Fast Fully Dynamic Landmark-based Estimation of Shortest Path Distances in Very Large Graphs

Konstantin Tretyakov [*] [†]
University of Tartu
Estonia
kt@ut.ee

Abel Armas-Cervantes
University of Tartu
Estonia
abel.armas@ut.ee

Luciano García-Bañuelos
University of Tartu
Estonia
luciano.garcia@ut.ee

Jaak Vilo
University of Tartu
Estonia
jaak.vilo@ut.ee

Marlon Dumas
University of Tartu
Estonia
marlon.dumas@ut.ee

ABSTRACT

Computing the shortest path between a pair of vertices in a graph is a fundamental primitive in graph algorithmics. Classical exact methods for this problem do not scale up to contemporary, rapidly evolving social networks with hundreds of millions of users and billions of connections. A number of approximate methods have been proposed, including several landmark-based methods that have been shown to scale up to very large graphs with acceptable accuracy. This paper presents two improvements to existing landmark-based shortest path estimation methods. The first improvement relates to the use of shortest-path trees (SPTs). Together with appropriate short-cutting heuristics, the use of SPTs allows to achieve higher accuracy with acceptable time and memory overhead. Furthermore, SPTs can be maintained incrementally under edge insertions and deletions, which allows for a fully-dynamic algorithm. The second improvement is a new landmark selection strategy that seeks to maximize the coverage of all shortest paths by the selected landmarks. The improved method is evaluated on the DBLP, Orkut, Twitter and Skype social networks.

Categories and Subject Descriptors

E.1 [**Data**]: Data Structures—*Graphs and networks, Trees*; H.2.4 [**Database Management**]: Systems—*Query processing*

General Terms

Algorithms, Experimentation, Performance

[*]Corresponding author.

[†]All authors are also affiliated with Software Technology and Applications Competence Center (STACC), Estonia

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.

Keywords

Graph Databases, Shortest Paths, Social Networks, Landmarks, Trees, Dynamic Updates

1. INTRODUCTION

Finding the shortest path between a given pair of vertices in a graph is a fundamental primitive in graph algorithmics. For example, in the context of social network analysis, this primitive is useful for *socially-sensitive* search [18, 20], whereby a user of a social network searches for an individual by name and expects to see the results ranked in the order of their shortest path distance to him. Similarly, a user may wish to know what chain of contacts allows him to reach another user in the network.

There exists a large body of methods to address this problem [21]. Existing methods can be broadly classified into exact and approximate. Exact methods, such as those based on Dijkstra's traversal, are prohibitively slow for performing online queries on graphs with hundreds of millions of vertices, which is a typical size for a contemporary social network. Among the approximate methods, a family of scalable algorithms for this problem are the so-called *landmark-based* (or *sketch-based*) approaches [4, 5, 7, 9, 18]. In this family of techniques, a fixed set of landmark nodes is selected and distances or actual shortest paths are precomputed from each vertex to some or all of the landmarks. Knowledge of the distances to the landmarks, together with the triangle inequality, typically allows one to compute approximate distance between any two vertices in $O(k)$ time, where k is the number of landmarks, and $O(kn)$ space, where n is the number of vertices in the network. Those estimates can then be used as-is, or exploited further as a component of a graph traversal or routing strategy in order to obtain an exact shortest path [7, 10].

Although landmark-based algorithms do not provide strong theoretical guarantees on approximation quality [11], they have been shown to perform well in practice, scaling up to graphs with millions or even billions of edges with acceptable accuracy and response times of under one second per query [5, 9, 16].

In this paper we describe two improvements to existing landmark-based estimation methods for undirected graphs.

These improvements allow us to achieve higher accuracy than existing methods with millisecond-execution times on a graph with over three billion edges.

The first improvement derives from the use of *shortest path trees (SPTs)* to maintain the paths between each landmark and every vertex in the graph. Based on this data structure, we present three novel strategies for computing an approximate shortest path between any pair of nodes. Moreover, the use of SPTs makes the proposed method suitable for continuously evolving graphs, since practically efficient algorithms exist for incremental SPT update [3, 6]. For completeness' sake we explicitly present a version of the incremental update algorithm from [6], simplified to unweighted graphs.

The second improvement relates to the selection of landmarks. Specifically, we propose a greedy approach to select those landmarks that provide the best coverage of all shortest paths in a random sample of vertex pairs.

We evaluate the proposed method on four large social network, the largest of them (Skype) consisting of circa 500 million users and 3 billion connections. The evaluation demonstrates improved precision of distance estimates gained by both the landmark selection technique and the use of SPTs.

The rest of the paper is structured as follows. Section 2 introduces the notation used to present the algorithms, which are themselves given in Section 3. Section 4 summarizes the experimental evaluation results. Finally, Section 5 discusses related work, while Section 6 provides concluding remarks.

2. BASIC DEFINITIONS

Let $G = (V, E)$ denote a graph with $n = |V|$ vertices and $m = |E|$ edges. For simplicity of exposition, we shall consider an undirected, unweighted graph, although our approach can be easily generalized to accommodate weighted directed graphs as well.

A *path* $\pi_{s,t}$ of length ℓ between two vertices $s, t \in V$ is defined as a sequence $\pi_{s,t} = (s, u_1, u_2, \ldots, u_{\ell-1}, t)$, where $\{u_1, u_2, \ldots, u_{\ell-1}\} \subseteq V$ and $\{(s, u_1), (u_1, u_2), \ldots, (u_{\ell-1}, t)\} \subseteq E$. We denote the length ℓ of a path $\pi_{s,t}$ as $|\pi_{s,t}|$. The *concatenation* of two paths $\pi_{s,t} = (s, \ldots, t)$ and $\pi_{t,v} = (t, \ldots, v)$ is the combined path $\pi_{s,v} = \pi_{s,t} + \pi_{t,v} = (s, \ldots, t, \ldots, v)$.

