Proceedings of the Eighth International AAAI Conference on Weblogs and Social Media

Memory-Efficient Fast Shortest Path Estimation in Large Social Networks

Volodymyr Floreskul and **Konstantin Tretyakov*** and **Marlon Dumas**†
Institute of Computer Science,
University of Tartu, Estonia

Abstract

As the sizes of contemporary social networks surpass billions of users, so grows the need for fast graph algorithms to analyze them. A particularly important basic operation is the computation of shortest paths between nodes. Classical exact algorithms for this problem are prohibitively slow on large graphs, which motivates the development of approximate methods. Of those, landmark-based methods have been actively studied in recent years.

Landmark-based estimation methods start by picking a fixed set of landmark nodes, precomputing the distance from each node in the graph to each landmark, and storing the precomputed distances in a data structure. Prior work has shown that the number of landmarks required to achieve a given level of precision grows with the size of the graph. Simultaneously, the size of the data structure is proportional to the product of the size of the graph and the number of landmarks. In this work we propose an alternative landmark-based distance estimation approach that substantially reduces space requirements by means of pruning: computing distances from each node to only a small subset of the closest landmarks.

We evaluate our method on the DBLP, Orkut, Twitter and Skype social networks and demonstrate that the resulting estimation algorithms are comparable in query time and potentially superior in approximation quality to equivalent non-pruned landmark-based methods, while requiring less memory or disk space.

1 Introduction

The shortest path problem is one of the core problems in graph theory. Effective algorithms have been developed for it long ago, which work well on small and medium-size graphs. In recent years more and more interest is concentrated on large social networks (such as Facebook, LinkedIn, Twitter, Skype), web and knowledge graphs. The size of these large graphs makes the basic *exact* shortest-path algorithms prohibitively slow. Even though the recently proposed exact algorithms can scale up to graphs with a few hundred million edges (Akiba, Iwata, and Yoshida 2013), they are still slow for contemporary social and web networks, which are in the order of billions or tens of billions of edges.

Approximate shortest path estimation methods provide an attractive alternative, which can scale to larger graphs. A promising family of such methods are those based on the use of *landmarks* (also referred to as *sketches*, *pivots* or *beacons* in various works). In a nutshell, these methods start by fixing a set of k *landmark nodes* and precomputing the shortest path distance between each node in the graph and each landmark. After this precomputation, an approximate distance between any two nodes can be computed using triangle inequality in $O(k)$ time.

The accuracy of landmark-based methods can be improved by using more landmarks. This, however, leads to linear increase in memory and disk space usage with only slight reduction of the approximation error.

In this work we describe an improvement to the landmark-based technique that can significantly reduce memory usage while keeping comparable accuracy and query running time. The idea of the modification is based on the fact that in the majority of cases it is enough to keep the distance from each node to only a small set of the closest landmarks rather than to all the landmarks. This optimization allows us to use a higher number of landmarks without a corresponding linear increase in memory usage.

2 Basic Definitions

This paper builds upon the methods and algorithms proposed in the work (Tretyakov et al. 2011). In this section we briefly review the basic definitions and the original landmark-based algorithms proposed in that paper.

As usual, by $G = (V, E)$ we denote a graph with $n = |V|$ vertices and $m = |E|$ edges. We shall consider undirected and unweighted graphs only, although the presented approach generalizes easily to accomodate weighted and directed graphs as well.

A *path* $\pi_{s,t}$ of length $\ell = |\pi_{s,t}|$ between two vertices $s, t \in V$ is defined as a sequence $\pi_{s,t} = (s, u_1, \ldots, u_{\ell-1}, t)$, where $\{u_1, \ldots, u_{\ell-1}\} \subseteq V$ and $\{(s, u_1), \ldots, (u_{\ell-1}, t)\}$

*Corresponding author.
†All authors are also affiliated with Software Technology and Applications Competence Center (STACC), Estonia. This work was partly supported by Microsoft/Skype Labs.
Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

$\subseteq E$. 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 $s, t, u \in V$

$$d(s,t) \leq d(s,u) + d(u,t)\,. \qquad (1)$$

If the node u lies on or near an actual shortest path from s to t, the upper bound is usually a good approximation to the true distance $d(s,t)$. This forms the core idea of all landmark-based approximation algorithms.

The simplest of them, *Landmarks-Basic*, first precomputes for a given landmark u the distance $d(s,u)$ between u and each node $s \in V$ in the graph. This allows to compute the upper bound approximation (1) for any s and t in constant time by simply adding two precomputed numbers.

The *Landmarks-LCA* algorithm, instead, precomputes a whole *shortest path tree (SPT)* for the landmark node u. This makes it possible to further increase the quality of approximation by merging the paths from the nodes s and u to the landmark and removing a cycle if it occurs. The resulting path will essentially pass through the "least common ancestor" of s and t in the shortest path tree for landmark u. The method presented in (Qiao et al. 2014) uses the same idea. Aside from computing distances, the method is capable of returning actual shortest paths.

Both *Landmarks-Basic* and *Landmarks-LCA* are typically used with a set of k different landmarks: the method is applied for each landmark separately and the best result is returned.

Finally, the *Landmarks-BFS* method precomputes shortest path trees like the *Landmarks-LCA* does. The approximation is computed, however, by collecting all nodes lying on all shortest paths from s and t to all the landmarks, and then running a standard breadth-first (or Dijkstra) search on the resulting subgraph.

