Aggregating Pairwise Information Over Optimal Routes

Grzegorz Herman*[a] and Grzegorz Gawryał
Theoretical Computer Science, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland

Keywords: Road Networks, Public Transport, Pairwise Aggregation, Contraction Hierarchies.

Abstract: Public transport planning is a complex task in which many factors must be considered. One of these factors is route profitability, highly dependent on the demand for a given connection. Computing such demands quickly in a potentially changing environment is crucial in suggesting and comparing multiple alternative routes. In this preliminary paper, we propose a mathematical model for this problem, adequate for transport modes with intermediate stops on their routes. We analyze similar problems in the literature, provide some efficient algorithms with good theoretical bounds, and evaluate them on real-life road networks. We also pose open research questions related to further generalization, improvement, and better understanding of the problem.

1 INTRODUCTION

Answering optimal path queries is a fundamental problem in transportation systems. With road graphs spanning millions of vertices and edges, algorithms whose query time is even linear in the size of the graph are far too costly to be practical. State-of-the-art solutions (see e.g., (Bast et al., 2016) for a survey) feature some sort of preprocessing phase, whose product—a semantically redundant data structure—allows actual answer times to be lower, logarithmic or even constant in the size of the graph. Here, we consider a closely related, yet different problem.

For a business-level perspective, imagine that we would like to propose new, profitable, bus routes in a given road network with bus stops located along the edges. A necessary prerequisite for a hypothetical route to be profitable is a high enough demand for transportation services along this route. Computing such demand is our primary concern in this paper. However, because profitability is influenced by many factors besides just the demand, instead of simply finding some high-demand routes, we want to be able to answer multiple demand queries.

To make this a bit more specific, suppose that we are given demand information for every directed *pair* of stops (e.g., an average daily number of people interested in getting from one to the other), and assume that a bus can serve all passengers along its route, i.e., satisfy the aggregated demand for all (directed) pairs of the stops it visits.

For a moment, let us also assume that the bus must follow a graph-optimal path between its initial and final stops. Our queries then take the form of **path demand queries**: for a graph-optimal path $u = u_0 \xrightarrow{e_1} u_1 \xrightarrow{e_2} \dots \xrightarrow{e_k} u_k = v$ (given by just its endpoints u and v), return the total demand

$$d_{u \rightsquigarrow v} := \sum_{1 \leq i < j \leq k} d_{e_i, e_j},$$

where $d_{e,f}$ denotes the demand for travel from (stops along) the edge e to (stops along) the edge f.

For a more realistic scenario, we may allow the bus route to be a concatenation of a few edge-disjoint optimal paths. This leads to a natural adaptation of the above query to a **pair demand query**: given the endpoints of *two* edge-disjoint graph-optimal paths $\pi_1 : u_1 \rightsquigarrow v_1$ and $\pi_2 : u_2 \to v_2$, return the total demand

$$d_{u_1 \rightsquigarrow v_1, u_2 \rightsquigarrow v_2} := \sum_{e_1 \in \pi_1} \sum_{e_2 \in \pi_2} d_{e_1, e_2}.$$

The two types of queries are tightly related. Consider a graph-optimal path $u \rightsquigarrow v$ which passes through some intermediate vertex w. We can decompose this path into two graph-optimal paths $u \rightsquigarrow w$ and $w \rightsquigarrow v$, thus expressing the path demand $d_{u \rightsquigarrow v}$ as

$$d_{u \rightsquigarrow v} = d_{u \rightsquigarrow w} + d_{u \rightsquigarrow w, w \rightsquigarrow v} + d_{w \rightsquigarrow v}.$$

Similarly, a pair demand query from a path $u_1 \rightsquigarrow w_1 \rightsquigarrow v_1$ to a path $u_2 \rightsquigarrow w_2 \rightsquigarrow v_2$ can be calculated from the pair demands between their fragments.

[a] https://orcid.org/0000-0001-6855-8316
*This work has been commissioned by Teroplan S.A. and partially financed by European Union funds (grant number: RPMP.01.02.01-12-0572/16-01)

Herman, G. and Gawryał, G.
Aggregating Pairwise Information Over Optimal Routes.
DOI: 10.5220/0011981900003479
In *Proceedings of the 9th International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS 2023)*, pages 344-352
ISBN: 978-989-758-652-1; ISSN: 2184-495X

This leads to a natural general strategy for solving both problems: find a decomposition hierarchy for all optimal paths in the graph, precompute aggregated answers for all possible path- and pair demand queries up to some level of this hierarchy, and use the higher levels to answer arbitrary queries.

For such a solution to be practical, the decomposition must balance between two goals. On one hand, the number of precomputed queries must be low enough so that their results can be stored. Because we are considering queries between *pairs* of paths, this means that the number of such **base paths** should be low—preferably log-linear in the size of the graph. On the other hand, answering an arbitrary query requires aggregating a number of precomputed values quadratic in the length of the decomposition. Therefore, each graph-optimal path should decompose into a number of base paths at most logarithmic in the size of the graph (or, if possible, even in the length of such a path).