The *distance* $d(s, t)$ between vertices s and t is defined as the length of the shortest path between s and t. The shortest path distance in a graph is a metric and satisfies the *triangle inequality*: for any $s, t, u \in V$

$$d(s, t) \leq d(s, u) + d(u, t)\,. \tag{1}$$

The upper bound becomes an equality iff there exists a shortest path $\pi_{s,t}$ which passes through u.

The *diameter* of a graph is the maximal length of a shortest path in the graph. In this work we rely on an important property of social networks – their diameter is small [19].

Centrality of a vertex is a general term used to refer to a number of metrics of importance of a vertex within a graph. *Betweenness centrality* corresponds to the mean proportion of shortest paths passing through a given vertex. *Closeness centrality* measures the average distance of a vertex to all other vertices in the graph. Computing centrality measures for all vertices exactly is typically hard (e.g., the best betweenness centrality algorithm so far requires $O(mn)$ time on unweighted graphs [2]), but can be approximated using random sampling [1].

Algorithm 1 LANDMARKS-BASIC

Require: Graph $G = (V, E)$, number of landmarks k.

```
1: procedure PRECOMPUTE
2:     U ← SELECT-LANDMARKS(G,k)        ▷ Algorithm 4
3:     for u ∈ U do
4:         Do a BFS traversal starting from u.
5:         For each v ∈ V store its distance from u as d_u[v].
6:     end for
7: end procedure
8: function LANDMARKS-BASIC(s, t)
9:     d_approx ← min_{u∈U}(d_u[s] + d_u[t])
10:    return d_approx
11: end function
```

3. ALGORITHM DESCRIPTION

3.1 Landmark-based distance estimation

As noted in Equation 1, if we fix a single landmark node u and precompute the distance $d(u, v)$ from this node to each other vertex v in the graph, we can get an upper bound approximation for the distance $d(s, t)$ between any two vertices s and t:

$$d^u_{\text{approx}}(s, t) = d(s, u) + d(u, t) \tag{2}$$

If we now select a set $U = \{u_1, u_2, \ldots, u_k\}$ of k landmarks, a potentially better approximation can be computed:

$$d^U_{\text{approx}}(s, t) = \min_{u \in U} d^u_{\text{approx}}(s, t) \tag{3}$$

In principle, the triangle inequality also allows to compute a lower bound on the distance, but previous work [16] indicates that lower-bound estimates are not as accurate as the upper-bound ones.

In the following we refer to this algorithm as LANDMARKS-BASIC (Algorithm 1). For unweighted graph, the algorithm requires $O(km)$ time to precompute distances using k BFS traversals and $O(kn)$ space to store the distances. Each query is answered in $O(k)$ time. Note that this approach only allows us to compute an approximate distance, but does not provide a way to retrieve the path itself.

3.2 Improved landmark-based algorithms

3.2.1 Shortest path trees

The central contribution of this paper is the idea of maintaining an explicit shortest path tree (SPT) for each landmark, instead of simply storing the distances to landmarks as numbers. More precisely, let $p_u[v]$ be the next vertex on an arbitrarily chosen shortest path from a vertex v to a landmark u. We shall refer to $p_u[v]$ as the *parent link* of v in the SPT of u (Figure 1). Similarly to distances, parent links can be computed in a straightforward manner during a BFS traversal of the graph in $O(m)$ time per landmark.

The availability of parent links enables us to recover an exact shortest path from each vertex v to each landmark u by simply following the corresponding links. Consequently, it also allows to compute the shortest path distance $d(u, v)$ and thus directly apply the ideas of the LANDMARKS-BASIC algorithm, with the only difference that each distance computation now requires $O(D)$ steps, where D is the diameter of the graph. As the diameters of social network graphs tend

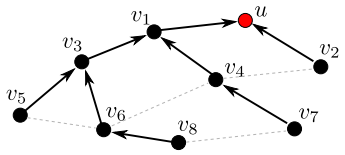

Figure 1: Shortest path tree for landmark u. Black arrows denote parent links. Dashed lines are graph edges that are not part of the tree.

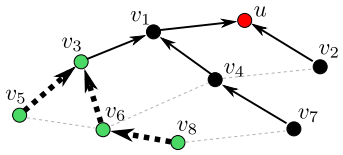

Figure 2: Lowest common ancestor technique. When approximating distance between v_5 and v_8 we use their lowest common ancestor v_3 instead of the landmark u as a reference.

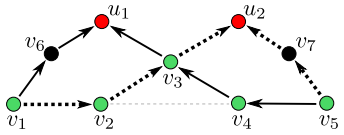

Figure 3: Landmark-BFS technique. Here, solid arrows belong to the SPT of u_1, and dotted arrows belong to the SPT of u_2.

to be small, the expected overhead of such a computation is minor.

Note that this approach allows to retrieve an actual path between any two vertices in addition to the distance approximation.

3.2.2 Lowest common ancestor method

Besides performing basic landmark-based approximation, the availability of the SPT allows us to significantly improve the upper bound estimates on distances for many vertex pairs. Consider the situation depicted in Figure 2 and suppose we wish to approximate the distance between v_5 and v_8. By applying the basic technique we obtain an upper bound estimate of $d(v_5, v_8) \leq d(v_5, u) + d(u, v_8) = 3 + 4 = 7$. Observe, however, that once we have the explicit shortest paths:

$$\pi_{v_5,u} = (v_5, v_3, v_1, u), \qquad \pi_{v_8,u} = (v_8, v_6, v_3, v_1, u)\,,$$

we can note that both of them pass through v_3 and thus the estimate

$$d(v_5, v_8) \leq d(v_5, v_3) + d(v_3, v_8) = 1 + 2 = 3\,.$$

will result in a better upper bound. In general, whenever two shortest paths $\pi_{s,u}$ and $\pi_{t,u}$ have a common vertex $v \neq u$, we have

$$d(s,t) \leq d(s,v) + d(v,t) < d(s,u) + d(u,t)\,, \qquad (4)$$

and thus if we use v instead of u to approximate $d(s,t)$ we obtain a tighter bound. Naturally, it makes sense to select the vertex v providing the best such approximation. It can be easily seen that this vertex is the *lowest common ancestor (LCA)* of s and t in the SPT of u.

