Online Landmark-Based Batch Processing of Shortest Path Queries

Manuel Hotz

University of Konstanz, Germany
manuel.hotz@uni.kn

Leonard Wörteler

University of Konstanz, Germany
leonard.woerteler@uni.kn

Theodoros Chondrogiannis

University of Konstanz, Germany
theodoros.chondrogiannis@uni.kn

Michael Grossniklaus

University of Konstanz, Germany
michael.grossniklaus@uni.kn

ABSTRACT

Processing shortest path queries is a basic operation in many graph problems. Both preprocessing-based and batch processing techniques have been proposed to speed up the computation of a single shortest path by amortizing its costs. However, both of these approaches suffer from limitations. The former techniques are prohibitively expensive in situations where the precomputed information needs to be updated frequently due to changes in the graph, while the latter require coordinates and cannot be used on non-spatial graphs. In this paper, we address both limitations and propose novel techniques for batch processing shortest paths queries using landmarks. We show how preprocessing can be avoided entirely by integrating the computation of landmark distances into query processing. Our experimental results demonstrate that our techniques outperform the state of the art on both spatial and non-spatial graphs with a maximum speedup of 3.61× in online scenarios.

CCS CONCEPTS

• **Information systems** → *Database query processing*; • **Theory of computation** → *Shortest paths*.

KEYWORDS

Batch Processing, Landmark Embedding, Shortest Paths

ACM Reference Format:

Manuel Hotz, Theodoros Chondrogiannis, Leonard Wörteler, and Michael Grossniklaus. 2021. Online Landmark-Based Batch Processing of Shortest Path Queries. In *33rd International Conference on Scientific and Statistical Database Management (SSDBM 2021)*, July 6–7, 2021, Tampa, FL, USA. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3468791.3468844>

1 INTRODUCTION

Shortest path computations are a basic building block of many graph algorithms used to route traffic in road and transportation networks, to measure the influence of people in a social network, or to answer questions in a knowledge graph. In order to find the shortest path between two nodes in a graph more efficiently, various

techniques to amortize computation costs have been proposed. One family of techniques applies preprocessing to the graph in order to build index structures that can later guide the shortest path search [30]. Another family of techniques groups shortest path queries into batches and then processes them together [24, 33].

However, both families of techniques have their shortcomings. Though the former significantly speed up single-source single-target shortest path computations, they are unsuited for situations where the precomputed information needs to be updated frequently due to changes in the graph. Techniques from the latter assume that the graph is spatially coherent by relying on coordinates and are therefore not applicable to non-spatial graphs.

In this paper, we address both of these problems by proposing two new techniques to compute batch shortest path queries in both spatial and non-spatial graphs alike. Our first technique uses landmark embedding, a well-known preprocessing technique, as an alternative to coordinates for grouping and batch processing shortest path queries on both spatial and non-spatial graphs. Our second technique fuses the preprocessing with the query phase by integrating the computation of the landmark distances into query processing. Our experimental evaluation on non-spatial graphs indicates that our techniques clearly outperform Dijkstra-1N and an “off-the-shelf” approach that combines the state-of-the-art method for batch shortest path query processing [33, 34] with FastMap [3, 8], an algorithm to generate distance-preserving node coordinates. Our results further show that our techniques are comparable to or outperform the state of the art even on spatial graphs.

In summary, the contributions of this paper are as follows:

- We propose the BALT algorithm for batch shortest path query processing using landmarks, along with a landmark-based decomposition method (Section 3).
- We present the OBALT algorithm, an extension of the BALT algorithm that requires no preprocessing, thus being able to process queries in online scenarios (Section 4).
- In a comprehensive experimental evaluation, we study our proposed solutions with regard to the state of the art and demonstrate a consistent speedup of up to 4.52× in offline and 3.61× in online scenarios (Section 5).

We begin in the next section by formally defining the batch shortest path query problem as well as giving a brief overview of goal-directed shortest path search algorithms and the current state-of-the-art solution for batch shortest path query processing.

This work is licensed under a Creative Commons Attribution International 4.0 License.

SSDBM 2021, July 6–7, 2021, Tampa, FL, USA
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8413-1/21/07.
<https://doi.org/10.1145/3468791.3468844>

General related work is discussed in Section 6. Concluding remarks are given in Section 7.

2 BACKGROUND AND DEFINITIONS

Let $G = (N, E)$ be a *directed weighted graph* with nodes N and edges $E \subseteq N \times N$. Each edge $e = (n, n') \in E$ is associated with a direction $n \rightarrow n'$, and has an assigned positive weight w_e that captures the cost of the transition from n to n' . A *path* $p(s \rightarrow t)$ from a source node s to a target node t is a connected and cycle-free sequence of edges $\langle (s, n_1), (n_1, n_2), \dots, (n_m, t) \rangle$. The length $\ell(p)$ of a path p is equal to the sum of the weights of all of its edges. A *shortest path* between two nodes, $p_{sp}(s \rightarrow t)$, is the path that has the shortest length among all paths from s to t . The length of $p_{sp}(s \rightarrow t)$ is also termed the (*network*) *distance* between s and t , i.e., $d(s, t) = \ell(p_{sp}(s \rightarrow t))$.

2.1 Problem Definition

In this paper, we study the problem of batch processing of shortest path queries. In practice, a *batch shortest path query* is a set of shortest path queries that are perceived as being issued simultaneously. Hence, the result of such a query contains the results of all the shortest path queries the batch consists of. The batch shortest path query is formally defined as follows:

Definition 2.1 (Batch Shortest Path Query). Let $G = (N, E)$ be a *directed weighted graph*. A *batch shortest path query* takes as input a set $Q = \{(s_1, t_1), \dots, (s_m, t_m)\}$ of source-target pairs such that $\{s_1, \dots, s_m\} \subseteq N$ and $\{t_1, \dots, t_m\} \subseteq N$, and returns a set R_Q of shortest paths between each pair of source-target nodes in Q , i.e., $R_Q = \{p_{sp}(s_1 \rightarrow t_1), \dots, p_{sp}(s_m \rightarrow t_m)\}$.

The simplest way to process a batch shortest path query Q is to answer each shortest path query separately using an existing algorithm [30]. However, it has been shown in previous work [18, 34] that the result of certain queries in Q can be utilized to speed up the processing of other queries in Q . This may happen as the processing of different queries may require traversing the same part of the graph. Our aim is to process the queries in Q in a way that reduces the total computation time of Q .

2.2 Goal-Directed Shortest Path Search

All algorithms discussed in this work are based on the A^* -search algorithm [12], which traverses a graph $G = (N, E)$ from a source node s to a target node t while employing a heuristic to direct the traversal towards the target. The heuristic $\hat{h} : N \times N \rightarrow \mathbb{R}$ returns an approximation for the distance $d(n, t)$ from node $n \in N$ to the target node. In order for A^* to always find the shortest path, $\hat{h}$ must be *admissible*, i.e., $\hat{h}(t, t) = 0$ and $\forall n \in N : \hat{h}(n, t) \leq d(s, n)$. A heuristic is additionally called *consistent* if it satisfies the triangle inequality $\hat{h}(a, t) \leq w_{(a,b)} + \hat{h}(b, t)$ for all neighboring nodes $(a, b) \in E$. Every consistent heuristic is also admissible [7]. A^* traverses the graph visiting nodes in increasing order of their tentative distance from the source plus their estimated distance to the target, i.e., $d(s, n) + \hat{h}(n, t)$. Assuming a consistent heuristic, each node $n \in N$ visited during the traversal of G is either *open* if the distance $d(s, n)$ has not yet been finalized, or *closed* if the distance $d(s, n)$ has been finalized. The set of open nodes is also called the search's *frontier*.