3 Algorithm Description

3.1 Pruned landmark trees

Traditional landmark-based methods require the computation of a *shortest-path tree (SPT)* for each landmark node u. A SPT is stored by keeping a *parent pointer* $p_u[v]$ for each $v \in V$, which indicates the node that follows v on the shortest path from v to u. For the *Landmarks-Basic* method, only the distance $d_u[v]$ from v to the landmark needs to be kept. In both cases, however, the space requirements for storing the precomputed data for each landmark is proportional to the number of nodes n in the graph. For k landmarks this results in the total memory requirements of $O(kn)$.

We propose to reduce this complexity by *pruning* the size of the shortest path trees that need to be stored. Formally, define a *pruned landmark tree (PLT)* as a shortest path tree on a subset of nodes $V' \subset V$, with the landmark node as the root.

There may be multiple pruning strategies. The method proposed in (Vieira et al. 2007) limits trees based on depth,

Algorithm 1 PLT-PRECOMPUTE

Require: Graph $G = (V, E)$, a set of landmarks $U \subset V$, number of landmarks per node r.

```
 1: procedure PLT-PRECOMPUTE
 2:     for v ∈ V do                          ▷ Initialize empty arrays
 3:         c[v] ← 0
 4:         for u ∈ U do
 5:             p_u[v] ← nil
 6:             d_u[v] ← ∞
 7:         end for
 8:     end for
 9:     Create an empty queue Q.
10:     for u ∈ U do                          ▷ Initialize queue
11:         Q.enqueue((u, u, 0))
12:         p_u[u] ← u
13:         d_u[u] ← 0
14:     end for
15:     while Q is not empty do
16:         (u, v, d) ← Q.dequeue()
17:         for x ∈ G.adjacentNodes(v) do
18:             if p_u[x] = nil and c[x] < r then
19:                 p_u[x] ← v
20:                 d_u[x] ← d + 1
21:                 c[x] ← c[x] + 1
22:                 Q.enqueue((u, x, d + 1))
23:             end if
24:         end for
25:     end while
26: end procedure
```

i.e. when building the tree it ignores all nodes with distance from the landmark larger than some fixed value. The drawbacks of this strategy are that nodes are inequally covered by landmarks and there may even exist nodes unconnected to any landmarks at all, which makes it impossible to approximate distances between them and any other vertices.

In the work (Akiba, Iwata, and Yoshida 2013), a pruning and landmark selection technique is proposed, which ensures that each pair of nodes in the graph would share at least one common landmark node, located on a shortest path between them. The resulting index can be used to quickly compute *exact* shortest path distance between any pair of nodes. The potential drawback of such an approach is the size of the index structure. As the size of the landmark set is not initially fixed, it can become prohibitively large for billion-node graphs.

We propose a somewhat intermediate solution. A fixed set of landmarks is selected first. Then for each node $v \in V$, our algorithm ensures that the size of the associated landmark set $L(v)$ of that node is limited to a fixed number r of its closest landmarks. The motivation comes from the observation that quite frequently landmarks that are close to a node tend to provide the best distance approximations.

This can be achieved using a modified BFS algorithm that we call *PLT-Precompute* (see Algorithm 1). Similarly to the regular BFS it is based on an iteration over a queue. This queue contains tuples (u, v, d), where u is a landmark, v is

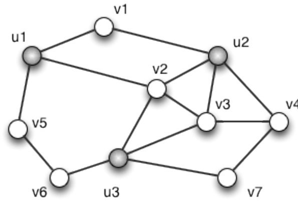


Figure 1: Example graph with landmarks u_1, u_2, u_3.

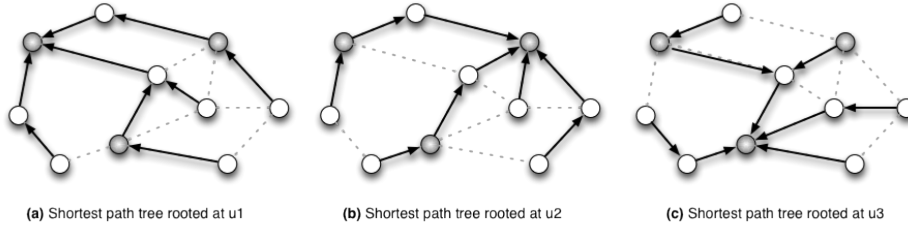


Figure 2: Shortest path trees

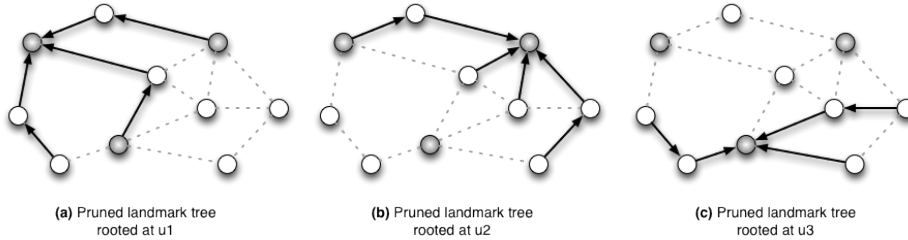


Figure 3: Pruned landmark trees

the next node to be processed in the SPT for the landmark u, and d is the distance from u to v. The queue is initialized with the set $\{(u, u, 0) : u \in U\}$, which, intuitively, corresponds to performing the BFS "in parallel" from all the landmarks. The difference with the regular BFS is that each node can be visited by at most r different landmarks. This is implemented by keeping track of the set of associated landmarks $L(v) = \{u : p_u[v] \neq nil\}$ for each node. No further traversal of a node is allowed when it has already been visited by r landmarks. The algorithm stops when the queue is empty.

After the algorithm completes, the resulting set $L(v)$ for each node v will contain its $\min(r, k')$ closest landmarks, where k' is the number of landmarks in the connected component of v. See Theorem 1 in Appendix A.