This observation provides the basis of our LCA approximation method (Distance-LCA, Algorithm 2). By substituting this distance estimate into Equation (3), we obtain a new algorithm Landmarks-LCA with increased accuracy. Note that this algorithm can also be trivially extended to return the actual path.

One way to understand the extent of improvement is to note that the basic algorithm will provide exact estimates only for shortest paths that pass through the landmark vertex. In Figure 2 those are only the paths connecting v_2 with v_1 and v_3. The LCA algorithm, however, will provide exact answers for all shortest paths that lie along the SPT, and there will be typically considerably more of those.

3.2.3 Shortcutting

Denote the lowest common ancestor of s and t by v. The LCA algorithm approximates $\pi_{s,t}$ by a concatenation of $\pi_{s,v}$ with $\pi_{v,t}$. However, it may happen that a vertex $w \in \pi_{s,v}$ is connected directly by an edge with a vertex $w' \in \pi_{v,t}$. In this case, an even shorter approximation to $\pi_{s,t}$ can be obtained by concatenating the paths $\pi_{s,w}$, $\pi_{w,w'}$ and $\pi_{w',t}$. For example, in Figure 2, the edge (v_5, v_6) acts as a shortcut from π_{v_5,v_3} to π_{v_3,v_8}. If we account for this edge, we may further improve the LCA distance estimate to the true shortest path $d(v_5, v_8) = 2$.

In order to locate shortcuts we can simply examine all pairs of vertices in $\pi_{s,v} \times \pi_{t,v}$ and if some of them are connected by an edge, find the edge providing the best distance estimate. This can be done in $|\pi_{s,v}| \times |\pi_{t,v}|$, i.e., at most $O(D^2)$ steps. We refer to the resulting distance approximation method as Distance-SC (Algorithm 2). By using this upper bound estimate in Equation (3) we obtain the landmark-based algorithm Landmarks-SC.

3.2.4 Landmarks-BFS

The algorithms Landmarks-Basic, Landmarks-LCA, and Landmarks-SC use each landmark for distance approximation independently of the other landmarks. This is not the best possible use of all available landmark data. Consider Figure 3, for example. When approximating distance between vertices v_1 and v_5, we would obtain a path of length 5 if we used the two landmarks independently. By combining the two subtrees we can find a better path of length 4. Finally, by taking other graph edges (i.e., "shortcuts") into account, we further improve the distance approximation to 3.

This suggests a very simple, yet powerful improvement over the previous approaches. In order to approximate distance between two vertices, collect all paths from those vertices to all landmarks, and perform a usual BFS (or, in the case of weighted graphs, Dijkstra) traversal on the subgraph, induced by the union of those paths. We refer to this algorithm as Landmarks-BFS (Algorithm 3).

For k landmarks, the size of the subgraph will be less than $2kD$. Consequently, the memory complexity of Landmarks-BFS is $O(kD)$ and the time complexity is at most $O(k^2D^2)$.

3.3 Landmark selection techniques

Although landmarks can be selected uniformly at random, experiments of Potamias *et al.* [16] show that selecting land-

Algorithm 2 LCA-based upper bounding algorithms

Require: Graph $G = (V, E)$, a landmark $u \in V$, a parent link $p_u[v]$ precomputed for each $v \in V$.

function PATH-TO$_u$(s,π)
  Returns the path in the SPT p_u from the vertex s to the closest vertex q belonging to the path π
  Result $\leftarrow (s)$ ▷ Sequence of 1 element.
  while $s \notin \pi$ **do**
    $s \leftarrow p_u[s]$
    Append s to Result
  end while
  return Result ▷ $(s, p_u[s], p_u[p_u[s]], \ldots, q), q \in \pi$
end function

function DISTANCE-LCA$_u$(s,t)
  $\pi^{(1)} \leftarrow$ PATH-TO$_u(s, (u))$
  $\pi^{(2)} \leftarrow$ PATH-TO$_u(t, \pi^{(1)})$
  LCA $\leftarrow$ Last element of $\pi^{(2)}$
  $\pi^{(3)} \leftarrow$ PATH-TO$_u(s, (\text{LCA}))$
  return $|\pi^{(2)}| + |\pi^{(3)}|$
end function

function DISTANCE-SC$_u$(s,t)
  $\pi^{(1)} \leftarrow$ PATH-TO$_u(s, (u))$
  $\pi^{(2)} \leftarrow$ PATH-TO$_u(t, \pi^{(1)})$
  LCA $\leftarrow$ Last element of $\pi^{(2)}$
  $\pi^{(3)} \leftarrow$ PATH-TO$_u(s, (\text{LCA}))$
  Best $\leftarrow |\pi^{(2)}| + |\pi^{(3)}|$
  for $(w, w') \in (\pi^{(2)} \times \pi^{(3)}) \cap E$ **do**
    Current $\leftarrow |\pi_{s,w}| + |\pi_{w,w'}| + |\pi_{w',t}|$
    Best $\leftarrow$ min(Current,Best)
  end for
  return Best
end function

Algorithm 3 LANDMARKS-BFS

Require: Graph $G = (V, E)$, a set of landmarks $U \subset V$, an SPT parent link $p_u[v]$ precomputed for each $u \in U, v \in V$.

function LANDMARKS-BFS(s,t)
  $S \leftarrow \emptyset$
  for $u \in U$ **do**
    $S \leftarrow S \cup$ PATH-TO$_u(s, (u))$ ▷ (see Algorithm 2)
    $S \leftarrow S \cup$ PATH-TO$_u(t, (u))$
  end for
  Let $G[S]$ be the subgraph of G induced by S.
  Apply BFS on $G[S]$ to find a path π from s to t.
  return $|\pi|$
end function

