Indexing Graphs for Shortest Beer Path Queries

David Coudert ✉ 📧
Université Côte d'Azur, Inria, I3S, CNRS, Sophia Antipolis, France

Andrea D'Ascenzo[1] ✉ 📧
Luiss University, Rome, Italy

Mattia D'Emidio ✉ 📧
University of L'Aquila, Italy

Abstract

A *beer graph* is an edge-weighted graph $G = (V, E, \omega)$ with *beer vertices* $B \subseteq V$. A *beer path* between two vertices s and t of a beer graph is a path that connects s and t and visits at least one vertex in B. The *beer distance* between two vertices is the weight of a *shortest beer path*, i.e. a beer path having minimum total weight. A *graph indexing scheme* is a two-phase method that constructs an *index* data structure by a one-time preprocessing of an input graph and then exploits it to compute (or accelerate the computation of) answers to *queries* on structures of the graph dataset. In the last decade, such indexing schemes have been designed to perform, effectively, many relevant types of queries, e.g. on reachability, and have gained significant popularity in essentially all data-intensive application domains where large number of queries have to be routinely answered (e.g. journey planners), since they have been shown, through many experimental studies, to offer extremely low query times at the price of limited preprocessing time and space overheads.

In this paper, we showcase that an indexing scheme, to efficiently execute queries on beer distances or shortest beer paths for pairs of vertices of a beer graph, can be obtained by adapting the highway labeling, a recently introduced indexing method to accelerate the computation of classical shortest paths. We design a preprocessing algorithm to build a WHL index, i.e. a weighted highway labeling of a beer graph, and show how it can be queried to compute beer distances and shortest beer paths. Through extensive experimentation on real networks, we empirically demonstrate its practical effectiveness and superiority, in terms of offered trade-off between preprocessing time, space overhead and query time, with respect to the state-of-the-art.

---

[1] Corresponding author.

1   Introduction

Determining detours constitutes a major decision process in our daily lives and in modern computing systems: whenever moving from point A to point B, we might need to deviate from the main itinerary in order to stop to a gas station or to buy a beer not to show up empty-handed to a friend one is going to visit. Similarly, in multi-agent systems, we might be interested in planning paths for agents that traverse one of any location of a certain type, for instance to pick-up a specific tool. Finally, in multi-hop communication networks, it might be useful to route data packets through nodes with certain characteristics, e.g. to achieve some form of resiliency. What is the fastest way to accomplish these goals? To model such decision processes from a computational viewpoint, Bacic et al. [3, 4] introduced the notions of *beer graph* and *beer paths*, and defined corresponding optimization problems. A *beer graph* is a graph $G = (V, E, \omega)$, with weight function $\omega : E \mapsto \mathbb{R}^+$ on the edges, where a set of special vertices $B \subseteq V$, called *beer vertices*, is given. A *beer path*, between two vertices s and t of a beer graph, is any path of G from s to t that visits at least one vertex in B whereas a *shortest beer path* for two vertices s and t is a beer path having minimum total weight (called the *beer distance*), i.e. minimizing the sum of the weights of its edges.

Surprisingly, while determining shortest beer paths (and corresponding distances) is a computational problem that arises naturally in a wide range of modern applications, and notwithstanding the fact that such problem can be seen as a special case of the *generalized shortest path* problem [29, 35], algorithmic issues related to such problem have been only recently formalized and investigated [3, 4, 11, 22, 23]. In more details, although a beer path may be a non-simple path (i.e. it might self-intersect), it can be easily shown that any shortest beer path, for a given pair of vertices s, t, always consists of two shortest paths: one from s to a beer vertex, say w, and one from w to t. In other terms, a beer distance can be always determined by finding the minimum, overall beer vertices $w_i \in B$, of the sums of the shortest path distances from s to w_i and from w_i to t. This characterization provides a baseline algorithm for determining shortest beer paths and beer distances for a pair s, t, namely: grow two shortest path trees by Dijkstra's algorithm [16] rooted, respectively, at s and t, and select the beer vertex that minimizes the sum of the distances from s and to t [3, 4]. Unfortunately, while this strategy is simple and considered efficient, as Dijkstra's algorithm runs in almost linear time in the size of the graph [16], many experimental works in the past two decades have shown how employing Dijkstra's algorithm is impractical in many real-world contexts where either the algorithm has to be executed on an interactive basis or when moderately to massively sized networks have to be handled since, in such contexts, it can take up to seconds to compute even a few shortest paths [1, 25, 32].

Practical limitations of shortest path algorithms have motivated the design of several so-called *graph indexing schemes* for shortest paths, i.e. two-phase methods that: (i) perform an *offline*, one-time *preprocessing* phase on the graph to compute auxiliary data, generally stored in a data structure called *index*; (ii) exploit the index, in an *online* phase and upon *query*, for very fast retrieval of shortest paths for (possibly many) pairs of vertices. Due to the excellent performance in practice, combining extremely low query times to find shortest paths (orders of magnitude smaller than methods without indexing) with limited preprocessing time and space overheads (even in large graphs) [1, 2], these schemes have gained significant popularity and have become the state-of-the-art for shortest paths retrieval in all application

domains where large number of queries have to be routinely answered (e.g. journey planners or network analytics software) [5, 6, 13–15, 20]. Moreover, such popularity has inspired the development of similar schemes to support, with comparable effectiveness, many other relevant types of queries on graphs (e.g. on reachability [31], on top-k shortest paths [12], on path counts [33], or on communities [34]).

Indexing schemes for queries on beer distances or shortest beer paths have been only very recently considered in the literature. Specifically, some schemes with theoretical guarantees, either on the space occupancy of the index or on preprocessing and query times, have been designed only for special graph classes [4, 11, 23]. For general graphs, instead, no indexing scheme able to support the computation of beer distances or shortest beer paths faster than the baseline, neither theoretically nor in practice, has been developed. To this end, a straightforward way to index a graph for accelerating queries on beer distances and shortest beer paths could be precomputing and storing, into a matrix, distances (or corresponding shortest paths) from all beer vertices to all other vertices of the graph. This approach, which we call B2ALL, translates into a simple and fast query algorithm which can retrieve, (i) the beer distance in $\mathcal{O}(|B|)$ time, by finding the beer vertex minimizing the sum of distances to the two queried vertices; (ii) a shortest beer path of weight ℓ in $\mathcal{O}(|B|\ell)$ time, by unrolling the path that minimizes the sum of the distances. An alternative to the above method, worth being considered, is adapting one of the many indexing schemes designed for the more general and complex problem of determining generalized shortest paths, i.e. paths having minimum weight among those which traverse at least one vertex for each of a set of vertex categories [17, 24, 27, 29, 30, 35]. To apply such methods to beer distances and shortest beer paths, in fact, it suffices to restrict vertex categories to be a single category that contains all beer vertices. For instance, by applying this restriction to the indexing scheme K-SKY, given in [24], one of the best in terms of offered trade-off between preprocessing cost and query performance for generalized shortest paths, one could obtain an algorithm that takes $\mathcal{O}(|B|^2 n)$ ($\mathcal{O}(|B|^2 n\ell)$, respectively) time to answer a query on the beer distance (on a shortest beer path, respectively), per vertex pair.

However, while both above strategies are appealing in terms of query performance, significantly better than that of two executions of Dijkstra's, it remains unclear whether they are applicable in real-world data-intensive scenarios, due to their high preprocessing time and space occupancy overheads. In both cases, in fact, the preprocessing step takes $\mathcal{O}(n^2|B|)$ time while the resulting index data structure has size $\Omega(n|B|)$ and $\mathcal{O}(n|B|)$, respectively, for any n-vertex graph. To the best of our knowledge, no experimental study has been concerned with the assessment of the average performance of existing indexing methods to support queries on beer distances or shortest beer paths, for both general graphs and special graph classes.

Our Contribution. In this paper, we move in this direction and advance the state-of-the-art with respect to graph indexing methods for queries on beer distances and shortest beer paths in general beer graphs. In particular, we first showcase that an indexing scheme, to efficiently execute such queries, can be obtained by adapting the highway labeling, a recently introduced indexing method to accelerate the computation of classical shortest paths [20]. We propose a preprocessing algorithm, similar but more intuitive than that in [20] for unweighted graphs, to build a WHL index, i.e. a weighted highway labeling of a beer graph; we adapt the query algorithm of [20] to retrieve beer distances or shortest beer paths by only accessing said index. Differently from [20], once the WHL index is computed, our method is oblivious to the graph, in the sense it does need to access it to answer queries. We prove the correctness and analyze the time and space complexities of our methodology.

Then, through extensive experimentation on real networks, we empirically demonstrate its practical effectiveness and superiority, in terms of offered trade-off between preprocessing time, space overhead and query time, with respect to the state-of-the-art. In particular, our experiments show that our query algorithm answers to queries on beer distances, on average, within microseconds per vertex pair, even for very large networks. This is: (i) orders of magnitude faster than both the baseline and the adaptation of the K-SKY method; (ii) comparable to the B2ALL method. At the same time, on the one hand our preprocessing routine preprocesses even very large beer graphs very quickly (within an hour), which is faster than any known indexing scheme for fast beer distance/shortest beer path query answering; on the other hand, our index is compact in size (few hundreds of MBs even for very large networks), with a space occupancy that is up to orders of magnitude smaller than any known index computed for queries on beer distances and shortest beer paths.