How does this **pairwise path aggregation problem** relate to the standard one of finding optimal paths? In one direction, all solutions to (non-aggregate) optimal path queries perform some sort of path decomposition. However, such decomposition might be highly dependent on the start and/or end node of the query, and the total number of "base" paths covering arbitrary queries might be too high for pairwise aggregation. Only when the decompositions do share enough paths, a viable solution for pairwise aggregation can be obtained.

In the other direction, any hierarchy featuring a logarithmic decomposition of optimal paths could be used to effectively answer optimal path queries. Note however, that for pairwise aggregation, we allow the preprocessed information to be at least quadratic in the size of the graph (in fact, even the input to our problem is quadratic, as it contains demand information for every pair of edges). Therefore, our preprocessing phase must be allowed at least this much time (for practical instances, even $O(n^3)$ time is acceptable with $O(n^2)$ input size)—this might be far too much for optimal path search, aimed usually at much larger graphs with possibly tens of millions of edges.

For the above reasons, even though the two problems are tightly related, it makes sense both to translate existing solutions, and to look for solutions to pairwise aggregation which are *not* suitable for shortest path queries.

This paper is a collection of ideas, results, and research questions. In particular, we:

- formally define the pairwise path aggregation problem (Section 2),
- theoretically analyse a solution based on separator hierarchies, showing that it allows $O(\log n)$ decomposition into a set of $O(n \log n)$ base paths (Section 3.1),
- discuss translations of solutions for optimal path queries, based on Contraction Hierarchies (Section 3.2) and Transit Nodes (Section 3.3),
- shortly discuss better bounds for simple graph classes (Section 3.4), and
- provide experimental evaluation of the above solutions on some practical data sets (Section 4).

We hope that this work might open a discussion on the pairwise aggregation problem. Throughout the paper, we state some research questions. Furthermore, Section 5 proposes some additional research directions.

2 THE PROBLEMS

Let us state the problems we are considering in a more formal way.

We are given a directed graph $G = (V, E)$ with edge costs $c : E \to \mathbb{R}^{\geq 0}$. A path between nodes $u, v \in V$ is called **optimal** iff its total cost is the least among such paths. Assuming optimal paths are unique, let us denote by $u \rightsquigarrow v$ the optimal path from u to v.

There is also some additional data associated with each *pair* of edges: $d : E^2 \to D$. This data can be aggregated in some commutative monoid[1] $(D, +, 0)$, from pairs of edges to paths and pairs of paths:

$$d_{u_0 \xrightarrow{e_1} \dots \xrightarrow{e_k} u_k} := \sum_{1 \leq i < j \leq k} d_{e_i, e_j},$$

$$d_{u_0 \xrightarrow{e_1} \dots \xrightarrow{e_k} u_k, v_0 \xrightarrow{f_1} \dots \xrightarrow{f_l} v_l} := \sum_{1 \leq i \leq k} \sum_{1 \leq j \leq l} d_{e_i, f_j}.$$

In the **pairwise path aggregation problem** we need to answer queries of two types:

- given two vertices u, v, return the aggregated data $d_{u \rightsquigarrow v}$, and
- given four vertices $u, v, w, x \in V$, return the aggregated data $d_{u \rightsquigarrow v, w \rightsquigarrow x}$.

We can preprocess the input data, however:

- preprocessing time should be reasonable—at most $O(|E|^3)$,
- storage for preprocessed data should not exceed the input size $\Theta(|E|^2)$ by more than a (poly)logarithmic factor, and
- query times should be low, at most $O(\log^2 |V|)$.

[1]Other algebraic structures are discussed in Section 5.2

As discussed in the Introduction, we could choose a suitable subset $B \subseteq V^2$ of ordered pairs of vertices, forming a set of **base paths**, precompute $d_{u \rightsquigarrow v}$ and $d_{u \rightsquigarrow v, w \rightsquigarrow x}$ for all $(u,v),(w,x) \in B$, and answer queries by decomposing each optimal path into base paths.

Therefore, in the **path base problem**, we are looking for a set $B \subseteq V^2$ with the following properties:

1. $|B| = O(|E| \log |V|)$,
2. for each optimal path $u_0 \to \ldots \to u_k$, there exist at most $O(\log |V|)$ indices $0 = b_0 < b_1 < \ldots < b_r = k$ such that $(u_{b_i}, u_{b_{i+1}}) \in B$ for each $0 \le i < r$.

3 ALGORITHMS

We now discuss a few solutions to the path base problem (and thus, to the pairwise path aggregation problem), beginning with one based on computing an edge separator hierarchy, and then on translations from the optimal path query problem.