Figure 1 presents an example of a small graph with three selected landmarks. Figure 2 illustrates the full shortest path trees obtained by the traditional landmark-based appoach, and Figure 3 demonstrates the pruned trees ensuring $r = 2$ landmarks per node.

3.2 Distance approximation with pruned trees

Basic method As described in Section 2, the core landmark-based approximation technique is based on the simple triangle inequality. The same algorithm cannot be directly applied to pruned landmark trees, as it is not guaranteed that for any pair of nodes (s, t) both of them will share any common landmarks (i.e., belong to the same landmarks shortest path trees). To address this problem we must use a *pair* of landmarks $u \in L(s)$ and $v \in L(t)$ in the shortest path distance approximation, including the precomputed distance $d(u, v)$ between the landmarks into the equation:

$$d_{\text{approx}}(s, t) \approx d(s, u) + d(u, v) + d(v, t).$$

To obtain the best approximation, we iterate over all pairs of landmarks $(u, v) \in L(s) \times L(t)$ and choose the one that produces the smallest approximation. We refer to this method as the *PLT-Basic*, see Algorithm 2. Clearly, if there are common landmarks between s and t, for those landmarks this method produces the same result as the *Landmarks-Basic* algorithm.

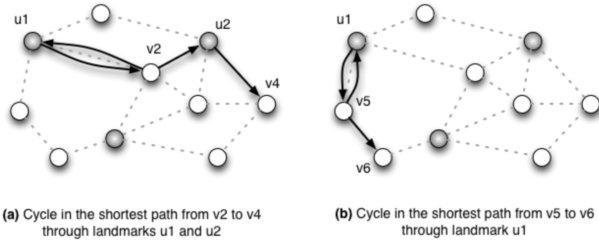
Consider the pruned landmark trees from Figure 3. Suppose that we want to estimate the distance between v_5 and v_4. When we use landmarks u_1 and u_2 the resulting approximate shortest path is computed to be $(v_5, u_1)+(u_1, v_1, u_2)+(u_2, v_4)$ of length 4. The two nodes are both present in the landmark tree rooted at u_3, hence the *PLT-Basic* algorithm will also find the path $(v_5, v_6, u_3) + (u_3, v_3, v_4)$, also of length 4.

Cycle elimination Consider again the PLTs on Figure 3. If we use the *PLT-Basic* algorithm to estimate the distance between v_2 and v_4 through landmarks u_1 and u_2, we may end

Algorithm 2 PLT-BASIC

Require: Graph $G = (V, E)$, a set of landmarks U, precomputed distance $d_u[x]$ from each node x to each landmark $u \in L(v)$, precomputed distance $d[u, v]$ for each pair of landmarks $(u, v) \in U \times U$.

```
 1: function PLT-BASIC(s, t)
 2:     d_min ← ∞
 3:     for u ∈ L(s) do
 4:         for v ∈ L(t) do
 5:             d ← d_u[s] + d[u, v] + d_v[t]
 6:             d_min ← min(d_min, d)
 7:         end for
 8:     end for
 9:     return d_min
10: end function
```



(a) Cycle in the shortest path from v2 to v4 through landmarks u1 and u2

(b) Cycle in the shortest path from v5 to v6 through landmark u1

Figure 4: Cycle elimination examples.

up with a path containing a cycle, as shown on Figure 4a. Analogously, when estimating the distance between v_5 and v_6, even through the same tree of the landmark u_1, the resulting path will contain a cycle of length 2 (see Figure 4b).

The *PLT-CE* algorithm (Algorithm 3) implements the cycle elimination technique to improve the results of the PLT-BASIC. To achieve that, it computes actual paths (not just distances), and relies on a fairly straighforward use of a stack and a set data structures to remove the loops. The issue of efficiently obtaining pieces of the path between the landmarks (the PATH-BETWEEN function) is discussed below in Section 3.2. The *PLT-CE* method can be regarded as a pruned version of the previous *Landmarks-LCA* approach.

Pruned landmark BFS Suppose that we want to get the shortest path between nodes u_1 and v_3 using pruned landmark trees depicted in Figure 3. Both *PLT-Basic* and *PLT-CE* algorithms can only return paths with distance 3 while the true shortest path (u_1, v_2, v_3) is of distance 2. The reason is that edge (v_2, v_3) is not present in any of the used PLTs.

The previous *Landmarks-BFS* algorithm proposes to approach this problem by running a BFS on a subgraph induced by the source and destination nodes and the paths from these to all the landmarks. This method makes use of *shortcuts* – edges that are present in the graph but are not present in landmark trees and therefore requires the graph itself. Another benefit of running BFS is that it always re-

Algorithm 3 PLT-CE

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

```
 1: function ELIMINATE-CYCLES(π)
 2:     S ← ∅
 3:     T ← Empty stack
 4:     for x ∈ π do
 5:         if x ∈ S then
 6:             while x ≠ T.top() do
 7:                 v ← T.pop()
 8:                 Remove v from S.
 9:             end while
10:         else
11:             Add x to S
12:             T.push(x)
13:         end if
14:     end for
15:     return T, converted from a Stack to a Path
16: end function
17: 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
18:     Result ← (s)                     ▷ Sequence of 1 element.
19:     while s ∉ π do
20:         s ← p_u[s]
21:         Append s to Result
22:     end while
23:     return Result    ▷ (s, p_u[s], p_u[p_u[s]], . . . , q), q ∈ π
24: end function
25: function PLT-CE(s,t)
26:     d_min ← ∞
27:     for u ∈ L(s) do
28:         for v ∈ L(t) do
29:             π ← PATH-TO(s, (u)) +
30:                     PATH-BETWEEN(u, v) +
31:                     REVERSED(PATH-TO(t, (v)))
32:             d ← |ELIMINATE-CYCLES(π)|
33:             d_min ← min(d_min, d)
34:         end for
35:     end for
36:     return d_min
37: end function
```

turns a path that does not contain cycles.