Related Works.   Bacic et al. [4] have designed a preprocessing-based method that, for *outerplanar graphs*, computes in $\mathcal{O}(n)$ time an index of size $\mathcal{O}(n)$ that allows to find, for a pair of vertices, the beer distance d in $\mathcal{O}(\alpha(n))$ time, where $\alpha(n)$ is the inverse Ackermann function, and a corresponding shortest beer path in $\mathcal{O}(d)$ time. Similarly, Das et al. [11] introduced a data structure for *interval graphs*, occupying $2n \log n + \mathcal{O}(n) + O(|B| \log n)$ bits of space, which allows to compute the beer distance d in $\mathcal{O}(\log^{\epsilon}(n))$ time, for any constant $\epsilon > 0$ and a corresponding shortest beer path in $\mathcal{O}(\log^{\epsilon}(n) + d)$ time. The same authors also show that, if one restricts the input to be a *proper interval graphs*, worst case running time and space occupancy can be further slightly improved [11]. Finally, on a similar line of research it is worth mentioning: (i) the method of Hanaka et al. [23] which, by computing a suited graph decomposition, achieves optimal query time on *series-parallel graphs* and linear preprocessing time on graphs having *bounded-size triconnected components*; (ii) the work of Gudmundsson and Sha [22] which showed how a graph with bounded treewidth t can be preprocessed in $\mathcal{O}(t^3 n)$ time to guarantee the retrieval of the beer distance d in $\mathcal{O}(t^3 \alpha(n))$ time and of the corresponding shortest beer path in $\mathcal{O}(d t^3 \alpha(n))$ time.

2   Notation and Definitions

We are given a weighted graph $G = (V, E, \omega)$, with $n = |V|$ vertices, $m = |E|$ edges and a *weight function* $\omega : E \mapsto \mathbb{R}^+$ that assigns a positive, real value to each edge of G. A *path* $P = (s = v_1, v_2, \ldots, t = v_\eta)$ in G, connecting a pair of vertices $s, t \in V$, is a sequence of η vertices such that $\{v_i, v_{i+1}\} \in E$ for all $i \in [1, \eta - 1]$. The *weight* $\omega(P)$ of a path P is the sum of the weights of the edges in P. For non-simple paths, path weights include multiplicities of occurrences of a same edge. A *shortest path* P_{st}, for a pair $s, t \in V$, is a path having minimum weight among all those in G that connect s and t. The *distance* $d(s, t)$ between s and t is the weight of a shortest path P_{st}. A *subpath* $(v_i, v_{i+1}, \ldots, v_{j-1}, v_j)$ of a path $P = (v_1, \ldots, v_i, v_{i+1}, \ldots, v_{j-1}, v_j, \ldots, v_\eta)$ is denoted by $P[v_i, v_j]$, for each $0 \leq i < j \leq \eta$. Given a vertex $v \in V$, we denote by $N(v) = \{u \in V | \{v, u\} \in E\}$ the set of neighbors of v. Given a set of vertices $S \subseteq V$ of a graph $G = (V, E, \omega)$, $G[S]$ denotes the subgraph of G induced by S, i.e., $E(G[S]) = \{\{u, v\} \in E \mid u, v \in S\}$. A *beer graph* is a weighted graph $G = (V, E, \omega)$ having a set of *beer vertices* $B \subseteq V$. A *beer path* between two vertices s and t of a beer graph is a path that connects s and t and includes at least one vertex in B. A *shortest beer path* for two vertices s and t is any beer path having minimum weight. The weight of a shortest beer path is called the *beer distance*. In what follows, for the sake of simplicity, we describe our approaches by assuming, w.l.o.g., that the given graph G is
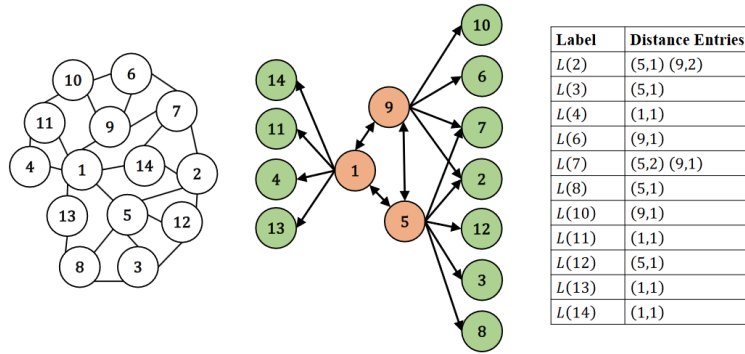
connected and undirected. All methods, described in this paper, can be used with digraphs by considering edge orientations and corresponding partitions of the neighbors of a vertex into outgoing and incoming neighbors.

2-Hop-Cover and Highway Labeling. The 2-hop-cover distance labeling is a graph indexing scheme, originally introduced in [7] and heuristically improved in [1]. It is based on the precomputation of an index, called 2-hop-cover distance labeling, that can be used to answer to queries on classical shortest paths and distances, as follows. Given a weighted graph $G = (V, E, \omega)$ and a subset $H \subseteq V$ of its vertices, called *hubs*, a 2-hop-cover distance labeling L of G is a collection of labels $L(v)$, one per vertex $v \in V$ such that: (i) a *label* $L(v)$ is a set of *entries* $(u, d(u, v))$ where $u \in H$; (ii) for each pair of vertices s, t, labels $L(s)$ and $L(t)$ store a set of entries that suffice to compute the distance $d(s, t)$ for any pair of vertices s, t of the graph, that is $d(s, t)$ can be obtained by a function $\delta_L : V \times V \mapsto \mathbb{R}^+$ that takes $L(s)$ and $L(t)$. Note that, it is known that computing a 2-hop-cover distance labeling of minimum size (i.e. number of label entries) is an NP-Hard problem [10].

The highway labeling is an hybrid indexing scheme that generalizes the 2-hop-cover distance labeling, introduced in [20] with the purpose of reducing the preprocessing time and space requirements of the approach of [1], at the price of a slight increase in the average query time. It is considered an hybrid indexing scheme in the sense that queries cannot be solved by accessing only the precomputed data structure and a search of the graph, even if bounded, must be employed to guarantee the correctness of the returned output. Given a graph $G = (V, E, \omega)$, and a subset of its vertices $R \subseteq V$, called *landmarks*, a *highway* $H(R, \delta_H)$ of G with landmarks $R \subseteq V$ is a pair (R, δ_H), where δ_H is a *distance decoding function*, i.e. a function $\delta_H : R \times R \mapsto \mathbb{R}^+$ such that, for any pair $r_i, r_j \in R$, we have $\delta_H(r_i, r_j) = d(r_i, r_j)$. In other words, a highway H is a data structure that stores the distance in G for any two landmarks in R in the form of a function δ_H (e.g. a look-up table). Given a vertex $r \in R \subseteq V$ and two vertices $s, t \in V \setminus R$, an *r-constrained shortest path* P_{st}^r from s to t in G is a path that passes through r (i.e. $r \in P_{st}^r$) and has minimum weight (called *r-constrained distance*) among all paths that connect s and t in G and include r.

Let $H(R, \delta_H)$ be a highway for a graph G with landmarks $R \subseteq V$. A *highway cover distance labeling* (or simply *highway labeling*) of G is a pair $(H(R, \delta_H), L)$ where H is a highway and L is a *labeling* i.e. a collection of labels $L(v)$, one per vertex $v \in V$ such that: (i) a *label* $L(v)$ consists of a set of *entries* in the form $(r_i, d(r_i, v))$ where $r_i \in R$; (ii) for any two vertices $s, t \in V \setminus R$, and for any $r \in R$, labels $L(s)$ and $L(t)$ store entries that suffice to compute the *r*-constrained distance $d^r(s, t)$. In other terms, any *r*-constrained distance between two vertices s and t can be found using only the labels of these two vertices. In more details, the latter condition, called *highway cover property* is guaranteed if, for any two vertices $s, t \in V \setminus R$ and for any $r \in R$, there exist $(r_i, d(r_i, s)) \in L(s)$ and $(r_j, d(r_j, t)) \in L(t)$ such that $r_i \in P_{rs}$, for some shortest path P_{rs} from s to t, and $r_j \in P_{rt}$, for some shortest path P_{rt} from r to t, where r_i and r_j may be equal to r. We say $H(R, \delta_H)$ covers G when the highway cover property holds. Moreover, if the label of a vertex v contains an entry $(r, d(r, v))$ for some $r \in R$ we say that vertex v is covered by landmark r in the (highway) labeling. Given any two vertices s and t, a highway labelling L can be used to find any *r*-constrained distance $d^r(s, t)$ by computing $d(r_i, s) + \delta_H(r_i, r_j) + d(r_j, t)$ where $r_i = r$ or $r_j = r$ and for $(r_i, d(r_i, s)) \in L(s)$ and $(r_j, d(r_j, t)) \in L(t)$. Moreover, an upper bound $d^T(s, t)$ on the shortest path distance from s to t is given by $d^T(s, t) = \min\{d(r_i, s) + \delta_H(r_i, r_j) + d(r_j, t) | (r_i, d(r_i, s)) \in L(s), (r_j, d(r_j, t)) \in L(t)\}$. Observe that such upper bound corresponds to the weight of a shortest path from s to t passing through

landmarks r_i and r_j which is exploited by the query routine [20] to compute the true distance $d(s,t)$ by running a distance-bounded bidirectional search on the subgraph $G[V \setminus R]$. In Figure 1 we show an example of highway labeling (H, L) of a graph G, taken from [20]. Consider the graph in the figure (left), the highway H has three landmarks, i.e. $R = \{1, 5, 9\}$. We have that $\langle 11, 1, 4 \rangle$ is a shortest path between vertices 11 and 4 constrained by landmark 1, i.e. it is 1-constrained shortest path between 11 and 4. In contrast, neither of the paths $\langle 11, 10, 9, 1, 4 \rangle$ and $\langle 11, 4 \rangle$ is a 1-constrained shortest path between 11 and 4. In Figure 1 (middle), the outgoing arrows from each landmark point to vertices in G that are covered by this landmark in the highway. The distance labelling in Figure 1 (right) satisfies the highway cover property because for any two vertices that are not landmarks and any landmark $r \in R = \{1, 5, 9\}$, we can find the r-constrained shortest path distance between these two vertices using their labels and the highway.