3.1 Separator-Based algorithm

Given a graph $G = (V,E)$ with $|V| \ge 2$, an **edge separator** is a set of edges $S \subseteq E$ such that the removal of S increases the number of connected components of G.

An edge separator is (α,β)**-balanced**, if it has size at most $\beta\sqrt{|V|}$, and each connected component formed by the removal of that separator has size at most $\alpha|V|$, for some positive constants $\alpha < 1$ and β.

A graph $G = (V,E)$ has an (α,β) **edge separator hierarchy** if its every sufficiently large subgraph has an (α,β)-balanced edge separator.

In this section, we model road networks as graphs embedded on the plane, whose edge crossing graphs have bounded degeneracy—see (Eppstein and Gupta, 2017) for details. Such graphs have vertex separator hierarchies (Eppstein and Gupta, 2017), but for the existence of edge separator hierarchy we also need a bounded degree of graph vertices. To achieve this, we can apply to each (high degree) vertex a local degree reduction scheme shown in Figure 1. This method reduces the degree to 3, while increasing the total numbers of vertices and edges by $4|E|$, and the degeneracy of the crossing graph by at most 1.

We will now present an algorithm for the path base problem for graphs with an (α,β) edge separator hierarchy. The algorithm will first construct a **separator tree**, and then use it to create the required set B.

Each node of the separator tree will correspond to some induced subgraph of the input graph G (the root covering the whole of G), represented by the set of its

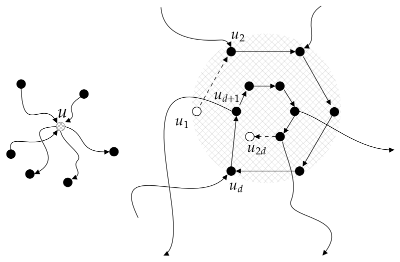

Figure 1: Degree reduction scheme. A vertex of degree d is replaced with a spiral of $2d$ vertices, consisting of an outer and an inner loop, handling respectively the incoming and outgoing edges. Edges on the spiral have zero cost and zero aggregable data. Queries related to the original vertex are remapped to one of the endpoints of the edge between the outer and the inner loop. After the reduction, previously edge-disjoint paths may share loop edges—this, however, has no influence on the aggregated data. Uniqueness of optimal paths is preserved.

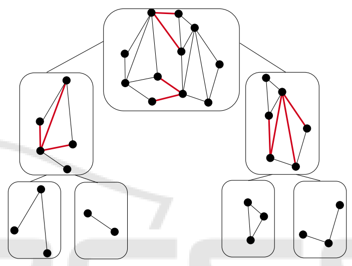

Figure 2: Separator tree.

vertices $X \subseteq V$. If $|X| < c$ (for some constant c), then X will be a leaf node. Otherwise, the node will have exactly two children Y and Z, where

- $\{Y,Z\}$ is a partition of X,
- the set S_X of edges between Y and Z forms an (α,β)-balanced separator of X.

A tree with these properties can be created by straightforward recursion, because all sufficiently large subgraphs of G are assumed to have (α,β)-balanced edge separators.

Additionally, for each leaf node X we define S_X to be the set of all edges in its induced subgraph. Because an (α,β)-balanced separator is also (α,β')-balanced for every $\beta' > \beta$, letting $\beta \ge c\sqrt{c}$ we can guarantee that $|S_X| \le \beta\sqrt{|X|}$ for every node X.

For an edge set $F \subseteq E$, let

$$\eta(F) := \{u : \exists v : (u,v) \in F \vee (v,u) \in F\}$$

be the set of endpoints of any edge in that set. Defining the depth of a node in the standard way (the root has depth 1, and children of a node of depth i have depth $i+1$), let us assign to each vertex $u \in V$ a **level** $l(u)$, being the minimum depth of a node X, such that $u \in \eta(S_X)$.

From the balanced separator property, we know that $|Y| < \alpha|X|$ whenever Y is a child node of X, and hence the height of the tree is at most

$$L = O(\log_{\frac{1}{\alpha}} |V|) = O(\log |V|).$$

Furthermore, since the children of each node form its partition, each vertex of G will be present in at most $O(\log |V|)$ nodes of the tree, each of different depth.

Now, we can define the set

$$B' := \{(u,v) : l(u) \geq l(v) \wedge (\exists X : u \in \eta(S_X) \wedge v \in X)\}$$

and show that $B := B' \cup \{(u,v) : (v,u) \in B'\}$ satisfies the requirements for a path base.

