columns. Let r be the row corresponding to an edge (u,v), and let c_u and c_v be the columns associated with nodes u and v. Then $\mathbf{m}[r,c_v] = \mathbf{m}[r,c_u] = 1$ and $\mathbf{m}[r,c] = 0$ for all other columns. Notice that $\mathbf{m}$ has at least a value of 1 on every row and column.

Let r be the matrix row corresponding to an edge (u,v). It is easy to see that, when a given ordering of the columns (nodes) makes the two nodes u and v adjacent, the number of sequential RLE 1-runs[5] for row r is 1. If the nodes are not adjacent, the number of sequential RLE 1-runs for row r is 2. See a few sample orderings in Table 2.

We claim the HPP has a solution iff SimMini1Runs has a solution with $t = 3e-n+2$ RLE 1-runs. Let $v_{i_1}, \ldots, v_{i_n}$

[5]All runs used in this proof are sequential.

$$\mathbf{o} = \begin{array}{c} \\ r_1 \\ r_2 \end{array} \begin{array}{c} \begin{array}{ccc} c_1 & c_2 & c_3 \end{array} \\ \left[\begin{array}{ccc} 0 & 1 & 1 \\ 1 & 0 & 1 \end{array}\right] \end{array}$$

Figure 2: Running example 0/1 matrix $\mathbf{o}$. Rows are labelled as r_i, whereas c_j represent column labels.

be a solution of HPP (i.e., a Hamiltonian path), and let P be the set of all edges included in this solution.

In every row corresponding to an edge not contained in P, switch one of its two 1-entries to 0. Then, order the columns with respect to the sequence of the nodes in the Hamiltonian path and rearrange the rows in lexicographical order.

The construction of the matrix, the trick of converting some of the 1s into 0s, and the ordering of the rows and columns are reused from Oswald and Reinelt's proof for the hardness of deciding whether a 0/1 matrix has the $C1S_k$ property (Oswald and Reinelt 2009). The rest of the proof, coming below, is significantly different.

Now, restore the previously replaced 1s. Since G has no duplicate edges, each of the $e - n + 1$ 1s that were replaced and restored have no adjacent 1s in the matrix. As such, each of them counts as two 1-runs, one horizontal and one vertical. This sums up to a total of $2(e - n + 1)$ 1-runs corresponding to the 1s replaced and restored. In addition, each row and column has one more 1-run. It follows that the matrix has $3e - n + 2$ 1-runs.

Conversely, consider a row and column ordering that creates $3e-n+2$ RLE 1-runs in total. We show that the matrix has at least $e+1$ vertical 1-runs, regardless of the row ordering. Consider the rows, in order, starting from the top. The first row introduces exactly 2 vertical 1-runs, one for each column where it contains a value of 1. Each subsequent row introduces at least one more vertical 1-run. Otherwise, the new row would be identical to the previous one, which contradicts the fact that the graph has no duplicate edges.

As there are at least $e + 1$ vertical 1-runs, the number of horizontal 1-runs will be at most $(3e - n + 2) - (e + 1) = 2e - n + 1$. We show that the column ordering is a Hamiltonian path. Assuming the contrary, there are only $p < n - 1$ edges such that their nodes are adjacent in the ordering. It follows that the number of horizontal 1-runs is $p + 2(e - p) = 2e - p > 2e - n + 1$. Contradiction. □

Theorem 3. *FMComp-uw is NP-complete.*

Proof. The NP-hardness is shown with a reduction from SimMini1Runs. Consider an arbitrary SimMini1Runs instance $\mathbf{o}$ with m rows and n columns. Figure 2 shows a running example. We build an undirected weighted graph $G = (V, A)$ as follows. V has 3 types of nodes, to a total of $m + n + 1$ nodes. Each *column* of $\mathbf{o}$ generates one node in V. We call these c-nodes. Each row generates one node as well (r-nodes). There is an extra node p called the hub node.

One r-node r_i and one c-node c_j are connected through a unit-cost edge iff $\mathbf{o}[r_i, c_j] = 1$. In addition, there is an edge with a weight of 0.75 between p and every other node. No other edges exist in graph G. See Figure 3 for an example.