Label	Distance Entries
$L(2)$	(5,1) (9,2)
$L(3)$	(5,1)
$L(4)$	(1,1)
$L(6)$	(9,1)
$L(7)$	(5,2) (9,1)
$L(8)$	(5,1)
$L(10)$	(9,1)
$L(11)$	(1,1)
$L(12)$	(5,1)
$L(13)$	(1,1)
$L(14)$	(1,1)

Figure 1 Example of Highway Labeling of a graph [20].

3   Indexing Scheme for Beer Distance Queries

In this section, we describe a graph indexing scheme to support queries on beer distances and shortest beer paths. Specifically, we first introduce an algorithm that preprocesses a beer graph to compute a WHL index, a data structure that can be used through a dedicated query algorithm, also given here, to compute beer distances and shortest beer paths. W.l.o.g, and for the sake of simplicity, we describe our method to support the execution of beer distance (BD, for short) queries only. All given approaches can be easily extended to return a corresponding shortest beer paths by the strategies similar to those used for shortest paths, e.g. by storing predecessors (see, e.g., [10, 14]).

Our methodology is based on the work of Farhan et al. [20]. The key idea is, given a beer graph $G = (V, E, \omega)$ with beer vertices B, to compute a highway labeling $X = (H, L)$ that is; (i) weighted; (ii) considers, as set of landmarks R, the set of beer vertices B; (iii) covers G. In fact, it is easy to show that a highway labeling that satisfies the above three properties can be used to retrieve beer distances for any pair of vertices of the given beer graph by the following observations. Specifically, while using this structure alone does not suffice to compute shortest path distances (see Section 2), it is easy to see that, under the above assumptions, the beer distance, for any pair of vertices $s, t \in V$, is returned in $\mathcal{O}(|B|^2)$ time by the following query routine:

$$Q(s, t, X) = \min_{b_i, b_j \in B} \{d(b_i, s) + \delta_H(b_i, b_j) + d(b_j, t) | (b_i, d(b_i, s)) \in L(s), (b_j, d(b_j, t)) \in L(t)\} \quad (1)$$

Indeed, by the highway cover property, any r-constrained distance $d^r(s,t)$ for landmark $r \in R$ can be found by computing $d(r_i, s) + \delta_H(r_i, r_j) + d(r_j, t)$ where $r_i = r$ or $r_j = r$ and for $(r_i, d(r_i, s)) \in L(s)$ and $(r_j, d(r_j, t)) \in L(t)$. Thus, the weight of a shortest beer path that includes a beer vertex $b \in B$ can be found by determining $d(r_i, s) + \delta_H(r_i, r_j) + d(r_j, t)$ where $r_i = b$ or $r_j = b$ and for $(r_i, d(r_i, s)) \in L(s)$ and $(r_j, d(r_j, t)) \in L(t)$. It follows that the upper bound $d^T(s,t)$ on the shortest path distance from s to t, provided by the labeling as $d^T(s,t) = \min\{d(r_i, s) + \delta_H(r_i, r_j) + d(r_j, t) | (r_i, d(r_i, s)) \in L(s), (r_j, d(r_j, t)) \in L(t)\}$, is the minimum of the weights of all r-constrained shortest paths, for each $r \in B$, which corresponds to the minimum of the weights of the shortest paths that include a beer vertex $b \in B$ for all beer vertices of B, which is the beer distance. We call a weighted highway labeling built on beer vertices, that satisfies the above equivalency, a WHL index.

Now, Farhan et al. [20] have shown that a highway labeling can be found in linear time for *unweighted* graphs by running $|R|$ modified breadth-first search (BFS) visits, one per landmark, and by incrementally constructing labels that satisfy the following. Whenever a vertex v is extracted from the queue of the BFS, rooted at landmark $r_i \in R$, if we call $P_{r_i v}$ the shortest path from r_i to v traversed by the search, and $d(r_i, v)$ its weight, then: (i) if vertex $v \in V \setminus R$, an entry $(r_i, d(r_i, v))$, for some $r_i \in R$, is added to $L(v)$ if and only if there does not exist any other landmark in $P_{r_i v}$, i.e. $P_{r_i v} \cap R = \{r_i\}$; (ii) if v is a landmark $r \in R \setminus \{r_i\}$, then $d(r_i, v)$ is added to the highway, i.e. to $\delta_H(r_i, v)$ to build the distance decoding function (e.g. in the form of a table). Unfortunately, no strategy for efficiently building a weighted highway labeling covering a weighted graph is given in [20]. In subsequent works, namely [18, 19], the authors claim that the construction of [20] can be adapted to weighted graphs by replacing the modified BFS with Dijkstra's algorithm. Differently from an unweighted one, in fact, both the label entries of a weighted highway labeling and the associated distance decoding function must store weights of shortest paths (i.e. sums of weights of edges), rather than path lengths (i.e. number of edges). However, no details on the extension are given in [18, 19] and adapting the approach from unweighted to weighted inputs seems to be all but straightforward. More specifically, in the unweighted version, each visit, rooted at a given vertex r_i, employs two queues, called $\mathcal{Q}_{label}$ and $\mathcal{Q}_{prune}$. The former is used for paths to vertices that do not traverse landmarks other than r_i, while the latter for paths that traverse at least one landmarks $r_j, j \in R \setminus \{r_i\}$. When an element is extracted from $\mathcal{Q}_{label}$ with some priority δ, all neighbors of the terminal vertex of the path are enqueued, with priority $\delta + 1$ into $\mathcal{Q}_{label}$ (if they are not landmarks) or into $\mathcal{Q}_{prune}$ (otherwise). In the latter case, elements in $\mathcal{Q}_{prune}$ having priority δ are all extracted, and neighboring vertices are enqueued in the same queue with priority $\delta + 1$. If the graph is unweighted, this mechanism guarantees that a vertex is never added to both queues. However, it is easy to see that this would not be true if the treated input graph is weighted, with simultaneous occurrences of a same vertex in both queues leading to unnecessary queue operations that have to be managed by the algorithm. Therefore, in what follows, we introduce a novel algorithm, called BUILDWHL, to efficiently compute a WHL index, i.e. a weighted highway labeling with beer vertices as landmarks that covers a beer graph, using only one priority-queue.

Algorithm BUILDWHL works as follows. Starting from each beer vertex in B as root, we run a modified version of Dijkstra's algorithm that searches the graph by relying on a single one-level priority queue (e.g. a min-heap). For each root $r \in B$, the algorithm assigns, to each vertex $v \in V$: a flag $flag[v]$ initially `false`; a tentative distance $d[v]$ initially equal to some infinity default value. Then, the visit starts by enqueueing the root with zero priority and, whenever a vertex v is dequeued, say with associated priority δ, two cases can occur. If v is a beer vertex $v \neq r$, first its flag is set to `true`, to trace the fact that a path from

the root to such vertex traverses a beer vertex. Then, the associated value δ, corresponding
to the weight $d(r, v)$ of a shortest path from r to v, is stored in the highway H (i.e. the
distance decoding function is built). Moreover, regardless of whether v is a beer vertex or
not, if the value of its flag is `true`, then value δ, which equals distance $d(r, v)$, is discarded
(since the shortest path inducing such distance traverses a beer vertex). Viceversa, if the
flag of v is `false`, we have that value δ is the distance from r to v induced by a shortest
path, say P_{rv}, that does not traverse any beer vertex, thus entry (r, δ) is added to $L(v)$.
Finally, a distance relaxation operation is performed on the neighbors of v. In details, for
each $w \in N(v)$, we check if $d[w] > \delta + \omega(v, w)$, i.e. if the weight of the path from r to v plus
the weight of edge (v, w) is less than the tentative distance $d[w]$. In the affirmative case, it
follows that the path P_{rv} to v combined with edge (v, w) has weight that is smaller than any
previously discovered path to w. Hence, we update $d[w] = \delta + \omega(v, w)$ and either enqueue w
with priority $d[w]$ or decrease its current priority to $d[w]$. Contextually, we update the flag
of w to that of v, to keep trace of traversal of beer vertices. Viceversa, the path from r to
w through v is either not a shortest path to w or a shortest path of equal weight, hence it
is not considered. The pseudo-code of BUILDWHL is given in Algorithm 1. We next state

---

■ **Algorithm 1** Algorithm BUILDWHL.