Fix an arbitrary vertex v, and denote by Y_i the unique node of depth i containing v. Now consider a pair $(u,v) \in B'$, witnessed by a node X with $u \in \eta(S_X)$ and $v \in X$. By definition, $l(u)$ must not exceed the depth of X, and therefore X must be equal to Y_i for some $i \geq l(u) \geq l(v)$. The number of such (u,v) pairs with $X = Y_i$ is bounded by

$$|\eta(S_{Y_i})| \leq 2\beta\sqrt{|Y_i|} \leq 2\beta\sqrt{\alpha}^{\,i-l(v)}\sqrt{|Y_{l(v)}|},$$

which summed over all $i \geq l(v)$ gives at most

$$\frac{2\beta}{1-\sqrt{\alpha}}\sqrt{|Y_{l(v)}|}.$$

We call all the pairs accounted for in this way **rooted at the set** $Y_{l(v)}$.

Now fix an arbitrary node Y, and denote by l its depth. Consider a vertex $v \in Y$ with $l(v) = l$. The level must be witnessed by $v \in \eta(S_X) \subseteq X$ for some node X at depth l. However, Y is the unique such node. Therefore, the number of such vertices in Y is bounded by

$$|\eta(S_Y)| \leq 2\beta\sqrt{|Y|},$$

from which we can bound the number of pairs in B' rooted at Y by

$$\frac{4\beta^2}{1-\sqrt{\alpha}}|Y| = O(|Y|).$$

Because the nodes at each single depth are disjoint, the number of pairs in B' rooted at such a depth is bounded by $O(|V|)$. Summing these over all possible depths gives us

$$|B| \leq 2|B'| = O(|V|\log|V|).$$

For assessing the size of decomposition of any optimal path $u \rightsquigarrow v$ into base paths, let w_0 (respectively, w_0') be the first (last) vertex having the lowest level on this path $u \to \cdots \to v$. Since B is symmetric, it suffices to compute the decomposition length of $u \to \ldots \to w_0$. The subpath $w_0' \to \ldots \to v$ can then be handled analogously, and the subpath $w_0 \to \ldots \to w_0'$ (if $w_0 \neq w_0'$) will be a base path because there are no vertices with smaller level between w_0 and w_0', and thus they must be endpoints of some edges in the same separator set.

Let us then focus on the path $u \to \ldots \to w_0$. If nonempty, it must be fully contained in a single child of some node X with $w_0 \in \eta(S_X)$—it is the node witnessing the level of w_0. Let w_1 be the leftmost vertex on our path with the lowest level (except for w_0). The level $l(w_1) > l(w_0)$ must be witnessed by some node Y, with w_1 being an endpoint of some edge in S_Y.

No separator at any depth $l(w_0) < l < l(w_1)$ separates w_1 from w_0—otherwise some vertex on this subpath would have level l, contradicting the choice of w_1. Thus, $w_0 \in Y$ and $(w_1, w_0) \in B$.

Now, all vertices on path $u \rightsquigarrow w_1$ have levels strictly smaller than $l(w_1)$, so we can recursively decompose this path. The depth of this recursion will be at most the height L of the separator tree, hence the total decomposition size of $u \rightsquigarrow v$ is at most $2L+1 = O(\log|V|)$.

3.2 Contraction Hierarchies

Contraction Hierarchies (Geisberger et al., 2012) are one of the most popular methods of answering shortest path queries. The preprocessing phase of this method sequentially contracts vertices in some chosen order, creating shortcuts (but preserving optimal path lengths), which are later used to quickly answer optimal path queries.

It can be adapted to perform base path decomposition by simply creating a single base path for every edge in the original graph and for every shortcut created during contraction. After the base paths are chosen, we can precompute for each pair of vertices the shortest decomposition into a sequence of base paths. To achieve this, it suffices to add shortcuts to the original graph (with appropriate weights) and then compute all pairs shortest paths in that network, where "shortest" should first minimize the original path cost and later the number of edges used. This can be computed in $O(|E||V|\log|V|)$ time, and stored in $O(|V|^2)$ space.

Various heuristics have been proposed to produce a contraction order yielding both a small number of shortcuts and guaranteeing a small search space afterwards (see for example (Geisberger et al., 2008) and (Funke and Storandt, 2015)). For our setting, the former directly translates to the total number of base paths, but the latter is not inherently related to the length of the decomposition. Therefore we revisit here the most popular heuristics.

3.2.1 Nested Dissection Order

Since road networks have small vertex separators, we can temporarily remove the vertices lying in the separator set, recursively contract the components of the resulting graph, and finally, contract the vertices of the separator. This "nested dissection" method was first analyzed in (Gilbert and Tarjan, 1986). Later, (Milosavljević, 2012) and (Columbus, 2013) applied it to contraction hierarchies, and shown that it would generate $O(|V|\log|V|)$ shortcuts.

Analogously to what we have done in Section 3.1, we can decompose any optimal path according to the "levels" corresponding to the recursive dissection, obtaining a similar $O(\log|V|)$ upper bound for the decomposition length.