The PLT-BFS algorithm (Algorithm 4) is the adapted version of LANDMARKS-BFS that operates on pruned landmark trees. This time the induced graph is constructed on the set of vertices composed of all shortest paths from the source and destination nodes s and t to their known landmarks $L(s)$ and $L(t)$ as well as all nodes on the interlandmark paths $\{\pi_{u,v} | u \in L(s), v \in L(t)\}$.

Computing paths between landmarks All the three proposed algorithms (*PLT-Basic*, *PLT-CE*, *PLT-BFS*) require the precomputation of the shortest path between each pair of

Algorithm 4 PLT-BFS

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

```
1:  function PLT-BFS(s,t)
2:      S ← ∅
3:      for u ∈ L(s) ∪ L(t) do
4:          S ← S ∪ PATH-TO(s, (u))
5:          S ← S ∪ PATH-TO(t, (u))
6:      end for
7:      for u ∈ L(s) do
8:          for v ∈ L(t) do
9:              S ← S ∪ PATH-BETWEEN(u, v)
10:         end for
11:     end for
12:     Let G[S] be the subgraph of G induced by S.
13:     Apply BFS on G[S] to find
14:      a path π from s to t.
15:     return |π|
16: end function
```

Algorithm 5 PATH-BETWEEN-LANDMARKS

Require: Graph $G = (V, E)$, a set of landmarks U, an SPT parent link $p_u[x]$ and a distance value $d_u[x]$ precomputed for each $u \in L(x), x \in V$.

```
1:  procedure CALCULATE-WITNESS-NODES
2:      for x ∈ V, u ∈ L(x), v ∈ L(x) do
3:          if w[u, v] = nil or
4:          (d_u[x] + d_v[x] <
5:          d_u[w[u, v]] + d_v[w[u, v]]) then
6:              w[u, v] ← x
7:          end if
8:      end for
9:  end procedure
10: function PATH-BETWEEN(u,v)
        Returns the path between landmarks u and v
11:     π ← PATH-TO(w[u, v], (u)) +
12:         REVERSED(PATH-TO(w[u, v], (v)))
13:     return π
14: end function
```

landmarks. The straightforward method to do it is to run BFS from each landmark and save distances to all other ones. Such a procedure, however, requires $O(k(m + n))$ time for k landmarks. The linear time dependency on k makes it prohibitive to use the number of landmarks significantly larger than in previous landmark-based methods, which somewhat reduces the benefits of the new approach.

We propose to tackle this problem by calculating approximations of interlandmark shortest path distances from the data already collected by the PLT-PRECOMPUTE algorithm. The idea is to find a *witness node* $w[u, v]$ for each pair of landmarks $u \in U$ and $v \in U$ such that $w[u, v]$ is present in the pruned landmark trees for both u and v, i.e. $\{u, v\} \subset L(w[u, v])$. The approximation of the distance between the landmarks can then be computed through this node as $d_u[w[u, v]] + d_v[w[u, v]]$. Also the approximate path between the landmarks can be restored via the witness.

Obviously, if several witness nodes exist for a pair of landmarks, we choose the one which minimizes the approximation. The implementation is provided in the CALCULATE-WITNESS-NODES procedure in Algorithm 5. When this procedure finishes, the approximated shortest paths between the landmarks can be obtained using the function PATH-BETWEEN.

Algorithm complexity The proposed modifications to the traditional landmark algorithms affect their runtime complexity twofold. On one hand, computation of pruned landmark trees requires visiting each node and each edge up to r times and therefore pruned trees can be built in $\Theta(r(m+n))$ time. This is more efficient compared to $\Theta(k(m + n))$ complexity of computing full SPTs in the regular landmark-based methods. On the other hand, the need to precompute distances between pairs of landmarks (Algorithm 5) introduces an additional $\Theta(r^2 n)$ term.

The time per query of the original methods was linear in the number of landmarks, $\Theta(k)$. In the proposed approaches the query time does not depend on the total number of landmarks, but rather is $\Theta(r^2)$ as the search is performed over pairs of landmarks.

Thus, both in precomputation and query time the new approaches are comparable to the previous ones whenever $r^2 \approx k$.

In terms of space complexity, the new methods require $\Theta(rn)$ space to keep landmark data plus $\Theta(k^2)$ for storing interlandmark witness nodes or distances. This compares favourably with the $\Theta(kn)$ complexity of the previous approaches whenever $n \gg k$, which is true for most large graphs.

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. Those are the same datasets that were used in (Tretyakov et al. 2011), the dataset descriptions below are presented verbatim from that paper.

- **DBLP.** The DBLP dataset contains bibliographic information of computer science publications (Ley and Reuther 2006). 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 (Mislove et al. 2007).
- **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 (Kwak et al. 2010). 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. The snapshot 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}. Note that the reported time to perform a BFS differs from the one given in (Tretyakov et al. 2011) due to the fact that we use a different programming language (Java) to implement our experiments.

Dataset	**DBLP**	**Orkut**	**Twitter**	**Skype**
$\|V\|$	770K	3.1M	41.7M	454M
$\|E\|$	2.6M	117M	1.2B	3.1B
$\overline{d}$	6.25	5.70	4.17	6.7
Δ	23	10	24	60
$\|S\|/\|V\|$	85%	100%	100%	85%
t_{BFS}	343 ms	25.4 sec	11 min	33 min

Table 1: Datasets.