Algorithm 4 SELECT-LANDMARKS

Require: Graph $G = (V, E)$, number of landmarks k, sample size M.

function HIGHEST-DEGREE
  For each $v \in V$ let $d[v] \leftarrow$ DEGREE(v).
  Sort V by $d[v]$.
  Let $v_{(i)}$ denote the vertex with i-th largest $d[v]$.
  return $\{v_{(1)}, v_{(2)}, \ldots, v_{(k)}\}$
end function

function BEST-COVERAGE
  $P \leftarrow \emptyset$
  for $i \in \{1, \ldots, M\}$ **do**
    $(s_i, t_i) \leftarrow$ Random pair of vertices
    $p_i \leftarrow$ SHORTESTPATH(s_i, t_i)
    $P \leftarrow P \cup \{p_i\}$
  end for
  $V_P \leftarrow \cup_{p \in P} p$
  for $i \in \{1, \ldots, k\}$ **do**
    For each $v \in V_P$ let $c[v] \leftarrow |\{p \in P : v \in p\}|$
    $u_i \leftarrow \text{argmax}_{v \in V_P} c[v]$ ▷ Resolve ties arbitrarily
    $P \leftarrow P \setminus \{u_i\}$
  end for
  return $\{u_1, \ldots, u_k\}$
end function

marks with the highest *degree* or lowest *closeness centrality* typically ensures better distance estimates, whereas it is shown that the two methods provide similar accuracy. In our experiments we use the highest degree landmark selection method and compare it with the following one.

Best coverage. When a landmark u lies on the shortest path between s and t, its upper bound distance estimate is exact. We say that such a landmark *covers* the pair (s, t). Consequently, the most desirable set of landmarks would be the one that covers as many vertex pairs as possible. It can be shown that the task of finding the *optimal* landmark set (i.e., the one with the maximal coverage) is NP-hard [16]. Instead, we propose a simple greedy strategy based on sampling (Algorithm 4). We sample a set of M vertex pairs, and compute exact shortest path for each pair. As the first landmark we select the vertex that is present in most paths of the sample. We remove the paths covered by that first landmark from the sample and proceed to select the second landmark as the vertex, which covers most of the remaining paths, etc.

3.4 Incremental updates

If the graph is subject to intensive edge insertion and deletion, landmarks that have been computed originally become outdated and the approximation performance deteriorates. Therefore, landmarks have to be maintained up to date. Although this can be achieved by means of daily or hourly full recomputation, such a solution is computationally expensive. Moreover, for certain applications, such as the above-mentioned social search, it can be particularly important to maintain landmarks up to date at all times. Indeed, if the social search feature is relied upon by new users to establish their first list of contacts, it is important that adding a new contact would be immediately reflected in the consequent search orderings.

Fortunately, when landmarks are maintained in the form of shortest path trees, they can be updated incrementally to accommodate edge insertions or deletions. The procedures for maintaining SPTs under insertions and deletions are well-known [3, 6]. In the particular case of unweighted graphs with a small diameter, they are fairly straightforward.

As an informal example, consider the SPT presented on Figure 1. Suppose that an edge $\{u, v_8\}$ has just been inserted

Algorithm 5 INSERT-EDGE

Require: Graph $G = (V, E)$, a landmark $u \in V$, a parent link $p_u[v]$ precomputed for each $v \in V$.

```
1: function INSERT-EDGE_u(s,t)
2:     if DISTTO_u(s) > DISTTO_u(t) then
3:         (s,t) = (t,s)
4:     end if
5:     DistChanged ← FIFOQueue()
6:     UPDATEIFBETTERPARENT_u(s, t,DistChanged)
7:     while not DistChanged.empty() do
8:         s ← DistChanged.pop()
9:         for t ∈Neighbors(s) do
10:            UPDATEIFBETTERPARENT_u(s, t,DistChanged)
11:        end for
12:    end while
13: end function
```

Helper routines

```
14: function DISTTO_u(s)
        Distance to u in the current SPT
        Note that we compute it in O(D) steps.
15:    d ← 0
16:    while p_u[s] ≠ u do
17:        s ← p_u[s]
18:        d ← d + 1
19:        if s ==null then
20:            return ∞                ▷ used in DELETE-EDGE
21:        end if
22:    end while
23:    return d
24: end function

25: function UPDATEIFBETTERPARENT_u(s,t,queue)
        Checks whether s is a better parent for t in the SPT.
        If so, updates p_u and pushes t to the queue.
26:    if (p_u[t] ≠ s)
27:        and (DISTTO_u(t) > DISTTO_u(s) + 1) then
28:        p_u[t] ← s
29:        queue.push(t)
30:    end if
31: end function
```

Algorithm 6 DELETE-EDGE

Require: Graph $G = (V, E)$, a landmark $u \in V$, a parent link $p_u[v]$ precomputed for each $v \in V$.

```
1: function DELETE-EDGE_u(s,t)
2:     if p_u[s] == t then
3:         (s,t) = (t,s)
4:     else if p_u[t] ≠ s then
5:         return                      ▷ SPT not affected
6:     end if
7:     prevDist ← DISTTO(t)
8:     p_u[t] ← null
9:     DistChanged ← PriorityQueue()
10:    PUSHIFDISTCHANGED_u(t, prevDist, DistChanged)
11:    while not DistChanged.empty() do
12:        t ← DistChanged.pop()
13:        Find the neighbor s of t
14:            with minimal DISTTO_u(s)
15:        p_u[t] ← s
16:    end while
17: end function
```