3.2.2 Random Contraction Order

A much simpler method is to contract vertices in random order. As we will show in this subsection, this yields (in expectation) very similar results to the nested dissection order.

Let us first look at the decomposition length. The following observation is a direct consequence of Lemma 5 in (Blum et al., 2021):

Observation 1. *For any optimal path $v_0 \to v_1 \to \ldots \to v_n$, the pair (v_0, v_n) forms a base path if and only if v_0 and v_n are both contracted later than all of $v_1, \ldots, v_{n-1}$.*

For any vertices $v_0 \neq v_n$ of a directed graph G, we will estimate the expected value of the decomposition length of the optimal path $v_0 \to \ldots \to v_n$.

Let v_m be the vertex on that path with the largest rank (i.e., contraction time). Then, the decomposition length of the path is equal to the sum of decomposition lengths of $v_0 \rightsquigarrow v_m$ and $v_m \rightsquigarrow v_n$. Since the analysis of both parts is very similar, let us only focus on the former.

For all $i \in \{0, \ldots, m\}$, let X_i be a random variable indicating whether $\text{rank}(v_i)$ is larger than all of $\text{rank}(v_0), \ldots, \text{rank}(v_{i-1})$. Note, that $X_0 = 1$ land $X_m = 1$. Let us look at the largest index $j < m$ for which $X_j = 1$. From Observation 1, (v_j, v_m) forms a base path, and since the rank of v_j is larger than all previous values, we can recursively decompose $v_0 \rightsquigarrow v_j$. Therefore, the expected decomposition length of the path $v_0 \rightsquigarrow v_m$ is equal to

$$\mathbb{E}(\sum_{i=1}^{m} X_i) = \sum_{i=1}^{m} \frac{1}{i} = O(\log m),$$

and hence the expected decomposition length of the path $v_0 \rightsquigarrow v_n$ is $O(\log n)$.

The above result holds for any class of graphs. The assessment of the expected size of the base path set has been done for the bounded growth model (Blum et al., 2021). In this model, we assume for all $r \in \mathbb{N}$, that the number of vertices reachable from any single node using at most r edges is $O(r^2)$. Under these conditions, the expected number of shortcuts created in the contraction hierarchy is $O(|V|\log|V|)$.

Remember however, that we are precomputing information for every *pair* of base paths and we are answering queries for *pairs* of optimal paths. Therefore, we actually need to calculate the expected value of the squared size of the base paths set and the expected value of the product of decomposition lengths for any two optimal paths. We can rephrase these as the following questions, yet unanswered:

Question 1. *When contracting vertices in random order, what is the variance of the size of the base path set?*

Question 2. *When contracting vertices in random order, for any two optimal paths, what is the covariance between the decomposition lengths of these paths?*

For real-life applications, the latter question does not need to be that general—if we will only be answering path aggregation queries, all we need is the expected value of the squared decomposition length of a single path $v_0 \to \ldots \to v_n$.

Note, that the previously introduced indicator variables X_i and X_j are independent for $i \neq j$, and so the expected value of the squared decomposition length is $O(\log^2 n)$, which matches the deterministic result for the separator-based algorithm.

For the case, mentioned in the Introduction, of a concatenation of a few optimal paths, we would also need pair queries over edge-disjoint paths $v_0 \rightsquigarrow v_n$ and $u_0 \rightsquigarrow u_m$, sharing only some small number k of vertices. To analyze this case, we can partition each of the two input paths into $k+1$ segments between these shared vertices, and take advantage of the following observation:

Observation 2 (Triangle inequality for decomposition length). *For a graph $G = (V, E)$, an optimal path $u \rightsquigarrow w$, and a vertex $v \in u \rightsquigarrow w$, we have*

$$\delta(u \rightsquigarrow v) + \delta(v \rightsquigarrow w) \geq \delta(u \rightsquigarrow w),$$

where $\delta(\pi)$ denotes the decomposition length of the path π.

Now, denoting the shared vertices as $u_{i_1}, \ldots, u_{i_k}$ (according to their positions on the path $u_0 \rightsquigarrow u_m$), we can estimate the expected product of decomposition lengths as

$$\mathbb{E}(\delta(v_0 \rightsquigarrow v_n)\delta(u_0 \rightsquigarrow u_m)) \leq$$

$$\mathbb{E}\Big(\delta(v_0 \rightsquigarrow v_n)\big(\delta(u_0 \rightsquigarrow u_{i_1-1})+$$

$$+\delta(u_{i_1-1} \rightsquigarrow u_{i_1+1}) + \delta(u_{i_1+1} \rightsquigarrow u_{i_2-1}) +$$
$$\dots$$
$$+\delta(u_{i_k-1} \rightsquigarrow u_{i_k+1}) + \delta(u_{i_k+1} \rightsquigarrow u_m))\Big).$$

For vertex-disjoint paths, their decomposition lengths are independent. The length-three subpaths introduce only a constant factor to the product of decomposition lengths. Therefore,

$$\mathbb{E}(\delta(v_0 \rightsquigarrow v_n)\delta(u_0 \rightsquigarrow u_m)) =$$
$$O(3k \log n + (k+1) \log n \log m) =$$
$$O(k \log^2 |V|).$$

In particular, for a constant k it is also $O(\log^2 |V|)$.