4.2 Experimental setup

In each experiment we randomly choose 500 pairs of vertices (s, t) from each graph and precompute the true distance between s and t for each pair by running the BFS algorithm. We then apply the proposed distance approximation algorithms to these pairs and measure the average *approximation error* and *query execution time*.

Approximation error is computed as $(\ell' - \ell)/\ell$, where ℓ' is the approximation and ℓ is the actual distance. *Query execution time* refers to the average time necessary to compute a distance approximation for a pair of vertices.

All experiments were run under Scientific Linux release 6.3 on a server with 8 Intel Xeon E7-2860 processors and 1024GB RAM. Only a small part of the computational resources was used in all experiments.

The described methods were implemented in Java. Graphs and intermediate data were stored on disk and accessed through memory mapping.

4.3 Landmark selection

As the proposed methods are focused on using larger number of landmarks than the previous techniques it becomes very important to choose scalable selection strategies. We use two strategies in our comparisons: *Random selection* and *Highest degree selection*.

One or both of these strategies have been used in many previous works that involve landmark-based methods (Goldberg and Harrelson 2005; Potamias et al. 2009; Tretyakov et al. 2011; Vieira et al. 2007; Zhao et al. 2010).

In random selection we make sure to use the same nodes in the experiments with equal landmark set sizes in order to make results more comparable.

4.4 Results

Approximation Error Figures 5, 6, 7 and 8 present the approximation error for DBLP, Orkut, Twitter and Skype graphs correspondingly. The error values are present for different landmark selection strategies (rows), algorithms (columns), numbers of landmarks per node (bar colors) and number of landmarks (x-axis). The dashed black line is the baseline. As the baseline for PLT-Basic we use Landmark-Basic, for PLT-CE we use Landmark-LCA and for PLT-BFS we use Landmark-BFS. Each of the baseline algorithms is used with 100 landmarks and the values are obtained from (Tretyakov et al. 2011).

Landmark selection strategy is a very significant factor for approximation quality, especially for PLT-Basic and PLT-CE algorithms. For the PLT-BFS method, however, randomly selected landmarks provide accuracy comparable with the highest degree method and sometimes even outperform them, as in the case for the Twitter graph. This effect was also observed for Landmark-Basic and Landmark-BFS in (Tretyakov et al. 2011).

Higher number r of landmarks per node leads to consistent reduction of the approximation error, as one might expect. Increasing the total number of landmarks k, however, may sometimes have no or even an opposite effect, as observed in the results for Orkut and Twitter with random landmark selection strategy. The reason for this lies in the fact that increasing the number of landmarks, while keeping the number of landmarks r per node fixed, results in the shrinking of pruned landmark trees and therefore using more distant pairs of landmarks for the approximation.

The obtained results also reconfirm that the accuracy of the different algorithm highly depends on the internal properties of graphs themselves. While the PLT-BFS method can return exact values in almost all cases on the DBLP graph (approximation error less than 0.01), the lowest obtained error for the Skype graph is still as high as 0.15.

The comparison with regular landmark-based algorithms confirms the idea that our methods can achieve similar accuracy with much less memory usage. For example, in Skype graph with highest degree landmark selection strategy, 5 landmarks/node and 10000 landmarks we achieve about the same approximation error as the regular landmark-based methods with 100 landmarks.

4.5 Query execution time

Query time was computed as the average value among 500 random queries in each graph. The total measured time excludes time needed to load the index into main memory, but as our implementation uses the **mmap** Linux operating system feature, which does not guarantee that all the data is immediately loaded into RAM, a part of the measured time may also include time for loading parts of the index file.

Figure 9 presents the results. The query time did not depend on the number of landmarks k, hence this aspect is not