Helper routines

```
18: function PUSHIFDISTCHANGED_u(t, prevDist, pqueue)
        Checks whether there exists a new parent for t with
        which its DISTTO will stay unchanged.
        If so, updates p_u. Otherwise, pushes t to queue and
        descends into leaves.
19:    Find the neighbor s of t
20:        with minimal DISTTO_u(s)
21:    For all neighbors of t that are in the pqueue,
22:        and have their key >DISTTO_u(s)+2,
23:        update their key to DISTTO_u(s)+2.
24:    if DISTTO_u(s) == prevDist-1 then
25:        p_u[t] ← s
26:    else
27:        pqueue.push(t, key=DISTTO_u(s)+1)
28:        for v ∈ children of t in the SPT do
29:            p_u[v] ← null
30:            PUSHIFDISTCHANGED_u(v,prevDist+1,pqueue)
31:        end for
32:    end if
33: end function
```

into the graph. The SPT update algorithm would proceed as follows. Firstly, note that the newly added edge provides a shorter path from v_8 to the landmark than what was previously available. Therefore, the parent pointer of v_8 has to be changed to make use of the new edge: $p_u[v_8] := u$. Now that the path to the landmark from v_8 has improved, we have to recursively examine all neighbors of v_8 (i.e., v_6 and v_7) and check, whether switching their parent pointer to v_8 would improve their previously known path to the landmark. This is true both for v_6 and v_7, hence we set $p_u[v_6] := v_8$, $p_u[v_7] := v_8$. We repeat this again for all neighbors of v_6 and v_7. Having found no new path improvements, we complete the update.

The deletion of an edge involves two passes. Consider again Figure 1 and suppose the edge $\{v_1, v_3\}$ was deleted from the graph. In order to find a new path to the landmark (and a new parent pointer) for v_3, we first examine its neighbors (v_5 and v_6). Unfortunately, both of them relied on v_3 for reaching the landmark, hence they provide no immediate fix. We record v_3 temporarily in a priority queue, using the best available new path length (∞ so far) as the key. We then recursively descend to process the children of v_3 in the SPT. Vertex v_5 has no immediate fix and gets recorded in the priority queue with key ∞. Vertex v_6, however, can be connected to v_4, retaining a path to the landmark of length 3. Consequently, there is no need to process children of v_6. After reconnecting v_6 we must update the keys of its neighbors (v_3 and v_5) in the priority queue – the new potential path of length 4 is better than the previously recorded ∞. This completes the first pass. In the second pass we empty the priority queue, rebuilding the rest of the SPT.

We provide a more formal description of the update procedures in Algorithms 5 and 6, and refer the reader to [6] for detailed explanations.

In theory, a single update may trigger the SPT recompu-

tation for the whole graph (e.g. deleting an edge that was a bridge between the landmark and all the other nodes). In practice, however, such situations are rare and, according to our experiments, the amortized time necessary to process a single update in a real Skype network is in the order of milliseconds (see Section 4.3).

4. EXPERIMENTAL EVALUATION

4.1 Datasets

We tested our approach on four real-world social network graphs, representing four different orders of magnitude in terms of network size.

- **DBLP.** The DBLP dataset contains bibliographic information of computer science publications [13]. Every vertex corresponds to an author. Two authors are connected by an edge if they have co-authored at least one publication. The snapshot from May 15, 2010 was used.
- **Orkut.** Orkut is a large social networking website. It is a graph, where each user corresponds to a vertex and each user-to-user connection is an edge. The snapshot of the Orkut network was published by Mislove *et al.* in 2007 [14].
- **Twitter.** Twitter is a microblogging site, which allows users to *follow* each other, thus forming a network. A snapshot of the Twitter network was published by Kwak *et al.* in 2010 [12]. Although originally the network is directed, in our experiments we ignore edge direction.
- **Skype.** Skype is a large social network for peer-to-peer communication. We say that two users are connected by an edge if they are in each other's contact list. A subset of the network comprising 454M users was obtained in February 2010.

The properties of these datasets are summarized in Table 1. The table shows the number of vertices $|V|$, number of edges $|E|$, average distance between vertices $\overline{d}$ (computed on a sample vertex pairs), approximate diameter Δ, fraction of vertices in the largest connected component $|S|/|V|$, and average time to perform a BFS traversal over the graph t_{BFS}.

Dataset	$\|V\|$	$\|E\|$	$\overline{d}$	Δ	$\|S\|/\|V\|$	t_{BFS}
DBLP	770K	2.6M	6.3	25	85%	345 ms
Orkut	3.1M	117M	5.7	10	100%	8 sec
Twitter	41.7M	1.2B	4.2	25	100%	9 min
Skype	454M	3.1B	6.7	60	85%	20 min

Table 1: Datasets.

4.2 Experimental setup

To evaluate the performance of our methods we first select a random sample of 500 vertex pairs from each graph and precompute the actual shortest path distance for every pair. We then apply the distance approximation techniques to those vertex pairs and measure their *Approximation error*, *Time*, and *Disk space*.

Approximation error is computed as $(\ell' - \ell)/\ell$, where ℓ' is the approximation and ℓ is the actual distance. *Time* refers to the average time required to compute a distance approximation for a pair of vertices. Finally, the *Disk space* corresponds to the amount of data stored for each landmark.

We run our measurements on a server with 32× Quad-core AMD Opteron 64bit 2.2GHz processors, 256G of RAM, accessing IBM DS 3400 FC SAN disk array, running Red Hat Enterprise Linux 5 operating system. We used C++ for algorithm implementation.

4.3 Results

4.3.1 Approximation Error

The approximation quality measurements are summarized in Figure 4 and Table 3. Observe that landmark selection strategy has a significant effect on approximation quality for the Landmarks-Basic and the Landmarks-LCA algorithms. The use of shortcutting in the Landmarks-SC and Landmarks-BFS seems to reduce the importance of landmark selection. Indeed, in case of Landmarks-BFS randomly selected landmarks often outperform the more sophisticated techniques. Overall, the *best coverage* and *highest degree* techniques perform nearly equally well, and depending on the particular graph either one or the other should be preferred to achieve the best possible results.

Note that the use of the *LCA* approach reduces the error of the basic technique consistently by 10–50%, with further slight improvements obtained by shortcutting and BFS methods.