3.2.3 Deleted Neighbours Order

Another contraction order suggested in (Geisberger et al., 2008) is the deleted neighbours order, in which we greedily contract a vertex with minimal number of already contracted neighbours. Even though this method can produce quadratic number of base paths for some trivial classes of graphs (e.g., star graphs), all such counterexample classes we found have vertices with non-constant degree. Moreover, this method performed best in our experimental evaluation. Hence, the following questions arises naturally:

Question 3. *What are the theoretical upper and/or lower bounds for both base path set size and decomposition length for the deleted neighbours order for road networks?*

3.3 Transit Nodes

Another algorithm for which some theoretical guarantees were stated is the Transit Nodes algorithm (Bast et al., 2007). In this method, we carefully pick a set of access nodes $A(u)$ for each vertex u of the original graph, and then form the set of transit nodes $T = \bigcup_{v \in V} A(v)$. Now, each long enough path from $u \rightsquigarrow v$ can be decomposed into three parts:

- from u to some node $a_u \in A(u)$,
- similarly, from some node $a_v \in A(v)$ to v, and
- from a_u to a_v.

Originally, we precompute the distance table between any pair of transit nodes to answer these queries rapidly. For short paths, we simply run another shortest path algorithm, e.g. bidirectional Dijkstra. The definition of being long enough and the set of access nodes can be specified in various ways to achieve different guarantees.

In our setting, it looks natural to create a base path for each edge, for each pair of transit nodes, and for each pair of a node and its access node. This approach would however produce shortest path decomposition of length at most 3 for all long enough paths, and thus it seems unlikely to yield subquadratic number of base paths at the same time.

3.4 Better Bounds for Special Graph Families

Except for Transit Nodes, all aforementioned algorithms achieve $O(|V| \log |V|)$ base path size and $O(\log |V|)$ decomposition length of any path for road networks.

A natural question to ask is whether these results are optimal. There could possibly exist some other decomposition methods that would result in only a slightly larger base path size, but with sublogarithmic decomposition length.

For some subfamilies of road network graphs, the answer to this question is positive. In the next subsections, we consider line graphs and trees and describe alternative algorithms based on Transitive Closure Spanners (Bhattacharyya et al., 2012).

3.4.1 Line Graphs

A k-transitive-closure-spanner (in short k-TC-spanner) of a directed graph $G = (V, E)$ is a directed graph $H = (V, E_H)$, where E_H is a subset of the edge set of the transitive closure of G, and the distance (in terms of the number of edges) between any two vertices connected in G is at most k.

For a line graph G, let us orient all the edges in one direction. In the transitive closure of a such graph, there will be a directed edge between every pair of vertices. Note, that if $H = (V, E_H)$ is a k-TC-spanner of G, then

$$\{(u,v) : (u,v) \in E_H \lor (v,u) \in E_H\}$$

is a base path set having $2|E_H|$ edges and decomposition length at most k for any path.

k-TC-spanners of line graphs have been analysed in (Raskhodnikova, 2010) and (Alon and Schieber, 1987). Particularly, for any k there exists a k-TC-spanner of size $O(|V|\lambda_k(|V|))$, where λ_k is the k-th row inverse Ackermann function. On the other hand, they have shown that with $O(|V|)$ base path set size, an optimal decomposition length is precisely $O(\lambda(|V|))$, where λ is the inverse Ackermann function.

3.4.2 Tree Graphs

Similarly, we can orient the edges of any tree from the root to the leaves, forming a directed tree G. Then, any path in the undirected tree can be decomposed into at most two paths from the lowest common ancestor of its endpoints. Therefore, if $H = (V, E_H)$ is a k-TC-spanner of G, then again, $\{(u,v) : (u,v) \in E_H \vee (v,u) \in E_H\}$ is a base path set, this time having decomposition length at most $2k$.

As noted in (Raskhodnikova, 2010) and proven in (Alon and Schieber, 1987), the optimal k-TC-spanners for lines and trees have asymptotically the same number of edges, and therefore the results obtained for line graphs also apply here.

These better algorithms raise the following question:

Question 4. *Can we achieve similar better bounds for wider classes of (or even all) road networks?*

4 EXPERIMENTAL EVALUATION

We have analysed the behaviour of the above algorithms in practice. We used two real-life road networks with distances, each having a few thousand vertices. Both datasets assumed that all roads are bidirectional, so the distance matrices were symmetric. For each data set, we selected the largest connected component and contracted all vertices of degree 2.