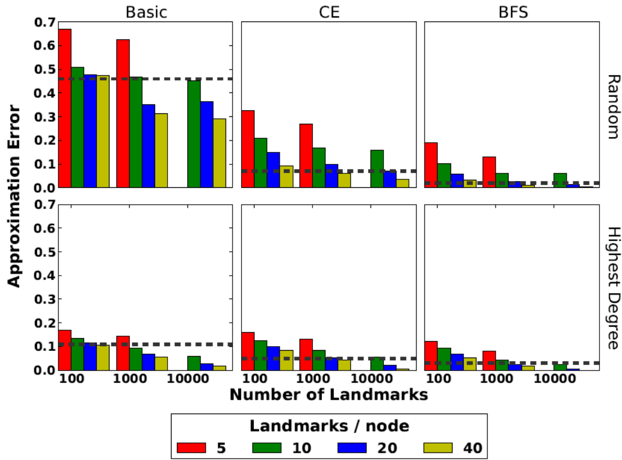


Figure 5: Approximation error for the DBLP graph

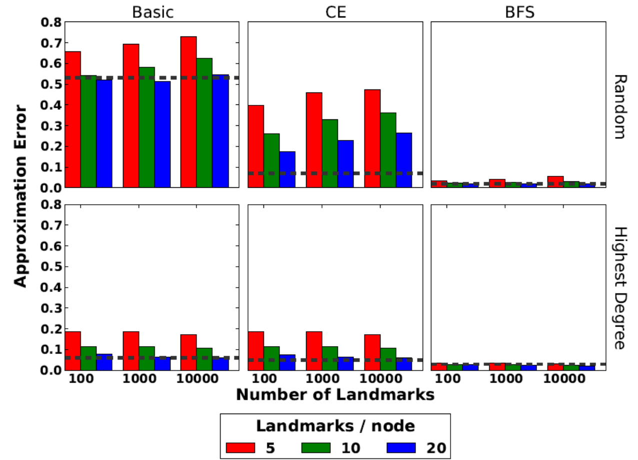


Figure 7: Approximation error for the Twitter graph

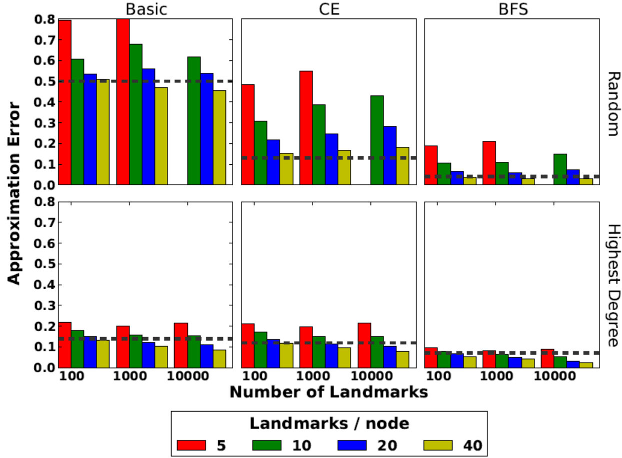


Figure 6: Approximation error for the Orkut graph

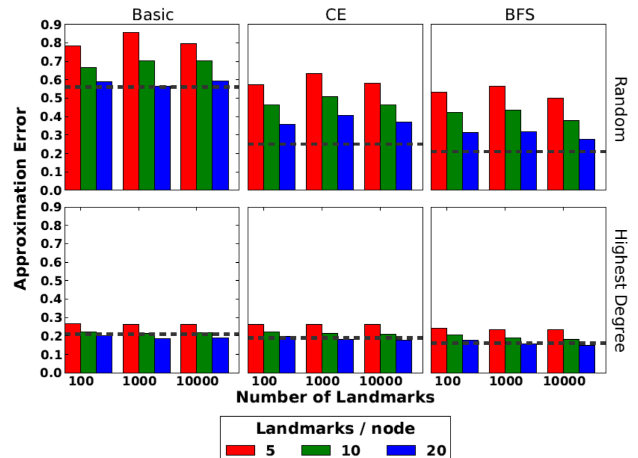


Figure 8: Approximation error for the Skype graph

shown. It has a quadratic dependency on the number of landmarks per node r, as expected.

Query time depends mostly on the choice of the algorithm and the graph. The average query time of PLT-Basic and PLT-CE methods never exceeds 7 milliseconds for 20 landmarks/node and is less than a millisecond for 5 landmarks/node in all cases. Unlike these two methods, the performance of the PLT-BFS highly depends on the dataset and the landmark selection strategy. For example, with 20 landmarks/node and the highest degree strategy the results vary from 4 milliseconds on the DBLP graph to 4 seconds on the Twitter graph.

4.6 Preprocessing time

The preprocessing time almost does not depend on the number of landmarks and their selection strategy. Table 2 contains time values obtained during the pruned landmark trees computation for different values of number of landmarks per node in each dataset. The data was collected for 1000 highest degree landmarks.

The pruned landmark tree computation heavily depends on the size of the graph. For example, for 20 landmarks/node it ranges from about 21 seconds in DBLP to almost 45 hours in Skype. The quadratic dependency of the preprocessing time on the number of landmarks per node prevents increasing this parameter for very large graphs.

Graph	Landmarks / Node		
	5	10	20
DBLP	3.6 s	8.6 s	21.1 s
Orkut	87 s	207 s	463 s
Twitter	48 m	105 m	247 m
Skype	4.4 h	18.6 h	44.9 h

Table 2: Preprocessing time for 1000 landmarks with highest degree selection strategy

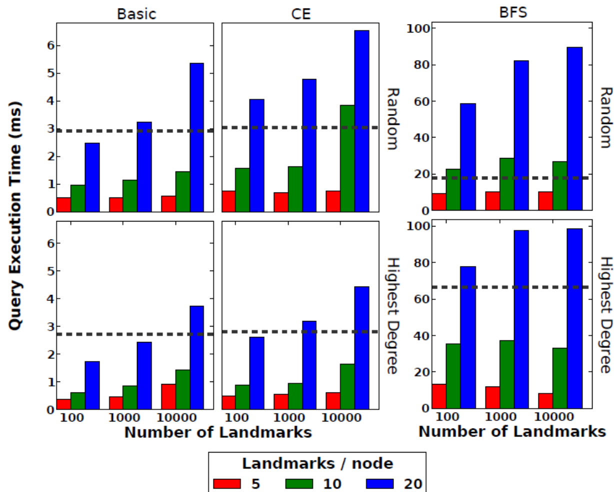


Figure 9: Average query time for the Skype graph

4.7 Memory usage

The main benefits of the proposed methods relates to memory savings. Whilst the previous approaches use $\Theta(kn)$ space to store k complete landmark trees, the requirements for pruned landmark trees are $\Theta(rn + k^2)$, which is significantly smaller whenever $k \ll n$.

The described property can be observed in Table 3, which shows the total amount of disk space consumed by the indexing structures. Notice how, for small r values, the sizes for DBLP and Orkut graph significantly depend on the total number of landmarks. For the larger Twitter and Skype graphs this effect is practically unnoticeable. The last column of Table 3 shows the baseline scenario of using 100 full landmark shortest path trees from (Tretyakov et al. 2011).

To store the trees we use a compact representation, where for each node we keep r (`landmark_id`, `node_id`) pairs. The nodes are identified using 32-bit integers.