ALT [10] is a preprocessing-based method that combines A^* -search with a graph topology-aware heuristic. Given a graph G , during the *preprocessing* phase, an ordered set of k nodes of G is chosen as landmarks $L = \{l_1, l_2, \dots, l_k\}$. For each of the selected landmarks $l \in L$, all distances $d(n, l)$ from each landmark $l \in L$ to a node n of the graph, and all distances $d(l, n)$ from a node n to each landmark $l \in L$ are computed and stored as two vectors associated with each node. We refer to these two vectors as the *forward* landmark vector for node n being $\vec{v}_n = \langle d(l_1, n), d(l_2, n), \dots, d(l_k, n) \rangle$ and the *backward* landmark vector being $\vec{v}_n^K = \langle d(n, l_1), d(n, l_2), \dots, d(n, l_k) \rangle$.

Given such a set of landmarks L , we can obtain a lower bound for the distance between any two nodes $s, t \in N$ using the triangle inequality, i.e.,

$$d(s, t) \geq \max_{l \in L} \max \{d(s, l) - d(t, l), d(l, t) - d(l, s), 0\}. \quad (1)$$

ALT executes A^* using this lower bound as a heuristic.

Naturally, obtaining tighter bounds, i.e., bounds that are close to the actual distance from s to t , is preferable. Hence, the selection of landmarks L can have a significant impact on the performance of the ALT algorithm. For instance, if a landmark lies far away from both s and t with its distance from/to s being almost equal to its distance from/to t , the value of the lower bound will be very close to zero. Instead if a landmark lies very close to either s or t , the lower bound will be very close to the distance $d(s, t)$. At the same time, the set of landmarks L must be small to avoid inflicting high computational and space overhead for the computation of the tightest possible lower bound. Towards this end, multiple landmark selection strategies have been proposed [23].

2.3 Batch Processing with Petal- A^*

To enable the shared computation of individual queries for efficient batch shortest path query processing on road networks, Zhang *et al.* [33, 34] have proposed BPQS as a solution, which is based on a multiple-targets A^* -search extension. The proposed approach first splits the input batch shortest path query into a set of $1-N$ shortest path queries (grouping) and then determines the sequence in which these queries are processed (scheduling) in order to maximize the shared computations. Since our techniques directly extend and improve upon the work of Zhang *et al.*, in what follows, we summarize the key ideas of BPQS in order to make this paper self-contained.

2.3.1 The Petal- A^* Algorithm. Zhang *et al.* [33] first introduced the A^* -1N algorithm as an extension of the classic A^* -search [12] to support the processing of single-source multiple-targets shortest path queries. Given a graph $G = (N, E)$, a source node s and a set of target nodes T , the A^* -1N algorithm first chooses one of the target nodes as the representative node t_r , towards which the heuristic is always computed. The algorithm traverses the graph from s towards t_r in the same fashion as the traditional A^* -search. For any target $t' \in T$ found during the traversal, including t_r , the shortest path $p_{sp}(s \rightarrow t')$ is retrieved. The traversal continues until the shortest paths to all the targets have been found, potentially continuing after t_r has already been reached.

Because A^* -1N enumerates all reachable nodes in non-descending order of $d(s, n) + \hat{h}(n, t)$, its efficiency in retrieving shortest paths to nodes other than t_r strongly depends on their closeness to $p_{sp}(s, t_r)$. Consider as an example a $1-N$ query from s to the

Figure 1: Search space illustration of answering a 1– N query (with the blue dot as source and the red dots as targets) by either one A^* -1 N search (b) or with two separate A^* -1 N searches (c) using two different representative nodes t_r and t'_r . Straight lines represent shortest-path distances, jagged lines show heuristic estimates towards representative nodes of each search.

target set $\{t_r, t', t'_r, t''\}$ with representative node t_r . Assuming a two-dimensional spatial network and a fairly accurate heuristic, the search space of A^* -1 N after reaching t_r resembles an ellipse around s and t_r as shown in Figure 1a. Then, finding target node t' only requires a fairly narrow region of the graph to be explored – the blue ellipse in Figure 1b grows around its foci s and t_r until reaching t' . However, including t'' and t'_r in the target set leads to all nodes in the hatched area also being visited, which expands the search space significantly until both nodes are found.

In order to reduce the search space, a second A^* -1 N search can be started from s with representative node t'_r and target set $\{t'', t'_r\}$. Figure 1c shows the resulting search space, i.e., the sum of both ellipses. Furthermore, given two consecutive 1– N searches from the same source node s , the second query can reuse the closed set and the open set (i.e., the frontier) of the previous query. The information about *closed* nodes, i.e., their distances from the source s , does not change since the subsequent search has the same source. However, *open* nodes have to be reordered taking into account the estimated distances towards the new representative node t'_r . Figure 1c shows the part of the subsequent search space (in blue) for which the previous search obtained results in its closed set as a dotted region. The open nodes that must be reordered are represented by the black dotted ellipse.

*Petal-A** [34] takes advantage of the aforementioned observations to optimize the processing of a set of 1– N queries. In order to split one 1– N query into multiple more focused 1– N queries, *Petal-A** uses *target set decomposition* to partition the target set of the input 1– N query. More specifically, *Petal-A** employs the *angle-based* decomposition method, which groups together targets that have a similar angle from the source node. This decomposition is parameterized by an *angle threshold* θ that dictates when nodes are to be put into the same or a different target set. When running the individual, more focussed A^* -1 N searches, *Petal-A** retains the closed and open sets between searches and modifies them as described above.

2.3.2 Query Grouping and Scheduling. BPQS processes batch shortest path queries by executing multiple *Petal-A** searches. To this end, the first step of BPQS is the *query grouping*, where a given batch shortest path query Q on a graph $G = (N, E)$ is split into multiple 1– N shortest path queries.

(a) Query order for Multi-BPQS.

(b) Query order for Zigzag-BPQS.

Figure 2: Example batch shortest path query Q answered by Multi-BPQS and Zigzag-BPQS. Order of queries is indicated by circles located near the source of each executed query. Note that Zigzag-BPQS executes one query (2.) on the graph with edges reversed.

Multi-BPQS first groups the queries in Q by source node into a partition Q_S , where subsets represent 1– N queries. For example, given the batch shortest path query $Q = \{q_1(s_1, t_1), q_2(s_1, t_2), q_3(s_2, t_3), q_4(s_3, t_3)\}$, depicted as jagged arrows in Figure 2a, *Multi-BPQS* groups Q into $Q_S = \{\{q_1(s_1, t_1), q_2(s_1, t_2)\}, \{q_3(s_2, t_3), q_4(s_3, t_3)\}\}$. To mark the resulting 1– N queries, each group is surrounded by dashed lines and the source node of each 1– N query is marked by a circle close by. In general, *Multi-BPQS* can also alternatively choose to group the queries by target node into a set Q_T . In the example, the resulting partition is $Q_T = \{\{q_1(s_1, t_1), q_2(s_1, t_2)\}, \{q_3(s_2, t_3), q_4(s_3, t_3)\}\}$. Queries in Q_S are executed on G while queries in Q_T are executed on the edge-reversed graph G^R . *Multi-BPQS* schedules *Petal-A** searches such that larger 1– N queries are computed first. A resulting example schedule is depicted in Figure 2a with the query order shown in blue circles.

In contrast to *Multi-BPQS*, *Zigzag-BPQS* considers partitions Q_S and Q_T simultaneously by scheduling them interleaved. Larger 1– N queries are again executed earlier. Since a query result obtained from Q_S can potentially answer queries in Q_T and vice versa, *Zigzag-BPQS* prunes the respective other query set after each *Petal-A**. Figure 2b depicts a reduced query set since *Zigzag-BPQS* can prune queries that were already answered in the reverse direction. For example, the 1– N query $\{q_3(s_2, t_3)\}$ from Q_S will be pruned by answering the query marked by 2. in an earlier *Petal-A** search.

Finally, *Zigzag Cluster* groups queries like *Zigzag-BPQS*. In the scheduling phase though, by executing A^* -1 N searches from different directions interleaved and answering large 1– N queries first,

Zigzag Cluster enables the pruning of smaller queries from the reverse query set. Since the authors report worse performance and higher memory usage than Multi-BPQS and Zigzag-BPQS [34], we omit this variant from our evaluation.