4.3.2 Query Time and Disk Usage

The disk usage in all the discussed methods is linear with respect to number of vertices, because we have to store a single value for each vertex-landmark pair. In the basic method this value is a distance and can be stored in a single byte, as distances in the studied social network graphs are small. When using SPTs, a vertex pointer needs to be stored for each vertex-landmark pair. For a graph as large as Skype, a 4-byte pointer is necessary. Consequently, the per-landmark disk space is 4 times greater when using SPTs than when storing distances only (see Table 2).

Dataset	Graph file	Landmark file	
		Basic	LCA/SC/LBFS
DBLP	27M	753K	3.0M
Orkut	918M	3.0M	12.0M
Twitter	9.3G	40M	160M
Skype	27G	433M	1.7G

Table 2: Disk usage.

Table 4 reports the average execution time per query. This is computed as the total time required process 500 random queries sequentially divided by 500. This also includes the time spent to load precomputed data from disk on-the-fly using Linux `mmap` facilities. We observe that the LCA and SC methods are 3-6 times slower than the basic algorithm, but the execution times still remain in the order of a few milliseconds even for the Skype graph. The Landmarks-BFS method can be considerably slower, especially so for the Twitter graph, due to the presence of nodes with very high degree in this network.

Dataset	Landmark selection	Method			
		Basic	LCA	SC	LBFS
DBLP	Random	0.46	0.07	0.04	0.02
	Highest Degree	0.11	0.05	0.03	0.03
	Best Coverage	0.07	0.04	0.03	0.02
Orkut	Random	0.50	0.13	0.08	0.04
	Highest Degree	0.14	0.12	0.10	0.07
	Best Coverage	0.13	0.11	0.08	0.05
Twitter	Random	0.53	0.07	0.02	0.02
	Highest Degree	0.06	0.05	0.03	0.03
	Best Coverage	0.07	0.06	0.03	0.02
Skype	Random	0.56	0.25	0.23	0.21
	Highest Degree	0.21	0.19	0.18	0.16
	Best Coverage	0.19	0.17	0.16	0.15

Table 3: Approximation error for different methods (using 100 landmarks).

Dataset	Method	No. of Landmarks		
		20	60	100
DBLP	Basic	0.03	0.09	0.14
	LCA	0.10	0.29	0.46
	SC	0.13	0.47	0.78
	LBFS	0.45	1.00	1.42
Orkut	Basic	0.08	0.11	0.39
	LCA	0.31	0.78	1.10
	SC	0.40	0.95	1.26
	LBFS	1.87	8.05	12.49
Twitter	Basic	0.15	0.49	0.84
	LCA	0.72	0.81	1.21
	SC	0.82	0.99	1.87
	LBFS	240	633	889
Skype	Basic	0.18	0.56	0.91
	LCA	1.06	2.43	3.69
	SC	1.22	2.92	4.85
	LBFS	5.10	13.24	16.25

Table 4: Query execution time averaged across 500 queries (in ms).

4.3.3 Precomputation Time

Precomputation time is comprised of two parts – landmark selection and the computation of SPTs (or distances, in the case of the basic method) for each landmark.

- **Landmark selection.** In the case of *random* landmark selection, the time is negligible. The *highest degree* method requires a single pass through the set of vertices of the graph, which is dominated by the time needed to read the graph data from disk. The most complex method is *best coverage*, which requires a sampling of pairs of vertices and the precomputations of their exact shortest paths (e.g. using BFS). Table 5 summarizes timing measurements obtained with a naïve sequential implementation (i.e. shortest paths were computed one after the other using BFS).
- **SPT computation per landmark.** The time needed to compute each landmark is equal to the time needed to perform a full BFS over the graph and is given in Table 1. For the DBLP graph each landmark can be computed in about 500ms, while for the Skype graph this time is approximately 20 minutes. Note that this operation can be trivially parallelized by assigning one thread per landmark.

Dataset	Highest degree	Best coverage
DBLP	140 ms	2 min
Orkut	2 s	15 min
Twitter	22 s	15 h
Skype	1 min	54 h

Table 5: Landmark selection timings.

4.3.4 Incremental Updates

We tested the performance of the incremental update procedures on our graphs by simulating a random sequence of 1000 edge modifications and using the landmark update procedures to maintain 100 landmarks up to date. Each edge modification was randomly chosen to be a deletion or an insertion of an edge between two random vertices. Table 6 presents the average time necessary to update one landmark.

	DBLP	Orkut	Twitter	Skype
Insertion	1μs	10μs	10μs	30μs
Deletion	100μs	2ms	12ms	11ms

Table 6: Average time to perform one update for one landmark.

5. RELATED WORK

The task of finding a shortest path between an arbitrary pair of vertices is known as a *point-to-point shortest path (PPSP)* problem. The naïve solution is to run the Dijkstra's traversal from the source to the destination vertex on every query. An improvement to this can be obtained by running a *bi-directional* search [15]. A further improvement is obtained by exploiting the A* search with landmark-based heuristics [7, 8, 10].