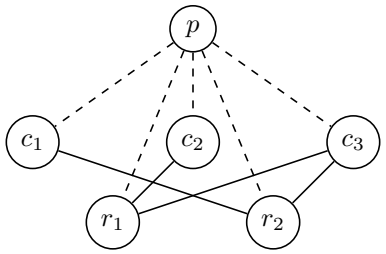

Figure 3: Graph in the running example. Dashed edges have a weight of .75, whereas solid lines are unit-cost edges.

$$\mathbf{m} = \begin{array}{c} \\ r_1 \\ r_2 \\ c_1 \\ c_2 \\ c_3 \\ p \end{array} \begin{array}{c} \begin{array}{cccccc} r_1 & r_2 & c_1 & c_2 & c_3 & p \end{array} \\ \left[\begin{array}{cccccc} 0 & 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 \\ 1 & 2 & 3 & 4 & 5 & 5 \end{array}\right] \end{array}$$

Figure 4: The first-move matrix for the running example. Without any generality loss, 0 is the move towards p. The incident edges of a given node are counted starting from 1.

Let $\mathbf{m}$ be the first-move matrix of G. The row of p has a fixed number of runs, namely $m + n$,[6] regardless of the ordering of $\mathbf{m}$'s columns. Let v be a c-node or r-node. Apart from v's adjacent nodes, all other nodes are reached through a shortest path whose first move is the edge (v, p). The matrix $\mathbf{m}$ for the running example is shown in Figure 4.

Let T_1 be the total number of occurrences of symbol 1 in matrix $\mathbf{o}$. We claim that there is an ordering of $\mathbf{o}$'s rows and columns that results in at most k sequential 1-runs (summed up for all rows and all columns) iff there is an ordering of the columns of $\mathbf{m}$ resulting in at most $k + 2T_1 + m + n$ cyclic RLE runs in total (summed up for all rows). Thus all rows of $\mathbf{m}$, except for p's row, have at most $k + 2T_1$ runs in total.

Let $r_{i_1}, \ldots r_{i_m}$ and $c_{j_1}, \ldots c_{j_n}$ be the row and column orderings in $\mathbf{o}$ that result in at most k sequential RLE runs on all rows and columns. We show that the ordering $r_{i_1}, \ldots r_{i_m}, c_{j_1}, \ldots c_{j_n}, p$ of $\mathbf{m}$'s columns generates at most $k + 2T_1 + m + n$ cyclic runs. Clearly, for every *row or column* σ in $\mathbf{o}$, there is a corresponding *row* σ' in $\mathbf{m}$. According to the steps explained earlier and illustrated in Figures 2 and 4, σ' is obtained from σ as follows. All original 0s are preserved. All original 1s are replaced with distinct consecutive integers starting from 1. In addition, σ' is padded with 0s at one or both of its ends. Since σ' has 0s at one or both its ends, $R^s_1(\sigma) = R^c_0(\sigma')$. It follows that $R^c_*(\sigma') = R^c_0(\sigma') + N_1(\sigma) = R^s_1(\sigma) + N_1(\sigma)$. Summing up $R^c_*(\sigma')$ over all rows σ' of $\mathbf{m}$, except for p's row, we obtain $\sum_{\sigma' \in \rho(\mathbf{m}) \setminus \{p\}} R^c_*(\sigma') = \sum_{\sigma \in \beta(\mathbf{o})} R^s_1(\sigma) + \sum_{\sigma \in \beta(\mathbf{o})} N_1(\sigma) \leq k + 2T_1$, where ρ denotes the set of rows of a matrix, κ is the set of columns, and $\beta = \rho \cup \kappa$. It follows

[6] Recall that $\mathbf{m}[p, p]$ is assigned the value of one of its neighbors in the ordering, so that $\mathbf{m}[p, p]$ does not impact the number of runs.

that $\mathbf{m}$'s rows have at most $k + 2T_1 + m + n$ cyclic RLE runs in total (that is, summed up for all rows).