3 LANDMARK-BASED BATCH PROCESSING

In this section, we propose *Batch-ALT* (BALT), a landmark-based technique to compute batch shortest path queries in spatial and non-spatial graphs. Our technique is based on the preprocessing and query phases of the ALT algorithm [10]. Even though ALT was originally developed for road networks, it is effective in non-road networks [23] as well, while having relatively low preprocessing overhead [1]. Since ALT does not use node coordinates in its heuristic function but uses the computed landmark embedding based on distances in the graph, our BALT technique does not require node coordinates as well, making it applicable in graphs where nodes are not associated with coordinates.

3.1 Shared Computation with Petal-ALT

In its preprocessing phase, BALT selects from a graph $G(N, E)$ a set of nodes $L \subset N$ as landmarks and precomputes distances to and from landmarks from and to all nodes in the graph similar to the ALT algorithm (Section 2.2). This preprocessing phase associates each node $n \in N$ with two vectors, one defined by distances to each landmark $\vec{d}(n, \cdot)_L$ and one defined by distances from each landmark $\vec{d}(\cdot, n)_L$, both of which are of size $|L|$. Thus, the information for the heuristic function used in BALT takes up space linear in the number of nodes in the graph: $O(|N| \cdot |L|)$, where $|L| \ll |N|$ usually.

While any of the proposed landmark selection techniques [10, 11, 23] can be used, one should consider the trade-off between preprocessing time and benefit through more accurate distance estimation. Based on existing work that measures the efficacy of landmark selection techniques [11, 13], we employ the *farthest* technique [10] which aims at heuristically maximizing the pairwise distances between selected landmarks.

In the query phase, BALT retains the input query grouping and query scheduling techniques—namely Multi or Zigzag—of BPQS. However, to process $1-N$ queries, BALT uses Petal-ALT, a version of Petal-A* that uses the landmark embedding to direct the search in the underlying A*-1N algorithm, which we therefore refer to as ALT-1N. To decompose $1-N$ queries into smaller subqueries, Petal-ALT uses a *landmark-based target set decomposition method*, which we describe in Section 3.2.

Algorithm 1 illustrates the pseudocode of Petal-ALT. The algorithm takes as input a graph G , a heuristic function $\hat{h}_L$, a target set decomposition threshold θ , and a $1-N$ query set Q_s . It obtains the source s and target set T_s from Q_s and starts with an initially empty query result R_{Q_s} in Line 2–4. Then, in Line 5, Petal-ALT initializes a search from s , setting up the open and closed sets for the search. In the following line, the landmark-based decomposition method partitions the target set T_s into multiple smaller target sets $\{C_0, \dots, C_i\}$ based on the landmarks L and the decomposition threshold θ . Petal-ALT processes these target sets in Line 7–16 sequentially as follows. First, the heuristically farthest away node among the targets is chosen in Line 7 as the representative node t_r . Previously found targets are removed from the current target set

Algorithm 1: PETAL-ALT.

Input: Weighted directed graph $G = (N, E)$, heuristic $\hat{h}_L$, landmarks L , angle threshold θ , $1-N$ query set Q_s
Data: T_s targets of query set to be found, T_C targets of query set to be found in each petal
Result: Query results R_{Q_s}

```

1 PetalALT( $G, L, \theta, Q_s$ ) begin
2    $s \leftarrow \text{source}(Q_s)$ 
3    $T_s \leftarrow \text{targets}(Q_s)$ 
4    $R_{Q_s} \leftarrow \emptyset$ 
   // initialize a new Petal-ALT search from  $s$ 
5    $\text{Search} \leftarrow \text{initFromSource}(s)$ 
6   foreach  $C \in \text{decompose}(L, \theta, s, T_s)$  do
7      $t_r \leftarrow \arg \max_{t \in T} \hat{h}_L(s, t)$ 
   // remove targets found earlier
8      $T_C \leftarrow C \cap T_s$ 
9     if  $T_C \neq \emptyset$  then
   // compute an ALT-1N petal from  $s$  to  $t_r$ 
10    foreach  $p_{sp}(s, r) \in \text{Search.towards}(t_r)$  do
11      if  $r \in T_s$  then // found a new target
12         $T_s \leftarrow T_s \setminus \{r\}$ 
13         $R_{Q_s} \leftarrow R_{Q_s} \cup \{p_{sp}(s, r)\}$ 
14         $T_C \leftarrow T_C \setminus \{r\}$ 
15        if  $T_C = \emptyset$  then // found all targets
16          break
17   return  $R_{Q_s}$ 

```

T_C in Line 8. If there remain targets to be found in T_C , Petal-ALT proceeds with an ALT-1N search towards t_r in Line 10–16. To direct the search towards the new t_r , the open set of the search is reordered taking into account the estimated distances towards t_r . When a target r is found during the search, it is removed from T_s and T_C and the path $p_{sp}(s, r)$ is added to the results. When T_C is empty, the current ALT-1N search stops and Petal-ALT proceeds with the next target set remaining.

3.2 Landmark-Based Target Set Decomposition

The existing decomposition methods proposed by Zhang *et al.*, which were proposed for road networks, cannot be used without the availability of node coordinates. In contrast, in order to be applicable in non-spatial networks, Petal-ALT is by design oblivious to node coordinates. Hence, to enable the shared computation of multiple $1-N$ queries with Petal-ALT, we propose the *landmark-based target set decomposition* which utilizes the distance vectors of a landmark embedding. Our decomposition method is based on the cosine similarity measure of two vectors, e.g., landmark vectors of query targets. Intuitively, this measure expresses the relation of one vector with another vector in terms of direction, ranging from “same direction” (a value of 1) to “opposite direction” (a value of -1). This measure captures information necessary to obtain target-focused searches, thus minimizing the combined search space of Petal-ALT.

In order to express this measure with respect to the source node of s the input query, our decomposition method calculates a *centered* cosine similarity by centering the landmark vector values of target nodes on s , and then calculating the cosine similarity on the resulting vectors. By taking the arc cosine of this similarity measure one can obtain a value as an angle, which might be more natural as a threshold parameter. With the forward landmark vector $\vec{v}_n$ for node n , we compute the centered cosine similarity between two nodes a and b in relation to node s as:

$$\text{sim}_L^s(a, b) := \frac{(\vec{v}_a - \vec{v}_s) \cdot (\vec{v}_b - \vec{v}_s)}{\|\vec{v}_a - \vec{v}_s\| \|\vec{v}_b - \vec{v}_s\|}$$

Based on this centered cosine similarity measure, our decomposition method partitions the input target set as follows. Iteratively, it selects an arbitrary unlabeled node $n \in T_s$. It assigns the label of n to all other unlabeled target nodes that have a cosine similarity centered on s which is over a specified similarity threshold θ . When no unlabeled nodes remain, the decomposition is finished and returns a partition where nodes having the same label are in the same subset.

3.3 Proof of Correctness

Since BALT transforms a batch shortest path query into a set of $1-N$ queries, its correctness depends solely on the correctness of ALT-1N, which in turn differs from A^* -1N in the heuristic that is used to guide the search. Zhang *et al.* present A^* -1N (and by extension Petal- A^* and BPQS) as a technique on road networks. Their correctness proof implicitly relies on the triangle inequality holding for the heuristic used by A^* [33, Lemma 1]. This is the case for both the euclidean and geodesic distance in spatial networks, but is not a requirement for heuristics in A^* -search in general.

In order to prove the correctness of ALT-1N, we show that the A^* -1N technique is correct with any heuristic as long as it is consistent, and that the landmark-based heuristic used by ALT-1N is in fact consistent.