Our datasets are listed below:

- Major road network of California, generated from (Li et al., 2005): 1364 vertices and 1959 edges.
- Road network of Oldenburg, generated from (Brinkhoff, 2000): 2930 vertices and 3750 edges.

We computed the base path set size and average squared decomposition length over all optimal paths. (for random contraction order, we averaged the results over 10 runs). For finding separators in the separator-based algorithm, we first planarized the graphs by creating auxiliary vertices in place of line crossings and then used a dynamic programming method over spanning trees of that graph and its dual.

To find the access nodes set in the Transit Nodes method, we used the ε-net construction, described in (Blum et al., 2021), attempting to find ε-net for radius $r = 30$. We started with $k = |V|$, trying 10 times to get an ε-net of size k by randomly selecting vertices. On failure, k was multiplied by 1.2, and the search was retried. The value $r = 30$ was chosen so that even the "short" queries (not going through an access node) were guaranteed to have a short decomposition.

The plots in Figures 3 and 4 show the distributions of squared decomposition lengths for both datasets. Numbers in parentheses represent the size of the base path set for each method.

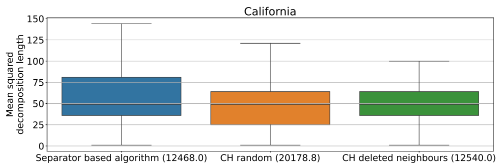

Figure 3: California: squared decomposition length (outliers omitted).

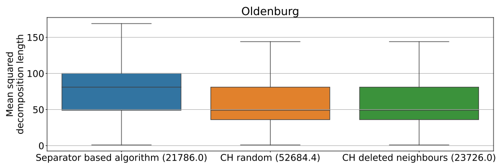

Figure 4: Oldenburg: squared decomposition length (outliers omitted).

As expected, the Transit Nodes method yielded an average decomposition length close to 3, but the number of base paths was huge—too large to be useful for pairwise aggregation. Therefore we have omitted it from the plots.

We can conclude, that the deleted neighbours method achieves the best results on both datasets, having the shortest average squared decomposition length while keeping the base path set size only slightly larger than the separator-based algorithm. On the other hand, the other methods (excluding Transit Nodes) still yield quite competitive results and can be useful in practice.

5 RESEARCH DIRECTIONS

5.1 Demand Updates

In real-life applications, we might want to be able to efficiently update the demand d_{e_i,e_j} for a single pair of edges. It requires modifying precomputed information on all base path pairs $(u_1 \rightsquigarrow v_1, u_2 \rightsquigarrow v_2)$, for which $e_i \in u_1 \rightsquigarrow v_1$ and $e_j \in u_2 \rightsquigarrow v_2$. The update time would be proportional to that number (assuming constant time operations in the data monoid), and so ideally, we would like to have a decomposition method with some guaranteed upper bounds.

Similarly to the decomposition length, in non-randomized algorithms we can focus just on the **affected base path set** $A(e)$ for a given edge e.

We have extended our experimental evaluation to include the distribution of $|A(e)|$ over all edges in each dataset. As shown in Figures 5 and 6, the average size of $A(e)$ for the deleted neighbours and separator-based methods is quite small, while for random contraction order it is significantly larger.

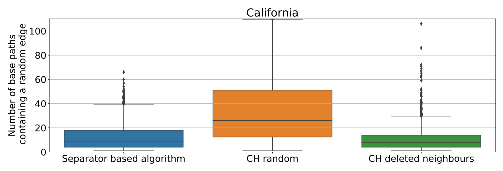

Figure 5: California: the expected number of base paths containing a random edge.

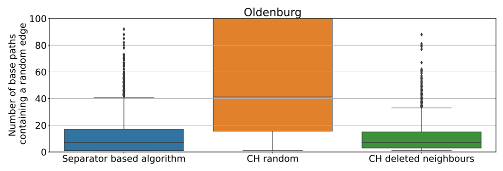

Figure 6: Oldenburg: the expected number of base paths containing a random edge.

For line graphs, using standard segment trees we can achieve guaranteed $O(|V|)$ base path size, $O(\log|V|)$ decomposition length, and $O(\log|V|)$ affected base path set. For the more general case, we leave open the following question:

Question 5. *Are there any upper/lower bounds for the average/pessimistic affected base path size for any of the presented decomposition algorithms?*

5.2 Subtractive and Idempotent Bases

So far, we have assumed the data domain to be a commutative monoid. It seems natural to consider a strengthening of its structure to a commutative *group*. This allows the use of **subtractive decomposition**, in which a path may be covered by overlapping "positive" and "negative" base paths, as long as the total number (sign included) of base paths containing each edge is exactly one.

In case of trees, simply connecting each vertex with the (arbitrarily chosen) root yields a path base of size $O(|V|)$ with decomposition length $O(1)$.