Conversely, assume an ordering of $\mathbf{m}$'s columns with at most $k + 2T_1 + m + n$ cyclic RLE runs in total (for all rows). This means that summing up the runs of all rows of $\mathbf{m}$, except for node p's row, results in at most $k + 2T_1$ runs. As there are exactly $2T_1$ distinct runs different from 0-runs, it follows that there are at most k 0-runs in total: $\sum_{\sigma' \in \rho(\mathbf{m}) \setminus \{p\}} R_0^c(\sigma') \leq k$.

Let $r_{i_1}, \ldots r_{i_m}, c_{j_1}, \ldots c_{j_n}, p$ be a re-arragement of $\mathbf{m}$'s columns so that: all r-nodes come in one contiguous block, and their relative ordering is preserved; all c-nodes are one contiguous block, and their relative ordering is preserved. Since G restricted to c-nodes and r-nodes is bi-partite, this rearrangement cannot possibly increase the number of RLE runs. (If anything, it could eliminate some 0-runs).

We order $\mathbf{o}$'s columns as $c_{j_1}, \ldots c_{j_n}$, and $\mathbf{o}$'s rows as $r_{i_1}, \ldots r_{i_m}$. With these orderings, the relation between a row or column σ of $\mathbf{o}$ and its corresponding row σ' in $\mathbf{m}$ is as follows. All non-zero values in σ' are converted into 1s in σ. Some of σ' 0s from one or both of its ends are cut away in σ. Since σ' contains some 0s at one or both ends, $R_1^s(\sigma) = R_0^c(\sigma')$. It follows that $\sum_{\sigma \in \rho(\mathbf{o}) \cup \kappa(\mathbf{o})} R_1^s(\sigma) = \sum_{\sigma' \in \rho(\mathbf{m}) \setminus \{p\}} R_0^c(\sigma') \leq k$. □

Fighting Complexity with Decomposition

So far all our results have been negative. We have shown that computing an optimal order on a large class of graphs is NP-hard. However, experiments from (Strasser, Harabor, and Botea 2014) show that on realistic graphs it is possible to construct good orders quickly. Two heuristics were proposed: use a depth-first preorder or recursively decompose the graph along edge cuts. We investigate these techniques in a more formal setting. We show that a depth-first preorder is optimal on trees and we show that graphs can be decomposed along articulation points (which are related to cuts of size 1). Further we are able to construct optimal orders efficiently on a broader class of graphs than trees: We show that the problem is fixed-parameter tractable in the size of the largest two-connected component of the graph.

Being able to decompose graphs along articulation points is useful on real-world road networks. These graphs tend to have a large two-connected component with many small trees attached. For example the Europe graph made available during the 9th DIMACS challenge (Demetrescu, Goldberg, and Johnson 2009) has about 18M nodes in total of which only 11.8M are within the largest two-connected component. Our result allows us to position 6.2M nodes in the order fast and optimally using only local information.

We focus on graphs G with an articulation point x, i.e., a node x whose removal decomposes the graph into disjoint connected subgraphs $\{x\}, G_1 \ldots G_n$. We call an *x-block ordering* any node ordering where x comes first, the nodes of G_1 come next as a contiguous block, and so on all the way to block G_n. Let $o|_{G'}$ be the projection of the node ordering o to a subset G' of G. We say that order o is a rotation of order o' when two sub-orders α and β exist such that $o' = \alpha, \beta$ and $o = \beta, \alpha$. Let Γ_i be a shorter notation for $G_i \cup \{x\}$.

In this section we assume that cyclic runs are used and that every cell $\mathbf{m}[s, s]$ gets its own distinct symbol, called the s-singleton. This makes the proofs easier and clearly does not have a significant impact on the number of runs.