Input: A beer graph $G = (V, E, \omega)$ with beer vertices $B \subseteq V$.
Output: A WHL index (H, L).

1 foreach $r \in R$ **do**
2 | **foreach** $v \in V$ **do**
3 | | $d[v] \leftarrow \infty$;
4 | | $flag[v] \leftarrow$ `false`;
5 | $PQ \leftarrow \emptyset$;   // PQ is a priority queue, e.g. min-heap
6 | Enqueue vertex r into PQ with priority 0;
7 | **while** $PQ \neq \emptyset$ **do**
8 | | Dequeue from PQ vertex v having minimum priority δ;   // Here $d[v] = \delta$
9 | | **if** $v \in B \setminus \{r\}$ **then**
10 | | | $flag[v] \leftarrow$ `true`;
11 | | | $\delta_H(r, v) \leftarrow \delta$;   // Store δ into entry $\delta_H(r, v)$ of the highway H
12 | | **if** $flag[v]$ *is false* **then**
13 | | | Add (r, δ) to $L(v)$;
14 | | **foreach** $w \in N(v)$ **do**
15 | | | **if** $d[w] > \delta + \omega(v, w)$ **then**
16 | | | | $flag[w] \leftarrow flag[v]$;
17 | | | | $d[w] \leftarrow \delta + \omega(v, w)$;
18 | | | | **if** $d[w] = \infty$ **then** Enqueue vertex w into PQ with priority $d[w]$;
19 | | | | **else** Decrease priority of vertex w in PQ to $d[w]$;

---

the correctness and the running time of procedure BUILDWHL. In particular, we are able
to prove that algorithm BUILDWHL computes a weighted highway labeling that covers the
input graph with landmarks B. This implies that the query routine of Eq. 1 on said labeling
returns the beer distance for every pair of vertices.

▶ **Lemma 1.** *Algorithm 1 adds entry (r, δ) to label $L(v)$ of a vertex $v \in V \setminus B$ if and only if
$r \in B$ is the only beer vertex in the shortest path P_{rv} inducing $\delta = d(r, v)$.*

Proof. Suppose that $P_{rv} \cap B \neq \{r\}$, i.e. there exists at least a beer vertex $r' \in B \setminus \{r\}$ in
the shortest path P_{rv} between r and v that causes v to be enqueued (i.e. path having weight
$\delta = d(r, v)$). Since r' lies in P_{rv}, it must be extracted from the priority queue before v, and

Line 10 in Algorithm 1 sets the flag of r' to `true`. By the subpaths optimality property of shortest paths, it follows that flag of r' is propagated in each distance relaxation operation of any vertex $v' \in P_{rv}[r', v]$ (cf. Lines 15-16). Therefore, when v is extracted from the priority queue, its flag is equal to `true`, and Line 13 is not executed. On the other hand, if Algorithm 1 inserts (r, δ) in $L(v)$ in Line 13, then we have $P_{rv} \cap B = \{r\}$. In fact, Line 13 is executed only if the flag of vertex v is set to `false`, which happens if and only if each distance relaxation applied on the vertices $v' \in P_{rv}$ was not induced by a path traversing a landmark $r' \neq r$. ◀

By Lemma 1 we can derive a corollary similar to that given in [20] for unweighted graphs.

▶ **Corollary 2.** *Let $r \in B$ be a beer vertex and let $v \in V \setminus B$ be a non-beer vertex. Let (H, L) be a labeling constructed by Algorithm 1. If $(r, d(r, v)) \notin L(v)$, then there must exist a beer vertex r_j such that $(r_j, d(r_j, v)) \in L(v)$, and $d(r, v) = d(r_j, v) + \delta_H(r_j, r)$.*

Finally, we can prove that the labeling computed by BuildWhl covers the given graph.

▶ **Theorem 3.** *The highway labeling (H, L) constructed by Algorithm 1 on a beer graph $G = (V, E, \omega)$ satisfies the highway cover property for G.*

Proof. We need to show that, for any two vertices $s, t \in V \setminus B$ and any $r \in B$, there exist $(r_i, d(r_i, s)) \in L(s)$ and $(r_j, d(r_j, t)) \in L(t)$ such that $r_i \in P_{rs}$ and $r_j \in P_{rt}$. To that aim, we can apply Corollary 2 to the following four cases: (i) r covers both s and t; (ii) r covers s but not t; (iii) t is covered by r, while s is not; (iv) neither s nor t is covered by r. For the sake of completeness, we also give it here. With a slight abuse of notation, in what follows we use $r' \in L(v)$ to denote that landmark r' covers a vertex v, i.e. that there exists an entry $(r', d(r', v))$ in $L(v)$. In Case (i), we have $r \in L(s)$ and $r \in L(t)$, thus $r = r_i = r_j$. Case (ii): $r_i = r$, while Corollary 2 ensures the existence of another landmark r_j such that r_j is in the shortest path between t and r and $r_j \in L(t)$. Case (iii) is treated similarly to the previous case. Finally, again by Corollary 2 applied to Case (iv), we know that there exist two landmarks r_i, r_j such that r_i (r_j, respectively) is in the shortest path between s (t, respectively) and r, and $(r_i, d(r_i, s)) \in L(s)$ and $(r_j, d(r_j, t)) \in L(t)$. ◀

▶ **Theorem 4.** *Algorithm 1 runs in $\mathcal{O}(|B|(m + n \log n))$ time.*

Proof. Observe that, for each landmark $r \in B$, the algorithm performs a call to a Dijkstra-like algorithm for each landmark. During the execution of such routine, rooted at a landmark r, the algorithm update the flags of the vertices. The initialization of the flags takes $\mathcal{O}(n)$ time. Then, when a vertex v is extracted from the min-heap data structure PQ, if $v \in B \setminus \{r\}$, the flag of v is set to `true` and the algorithm inserts distance δ into highway H in constant time. Otherwise, the algorithm adds entry (r, δ) to label $L(v)$, which again can be done in constant time. Finally, the algorithm checks if value $d[w]$ can be decreased for neighbors w of v. In the affirmative case, the flag of w is set to the flag of v and a queue operation is performed, in $\mathcal{O}(\log n)$ time. Overall, the maintenance of the flags requires $\mathcal{O}(m + n)$ constant time operations while queue operations account for $\mathcal{O}(n \log n)$ time. Furthermore, the algorithm performs $|B| - 1$ insertions into the highway H and at most $n - |B|$ insertions into the labels of the vertices, so overall $\mathcal{O}(n)$ insertions operations, each taking $\mathcal{O}(1)$ time. Therefore, the time complexity per landmark $r \in B$ is $\mathcal{O}(m + n \log n)$ and the claim follows. ◀

4   Experimental Evaluation

In this section, we present the results of an experimental evaluation we conducted to assess the effectiveness of our new graph indexing method for BD queries.

Setup and Executed Tests. We implemented: (i) algorithm BUILDBM that precomputes and stores, in a corresponding B2ALL matrix, all distances from beer vertices to other vertices of the graph; (ii) the corresponding query algorithm, called QUERYBM, that solves a BD query for a pair of vertices by determining the minimum of the sums of the distances between all beer vertices and the two queried vertices; (iii) our method BUILDWHL to compute a WHL index; (iv) the corresponding query algorithm, denoted by QUERYWHL, that computes the BD for a pair of vertices by accessing the index as per Eq. 1; (v) the baseline method, denoted by BASELINE, which executes Dijkstra's algorithm twice to compute the shortest path trees rooted at the two queried vertices s, t and selects the beer vertex providing the minimum sum of distances from s and to t. All our code is written in C++, compiled with GCC 10.5 with opt. level $O3$. All tests have been executed on a workstation equipped with an Intel Xeon© CPU E5-2643, clocked at 3.40 GHz, and 96 GB of RAM, running Ubuntu Linux.