Instead of having a group structure, we might require the monoid operation to be idempotent (this happens for example when the aggregated information is some kind of maximum). Then the base path set must allow an **idempotent decomposition** of any path, in which each edge may be covered an arbitrary positive number of times.

Note that in both above cases, solutions to the aggregation and path base problems can no longer be directly used to efficiently answer optimal path queries, as the summed number of edges covered by base paths in a decomposition could is general be significantly larger than the length of the resulting path. This opens up the following interesting research direction:

Question 6. *Find efficient solutions to the path base problem on road networks, when the path base is allowed to be subtractive or idempotent.*

5.3 Sparse Demand Matrix

To this point, we have assumed the input demands to be given as a full matrix, of size $\Theta(|E|^2)$. In practice, many entries of this matrix could be negligibly small or even equal to zero. Thus, we could approximate it with a sparse matrix, having only some set Q of non-zero entries (with $0 \leq |Q| \leq |E|^2$). If the pairwise aggregation problem can in this case be solved efficiently with smaller amount of preprocessed data, one could apply it to larger graphs.

For solutions with small affected base path sets, one could start with an empty sparse matrix with precomputed pairwise base path data, and treat each member of Q as an update. For example, with affected base path sets of size $O(\log|V|)$, one would obtain a final preprocessed data structure of size $(|Q|\log^2|V|)$, scaling nicely with the amount of input information[2].

We thus pose a final open question:

Question 7. *Can the pairwise aggregation problem be solved efficiently with space requirements scaling gently with the number of non-zero data entries?*

6 CONCLUSIONS

In this work, we have stated a new problem of aggregating pairwise information over optimal graph routes. In our opinion, the problem is worth gaining more attention because of both its practical applications and interesting theoretical properties, derived from relationships with shortest path query problems and segment-tree-like data structures.

We have presented algorithms leading to acceptable time and space complexities, and hope that they can be improved even further, especially wrt. space consumption. Practical, implemented solutions to the proposed problem could form an important component of a larger system for optimizing and proposing new routes for public transportation. In particular, we hope that answering some of the open questions stated in this paper can bring us closer to this goal.

[2]This does not include the information necessary to perform decomposition of each arriving query, which may depend on the decomposition method.

REFERENCES

Alon, N. and Schieber, B. (1987). *Optimal Preprocessing for Answering On-line Product Queries*. Tel-Aviv University. The Moise and Frida Eskenasy Institute of Computer Sciences.

Bast, H., Delling, D., Goldberg, A., Müller-Hannemann, M., Pajor, T., Sanders, P., Wagner, D., and Werneck, R. (2016). *Route Planning in Transportation Networks*, volume 9220, pages 19–80.

Bast, H., Funke, S., Sanders, P., and Schultes, D. (2007). Fast routing in road networks with transit nodes. *Science*, 316(5824):566–566.

Bhattacharyya, A., Grigorescu, E., Jung, K., Raskhodnikova, S., and Woodruff, D. P. (2012). Transitive-closure spanners. *SIAM Journal on Computing*, 41(6):1380–1425.

Blum, J., Funke, S., and Storandt, S. (2021). Sublinear search spaces for shortest path planning in grid and road networks. *J. Comb. Optim.*, 42(2):231–257.

Brinkhoff, T. (2000). Generating network-based moving objects. pages 253 – 255.

Columbus, T. (2013). Search-space size in contraction hierarchies. In *40th International Colloquium on Automata, Languages, and Programming (ICALP'13), volume 7965 of Lecture Notes in Computer Science, pages 93–104.* Springer.

Eppstein, D. and Gupta, S. (2017). Crossing patterns in nonplanar road networks. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '17, New York, NY, USA. Association for Computing Machinery.

Funke, S. and Storandt, S. (2015). Provable efficiency of contraction hierarchies with randomized preprocessing. pages 479–490.

Geisberger, R., Sanders, P., Schultes, D., and Delling, D. (2008). Contraction hierarchies: Faster and simpler hierarchical routing in road networks. pages 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.

Gilbert, J. R. and Tarjan, R. E. (1986). The analysis of a nested dissection algorithm. *Numerische Mathematik*, 50:377–404.

Li, F., Cheng, D., Hadjieleftheriou, M., Kollios, G., and Teng, S.-H. (2005). On trip planning queries in spatial databases. volume 3633, pages 273–290.

Milosavljević, N. (2012). On optimal preprocessing for contraction hierarchies. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, IWCTS '12, page 33–38, New York, NY, USA. Association for Computing Machinery.

Raskhodnikova, S. (2010). Transitive-closure spanners: A survey. In Goldreich, O., editor, *Property Testing: Current Research and Surveys*, pages 167–196, Berlin, Heidelberg. Springer Berlin Heidelberg.