Sometimes it makes sense to precompute all pairwise distances. The corresponding algorithms are typically referred to as *all pairs shortest paths (APSP)*. Although a variety of elegant techniques have been proposed for this task, such as dynamic programming, path-forming and Boolean matrix multiplications, most known exact APSP algorithms are about $O(n^3)$ in time, with a few subcubic solutions for cer-

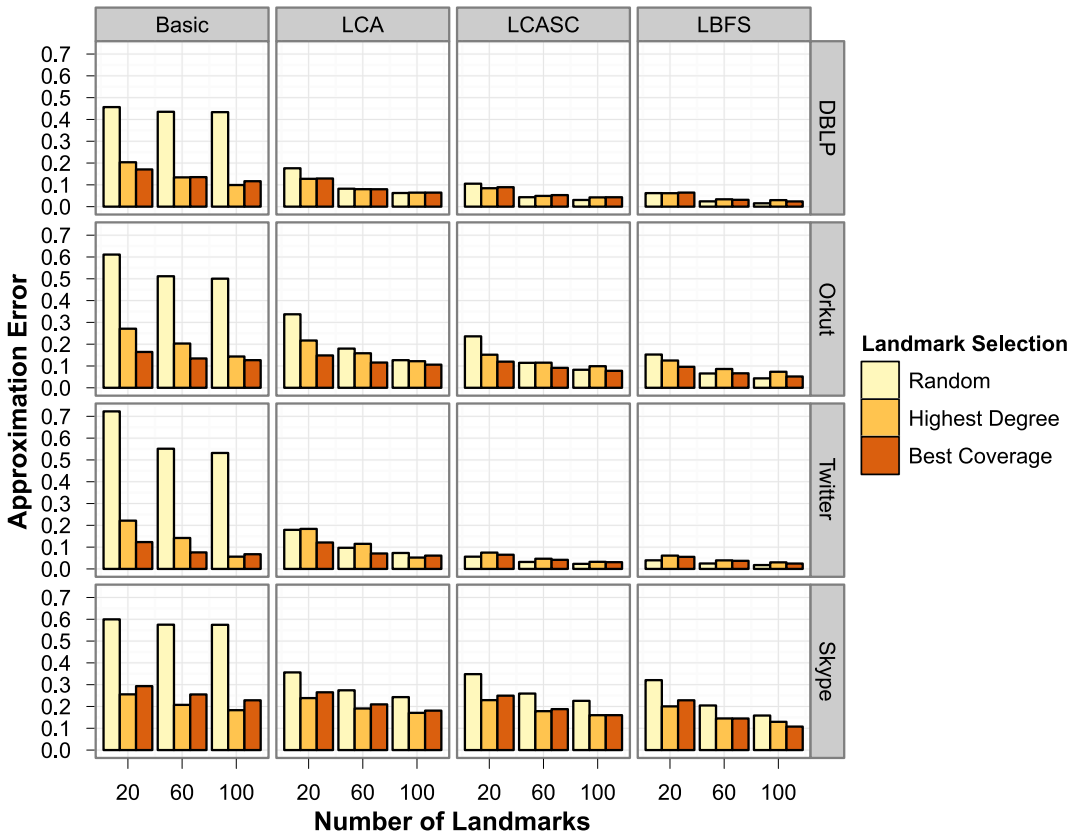

Figure 4: Approximation error results. The best accuracy is consistently obtained using *Landmarks-BFS* distance estimation method.

tain types of graphs [21]. This is not significantly better than simply performing a separate SSSP run for each vertex.

For many practical applications, finding out approximate distances between vertices can be sufficient. Careful indexing of the graph can provide a way to later provide approximate answers to arbitrary distance queries in constant or logarithmic time. Thorup and Zwick [17] coin the term *approximate distance oracles* for such algorithms. They present a method which requires $O(kn^{1+1/k})$ space and $O(kmn^{1/k})$ expected time for preprocessing and allows constant $O(k)$ query time. The idea of their method is to compute a sequence of nested sets of randomly picked landmarks. The closest landmark from each set is located and stored for each vertex together with the distance to it. A distance query $d(s,t)$ can then be answered by finding a landmark u which is one of the closest to both s and t and using the approximation given by Equation (2). The authors prove that their algorithm provides distance estimates which can be wrong up to a factor of $2k-1$.

An interesting analogy of our method to this algorithm can be observed. Indeed, let u be a single landmark. Consider the collection of nested sets $A_0 \subseteq A_1 \cdots \subseteq A_D$, where $A_i = \{v \in V\ ,\ d(u,v) \leq i\}$, and D is the diameter of the graph. Then the complete shortest path $\pi_{s,u}$ corresponds to the *bunch* concept from Thorup's algorithm, and the process of finding the LCA is similar to the distance estimation technique used in that paper. However, the strategy of selecting the nested sets A_i is completely different. Besides, the analogy does not hold as soon as more than one landmark is used in our algorithm.

In general, the idea of landmark-based estimation has been widely studied in previous works (see, e.g., [4, 5, 7, 9, 16, 18]). A closely related work is that of Potamias *et al.* [16], where the Landmarks-Basic algorithm is evaluated under different landmark selection strategies. The major innovation of our work is the use of SPTs instead just keeping the distance from each landmark to every vertex.

The algorithms Landmarks-LCA, Landmarks-SC and Landmarks-BFS are also similar to those suggested by Gubichev *et al.* [9]. However, the algorithms in [9] use different sets of landmarks for each vertex and thus store complete shortest paths to each landmark at each vertex. This results in higher memory requirements ($O(Dkn)$ instead of $O(kn)$ for storing only a single pointer per vertex-landmark pair in our approach) and prevents the possibility of performing incremental updates. The execution times reported in [9] are considerably slower when compared to ours – more than 4 seconds on a graph with 10 times less edges and 100 times less vertices than the Skype graph.

An important aspect of landmark-based methods is the way landmarks are selected – a careful selection strategy can have a significant positive effect. Strategies, which rely on selecting landmarks with high *degree*, *betweenness*- and *closeness centrality* as well as ensuring proper *dispersion* of landmarks over the graph and its parts, have been suggested, with *highest degree* and *closeness centrality* being shown to

typically yield highest accuracy in [16]. In our work, we proposed and evaluated an additional method that can compete with the *highest degree* approach, outperforming it slightly on some datasets. The combined effect of our improvements (*best coverage* landmark selection with LANDMARKS-BFS algorithm) for the large Skype network is a 40% smaller error.

6. CONCLUSION

This paper described and evaluated two improvements to existing approaches for landmark-based estimation of shortest paths. These improvements strike a tradeoff between accuracy, query execution time and disk usage for precomputed data. With respect to previous related work, we achieve notable accuracy improvements while maintaining the response time per query within a few milliseconds – even for a graph with billions of edges – and a space consumption comparable to previous state of the art methods. An exclusive property of the proposed methods is the support for dynamic updates.