Input Instances. As inputs to our experiments we considered real-world road graphs taken from publicly available repositories [26]. Details on used inputs, including number of vertices $|V|$ and edges $|E|$, average vertex degree, and size of the graph file $|G|$ are reported in Table 1. Graphs are sorted from top to bottom by $|V| + |E|$. Concerning the number of

Table 1 Overview of Input Graphs.

| Graph	$|V|$	$|E|$	Avg. Deg.	$|G|$
LUX	30 647	37 773	2.46	1.1 MB
NY	264 346	365 050	2.76	13 MB
BAY	321 270	397 415	2.47	14 MB
COL	435 666	521 200	2.39	19 MB
DNK	469 110	545 019	2.32	18 MB
FLA	1 070,376	1 343 951	2.51	48 MB
NW	1 207 945	1 410 387	2.33	52 MB
NE	1 524 453	1 934 010	2.53	71 MB
CAL	1 890 815	2 315 222	2.44	87 MB
ITA	2 077 709	2 589 431	2.49	91 MB
DEU	4 047 577	4 907 447	2.42	178 MB
USA	23 947 347	28 854 312	2.40	1.2 GB

beer vertices b, to study the scalability properties of the proposed approach, as suggested in [28], we measure and report performance indicators for doubling values of parameter b that are relevant to the domain, i.e. for $b \in \{25, 50, 100, 200, 400\}$. Beer vertices are placed in each graph by a strategy called *distance-δ bounded dominating set*, a policy that is often considered in network design problems, and adopted in real-world scenarios, to identify a subset of important vertices that have to be traversed by paths between other vertices (e.g. routers responsible for specific messages) [21]. For each input graph and value of b, we: (i) run BUILDWHL to construct a WHL index; (ii) execute BUILDBM to precompute the B2ALL matrix. We measure the running times of both BUILDWHL and BUILDBM (denoted by PT_{WHL} and PT_{B2ALL}, respectively), and the space occupancy, in MB, of the WHL index (which includes both the distance deconding function and the label entries) and the B2ALL matrix (denoted by IS_{WHL} and IS_{B2ALL}, respectively). Note that, in all our experiments, we represent distances and landmarks/beer vertices as 32-bit integers. Moreover, we execute QUERYWHL and QUERYBM to solve 10^5 BD queries for randomly selected vertex pairs and measure their average execution time (denoted by QT_{WHL} and QT_{B2ALL}, respectively). Finally, we run algorithm BASELINE for a subset of 10^4 of the aforementioned vertex pairs, and measure the average execution time. Such reduction is necessary due to the moderately large running time per query of BASELINE.

Analysis. The results of our experimentation are summarized in Tables 2–4. For both performance indicators indexing time and space occupancy, in Tables 2–4 we give the ratio of the value achieved by B2ALL and that achieved by WHL. In terms of indexing/preprocessing time, our data show that algorithms BuildWhl and BuildBM are comparable and both can be considered reasonably practical since both average execution times range from less than half a second in the smallest instance (i.e. LUX with $b = 25$) to around one hour in the largest one (i.e. USA with $b = 400$). However, we notice that BuildWhl is always

Table 2 Results of the experimentation with $b = 25$ beer vertices.

Graph	Preprocessing			Space Occupancy			Query (s)		
	PT_{WHL} (s)	PT_{B2ALL} (s)	PT_{B2ALL}/PT_{WHL}	IS_{WHL} (MB)	IS_{B2ALL} (MB)	IS_{B2ALL}/IS_{WHL}	QT_{WHL}	QT_{B2ALL}	BaseLine
LUX	0.12	0.13	1.09	0.13	2.92	22.46	$1.2 \cdot 10^{-7}$	$1.1 \cdot 10^{-7}$	0.01
NY	1.65	1.78	1.08	3.80	25.21	6.63	$4.5 \cdot 10^{-7}$	$3.2 \cdot 10^{-7}$	0.10
BAY	1.94	2.40	1.24	2.59	30.64	11.83	$4.3 \cdot 10^{-7}$	$3.7 \cdot 10^{-7}$	0.13
COL	2.39	2.96	1.24	1.77	41.55	23.47	$3.5 \cdot 10^{-7}$	$3.3 \cdot 10^{-7}$	0.18
DNK	2.56	3.30	1.29	5.36	44.74	8.35	$4.1 \cdot 10^{-7}$	$3.8 \cdot 10^{-7}$	0.17
FLA	6.95	7.40	1.07	12.36	102.08	8.26	$4.4 \cdot 10^{-7}$	$3.4 \cdot 10^{-7}$	0.43
NW	7.25	8.43	1.16	8.30	115.20	13.88	$4.2 \cdot 10^{-7}$	$3.5 \cdot 10^{-7}$	0.51
NE	11.04	12.41	1.12	21.63	145.38	6.72	$4.6 \cdot 10^{-7}$	$3.9 \cdot 10^{-7}$	0.70
CAL	13.63	15.43	1.13	15.44	180.32	11.68	$5.2 \cdot 10^{-7}$	$4.3 \cdot 10^{-7}$	0.87
ITA	10.56	12.62	1.19	7.93	198.15	24.99	$4.9 \cdot 10^{-7}$	$5.1 \cdot 10^{-7}$	0.72
DEU	26.81	31.80	1.19	48.44	386.01	7.97	$6.9 \cdot 10^{-7}$	$5.2 \cdot 10^{-7}$	1.63
USA	246.94	280.49	1.14	91.35	2283.80	25.00	$9.7 \cdot 10^{-7}$	$6.7 \cdot 10^{-7}$	10.90

faster than BuildBM, by factors that span in the orders of tens of percentage points (see PT_{B2ALL}/PT_{WHL} column in Tables 2–4). This is most likely due to the fact that BuildWhl stores label entries only when the flag of a vertex is false, i.e. when the root r is the only beer vertex lying on the path found by the visit. Furthermore, to characterize of the scalability properties of BuildWhl, in Fig 2 (left) and 3 (left) we plot its running time as a function of b for all inputs. Our data suggest a linear trend of the indexing time with respect to to b, which matches our analysis of Theorem 4. On top of that, Figures 2 (middle) and 3 (middle) suggest that BuildWhl scales better with respect to b than BuildBM, in terms of running time, with the ratio between the execution times of the two algorithms increasing with b.
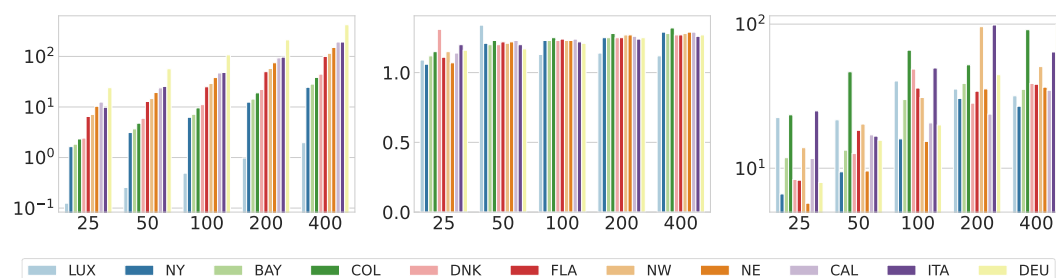
Figure 2 Running time (in seconds) of BuildWhl (left); ratio of the running time of BuildWhl to that of BuildBM (middle); ratio of the space occupancy of the B2ALL matrix index to that of the WHL (right), for all graphs, except USA, as a function of b (x-axis).

Concerning the sizes of the WHL index and the B2ALL matrix, instead, we observe that our new scheme significantly outperforms method B2ALL. In fact, BuildWhl computes very compact WHL indices, even for large graphs, with an average space occupancy that is up to orders of magnitude smaller than that of B2ALL matrices, which contain precisely one distance value per beer vertex and for all vertices of the graph (cf. column "Space Occupancy" of Tables 2-4). In details, we observe that, even in the largest considered graph, i.e. USA (see

Table 3 Results of the experimentation with $b \in \{50, 100\}$ beer vertices.

b	Graph	Preprocessing			Space Occupancy			Query (s)		
		PT_{WHL} (s)	PT_{B2ALL} (s)	PT_{B2ALL}/PT_{WHL}	IS_{WHL} (MB)	IS_{B2ALL} (MB)	IS_{B2ALL}/IS_{WHL}	QT_{WHL}	QT_{B2ALL}	BASELINE
50	LUX	0.25	0.34	1.34	0.27	5.85	21.67	$1.5 \cdot 10^{-7}$	$1.4 \cdot 10^{-7}$	0.01
	NY	3.11	3.77	1.21	5.34	50.42	9.44	$3.5 \cdot 10^{-7}$	$3.4 \cdot 10^{-7}$	0.11
	BAY	3.68	4.41	1.20	4.59	61.28	13.35	$3.3 \cdot 10^{-7}$	$2.2 \cdot 10^{-7}$	0.13
	COL	4.77	5.89	1.23	1.78	83.10	46.69	$4.9 \cdot 10^{-7}$	$2.2 \cdot 10^{-7}$	0.17
	DNK	5.97	7.14	1.20	7.08	89.48	12.64	$3.8 \cdot 10^{-7}$	$2.4 \cdot 10^{-7}$	0.21
	FLA	12.82	15.59	1.22	11.14	204.16	18.33	$4.2 \cdot 10^{-7}$	$2.5 \cdot 10^{-7}$	0.46
	NW	14.67	17.79	1.21	11.37	230.40	20.26	$4.3 \cdot 10^{-7}$	$2.7 \cdot 10^{-7}$	0.55
	NE	19.29	23.46	1.22	30.37	290.77	9.57	$4.4 \cdot 10^{-7}$	$2.6 \cdot 10^{-7}$	0.71
	CAL	23.86	29.27	1.23	21.17	360.64	17.04	$4.5 \cdot 10^{-7}$	$2.7 \cdot 10^{-7}$	0.88
	ITA	25.47	30.61	1.20	23.73	396.29	16.70	$4.6 \cdot 10^{-7}$	$2.9 \cdot 10^{-7}$	0.96
	DEU	57.03	66.62	1.17	49.37	772.01	15.64	$6.0 \cdot 10^{-7}$	$3.5 \cdot 10^{-7}$	2.05
	USA	412.75	469.48	1.13	161.71	4 567.59	28.24	$5.7 \cdot 10^{-7}$	$3.3 \cdot 10^{-7}$	13.91
100	LUX	0.49	0.55	1.13	0.29	11.69	40.31	$1.4 \cdot 10^{-7}$	$1.7 \cdot 10^{-7}$	0.01
	NY	6.25	7.68	1.23	6.31	100.84	15.98	$3.7 \cdot 10^{-7}$	$2.9 \cdot 10^{-7}$	0.11
	BAY	7.16	8.83	1.23	4.09	122.55	29.96	$3.4 \cdot 10^{-7}$	$2.9 \cdot 10^{-7}$	0.13
	COL	9.60	12.03	1.25	2.52	166.19	65.95	$4.1 \cdot 10^{-7}$	$3.0 \cdot 10^{-7}$	0.18
	DNK	11.11	13.71	1.23	3.67	178.95	48.76	$3.8 \cdot 10^{-7}$	$3.0 \cdot 10^{-7}$	0.20
	FLA	25.08	31.12	1.24	11.34	408.32	36.01	$4.3 \cdot 10^{-7}$	$3.4 \cdot 10^{-7}$	0.46
	NW	29.08	35.90	1.23	14.89	460.79	30.95	$4.4 \cdot 10^{-7}$	$3.7 \cdot 10^{-7}$	0.55
	NE	38.17	47.12	1.23	37.89	581.53	15.35	$4.7 \cdot 10^{-7}$	$3.6 \cdot 10^{-7}$	0.71
	CAL	47.01	58.20	1.24	34.96	721.29	20.63	$4.9 \cdot 10^{-7}$	$3.6 \cdot 10^{-7}$	0.88
	ITA	48.14	58.72	1.22	15.98	792.58	49.60	$4.8 \cdot 10^{-7}$	$3.6 \cdot 10^{-7}$	0.92
	DEU	107.30	130.06	1.21	77.34	1 544.03	19.96	$5.2 \cdot 10^{-7}$	$3.8 \cdot 10^{-7}$	1.98
	USA	836.25	981.31	1.17	341.28	9 135.19	26.76	$6.5 \cdot 10^{-7}$	$4.2 \cdot 10^{-7}$	13.81