LEMMA 3.1. *Given a graph $G = (N, E)$, nodes $s, t \in N$ and a consistent heuristic $\hat{h}$, an A^* -search from s towards t*

- (a) *closes all nodes $n \in N$ reachable from s in non-descending order of $\hat{f}(n) := d(s, n) + \hat{h}(n, t)$, and*
- (b) *has found the shortest path $p_{sp}(s, n)$ to any node $n \in N$ as soon as n is added to the closed set.*

PROOF. Part (a) and (b) are proven as Lemma 3 and Lemma 2 respectively in the paper by Hart *et al.* originally introducing the A^* algorithm [12]. $\square$

COROLLARY 3.1.1. *Given a graph $G = (N, E)$, a start node $s \in N$, a set of target nodes $T \subseteq N$ and a consistent heuristic $\hat{h} : N \times N \rightarrow \mathbb{R}$, A^* -1N correctly computes the shortest paths $\{p_{sp}(s, t) \mid t \in T\}$, since all reachable target nodes $t \in T$ are found and their respective shortest path $p_{sp}(s, t)$ is returned.*

LEMMA 3.2. *Landmark-based lower bounds $\hat{h}_L$ for shortest-path distances on a graph $G = (N, E)$ are a consistent heuristic, i.e., for every $L \subseteq N$, $(a, b) \in E$ and every $t \in N$, $\hat{h}_L$ obeys (1) $\hat{h}_L(t, t) = 0$ and (2) the triangle inequality $\hat{h}_L(b, t) \geq \hat{h}_L(a, t) - w_{(a, b)}$.*

PROOF. Using the lower bounds of Equation 1, the heuristic function for ALT is:

$$\hat{h}_L(u, v) := \max_{l \in L} \max\{d(l, v) - d(l, u), d(u, l) - d(v, l), 0\} \quad (2)$$

We prove the two properties required for consistency (and therefore also admissibility) separately.

- (1) $\hat{h}_L(t, t) = \max_{l \in L} \max\{\underbrace{d(l, t) - d(l, t)}_{=0}, \underbrace{d(t, l) - d(t, l)}_{=0}, 0\} = 0$
- (2) Given an edge $(a, b) \in E$, the two “max” operations make it so that in order for the triangle inequality to hold, *at least one* landmark must produce an estimation for $d(b, t)$ that is greater than or equal to $\hat{h}_L(a, t) - w_{(a, b)}$. This is in fact true for any landmark node $l^* \in L$ that produces the maximum estimation for $d(a, t)$.

We now distinguish which component of the set in Equation 2 produced $\hat{h}_L(a, t)$, i.e., which of the three elements of the set $\{d(l^*, t) - d(l^*, a), d(a, l^*) - d(t, l^*), 0\}$ is largest. We mark the steps in which we apply the triangle inequality on shortest-path distances with a “ Δ ”.

- (a) If $\hat{h}_L(a, t) = 0$ we have:

$$\begin{aligned} \hat{h}_L(b, t) &= \max_{l \in L} \max\{d(l, t) - d(l, b), d(b, l) - d(t, l), 0\} \geq 0 \\ &= \hat{h}_L(a, t) > \hat{h}_L(a, t) - \underbrace{w_{(a, b)}}_{>0} \end{aligned}$$

- (b) If $\hat{h}_L(a, t) = d(l^*, t) - d(l^*, a)$ we have:

$$\begin{aligned} \hat{h}_L(b, t) &= \max_{l \in L} \max\{d(l, t) - d(l, b), d(b, l) - d(t, l), 0\} \\ &\geq d(l^*, t) - d(l^*, b) \\ &\stackrel{\Delta}{\geq} d(l^*, t) - (d(l^*, a) + w_{(a, b)}) \\ &= (d(l^*, t) - d(l^*, a)) - w_{(a, b)} \\ &= \hat{h}_L(a, t) - w_{(a, b)} \end{aligned}$$

- (c) If $\hat{h}_L(a, t) = d(a, l^*) - d(t, l^*)$ we have:

$$\begin{aligned} \hat{h}_L(b, t) &= \max_{l \in L} \max\{d(l, t) - d(l, b), d(b, l) - d(t, l), 0\} \\ &\geq d(b, l^*) - d(t, l^*) \\ &\stackrel{\Delta}{\geq} (d(a, l^*) - w_{(a, b)}) - d(t, l^*) \\ &= (d(a, l^*) - d(t, l^*)) - w_{(a, b)} \\ &= \hat{h}_L(a, t) - w_{(a, b)} \end{aligned}$$

$\square$

THEOREM 3.3. *Given a graph $G = (N, E)$, a start node $s \in N$, a set of target nodes $T \subseteq N$ and a set of landmark nodes $L \subseteq N$, ALT-1N using the landmark heuristic $\hat{h}_L$ correctly computes the shortest paths $\{p_{sp}(s, t) \mid t \in T\}$.*

PROOF. The theorem follows from Corollary 3.1.1 and Lemma 3.2. $\square$

4 ONLINE-BATCH-ALT

Like any other preprocessing-based algorithm, BALT is not well-suited for scenarios in which the graph structure or edge weights change frequently. Such scenarios can arise naturally in both spatial and non-spatial graphs. For example, in a spatial graph representing a road network, road construction and traffic congestion lead to changes in the structure and edge weights of the graph, respectively. In non-spatial graphs, e.g., a large-scale social network, user activity may lead to relationships between participants being added or removed, which changes the structure of the underlying graph. In such a scenario, any precomputed landmark distances may be outdated by the time a new batch query is issued and could lead to the resulting heuristic not being admissible anymore. To ensure the correctness of the computation, the distances from/to the landmarks therefore have to be recomputed from scratch prior to answering the query.

One way to profit from the additional work needed for landmark computation is to choose some nodes of the batch query as landmark nodes. All $1-N$ queries starting from landmarks can then be answered through the corresponding landmark computation. Moreover, this choice of landmarks benefits other $1-N$ queries that start or end nearby. This means that areas of the graph where many queries start or end have a high probability of receiving a landmark. As earlier work demonstrated [10, 13], estimations from or to such areas are very accurate. However, placing additional landmarks in the vicinity of already existing ones imposes additional computational and space overhead without improving the estimations considerably. As such, it is desirable to choose a set of landmarks that are far away from each other.

Following from the aforementioned observations, we propose *Online-BALT* (OBALT) to answer batch shortest path queries efficiently without any need for preprocessing. OBALT fuses the preprocessing phase of BALT with the query phase, i.e., selects k of the query nodes as landmarks while maximizing their pairwise distances. We present the pseudocode of OBALT in Algorithm 2. OBALT takes as input a graph G , a user-defined parameter k for the number of landmarks to be computed, and the batch shortest path query Q . The algorithm starts by grouping the batch into a query set Q_S , where subsets share the same source and represent $1-N$ queries. In its first stage, in Line 3 – 15, OBALT uses the first k $1-N$ queries in Q_S to compute landmark vectors from, using their source nodes as landmarks. A mapping $d_{\min}$ from all query sources to minimum distances to and from the current landmarks is maintained throughout the procedure and initialized in Line 3.

Starting from Line 4, the algorithm iteratively selects k $1-N$ queries to compute landmarks with. The first $1-N$ query is chosen as the one with the largest target set size (Line 7). Each subsequent query is chosen such that the distance of its source to the existing landmarks is maximized (Line 9) according to the distances in $d_{\min}$. The source l of the chosen query Q_l is the next landmark and is added to the current set of landmarks L in Line 11. In order to obtain two landmark distance vectors for each node in the graph based on the landmark l , the algorithm computes the landmark embedding for l according to ALT in Line 12, i.e., by running one Dijkstra-1-to-All in G and one Dijkstra-1-to-All in G^R . Note that only through the search in G we can obtain the result R_{Q_l} for

Algorithm 2: ONLINE-Batch-ALT.