5 Related Work

A large body of work exists on the problem of finding shortest paths between nodes in a graph. The methods can roughly be divided in to *exact* and *approximate*. The simplest example of an exact shortest path method is the *Dijkstra's algorithm* (Dijkstra 1959). In a general graph with n nodes and m edges, this algorithm computes paths from a single source to all other vertices in $O(m)$ space and no less than $O(m + n \log n)$ time. The runtime of the approach can be improved by running a *bi-directional* search (Pohl 1971) or exploiting the A* search algorithm (Ikeda et al. 1994; Goldberg and Harrelson 2005; Goldberg, Kaplan, and Werneck 2006).

Sometimes it makes sense to precompute shortest paths between all pairs of nodes. Numerous techniques have been proposed for this *all-pairs-shortest-path (APSP)* problem (Zwick 2001). Most of them run in $O(n^3)$ time, with a few subcubic solutions for certain types of graphs. This is not much better than simply running a separate single-source shortest path (SSSP) traversal from each vertex. The latter approach, however, can be optimized by *pruning* the traversals in a smart way. A recent algorithm (Akiba, Iwata, and Yoshida 2013) computes for each node a limited set of *distances to landmarks*, ensuring that any pair of nodes shares at least one landmark on the shortest path between them. Such a data structure makes it possible to answer exact shortest path queries. Although it is computed by performing a SSSP traversal from each node, the traversals can be heavily pruned and the method is shown to scale to graphs with millions of nodes and hundreds of millions of edges.

Approximate shortest path algorithms trade off accuracy in exchange for better time or memory requirements. Most approximate shortest path methods rely, in one way or another, on the idea of precomputing some distances in the graph and then using those to infer all other distances. Most commonly the distances are precomputed to a fixed set of *landmark* nodes (Cowen and Wagner 2004; Goldberg and Harrelson 2005; Vieira et al. 2007; Potamias et al. 2009; Das Sarma et al. 2010; Gubichev et al. 2010; Tretyakov et al. 2011; Agarwal et al. 2012; Cheng et al. 2012; Jin et al. 2012; Fu and Deng 2013; Qiao et al. 2014), which enables the use of the *Landmarks-Basic* algorithm and its derivatives. Some variations of the basic algorithm allow to compute actual shortest paths rather than just distances (Gubichev et al. 2010; Tretyakov et al. 2011). It allows to further increase the accuracty and support dynamic updates to the data structure.

A variation suggested in (Agarwal et al. 2012) computes, for each node, besides the distances to the landmarks, also the distances to all nodes in its *vicinity*. At the cost of some additional memory, the resulting algorithm is capable of answering shortest path queries exactly for as much as 99.9% of node pairs in the graph.

So far there are no strong theoretical guarantees on approximation quality of landmark-based methods (Kleinberg, Slivkins, and Wexler 2004). However, they have been shown to provide good accuracy while keeping the query time in the order of milliseconds, even for very large graphs (Potamias et al. 2009; Das Sarma et al. 2010; Gubichev et al. 2010; Tretyakov et al. 2011; Agarwal et al. 2012).

All of the approaches mentioned above, however, require no less than $O(kn)$ disk space to store the index structure, where k is the number of landmarks. Reducing this memory requirement without significantly compromising the accuracy or query time is the central problem addressed in this work.

Finally, a smart choice of a landmark selection strategy can have a significant positive effect on accuracy. Several strategies have been proposed and evaluated in previous works (Potamias et al. 2009; Tretyakov et al. 2011). The general result seems to be that picking the landmarks with the highest degree would often provide very good results at a low computational cost.

6 Conclusion

In this work we introduced and evaluated *pruned landmark trees* as an improvement for landmark-based estimation of shortest paths. With respect to previous related work, this

Graph	Landmarks	Landmarks / Node			Baseline
		5	10	20	(100 SPTs)
	100	30M	59M	117M	
DBLP	1000	34M	63M	121M	300M
	10000	411M	441M	499M	
	100	118M	235M	469M	
Orkut	1000	122M	239M	473M	1.2G
	10000	499M	616M	851M	
	100	1.6G	3.2G	6.3G	
Twitter	1000	1.6G	3.2G	6.3G	16G
	10000	2.0G	3.5G	6.6G	
	100	17G	34G	68G	
Skype	1000	17G	34G	68G	170G
	10000	18G	35G	69G	

Table 3: Total PLT index memory usage

allows to achieve comparable or better accuracy and similar query time with decreased memory and disk space usage.

For example, when compared to the baseline 100-landmark methods from (Tretyakov et al. 2011), the proposed methods with $k = 1000$ highest degree landmarks and $r = 20$ landmarks per node show consistently better performance in terms of accuracy on all the tested graphs, require 2.5 times less disk space, yet only use a factor of 1.5 more time. With $k = 100$ and $r = 5$ the PLT approach underperforms only slightly in terms of accuracy, yet requires 10 times less space and 5 times less time per query.

The methods were presented for the case of undirected unweighed graphs, but they can be generalized to support weighted and directed graphs by replacing BFS with Dijkstra traversal and storing two separate trees for each landmark – one for incoming paths and another for outgoing ones. We also foresee that pruned landmark trees could be dynamically updated under edge insertions and deletions using techniques similar to those outlined in (Tretyakov et al. 2011).

7 Acknowledgements

The authors acknowledge the feedback from Ando Saabas from Skype/Microsoft Labs. This research is funded by ERDF via the Estonian Competence Center Programme and Microsoft/Skype Labs.

References

Agarwal, R.; Caesar, M.; Godfrey, B.; and Zhao, B. Y. 2012. Shortest paths in less than a millisecond. In *Proc. of the Fifth ACM SIGCOMM Works. on Social Networks (WOSN)*, 37–42. ACM.