Several extensions to the proposed method are possible.

Generalization to directed weighted graphs. We have presented the algorithms for the case of an undirected unweighted graph. The generalization to weighted graphs is obtained by replacing the BFS in the SPT precomputation phase and in the LANDMARK-BFS algorithm with a Dijkstra traversal. The generalization to directed graphs requires computing two shortest path trees for each landmark – the first one holding distances *to* the landmark, and the second one with distances *from* the landmark. The algorithms then need to be updated slightly to use both trees appropriately (e.g. lines 4 and 5 of Algorithm 3 will refer to two different trees rather than one).

Exact distance estimation. Being a distance approximation scheme, a landmark-based algorithm can be used as a heuristic in the (unidirectional or bidirectional) A* search, as described in [7, 10]. In particular, this ability to efficiently estimate exact shortest paths allows us to take larger samples for *best coverage* landmark selection. Note that due to the incremental update capabilities of our approach, the result is a fast fully-dynamic exact shortest path algorithm.

Evolutionary landmark selection. In the presented approach, all landmarks are selected *ex ante* and the selection of landmarks is never revised. A further improvement might be obtained by using information collected during the processing of queries in order to add or remove landmarks. Each time a query is answered, we can identify which vertices are used in the shortest path. Based on this information we could promote certain vertices, which lie on shortest paths frequently, to become landmarks, or we could drop landmarks that are infrequently used.

7. ACKNOWLEDGMENTS

The authors gratefully acknowledge the contributions of Andres Kütt and André Karpištšenko from Skype Technologies. This research is funded by the ERDF via the Estonian Software Technology and Applications Competence Center.

8. REFERENCES

[1] D. A. Bader, S. Kintali, K. Madduri, and M. Mihail. Approximating betweenness centrality. In *Proceedings of the 5th international conference on Algorithms and models for the web-graph*, WAW'07, pages 124–137, Berlin, Heidelberg, 2007. Springer-Verlag.

[2] U. Brandes. A faster algorithm for betweenness centrality, 2001.

[3] E. P. F. Chan and Y. Yang. Shortest path tree computation in dynamic graphs. *IEEE Trans. Comput.*, 58(4):541–557, 2009.

[4] L. J. Cowen and C. G. Wagner. Compact roundtrip routing in directed networks. *Journal of Algorithms*, 50(1):79 – 95, 2004.

[5] A. Das Sarma, S. Gollapudi, M. Najork, and R. Panigrahy. A sketch-based distance oracle for web-scale graphs. In *Proceedings of the third ACM international conference on Web search and data mining*, WSDM '10, pages 401–410, New York, NY, USA, 2010. ACM.

[6] D. Frigioni. Fully dynamic algorithms for maintaining shortest path trees, 2000.

[7] A. V. Goldberg and C. Harrelson. Computing the shortest path: A* search meets graph theory. In *Proc. 16th ACM-SIAM Symposium on Discrete Algorithms*, pages 156–165, 2005.

[8] A. V. Goldberg, H. Kaplan, and R. F. Werneck. Abstract reach for A*: Efficient point-to-point shortest path algorithms, 2006.

[9] A. Gubichev, S. J. Bedathur, S. Seufert, and G. Weikum. Fast and accurate estimation of shortest paths in large graphs. In *CIKM '10: Proceeding of the 19th ACM conference on Information and knowledge management*, pages 499–508. ACM, 2010.

[10] T. Ikeda, M.-Y. Hsu, H. Imai, S. Nishimura, H. Shimoura, T. Hashimoto, K. Tenmoku, and K. Mitoh. A fast algorithm for finding better routes by ai search techniques. In *Proc. Vehicle Navigation and Information Systems Conf.*, pages 291–296, 1994.

[11] J. Kleinberg, A. Slivkins, and T. Wexler. Triangulation and embedding using small sets of beacons. In *Proc. 45th Annual IEEE Symp. Foundations of Computer Science*, pages 444–453, 2004.

[12] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? In *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 591–600, New York, NY, USA, 2010. ACM.

[13] M. Ley and P. Reuther. Maintaining an online bibliographical database: the problem of data quality. in egc, ser. revue des nouvelles technologies de l' information, vol. rnti-e-6. *Cépadués Éditions*, 2006:5–10, 2006.

[14] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and Analysis of Online Social Networks. In *Proceedings of the 5th ACM/Usenix Internet Measurement Conference (IMC'07)*, San Diego, CA, October 2007.

[15] I. Pohl. Bi-directional search. In D. Meltzer, Bernard; Michie, editor, *Machine Intelligence*. Edinburgh University Press, 1971.

[16] M. Potamias, F. Bonchi, C. Castillo, and A. Gionis.

Fast shortest path distance estimation in large networks. In *CIKM '09: Proceeding of the 18th ACM conference on Information and knowledge management*, pages 867–876, New York, NY, USA, 2009. ACM.

[17] M. Thorup and U. Zwick. Approximate distance oracles. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, STOC '01, pages 183–192, New York, NY, USA, 2001. ACM.

[18] M. V. Vieira, B. M. Fonseca, R. Damazio, P. B. Golgher, D. d. C. Reis, and B. Ribeiro-Neto. Efficient search ranking in social networks. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, CIKM '07, pages 563–572, New York, NY, USA, 2007. ACM.

[19] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, Jun 1998.

[20] X. Zhao, A. Sala, C. Wilson, H. Zheng, and B. Y. Zhao. Orion: shortest path estimation for large social graphs. In *Proceedings of the 3rd conference on Online social networks*, WOSN'10, pages 9–9, Berkeley, CA, USA, 2010. USENIX Association.

[21] U. Zwick. Exact and approximate distances in graphs – a survey. In *ESA '01: 9th Annual European Symposium on Algorithms*, pages 33–48. Springer, 2001.