Input: Weighted directed graph $G = (N, E)$, number of landmarks k , batch query set Q , angle threshold θ
Data: map of minimum distances $d_{\min} : N \mapsto \mathbb{R} \times \mathbb{R}$ between query sources and current landmark set L , landmark heuristic $\hat{h}_L$
Result: Query results R_Q

```

1 OBALT( $G, k, Q$ ) begin
2    $Q_S \leftarrow \text{groupBySource}(Q)$ 
3   foreach  $Q_S \in Q_S$  do  $d_{\min}[\text{source}(Q_S)] \leftarrow (\infty, \infty)$ 
4    $L \leftarrow \emptyset$ 
5   while  $|L| < k$  do
6     if  $L = \emptyset$  then
7        $Q_l \leftarrow \arg \max_{q \in Q_S} |\text{targets}(q)|$ 
8     else
9        $Q_l \leftarrow \text{chooseByFarthestSource}(Q_S, d_{\min})$ 
10     $Q_S \leftarrow Q_S \setminus \{Q_l\}$ 
11     $l \leftarrow \text{source}(Q_l)$ 
12    // compute landmark  $l$  with Dijkstra-1-to-All
13     $(vec_l, R_{Q_l}) \leftarrow \text{computeLandmark}(G, Q_l)$ 
14     $R_Q \leftarrow R_Q \cup R_{Q_l}$ 
15     $L \leftarrow L \cup \{l \mapsto vec_l\}$ 
16    updateMinDists( $d_{\min}, \hat{h}_L, l$ )
17  foreach  $Q_S \in Q_S$  with  $|\text{targets}(Q_S)|$  non-ascending do
18     $R_Q \leftarrow R_Q \cup \text{PetalALT}(G, L, \theta, Q_S)$ 
19  return  $R_Q$ 

```

the $1-N$ query Q_l . As such, the search in G^R could be omitted to improve the run time of the current stage, since it does not directly answer any queries of Q_l . However, we show in Section 5.2.2 that computing the landmark distances bi-directionally improves the overall performance of OBALT. After the landmark embedding is computed, the query result R_{Q_l} and landmark distance vectors vec_l are saved (Line 13 – 14). In Line 15, the distance mapping $d_{\min}$ is updated to account for the newly added landmark l .

After the desired number of landmarks is computed on the first k $1-N$ queries, the method proceeds to the second stage in Line 16–18. In this stage, OBALT answers the remaining queries using Petal-ALT using the landmark heuristic from the previous stage. Queries are scheduled according to query size again (Line 16). Finally, in Line 18 the result set R_Q is returned from the algorithm.

5 EXPERIMENTAL EVALUATION

In this section, we present the results of our experimental evaluation. We examine the efficiency of our proposed solutions presented in Sections 3 and 4 with respect to batch processing time. First, we compare our solutions to BPQS, the state-of-the-art for batch processing in road networks, and a baseline solution based on Dijkstra’s algorithm. Second, we evaluate the performance of our offline and online approaches in a diverse set of non-spatial graphs in comparison to an “off-the-shelf” approach and the baseline solution.

Table 1: OSM road networks were obtained from OpenStreetMap [21], DIMACS9 road networks were obtained from DIMACS [4], and non-road networks were obtained from SNAP [17].

Name	Direction	Edge Weights	Network Type	N	E	Density	Est. Diameter
NYC OSM	Directed	Distance	Road	67,106	158,664	$3.52 \cdot 10^{-5}$	$66 \cdot 10^3$
Porto OSM	Directed	Distance	Road	84,587	192,980	$2.70 \cdot 10^{-5}$	$119 \cdot 10^3$
DIMACS9 BAY	Undirected	Distance	Road	321,270	397,415	$7.70 \cdot 10^{-6}$	$355 \cdot 10^3$
DIMACS9 COL	Undirected	Distance	Road	435,666	521,200	$5.49 \cdot 10^{-6}$	$935 \cdot 10^3$
DIMACS9 FLA	Undirected	Distance	Road	1,070,376	1,343,951	$2.35 \cdot 10^{-6}$	$1.2 \cdot 10^6$
p2p-Gnutella31	Directed	Unweighted	Peer-to-peer	14,149	50,916	$2.54 \cdot 10^{-4}$	23
cit-HepPh	Directed	Unweighted	Citation	12,711	139,965	$8.66 \cdot 10^{-4}$	38
ca-AstroPh	Undirected	Unweighted	Co-Author	17,903	196,972	$1.23 \cdot 10^{-3}$	14
com-DBLP	Undirected	Unweighted	Co-Author	317,080	1,049,866	$2.09 \cdot 10^{-5}$	23
wiki-Talk	Directed	Unweighted	Communication	111,881	1,477,893	$1.18 \cdot 10^{-4}$	7
web-Google	Directed	Unweighted	Hyperlinks	434,818	3,419,124	$1.81 \cdot 10^{-5}$	33

Table 2: Real-world taxi trip data obtained from Kaggle.com.

Network	Trips	Date Range
NYC [14]	82540	June 1 st –June 30 th 2016
Porto [6]	142490	June 1 st –June 30 th 2014

Figure 3: Cumulative distributions of real world batch sizes for 1-24 hourly tumbling windows generated from the datasets shown in Table 2.

We conducted our tests on a server-grade machine with two AMD EPYC 7351 16-Core processors, 512GiB 2666Mhz DDR4 memory, Intel 660P NVMe SSD, running GNU/Linux 5.4.0-66, and OpenJDK 15. All methods were implemented single-threaded and in-memory using JGraphT 1.5 [20].

5.1 Networks & Query Workloads

Table 1 gives an overview of the networks used in the evaluation. We divide the networks into two categories: road and non-road networks. The non-road networks come in a variety of network types, e.g., peer-to-peer or citation network. In the experiments, we assign unit weight to edges of unweighted networks and treat undirected edges as directed ones in both directions. We additionally reduce all networks to their largest strongly connected component in order to get results for all generated (source, target)-pairs. The reported diameters and numbers of nodes and edges are those of the reduced networks.

5.1.1 Real-World Query Datasets. The real-world query datasets consist of recorded, geographically located taxi trips in New York City, USA and Porto, Portugal. In order to obtain batches of varying sizes comprised of real-world queries for the experiments, we employ tumbling windows of time intervals of 1, 3, 6, 12, and 24 hours, giving us a wider range of batch sizes to evaluate with. Figure 3 shows the resulting cumulative distributions for batch sizes in the query sets of NYC and Porto, verifying that the obtained batches cover a wide range of sizes. Note that a production environment would likely receive large batch sizes even in very small windows.

5.1.2 Synthetic Query Workloads. The performance of all techniques based on Petal-A* is influenced by the number of distinct source and target nodes in the workload and therefore the number of $1-N$ queries to answer. Additionally, they benefit from query workloads in which target sets per source are decomposable into a low number of highly coherent $1-N$ subqueries that can be answered efficiently by A*-1N.

In order to vary these aspects of spatial cohesion, we generate synthetic workloads as follows. We first partition the graph into a fixed number of clusters using the METIS graph partitioning library [15], and we set one node $c \in N$ in each cluster to be the cluster center. Then, we randomly pick k nodes within a configurable distance d around each cluster center into candidate sets CS_c . To generate a workload of size n , we generate each query (s_i, t_i) , $i \in \{1, \dots, n\}$ by picking source node s_i and target node t_i randomly from different candidate sets. Distance d controls the spatial coherence of each candidate set, while candidate set size $k = |CS|$ influences the probability that a single node occurs multiple times in the workload.

5.2 Batch Processing in Road Networks

We start our experimental evaluation on road networks by comparing our proposed solutions to the state-of-the-art BPQS method. BPQS works on the geodesic distance computed on the node coordinates. We also include a baseline method that is based on answering each $1-N$ query resulting from the input with Dijkstra-1N.

5.2.1 Run Time Comparison With BPQS. Figure 4 and Table 3 show the comparison of our proposed BALT and OBALT techniques

Figure 4: Mean batch execution time on real-world workloads.

with Multi-BPQS, denoted by BPQS-GM, Zigzag-BPQS, denoted by BPQS-GZ, and the baseline approach. Both of our techniques use four landmarks and a bi-directional landmark embedding. For BALT we employ the “farthest” selection technique and “Multi” scheduling. We present the mean batch execution time for queries derived from the taxi trip datasets over various batch sizes. For BALT, we additionally include the preprocessing time that one would have to invest before answering the first batch query.