Lemma 1. *Let x be an articulation point of a graph G. Every node order π can be rearranged into an x-block ordering π' without increasing the number of runs on any row.*

Proof. For every subgraph G_i we construct a node order π_i as following: Consider the sub-order $o|_{\Gamma_i}$ that only contains the nodes of $\Gamma_i = G_i \cup \{x\}$. We then rotate this sub-order such that x comes first. The resulting sub-order is projected onto G_i (in other words, remove node x from the ordering), obtaining π_i. The desired global order o' is then constructed as: $o' = x, \pi_1, \ldots, \pi_n$. It is clear by construction that the nodes in every subgraph are consecutive. It remains to show that the number of runs per row does not grow.

Denote by s a source node. We distinguish two cases: (a) $s \in G_i$ for some i, and (b) $s = x$. The key insight in (a) is that all first-moves for shortest paths with target nodes outside of G_i have the same first move as the shortest path towards x. One can check that the reordering of nodes from o to o' has the following impact on the first-move row of s. We temporarily remove all first moves towards vertices outside of Γ_i, and rotate the row such that x comes first. Neither removing elements nor rotating elements can increase the number of runs. We then re-insert the first-moves towards the nodes outside of Γ_i, in such a way that the newly re-inserted values are cyclic-adjacent to a run with the same value, namely the run that includes the move towards x. This only increases the size of that run, but it does *not* create a new run. This concludes case (a).

For case (b), the key insight is that the first-moves from x to nodes in two different subgraphs must be different. Thus, no run can contain first-moves to multiple subgraphs G_i. Using this we can show that going from o to o' only rearranges the runs but does not cut any run into two runs. (However, it is possible that runs with the same value are arranged consecutively thus reducing the number of runs.)

Re-arranging the row r of x from o onto o' is equivalent to the following steps. We construct for every i a subrow r_i that only contains the nodes in Γ_i. Note that every run in r is in a single r_i, with the exception of the x-singleton run from x to x, which occurs in each r_i. Next we rotate all r_i, such that x comes first, to obtain r'_i. This places the x-singleton at the front and, as singletons can not wrap around, no r'_i can contain a cyclic run. To get a re-arranged row r' corresponding to o' we concatenate the x-singleton, followed by r'_1 (without x),. . ., followed by r'_n (without x). As the construction never needed to split runs, it is clear that r' contains no more non-x-singleton runs than r. Both r and r' contain exactly one x-singleton, which concludes the proof. □

Given a graph G, a node ordering o and a row subset S, let $N(o, G, S)$ be the number of runs restricted to subset S. Clearly, $N(o, G, G)$ is the total number of runs.

Lemma 2. *Given any x-block ordering o, we have that:*

1. $N(o, G, G_i) = N(o|_{\Gamma_i}, \Gamma_i, G_i)$*; and*

2. $N(o, G, \{x\}) = 1 - n + \sum_i N(o|_{\Gamma_i}, \Gamma_i, \{x\})$; *and*
3. $N(o, G, G) = 1 - n + \sum_i N(o|_{\Gamma_i}, \Gamma_i, \Gamma_i)$.

Theorem 4. *Given an optimal order o_i for every subgraph induced by Γ_i, we can construct an optimal global ordering o for G as following. Obtain new orderings o'_i by rotating o_i such that x comes first, and then removing x. Then, $o = x, o'_1, \ldots, o'_n$ is optimal.*

Proof. We show, by contradiction, that the global ordering o is optimal. Notice that $o|_{\Gamma_i}$ is optimal for Γ_i. Assume there is a strictly better ordering o'. According to Lemma 1, there exists an x-block ordering o'' at least as good as o'. We have $N(o, G, G) = 1 - n + \sum_i N(o|_{\Gamma_i}, \Gamma_i, \Gamma_i) \leq 1 - n + \sum_i N(o''|_{\Gamma_i}, \Gamma_i, \Gamma_i) = N(o'', G, G) \leq N(o', G, G)$ which is a contradiction with o' being strictly better, i.e., $N(o', G, G) < N(o, G, G)$. □

Lemma 3. *If G is a tree and o is a depth-first preorder of G (with aribtrary root) then o is a rotated x-block order for every node x.*