Table 4 Results of the experimentation with $b \in \{200, 400\}$ beer vertices.

| b	Graph	Preprocessing			Space Occupancy			Query (s)		
		PT_{WHL} (s)	PT_{B2ALL} (s)	PT_{B2ALL}/PT_{WHL}	IS_{WHL} (MB)	IS_{B2ALL} (MB)	IS_{B2ALL}/IS_{WHL}	QT_{WHL}	QT_{B2ALL}	BASELINE
200	LUX	0.97	1.11	1.14	0.66	23.38	35.42	$1.7 \cdot 10^{-7}$	$2.9 \cdot 10^{-7}$	0.01
	NY	12.46	15.56	1.25	6.61	201.68	30.51	$4.4 \cdot 10^{-7}$	$4.0 \cdot 10^{-7}$	0.11
	BAY	14.33	17.95	1.25	6.32	245.11	38.78	$3.7 \cdot 10^{-7}$	$4.1 \cdot 10^{-7}$	0.13
	COL	19.00	24.28	1.28	6.37	332.39	52.18	$4.1 \cdot 10^{-7}$	$4.2 \cdot 10^{-7}$	0.17
	DNK	22.15	27.77	1.25	12.65	357.90	28.29	$4.2 \cdot 10^{-7}$	$4.3 \cdot 10^{-7}$	0.20
	FLA	49.94	62.58	1.25	23.84	816.63	34.25	$4.9 \cdot 10^{-7}$	$4.5 \cdot 10^{-7}$	0.46
	NW	57.63	72.91	1.27	9.60	921.59	96.00	$4.7 \cdot 10^{-7}$	$4.8 \cdot 10^{-7}$	0.55
	NE	74.85	94.99	1.27	32.77	1 163.07	35.49	$4.9 \cdot 10^{-7}$	$4.6 \cdot 10^{-7}$	0.71
	CAL	93.64	117.93	1.26	60.73	1 442.58	23.75	$5.6 \cdot 10^{-7}$	$4.6 \cdot 10^{-7}$	0.88
	ITA	96.81	120.30	1.24	16.08	1 585.17	98.58	$4.9 \cdot 10^{-7}$	$4.6 \cdot 10^{-7}$	0.94
	DEU	212.27	264.37	1.25	69.28	3 088.06	44.75	$5.4 \cdot 10^{-7}$	$4.8 \cdot 10^{-7}$	1.98
	USA	1 658.4	1 967.24	1.18	529.48	18 270.38	34.50	$6.8 \cdot 10^{-7}$	$5.3 \cdot 10^{-7}$	13.88
400	LUX	1.96	2.19	1.12	1.47	46.76	31.81	$2.6 \cdot 10^{-7}$	$4.5 \cdot 10^{-7}$	0.01
	NY	24.54	31.71	1.29	14.97	403.36	26.94	$6.5 \cdot 10^{-7}$	$5.7 \cdot 10^{-7}$	0.11
	BAY	28.33	36.30	1.28	13.90	490.22	35.27	$5.4 \cdot 10^{-7}$	$5.7 \cdot 10^{-7}$	0.13
	COL	38.41	50.56	1.32	7.27	664.77	91.44	$4.5 \cdot 10^{-7}$	$5.8 \cdot 10^{-7}$	0.17
	DNK	44.66	56.75	1.27	18.47	715.81	38.76	$4.6 \cdot 10^{-7}$	$5.8 \cdot 10^{-7}$	0.21
	FLA	99.92	127.31	1.27	42.71	1 633.26	38.24	$6.0 \cdot 10^{-7}$	$5.9 \cdot 10^{-7}$	0.46
	NW	115.23	147.90	1.28	36.33	1 843.18	50.73	$5.3 \cdot 10^{-7}$	$6.0 \cdot 10^{-7}$	0.53
	NE	150.59	194.48	1.29	63.70	2 326.13	36.52	$6.1 \cdot 10^{-7}$	$6.7 \cdot 10^{-7}$	0.73
	CAL	192.78	248.61	1.29	83.07	2 885.15	34.73	$6.3 \cdot 10^{-7}$	$6.4 \cdot 10^{-7}$	0.91
	ITA	192.47	242.78	1.26	49.50	3 170.33	64.05	$5.2 \cdot 10^{-7}$	$6.1 \cdot 10^{-7}$	0.92
	DEU	423.58	539.66	1.27	62.38	6 176.11	99.01	$5.7 \cdot 10^{-7}$	$6.6 \cdot 10^{-7}$	2.02
	USA	3 178.94	3 962.77	1.24	692.82	36 540.75	52.74	$7.5 \cdot 10^{-7}$	$6.9 \cdot 10^{-7}$	13.95

Table 4), the space occupancy of the WHL index, with $b = 400$, is less than 700 MB, which can be considered a negligible amount of memory for modern commodity hardware. On the contrary, the space occupancy of the B2ALL matrix is more than 36 GB of data, roughly 50 times more than the WHL index. Moreover, as shown in Figures 2 (right) and 3 (right) the gap between the two occupancies tends to increase with b, which suggests that the WHL schemes scales better in terms of space occupancy with respect to b. Even more remarkably, the space to store WHL index is often lower than the space occupancy of the input graph (cf. column $|G|$ of Table 1 and column IS_{WHL} of Tables 2–4) and always lower for large graphs and
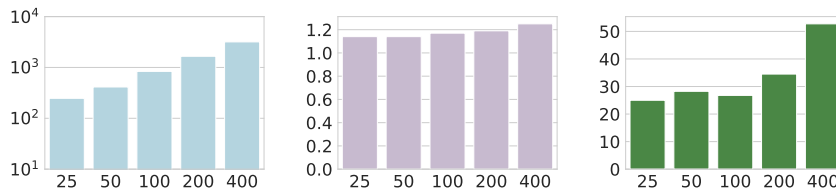
Figure 3 Running time of BuildWhl (left, in seconds); ratio of the running time of BuildWhl to that of BuildBM (middle); ratio of the space occupied by the b2all matrix index to that occupied by the whl (right) for graph usa, as a function of b (x-axis).

values of b. Note that, differently from previously indexing strategies based on the highway labeling [18–20], our scheme is oblivious of the graph topology, that is, once the whl index is built, algorithm QueryWhl does not need to access the underlying graph to answer BD queries. This is an interesting feature with respect to both space efficiency (the whl index can be seen as a compressed representation of all pairs beer distances) and usage of the index in a distributed environment (distances can be computed by only accessing the labels of the queried vertices and the distance decoding function [8]).