We observe that both BPQS variants and our proposed approaches offer a considerable speedup over the baseline (cf. Table 3). On the smaller network of NYC, both our approaches outperform BPQS when batches become larger. In the cold start case, when BALT has to preprocess the graph, OBALT benefits from being able to answer the first queries already while building its index. Evidently, when precomputed information is available, BALT outperforms OBALT and BPQS, especially on larger batches. When executed on very small batches and with no precomputed information available, BALT and OBALT do not outperform BPQS. Hence, the preprocessing overhead of landmark embedding does not pay off for small batch sizes compared to BPQS.

On the larger network of the county of Porto, OBALT is slightly slower than BPQS on small batch sizes, while it is on par with BPQS on the larger batch sizes. This difference from NYC where OBALT outperforms BPQS for large batches, can be explained by the geographic distribution of taxi trips in the two datasets. By examining the geographic locations of trip data in the Porto network, we see that nearly all trips occur in a very small region of the network, the city of Porto. In such cases where queries are concentrated only on a small part of the input network, our methods incur a lot of overhead by considering parts of the network that are almost never included in the search space.

5.2.2 Impact of Bi-Directional Landmarks for OBALT. In order to demonstrate that the inclusion of calculated distances from nodes to landmarks can have a positive impact on the query performance

Table 3: Speedup (with and without preprocessing time) in relation to baseline over different batch sizes.

Data	Q	BALT L = 4		OBALT k = 4	BPQS GM	BPQS GZ
		w/o	w/			
NYC	(0, 500]	6.86×	0.77×	1.55×	4.53×	4.42×
	(500, 1500]	6.78×	1.68×	3.02×	4.30×	4.25×
	(1500, 2500]	6.47×	2.61×	4.18×	4.27×	4.19×
	(2500, 4500]	6.32×	3.50×	5.16×	4.19×	4.25×
	(4500, 9000]	6.42×	4.40×	5.58×	4.10×	4.31×
Porto	(0, 500]	5.76×	0.50×	1.00×	4.77×	5.90×
	(500, 1500]	5.37×	1.28×	2.60×	4.40×	5.46×
	(1500, 2500]	5.11×	2.12×	4.08×	4.08×	5.33×
	(2500, 4500]	4.83×	2.58×	4.71×	3.92×	5.35×
	(4500, 9000]	4.64×	2.81×	4.96×	3.91×	5.11×

Figure 5: Impact of bi-directional landmarks (+) for mean batch execution time of OBALT.

of OBALT, we show the mean batch execution time on the taxi trip workload in Figure 5. Here, we compare OBALT with and without these bi-directional landmarks while varying the number of landmarks used. Larger batch sizes clearly benefit from the additional distance vectors for up to 4 landmarks. If more landmarks are involved, computing landmark distances from a search that does not also answer queries from the batch, *i.e.*, in the reverse graph, seems to be detrimental to performance. For very small batches consisting of up to 500 query pairs, calculating bi-directional landmark distances does not have a benefit over the uni-directional variant when many landmarks are involved. The performance difference between the two variants using 2–3 landmarks is negligible, since

Figure 6: Influence of increasing the number of landmarks for BALT and OBALT using bi-directional landmarks. Batch size is (2500, 4500).

the bi-directional variant only has to run two additional Dijkstra-1-to-All searches. The time invested in computing many landmarks in the first stage of OBALT seems to be better invested in answering the remaining parts of the batch query. In conclusion, for a small number of landmarks the bi-directional variant is preferable to the uni-directional variant to achieve higher query performance.

5.2.3 BALT vs. OBALT. Having established the benefits of OBALT with bi-directional landmarks over the uni-directional variant, we now compare it with the offline BALT algorithm. Figure 6 shows the batch query execution time when using various numbers of landmarks on batch queries of 2500–4500 query pairs. The preprocessing time of BALT is depicted as hatched areas. The results show that BALT has an advantage over OBALT when the preprocessing can be done ahead of time and does not factor into the cost of the query execution. As such, BALT is consistently faster than the fastest instance of OBALT when this preprocessing time is excluded. In an online scenario though, *i.e.*, when the structure of the graph or edge weights change prior to a batch query, BALT first has to compute landmark distances from scratch and only then it can answer the batch query. In contrast, OBALT can immediately start computing the batch query, thus offering a faster solution in such a scenario. Furthermore, BALT including preprocessing costs as well as OBALT cannot benefit from more than four landmarks, because the landmark computation cost dominates the benefits gained from a more precise heuristic. Even excluding the preprocessing time in BALT, performance degrades gradually for more than 10 landmarks, since the heuristic function itself, which is called for each node visited during the search, becomes more expensive to compute.

5.2.4 Synthetic Workload. In order to compare the scalability of our proposed solutions with respect to larger road networks, we use a synthetically generated query set as described in Section 5.1.2 on the three road networks from DIMACS [4] presented in Table 1.

Table 4 shows the comparison of BALT and OBALT on 8 landmarks each with Multi-BPQS and the simple Dijkstra-1N technique as a baseline. Over all three networks, the relative order regarding query performance of the tested algorithms stays the same. Both BALT and OBALT are able to complete batch queries nearly twice

Table 4: Mean batch execution time and speedup (with and without preprocessing time) in relation to baseline on synthetic workload in road networks ($|Q| \in (1500, 2000]$, $|CS| = 10$, $d = 20000$).

Method	Time/Speedup	BAY	COL	FLA
BALT-08	<i>Preproc.</i>	8.32s	14.57s	38.90s
	<i>Query</i>	32.26s	43.69s	177.16s
	<i>Speedup w/</i>	4.02×	4.27×	3.59×
	<i>Speedup w/o</i>	5.06×	5.70×	4.37×
OBALT-08	<i>Query</i>	40.23s	58.57s	196.69s
	<i>Speedup</i>	4.06×	4.25×	3.94×
BPQS-GM	<i>Query</i>	74.06s	112.92s	347.48s
	<i>Speedup</i>	2.20×	2.20×	2.23×
Dijkstra-1N	<i>Query</i>	163.24s	248.94s	774.83s

as fast as BPQS when no preprocessing information is available. When BALT can use an existing landmark embedding, it is 2–2.6× faster than BPQS. Compared to the baseline, our techniques achieve a speedup of approximately 4× with no preprocessing beforehand or up to 5× when BALT can use an existing landmark embedding. In the largest network, OBALT achieves a higher speedup than BALT when no preprocessing is available.

5.3 Batch Processing in Non-Road Networks

Zhang *et al.* propose BPQS for batch shortest path query processing on road networks. As such, BPQS employs the geodesic distance between two nodes computed using their coordinates as a heuristic for Petal-A*. In non-road networks, however, such coordinates typically do not exist. For example, most of the networks provided by the Stanford Large Network Dataset Collection [17] do not carry node coordinates. Therefore, in contrast to our solutions, BPQS cannot be used directly for batch shortest path query processing on networks without coordinates.

In this section, we first demonstrate that an “off-the-shelf” approach combining the state-of-the-art BPQS with a distance-preserving method that generates node coordinates does not lead to significant improvements over the Dijkstra-1N baseline technique. Afterwards, we show that our proposed solutions, based on landmark embedding, outperform this “off-the-shelf” approach, achieving a considerable speedup compared to the baseline.

5.3.1 Generated Coordinates with FastMap. An “off-the-shelf” solution to make BPQS applicable to such networks is to generate coordinates for each node in a preprocessing phase. To do so, we use *FastMap* [3, 8] which allows us to generate coordinates and obtain a consistent heuristic using the *Manhattan distance*. Since the resulting heuristic is consistent, this method is suitable to generate coordinates for A*-1N and thus applicable in BPQS. Then, both the heuristic, *i.e.*, the Manhattan distance, and the angle-based decomposition are computed using these generated coordinates. In the experiments we denote BPQS using the heuristic and decomposition on d FastMap dimensions as “BPQS-F- d ”.

Figure 7: Mean batch execution time using BPQS on FastMap coordinates. We selected the networks as representatives, since the remaining networks showed similar effects.