Proof. Every preorder induces a rooted tree. With respect to this root every node x (except the root) has a parent p and a possibly empty sequence of direct children $c_1 \ldots c_n$ ordered in the way that the depth-first search visited them. When removing x, G is decomposed into the subgraphs G_p, $G_{c_1} \ldots G_{c_n}$. If x is the root then G_p is the empty graph. The order o has the following structure: some nodes of G_p, x, all nodes of G_{c_1} … all nodes of G_{c_n}, the remaining nodes of G_p. Clearly this is a rotated x-block ordering. □

Theorem 5. *If $G = (V, E)$ is a tree and o a depth-first preorder of G then $S(o, G, G) = 3|V| - 2$.*

Proof. A direct consequence of Lemma 3 is that every node v has as many runs as its degree $d(v) + 1$. The $+1$ comes from the v-singleton. We have thus $N(o, G, G) = \sum_{v \in V}(d(v) + 1) = 2|E| + |V| = 3|V| - 2$. □

Theorem 6. *Computing an optimal order for graph G is fixed-parameter tractable in the size of largest two-connected component of G.*

Proof. Recursively decompose G at articulation points until only two-connected parts are left. As the size of the parts does not depend on the size of G we can enumerate all orders and pick the best one. Given optimal orders for every part we can use Theorem 4 to construct an optimal global order. □

Related Work

Contraction Hierarchies (CH) are a popular technique for quickly finding shortest st-paths and st-distances (Geisberger et al. 2008). This preprocessing-based algorithm takes as input a graph and an ordering over its vertices to construct a new hierarchical search space. A "good" vertex ordering in this case is one that yields a small search space. Typically CH vertex orderings have been computed using simple and efficient heuristics that rely on local graph properties; e.g. as in (Geisberger et al. 2012). More recent work (Bauer et al. 2013; Dibbelt, Strasser, and Wagner 2014) has shown that for certain types of graphs, e.g. those with bounded tree-width, there exist efficiently computable orderings which can also bound the size of the resultant search space. Computing an optimal ordering meanwhile – i.e. one which minimizes the size of the resulting search space – is known to be NP-hard (Bauer et al. 2010).

Hub Labels (HL) (Cohen et al. 2002; Abraham et al. 2011) and Hierarchical Hub Labels (HHL) (Abraham et al. 2012) are popular and successful techniques for speeding up shortest st-path and st-distance queries on road graphs. Rather than storing a compressed database of first-moves, HL and HHL store a database of compressed distances. In particular, for each node there is a forward and backward label containing a list of *hub nodes* and the exact distances to them. For each st-pair there must exist a *meeting* hub h that is a forward hub of s and a backward hub of t and is on a shortest st-path. A meeting hub can be determined using a simultaneous scan over both labels. A "good" labelling is one which reduces the overall size of the distance database. In the case of HHL a labelling is computed by first defining a partial order over the set of vertices in the input graph. Proposed orderings for HHL involve bottom-up graph contraction and greedy top-down strategies (Abraham et al. 2012) and sampling-based variations thereof (Delling et al. 2014b). Such orderings are efficient to compute in practice and have been shown to yield small labels but they do not come with any guarantees about the size of the resultant database. Computing an optimal vertex ordering for HHL is NP-hard (Delling et al. 2014a).

Conclusion

We have studied the problem of minimizing the size of a compressed path database based on run-length encoding. We have found that in the general case, for both directed and undirected graphs, the problem is NP-complete. In more specific cases, such as graphs that can be decomposed along articulation points, the graph decomposition provides a decomposition of the problem in independent sub-problems. For example, since all interior points of a tree are articulation points, trees can recursively be decomposed all the way down to sub-graphs of size one. In effect, a depth-first traversal of a tree provides an optimal node ordering.

Our results give a first theoretical underpinning to the problem of creating space-efficient CPDs using RLE. We provide a framework for understanding the very promising empirical results recently reported by Strasser, Harabor, and Botea (2014) which obtained good orderings for road networks and game grid maps. Our work also extends theoretical results from the areas of information processing and databases (Oswald and Reinelt 2009; Mohapatra 2009).

A future work topic is developing tractable approximation techniques that improve the performance of RLE in CPDs. Harnessing available results on efficient, approximate TSP algorithms might be an interesting direction to pursue.