For the sake of completeness, in Figures 4 and 5, we provide measures of number of label entries stored by algorithm BuildWhl in the labeling part of the computed whl index. Despite, in the worst case, a whl stores an entry per beer vertex in each label, here we observe that, in practice, the amount of entries is generally around two orders of magnitude lower than such a worst case.
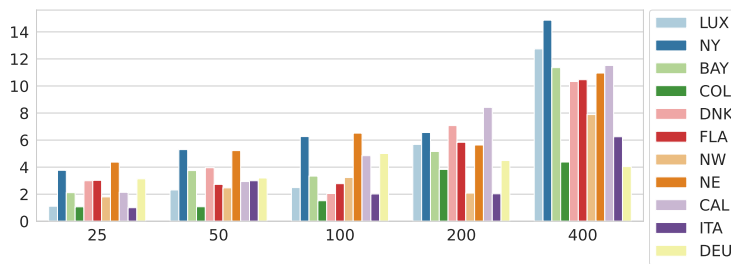
Figure 4 Average number of label entries per vertex (y-axis) stored by algorithm BuildWhl into the labeling part of the whl index, for each of the considered graphs, except usa, as a function of the number b of beer vertices (x-axis).
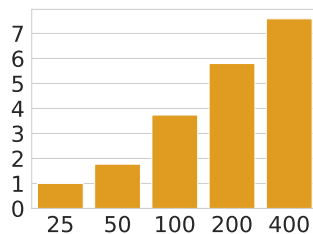
Figure 5 Average number of label entries per vertex (y-axis) stored by algorithm BuildWhl into the labeling part of the whl index, for graph usa, as a function of the number b of beer vertices (x-axis)

Such behavior is reflected into the low space occupancy requirements discussed above and shows how our preprocessing algorithm, and the highway labeling properties, allow to compute compact representations of shortest beer paths. In this regard, we leave the problem of evaluating whether such effectiveness is influenced by the centrality of beer vertices open for future investigation.

Concerning query times, our experiments highlight that both QueryWhl and QueryBM are extremely fast at answering BD queries (few hundreds of nanoseconds even for large graphs and values of b, cf. column "Query" of Tables 2-4). As expected, QueryWhl is slightly slower than QueryBM, since the query algorithm must consider distances between beer vertices (cf. second term of Eq. 1). Nonetheless, both strategies are competitive under such measure and suited for the requirements of modern data-intensive applications. Our data also confirm the poor performance of BaseLine in this sense: the algorithm does not scale well with both the graph size and b. Indeed, BaseLine's average running time ranges from around 0.01 seconds, on the smallest considered graph and value of b, to 14 seconds on the largest graph and value of b (see Tables 2-4, cf. column BaseLine). In order to give further insights on the effectiveness of employing a whl index for BD queries, in what follows, we provide a cumulative analysis that compares BaseLine, the best known approach without indexing, and our method. In particular, we design and run an experiment where, on the one hand, we execute BaseLine to solve 10^5 queries for randomly selected vertex pairs and measure the total running time. We call this quantity CMT_{BSL} the cumulative running time of BaseLine. On the other hand, we execute BuildWhl to build a whl index and run QueryWhl to answer the same set of queries. We measure the preprocessing time and sum it to the time for executing QueryWhl for all queries. We call this quantity CMT_{WHL} the cumulative running time of whl. The purpose of this experiment is to assess whether the time taken by BuildWhl to construct the index is amortized by the time saved by running QueryWhl to answer BD queries instead of BaseLine, hence to determine the most effective solution in terms of amortized running time.

In Table 5 we present the results of the cumulative experiment for a subset of the considered inputs and values of b. Results for other graphs and values of b are similar and hence omitted. Our data show that the cumulative running time of whl is almost three

■ **Table 5** Cumulative running time of whl and BaseLine to answer to 10^5 BD queries with $b = 200$.

Graph	CMT_{WHL} (s)	CMT_{BSL} (s)	CMT_{BSL}/ CMT_{WHL}
BAY	$1.4 \cdot 10^1$	$1.3 \cdot 10^4$	$9.2 \cdot 10^2$
DNK	$2.2 \cdot 10^1$	$2.0 \cdot 10^4$	$9.3 \cdot 10^2$
CAL	$9.3 \cdot 10^1$	$8.8 \cdot 10^4$	$9.4 \cdot 10^2$
ITA	$9.6 \cdot 10^1$	$9.4 \cdot 10^4$	$9.7 \cdot 10^2$
DEU	$2.1 \cdot 10^2$	$1.9 \cdot 10^5$	$9.3 \cdot 10^2$

orders of magnitude lower than that of BaseLine on any combination of graph and number of beer vertices. This is a remarkable result, especially if one considers that the time for precomputing the highway labeling via BuildWhl can be as high as around two hours on usa, and represents a strong experimental evidence of the fact that whl is the most effective framework in practical contexts to answer BD queries, even when large graphs and volumes of queries have to be managed.

Comparison against Frameworks for Generalized Shortest Paths.   In this section we complete the experimental evaluation of our framework for BD queries by comparing it with the indexing scheme k-sky of [24], which is considered the best performing method to answer

queries on generalized shortest paths, where the aim is computing minimum-weighted paths that traverse at least one vertex for each of k vertex *categories*. Answering to queries on beer distances or shortest beer paths is equivalent to the special case of generalized shortest paths where $k = 1$. For the purpose, method K-SKY preprocesses the graph to store both a 2-hop cover distance labeling and, for each vertex, a so-called *keyword-skyline*. Upon query, the former is exploited to compute quickly shortest paths while the latter, which is a collection of sets of vertices, one per category, is used to reduce the computation of distances toward vertices of each sought category (see [24] for more details). In order to compare K-SKY and WHL, we implemented K-SKY by considering a single POI category (set B) and executed it in the same settings considered for WHL in the previous section. For K-SKY we measure: (i) the running time to build the 2-hop cover distance labeling and to populate the keyword skyline ($PT_{\text{K-SKY}}$); (ii) the space occupancy to store such two data structures ($IS_{\text{K-SKY}}$); (iii) the average execution time to answer 10^5 BD queries for randomly generated vertex pairs ($QT_{\text{K-SKY}}$). For WHL, we measured the same performance indicators discussed in the previous part of the experimentation. Clearly, QueryWhl is run with the same set of queries.

An excerpt of the results of the above experiment is given in Table 6. Data for other graphs and values of b are omitted due to space limitations. The main conclusion that can be drawn from our experimental data is that our method WHL outperforms method K-SKY with respect to all performance metrics. Specifically, the preprocessing time of K-SKY is orders of magnitude larger than the running time of BuildWhl, as it runs for more than one hour even for small inputs. This is expected, since the index constructed by BuildWhl can be seen as a compact version of the 2-hop cover labeling. Also from a space occupancy viewpoint, K-SKY stores at least two orders of magnitude more information with respect to the WHL index. Finally, the average query time offered by K-SKY is always around 3 orders of magnitude larger than that of QueryWhl.

Table 6 Performance of WHL and K-SKY on graphs NY, BAY, COL with $b \in \{50, 100\}$ beer vertices.

b	Graph	Preprocessing (s)		Space Occupancy (MB)		Query (s)	
		PT_{WHL}	$PT_{\text{K-SKY}}$	IS_{WHL}	$IS_{\text{K-SKY}}$	QT_{WHL}	$QT_{\text{K-SKY}}$
50	NY	3.11	> 3600	5.34	795.02	$3.5 \cdot 10^{-7}$	$1.3 \cdot 10^{-4}$
	BAY	3.68	> 3600	4.59	1385.53	$3.3 \cdot 10^{-7}$	$1.6 \cdot 10^{-4}$
	COL	4.77	> 3600	1.78	1982.95	$4.9 \cdot 10^{-7}$	$1.5 \cdot 10^{-4}$
100	NY	6.25	> 3600	6.31	1467.21	$3.7 \cdot 10^{-7}$	$2.6 \cdot 10^{-4}$
	BAY	7.16	> 3600	4.09	2726.54	$3.4 \cdot 10^{-7}$	$3.3 \cdot 10^{-4}$
	COL	9.60	> 3600	2.52	3794.05	$4.1 \cdot 10^{-7}$	$3.0 \cdot 10^{-4}$

5 Conclusion and Future Work

We have showcased that an indexing scheme, to efficiently execute queries on beer distances or shortest beer paths for pairs of vertices of a beer graph, can be built by adapting the highway labeling. Through extensive experimentation on real-world graphs, we have empirically demonstrated its practical effectiveness and superiority, in terms of offered trade-off between preprocessing time, space occupancy and query time, with respect to the state-of-the-art. Our work leaves several questions open for future investigation. First and foremost, it would be interesting to understand whether an indexing method with a space/computational time trade-off better than those mentioned in this paper (either empirically or in the worst case), can be designed for general graphs. Another relevant direction to explore would be extending the experimental comparison of WHL, B2ALL K-SKY given here to digraphs and to queries on shortest beer paths. A more challenging, but certainly of interest, objective to pursue would be designing a dynamic algorithm to maintain the WHL index under changes of the set B.

─── **References** ───

1   Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, USA*, pages 349–360. ACM, 2013. `doi:10.1145/2463676.2465315`.

2   Shikha Anirban, Junhu Wang, and Md. Saiful Islam. Experimental evaluation of indexing techniques for shortest distance queries on road networks. In *39th IEEE International Conference on Data Engineering (ICDE 2023) Anaheim, USA*, pages 624–636. IEEE, 2023. `doi:10.1109/ICDE55515.2023.00054`.

3   Joyce Bacic, Saeed Mehrabi, and Michiel Smid. Shortest beer path queries in outerplanar graphs. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation (ISAAC 2021), Fukuoka, Japan*, volume 212 of *LIPIcs*, pages 62:1–62:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPICS.ISAAC.2021.62`.

4   Joyce Bacic, Saeed Mehrabi, and Michiel Smid. Shortest beer path queries in outerplanar graphs. *Algorithmica*, 85(6):1679–1705, 2023. `doi:10.1007/S00453-022-01045-4`.