Figure 7 shows the two representative cases of results we obtained from testing with the non-road networks in Table 1. In one case (Figure 7a), BPQS-F is providing only a marginal benefit compared to the Dijkstra-1N baseline, even if one excludes the expensive preprocessing time. In the other case (Figure 7b), BPQS-F improves upon the baseline technique only by up to a third, if preprocessing cost is excluded. For the shown networks, more than eight FastMap dimensions do not improve performance and only add to the preprocessing cost.

5.3.2 Impact of Number of Landmarks and Batch Size. Since the previous section demonstrates that the “off-the-shelf” solution does not improve over the baseline technique considerably, we now investigate the benefits of our proposed techniques on the non-road networks with respect to whether or not preprocessed information can be re-used.

Based on Figure 8, we can conclude that the baseline is only competitive with our proposed solutions when answering large batch sizes using fewer than three landmarks. In a static scenario, where BALT can re-use an existing landmark embedding, it always dominates OBALT’s performance. However, if preprocessing costs have to be taken into consideration, *i.e.*, when changes to the graph invalidate the preprocessed information or after a cold start, OBALT is always faster than BALT, especially when a high number of landmarks is used.

5.3.3 BALT and OBALT on Various Types of Graphs. Figure 9 shows the configurations for BALT, OBALT, and the off-the-shelf-solution BPQS-F with the shortest overall run times for each of the non-spatial graphs. As can be seen, BPQS-F cannot speed up the computation compared to the baseline in five of the graphs. In the remaining graph, its advantage is fairly unsubstantial. Between BALT and OBALT no clear winner emerges. In the citation and peer-to-peer graph OBALT performs best overall. In the smaller co-authorship graph, and the communication and hyperlink graphs, BALT is faster than OBALT if preprocessing costs are ignored, slower otherwise. In the larger co-authorship graph, BALT beats OBALT even if preprocessing costs are considered. Table 5 shows that OBALT provides a speedup of 1.12–3.61× depending on the type of the graph.

6 RELATED WORK

The efficient computation of shortest path queries is a problem that has attracted significant attention from both the academic community and the industry. Classical solutions like Dijkstra’s algorithm [5] and *A*-search* [22] fail to meet the increasing demand for fast shortest path query processing. To achieve scalability, various preprocessing-based methods have been proposed [2, 30], including methods that employ landmark embedding [10, 13, 23]. Naturally, all such methods can process batch shortest path queries, by processing all queries sequentially. To the best of our knowledge though, the only preprocessing-based method that supports query processing in batches is BBQ [32]. Moreover, variants of some methods for shortest path query processing have been developed to process distance joins [9, 16, 26, 28]. However, apart from not supporting the online computation of batch shortest path queries, the aforementioned preprocessing-based methods are limited to computing the distance between source–target pairs and do not retrieve the actual path.

The work most related to ours is BPQS [33, 34], the method proposed by Zhang *et al.* that we describe in Section 2.3. To the best of our knowledge, this is the only method that tackles the exact computation of batch shortest path queries. However, the approximate computation of batch shortest path queries has also attracted significant attention. Such methods focus on road networks where they take advantage of the spatial coherence of such networks, *i.e.*, the fact that many shortest paths between different nodes share common parts [25, 27, 29, 31]. Mahmud *et al.* [19] proposed GBPQ, an approximate method that uses clustering to split the input batch shortest path query into groups of queries where sources and targets are located close-by. For each group, a source and a target region that contain all source and all target nodes of the input query respectively are defined. Every query in the input batch is then processed by concatenating the shortest paths connecting the source and target nodes to border nodes of the respective regions, and a shortest path connecting the two regions. Reza *et al.* [24] extend the work of Mahmud *et al.* with a faster but less accurate method for computing queries in the same group simultaneously. Li *et al.* [18] propose an alternative to the GBPQ technique in which the batch shortest path query is split into groups of queries with sources and targets in the same regions in such a way that the error of the resulting approximation can be bounded.

Figure 8: Impact of increasing the number of landmarks for BALT and OBALT on the synthetic query workload.

Figure 9: Comparison of OBALT, offline BALT and BPQS on FastMap coordinates with Dijkstra-1N over various datasets. The numerical suffix indicates the number of landmarks or FastMap dimensions. OBALT with bi-directional landmarks is marked with a “+” ($|Q| \in (1500, 2000]$, $|CS| = 10$, $d = 5$).Table 5: Speedup and mean batch execution time results for synthetic workload on all non-road networks ($|Q| \in (1500, 2000]$, $|CS| = 10$, $d = 5$).

Network	Method	Time (seconds)		Speedup	
		Preproc.	Query	w/	w/o
p2p-Gnutella31	BALT-12	0.65	1.58	1.30	1.84
	+OBALT-16	—	1.46	1.99	—
	BPQS-F-03	1.50	2.50	0.72	1.16
	Dijkstra1N	—	2.90	1.00	—
cit-HepPh	BALT-03	0.53	2.87	0.90	1.06
	+OBALT-03	—	2.72	1.12	—
	BPQS-F-03	3.81	2.83	0.46	1.08
	Dijkstra1N	—	3.05	1.00	—
ca-AstroPh	BALT-16	2.91	8.90	3.40	4.52
	+OBALT-16	—	11.14	3.61	—
	BPQS-F-08	11.48	25.65	1.08	1.57
	Dijkstra1N	—	40.22	1.00	—
com-DBLP	BALT-16	33.99	68.18	3.01	4.51
	+OBALT-16	—	135.40	2.27	—
	BPQS-F-03	55.13	261.19	0.97	1.18
	Dijkstra1N	—	307.56	1.00	—
wiki-Talk	BALT-08	12.16	97.73	1.74	1.96
	+OBALT-08	—	107.76	1.78	—
	BPQS-F-03	62.61	173.33	0.81	1.11
	Dijkstra1N	—	191.73	1.00	—
web-Google	BALT-08	40.59	86.03	2.83	4.17
	+OBALT-08	—	118.60	3.02	—
	BPQS-F-02	103.09	344.29	0.80	1.04
	Dijkstra1N	—	358.55	1.00	—

7 CONCLUSIONS AND FUTURE WORK

In this paper, we have presented two landmark-based techniques to efficiently answer batch shortest path queries. First, we presented the BALT algorithm, a preprocessing-based method that uses landmarks to direct its search, resulting in its applicability in spatial and non-spatial graphs alike. BALT uses a novel landmark-based decomposition method that aims to maximize shared computations during the search. Our final contribution is the OBALT algorithm, which is an extension of BALT and requires no preprocessing, thus being applicable in situations where preprocessed information is invalidated due to changes in the graph. Our experiments demonstrate that while OBALT is competitive with the state of the art approach on road networks, it beats the baseline technique for non-spatial graphs by up to 3.61× even when no preprocessed information is available. Furthermore, we have shown that both BALT and OBALT beat an “off-the-shelf” solution to make the state-of-the-art approach applicable in non-spatial networks by a wide margin.

In the future, we want to investigate how the type and structure of the graph influences the performance of our proposed solutions. Furthermore, we plan to investigate the concurrent and distributed computation of batch shortest path queries. In addition, we intend to extend our solutions to support the processing of complex queries and analyses that rely on shortest path computations. Finally, we plan to study the batch processing of path queries under multiple optimization criteria, label and resource constraints.

ACKNOWLEDGMENTS

This work is supported by Grant No. **CH 2464/1-1** of the Deutsche Forschungsgemeinschaft (DFG). Map data copyrighted OpenStreetMap contributors and available from <https://openstreetmap.org>.