References

Abraham, I.; Delling, D.; Goldberg, A.; and Werneck, R. 2011. A hub-based labeling algorithm for shortest paths on road networks. In *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630, 230–241.

Abraham, I.; Delling, D.; Goldberg, A.; and Werneck, R. 2012. Hierarchical hub labelings for shortest paths. In Epstein, L., and Ferragina, P., eds., *European Symposium on Algorithms (ESA)*, volume 7501 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 24–35.

Baier, J.; Botea, A.; Harabor, D.; and Hernández, C. 2014. A fast algorithm for catching a prey quickly in known and partially known game maps. *Computational Intelligence and AI in Games, IEEE Transactions on* PP(99).

Bauer, R.; Columbus, T.; Katz, B.; Krug, M.; and Wagner, D. 2010. Preprocessing speed-up techniques is hard. In *Algorithms and Complexity, 7th International Conference, CIAC 2010, Rome, Italy, May 26-28, 2010. Proceedings*, 359–370.

Bauer, R.; Columbus, T.; Rutter, I.; and Wagner, D. 2013. Search space size in contraction hierarchies. In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP)*, 93–104.

Botea, A., and Harabor, D. 2013. Path planning with compressed all-pairs shortest paths data. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*.

Botea, A.; Baier, J. A.; Harabor, D.; and Hernández, C. 2013. Moving target search with compressed path databases. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS'13)*.

Botea, A. 2011. Ultra-fast optimal pathfinding without runtime search. In *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE'11)*, 122–127.

Botea, A. 2012. Fast, optimal pathfinding with compressed path databases. In *Proceedings of the Symposium on Combinatorial Search (SoCS'12)*.

Cohen, E.; Halperin, E.; Kaplan, H.; and Zwick, U. 2002. Reachability and distance queries via 2-hop labels. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, 937–946. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

Delling, D.; Goldberg, A. V.; Savchenko, R.; and F.Werneck, R. 2014a. Hub labels: Theory and practice. In *Proceedings of the 13th International Symposium on Experimental Algorithms (SEA'14)*. Springer.

Delling, D.; Goldberg, A.; Pajor, T.; and Werneck, R. 2014b. Robust distance queries on massive networks. In Schulz, A., and Wagner, D., eds., *European Symposium on Algorithms (ESA)*, volume 8737 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 321–333.

Demetrescu, C.; Goldberg, A. V.; and Johnson, D. S., eds. 2009. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Book*. American Mathematical Society.

Dibbelt, J.; Strasser, B.; and Wagner, D. 2014. Customizable contraction hierarchies. In *Experimental Algorithms - 13th International Symposium, SEA 2014, Copenhagen, Denmark, June 29 - July 1, 2014. Proceedings*, 271–282.

Geisberger, R.; Sanders, P.; Schultes, D.; and Delling, D. 2008. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *Proceedings of the 7th International Conference on Experimental Algorithms (WEA'08)*, 319–333.

Geisberger, R.; Sanders, P.; Schultes, D.; and Vetter, C. 2012. Exact routing in large road networks using contraction hierarchies. *Transportation Science* 46(3):388–404.

Kou, L. T. 1977. Polynomial complete consecutive information retrieval problems. *SIAM Journal on Computing* 6(1):67–75.

Mohapatra, A. 2009. Optimal Sort Ordering in Column Stores is NP-Complete. Technical report, Stanford University.

Oswald, M., and Reinelt, G. 2009. The simultaneous consecutive ones problem. *Theoretical Computer Science* 410(21-23):1986–1992.

Sankaranarayanan, J.; Alborzi, H.; and Samet, H. 2005. Efficient query processing on spatial networks. In *Proceedings of the 13th Annual ACM International Workshop on Geographic Information Systems (GIS'05)*, 200–209.

Strasser, B.; Harabor, D.; and Botea, A. 2014. Fast First-Move Queries through Run Length Encoding. In *Proceedings of the Symposium on Combinatorial Search (SoCS'14)*.