5   Valentin Buchhold, Dorothea Wagner, Tim Zeitz, and Michael Zündorf. Customizable Contraction Hierarchies with Turn Costs. In Dennis Huisman and Christos D. Zaroliagis, editors, *20th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2020)*, volume 85 of *Open Access Series in Informatics (OASIcs)*, pages 9:1–9:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.ATMOS.2020.9`.

6   Alessio Cionini, Gianlorenzo D'Angelo, Mattia D'Emidio, Daniele Frigioni, Kalliopi Giannakopoulou, Andreas Paraskevopoulos, and Christos D. Zaroliagis. Engineering graph-based models for dynamic timetable information systems. In Stefan Funke and Matús Mihalák, editors, *14th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2014, September 11, 2014, Wroclaw, Poland*, volume 42 of *OASIcs*, pages 46–61. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. `doi:10.4230/OASICS.ATMOS.2014.46`.

7   Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. *SIAM J. Comput.*, 32(5):1338–1355, 2003. `doi:10.1137/S0097539702403098`.

8   Feliciano Colella, Mattia D'Emidio, and Guido Proietti. Simple and practically efficient fault-tolerant 2-hop cover labelings. In Dario Della Monica, Aniello Murano, Sasha Rubin, and Luigi Sauro, editors, *Joint Proceedings of the 18th Italian Conference on Theoretical Computer Science (ICTCS 2017) and the 32nd Italian Conference on Computational Logic (CILC 2017), Naples, Italy*, volume 1949 of *CEUR Workshop Proceedings*, pages 51–62. CEUR-WS.org, 2017.

9   David Coudert, Andrea D'Ascenzo, and Mattia D'Emidio. D-hash/ShortestBeerDistanceQueries. Software (visited on 2024-09-20). URL: `https://github.com/D-hash/ShortestBeerDistanceQueries`.

10 Gianlorenzo D'Angelo, Mattia D'Emidio, and Daniele Frigioni. Fully dynamic 2-hop cover labeling. *ACM J. Exp. Algorithmics*, 24(1):1.6:1–1.6:36, 2019. `doi:10.1145/3299901`.

11 Rathish Das, Meng He, Eitan Kondratovsky, J. Ian Munro, Anurag Murty Naredla, and Kaiyu Wu. Shortest beer path queries in interval graphs. In Sang Won Bae and Heejin Park, editors, *33rd International Symposium on Algorithms and Computation (ISAAC 2022), Seoul, Korea*, volume 248 of *LIPIcs*, pages 59:1–59:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPICS.ISAAC.2022.59`.

12 Andrea D'Ascenzo and Mattia D'Emidio. Top-k distance queries on large time-evolving graphs. *IEEE Access*, 11:102228–102242, 2023. `doi:10.1109/ACCESS.2023.3316602`.

13 Daniel Delling, Julian Dibbelt, Thomas Pajor, and Tobias Zündorf. Faster Transit Routing by Hyper Partitioning. In Gianlorenzo D'Angelo and Twan Dollevoet, editors, *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS*

2017), volume 59 of *Open Access Series in Informatics (OASIcs)*, pages 8:1–8:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.ATMOS.2017.8`.

14   Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Robust distance queries on massive networks. In Andreas S. Schulz and Dorothea Wagner, editors, *Proceedings of the 22th Annual European Symposium on Algorithms (ESA 2014), Wroclaw, Poland*, volume 8737 of *Lecture Notes in Computer Science*, pages 321–333. Springer, 2014. `doi:10.1007/978-3-662-44777-2_27`.

15   Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Connection scan algorithm. *ACM J. Exp. Algorithmics*, 23, 2018. `doi:10.1145/3274661`.

16   Edsger W. Dijkstra. A note on two problems in connexion with graphs. In Krzysztof R. Apt and Tony Hoare, editors, *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, volume 45 of *ACM Books*, pages 287–290. ACM / Morgan & Claypool, 2022. `doi:10.1145/3544585.3544600`.

17   Jochen Eisner and Stefan Funke. Sequenced route queries: getting things done on the way back home. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL 2012, pages 502–505, New York, NY, USA, 2012. ACM. `doi:10.1145/2424321.2424400`.

18   Muhammad Farhan, Koehler Henning, and Qing Wang. BatchHL$^+$: batch dynamic labelling for distance queries on large-scale networks. *The VLDB Journal*, pages 1–29, 2023. `doi:10.1007/s00778-023-00799-9`.

19   Muhammad Farhan and Qing Wang. Efficient maintenance of highway cover labelling for distance queries on large dynamic graphs. *World Wide Web (WWW)*, 26(5):2427–2452, 2023. `doi:10.1007/S11280-023-01146-2`.

20   Muhammad Farhan, Qing Wang, Yu Lin, and Brendan D. McKay. A highly scalable labelling approach for exact distance queries in complex networks. In Melanie Herschel, Helena Galhardas, Berthold Reinwald, Irini Fundulaki, Carsten Binnig, and Zoi Kaoudi, editors, *Proceedings of 22nd International Conference on Extending Database Technology (EDBT 2019), Lisbon, Portugal*, pages 13–24. OpenProceedings.org, 2019. `doi:10.5441/002/EDBT.2019.03`.

21   Roy Friedman and Alex Kogan. Deterministic dominating set construction in networks with bounded degree. In Marcos K. Aguilera, Haifeng Yu, Nitin H. Vaidya, Vikram Srinivasan, and Romit Roy Choudhury, editors, *Distributed Computing and Networking*, pages 65–76, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

22   Joachim Gudmundsson and Yuan Sha. Shortest beer path queries in digraphs with bounded treewidth. In Satoru Iwata and Naonori Kakimura, editors, *34th International Symposium on Algorithms and Computation, ISAAC 2023, December 3-6, 2023, Kyoto, Japan*, volume 283 of *LIPIcs*, pages 35:1–35:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. `doi:10.4230/LIPICS.ISAAC.2023.35`.

23   Tesshu Hanaka, Hirotaka Ono, Kunihiko Sadakane, and Kosuke Sugiyama. Shortest beer path queries based on graph decomposition. In Satoru Iwata and Naonori Kakimura, editors, *34th International Symposium on Algorithms and Computation (ISAAC 2023), Kyoto, Japan*, volume 283 of *LIPIcs*, pages 37:1–37:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. `doi:10.4230/LIPICS.ISAAC.2023.37`.

24   Vassilis Kaffes, Alexandros Belesiotis, Dimitrios Skoutas, and Spiros Skiadopoulos. Finding shortest keyword covering routes in road networks. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*, pages 1–12, 2018.

25   Vassilissa Lehoux and Christelle Loiodice. Faster preprocessing for the trip-based public transit routing algorithm. In Dennis Huisman and Christos D. Zaroliagis, editors, *20th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2020, September 7-8, 2020, Pisa, Italy (Virtual Conference)*, volume 85 of *OASIcs*, pages 3:1–3:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/OASICS.ATMOS.2020.3`.

26   Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Trans. Intell. Syst. Technol.*, 8(1), July 2016. `doi:10.1145/2898361`.

27   Huiping Liu, Cheqing Jin, Bin Yang, and Aoying Zhou. Finding top-k optimal sequenced routes. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*, pages 569–580. IEEE Computer Society, 2018. `doi:10.1109/ICDE.2018.00058`.

28   Catherine C. McGeoch. *A Guide to Experimental Algorithmics*. Cambridge University Press, 2012.

29   Michael N. Rice and Vassilis J. Tsotras. Engineering generalized shortest path queries. In Christian S. Jensen, Christopher M. Jermaine, and Xiaofang Zhou, editors, *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 949–960. IEEE Computer Society, 2013. `doi:10.1109/ICDE.2013.6544888`.

30   Mehdi Sharifzadeh, Mohammad Kolahdouzan, and Cyrus Shahabi. The optimal sequenced route query. *The VLDB journal*, 17:765–787, 2008.

31   Chao Zhang, Angela Bonifati, and M. Tamer Özsu. An overview of reachability indexes on graphs. In *Companion of the 2023 International Conference on Management of Data*, SIGMOD 2023, pages 61–68, New York, NY, USA, 2023. ACM. `doi:10.1145/3555041.3589408`.

32   Junhua Zhang, Long Yuan, Wentao Li, Lu Qin, Ying Zhang, and Wenjie Zhang. Label-constrained shortest path query processing on road networks. *The VLDB Journal*, 33:569–593, 2024. `doi:10.1007/s00778-023-00825-w`.

33   Yikai Zhang and Jeffrey Xu Yu. Hub labeling for shortest path counting. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD 2020, pages 1813–1828, New York, NY, USA, 2020. Association for Computing Machinery. `doi:10.1145/3318464.3389737`.

34   Zibin Zheng, Fanghua Ye, Rong-Hua Li, Guohui Ling, and Tan Jin. Finding weighted k-truss communities in large networks. *Information Sciences*, 417:344–360, 2017. `doi:10.1016/j.ins.2017.07.012`.

35   Huaijie Zhu, Wenbin Li, Wei Liu, Jian Yin, and Jianliang Xu. Top k optimal sequenced route query with POI preferences. *Data Sci. Eng.*, 7(1):3–15, 2022. `doi:10.1007/S41019-022-00177-5`.