REFERENCES

- [1] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. 2016. *Route Planning in Transportation Networks*. Springer, 19–80. https://doi.org/10.1007/978-3-319-49487-6_2
- [2] Theodoros Chondrogiannis and Johann Gamper. 2016. ParDiSP: A Partition-based Framework for Distance and Shortest Path Queries on Road Networks. In *Proc. Intl. Conf. on Mobile Data Management (MDM)*. 242–251. <https://doi.org/10.1109/MDM.2016.45>
- [3] Liron Cohen, Tansel Uras, Shiva Jahangiri, Aliyah Arunasalam, Sven Koenig, and T. K. Satish Kumar. 2018. The FastMap Algorithm for Shortest Path Computations. In *Proc. Intl. Conf. on Artificial Intelligence (IJCAI)*. 1427–1433. <https://doi.org/10.24963/ijcai.2018/198>
- [4] Camil Demetrescu, Andrew V. Goldberg, and David Johnson. 2006. 9th DIMACS Implementation Challenge: Shortest Paths. <http://users.diag.uniroma1.it/challenge9/>. Accessed: 2021-03-05.
- [5] Edsger W. Dijkstra. 1959. A note on two problems in connexion with graphs. *Numer. Math.* 1, 1 (1959), 269–271. <https://doi.org/10.1007/BF01386390>
- [6] ECML/PKDD 2015 Competition. 2015. ECML/PKDD 15: Taxi Trajectory Prediction (I). <https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i>. Accessed: 2021-03-05.
- [7] Stefan Edelkamp and Stefan Schrödl. 2012. Chapter 2 – Basic Search Algorithms. In *Heuristic Search*, Stefan Edelkamp and Stefan Schrödl (Eds.). Morgan Kaufmann, 47–87. <https://doi.org/10.1016/B978-0-12-372512-7.00002-X>
- [8] Christos Faloutsos and King-Ip Lin. 1995. FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets. In *Proc. Intl. Conf. on Management of Data (SIGMOD)*. 163–174. <https://doi.org/10.1145/223784.223812>
- [9] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. 2008. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *Proc. Intl. Conf. on Experimental Algorithms (WEA)*. 319–333. https://doi.org/10.1007/978-3-540-68552-4_24
- [10] Andrew V. Goldberg and Chris Harrelson. 2005. Computing the Shortest Path: A* Search Meets Graph Theory. In *Proc. Symp. on Discrete Algorithms (SODA)*. 156–165.
- [11] Andrew V. Goldberg and Renato Werneck. 2005. Computing Point-to-Point Shortest Paths from External Memory. In *Proc. Workshop on Algorithm Engineering and Experiments (ALENEX)*. <https://doi.org/10.1137/1.9781611972863.13>
- [12] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern.* 4, 2 (1968), 100–107. <https://doi.org/10.1109/TSSC.1968.300136>
- [13] Manuel Hotz, Theodoros Chondrogiannis, Leonard Wörteler, and Michael Grossniklaus. 2019. Experiences with Implementing Landmark Embedding in Neo4j. In *Proc. Intl. Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*. 1–9. <https://doi.org/10.1145/3327964.3328496>
- [14] Kaggle. 2017. New York City Taxi Trip Duration. <https://www.kaggle.com/c/nyc-taxi-trip-duration>. Accessed: 2021-03-05.
- [15] George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.* 20, 1 (1998), 359–392. <https://doi.org/10.1137/S1064827595287997>
- [16] Sebastian Knopp, Peter Sanders, Dominik Schultes, Frank Schulz, and Dorothea Wagner. 2007. Computing Many-to-Many Shortest Paths Using Highway Hierarchies. In *Proc. Workshop on Algorithm Engineering and Experiments (ALENEX)*. 36–45. <https://doi.org/10.1137/1.9781611972870.4>
- [17] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. <http://snap.stanford.edu/data>.
- [18] Lei Li, Mengxuan Zhang, Wen Hua, and Xiaofang Zhou. 2020. Fast Query Decomposition for Batch Shortest Path Processing in Road Networks. In *Proc. Intl. Conf. on Data Engineering (ICDE)*. 1189–1200. <https://doi.org/10.1109/ICDE48307.2020.00107>
- [19] Hossain Mahmud, Ashfaq Mahmood Amin, Mohammed Eunus Ali, Tanzima Hashem, and Sarana Nutanong. 2013. A Group Based Approach for Path Queries in Road Networks. In *Proc. Intl. Symp. on Advances in Spatial and Temporal Databases (SSTD)*. 367–385. https://doi.org/10.1007/978-3-642-40235-7_21
- [20] Dimitrios Michail, Joris Kinable, Barak Naveh, and John V. Sichi. 2020. JGraphT – A Java Library for Graph Data Structures and Algorithms. *ACM Trans. Math. Softw.* 46, 2 (May 2020). <https://doi.org/10.1145/3381449>
- [21] OpenStreetMap contributors. 2021. Planet dump retrieved from <https://planet.osm.org>. <https://www.openstreetmap.org>.
- [22] Ira S. Pohl. 1969. *Bi-directional and Heuristic Search in Path Problems*. Ph.D. Dissertation. Stanford University, Stanford, CA, USA.
- [23] Michalis Potamias, Francesco Bonchi, Carlos Castillo, and Aristides Gionis. 2009. Fast Shortest Path Distance Estimation in Large Networks. In *Proc. Intl. Conf. on Information and Knowledge Management (CIKM)*. 867–876. <https://doi.org/10.1145/1645953.1646063>
- [24] Radi Muhammad Reza, Mohammed Eunus Ali, and Tanzima Hashem. 2015. Group Processing of Simultaneous Shortest Path Queries in Road Networks. In *Proc. Intl. Conf. on Mobile Data Management (MDM)*. 128–133. <https://doi.org/10.1109/MDM.2015.70>
- [25] Hanan Samet, Jagan Sankaranarayanan, and Houman Alborzi. 2008. Scalable Network Distance Browsing in Spatial Databases. In *Proc. Intl. Conf. on Management of Data (SIGMOD)*. 43–54. <https://doi.org/10.1145/1376616.1376623>
- [26] Peter Sanders and Dominik Schultes. 2005. Highway Hierarchies Hasten Exact Shortest Path Queries. In *Proc. European Conference on Algorithms (ESA)*. 568–579. https://doi.org/10.1007/11561071_51
- [27] Jagan Sankaranarayanan, Houman Alborzi, and Hanan Samet. 2005. Efficient Query Processing on Spatial Networks. In *Proc. Intl. Workshop on Geographic Information Systems (GIS)*. 200–209. <https://doi.org/10.1145/1097064.1097093>
- [28] Jagan Sankaranarayanan, Houman Alborzi, and Hanan Samet. 2006. Distance Join Queries on Spatial Networks. In *Proc. Intl. Symp. on Advances in Geographic Information Systems (GIS)*. 211–218. <https://doi.org/10.1145/1183471.1183506>
- [29] Jagan Sankaranarayanan, Hanan Samet, and Houman Alborzi. 2009. Path Oracles for Spatial Networks. *Proc. VLDB Endow.* 2, 1 (2009), 1210–1221. <https://doi.org/10.14778/1687627.1687763>
- [30] Christian Sommer. 2014. Shortest-Path Queries in Static Networks. *ACM Comput. Surv.* 46, 4 (March 2014). <https://doi.org/10.1145/2530531>
- [31] Jeppe Rishede Thomsen, Man Lung Yiu, and Christian S. Jensen. 2012. Effective Caching of Shortest Paths for Location-Based Services. In *Proc. Intl. Conf. on Management of Data (SIGMOD)*. 313–324. <https://doi.org/10.1145/2213836.2213872>
- [32] Qiongwen Xu, Xu Zhang, Jin Zhao, Xin Wang, and Tilman Wolf. 2016. Fast shortest-path queries on large-scale graphs. In *Proc. Intl. Conf. on Network Protocols (ICNP)*. 1–10. <https://doi.org/10.1109/ICNP.2016.7784419>
- [33] Mengxuan Zhang, Lei Li, Wen Hua, and Xiaofang Zhou. 2019. Batch Processing of Shortest Path Queries in Road Networks. In *Proc. Australasian Database Conference (ADC)*. 3–16. https://doi.org/10.1007/978-3-030-12079-5_1
- [34] Mengxuan Zhang, Lei Li, Wen Hua, and Xiaofang Zhou. 2019. Efficient Batch Processing of Shortest Path Queries in Road Networks. In *Proc. Intl. Conf. on Mobile Data Management (MDM)*. 100–105. <https://doi.org/10.1109/MDM.2019.00-69>