Akiba, T.; Iwata, Y.; and Yoshida, Y. 2013. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, 349–360. New York, NY, USA: ACM.

Cheng, J.; Ke, Y.; Chu, S.; and Cheng, C. 2012. Efficient processing of distance queries in large graphs: A vertex cover approach. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, 457–468. New York, NY, USA: ACM.

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

Das Sarma, A.; Gollapudi, S.; Najork, M.; and Panigrahy, R. 2010. 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, 401–410. New York, NY, USA: ACM.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271.

Fu, L., and Deng, J. 2013. Graph calculus: Scalable shortest path analytics for large social graphs through core net. In *Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2013 IEEE/WIC/ACM International Joint Conferences on*, volume 1, 417–424.

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

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

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

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

Jin, R.; Ruan, N.; Xiang, Y.; and Lee, V. E. 2012. A highway-centric labeling approach for answering distance queries on large sparse graphs. In Candan, K. S.; 0001, Y. C.; Snodgrass, R. T.; Gravano, L.; and Fuxman, A., eds., *SIGMOD Conference*, 445–456. ACM.

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

Kwak, H.; Lee, C.; Park, H.; and Moon, S. 2010. What is

Twitter, a social network or a news media? In *WWW '10: Proceedings of the 19th international conference on World wide web*, 591–600. New York, NY, USA: ACM.

Ley, M., and Reuther, P. 2006. 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.

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

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

Potamias, M.; Bonchi, F.; Castillo, C.; and Gionis, A. 2009. Fast shortest path distance estimation in large networks. In *CIKM '09: Proceeding of the 18th ACM conference on Information and knowledge management*, 867–876. New York, NY, USA: ACM.

Qiao, M.; Cheng, H.; Chang, L.; and Yu, J. X. 2014. Approximate shortest distance computing: A query-dependent local landmark scheme. *IEEE Transactions on Knowledge and Data Engineering* 26(1):55–68.

Tretyakov, K.; Armas-Cervantes, A.; García-Bañuelos, L.; Vilo, J.; and Dumas, M. 2011. Fast fully dynamic landmark-based estimation of shortest path distances in very large graphs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, CIKM '11, 1785–1794. New York, NY, USA: ACM.

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

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

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

A Proofs

Theorem 1 *The Algorithm 1 (*PLT-Precompute*) selects the set $L(v)$ of the closest landmarks for each node $v \in V$. The size of the set $|L(v)|$ is equal to* $\min(r, k')$*, where k' is the number of landmarks in the connected component of v.*

Before we can prove this theorem, we need an auxiliary result.

Let the set of landmarks be $U = \{u_1, \ldots, u_k\}$. Without loss of generality, we shall assume there is an ordering among the landmarks (e.g. landmark u_1 will be considered to be *preceding* u_2, denoted as $u_1 \prec u_2$) and that the landmarks are first pushed into the queue on lines 10–14 of the algorithm in this particular order.

Note that at first the queue Q contains k tuples of the form $(u, u, 0)$, ordered according to the landmark ordering. After k iterations of line 16, those tuples are removed from the queue and instead a number of elements of the form $(u, x, 1)$ is enqueued, where the distance value is 1 and the landmarks are again in the correct order. Continuing in this fashion, for the dequeued distance-1 elements, some new elements with distance value 2 are pushed again in the correct order of landmarks, and so on. It is thus easy to see that the following must hold:

Lemma 1 *Tuple (u, x, d_1) can be enqueued before (ℓ, y, d_2) only if $d_1 < d_2$ or ($d_1 = d_2$ and $u \prec \ell$).*

Proof of Theorem 1. Consider some node $v \in V$. If there are $k' \leq r$ landmarks in the connected component of v, the condition on line 18 may become false for some node only after it is already associated with all the landmarks. Thus, a full traversal of the component will be performed for each landmark and $L(v)$ will contain all k' of them (possibly zero, if $k' = 0$).

The remainder of the proof assumes there are at least $r + 1$ landmarks in the same connected component as v. Suppose that after completing the algorithm a landmark $u \in U$ (from the same connected component) is not in $L(v)$, that is, $p_u[v] = nil$. We will now demonstrate that from this it follows that there exist at least r other landmarks $\{\ell_1, \ldots, \ell_r\}$ such that for each ℓ_i, either it is closer to v than u or at the same distance, but *preceding* u (i.e. $\ell_i \prec u$).

Consider two cases. a) There exists a neighbor w of v, such that $p_u[w] \neq nil$. In this case a tuple $(u, w, \cdot)$ must have been added to Q at some point (as executing line 19 implies executing line 22 too). At some later moment this tuple was dequeued on line 16 and all neighbors of w, including v were iterated over. We know that $(u, v, \cdot)$ was not enqueued, hence at that moment $c[v] = r$, which means that for r other landmarks ℓ_i a tuple $(\ell_i, v, \cdot)$ had been enqueued already. It follows from Lemma 1 that those landmarks were either closer to v than u or at the same distance, but *preceding*.

The second case. b) No neighbors of v have u in their landmark sets. Consider the shortest path $\pi_{v,u} = (v, w_1, w_2, \ldots, u)$. As we know that $u \in L(u)$ (due to line 12), and $u \notin L(v)$, there must exist a node w_j along the path such that $u \notin L(w_j)$, but $u \in L(w_{j+1})$. Repeating the logic of case a) we conclude that there exist r distinct landmarks ℓ_i which are closer to w_j than u (or at the same distance, but preceding). But if $d(w_j, \ell_i) \leq d(w_j, u)$, then necessarily $d(v, \ell_i) \leq d(v, w_j) + d(w_j, \ell_i) \leq d(v, w_j) + d(w_j, u) = d(v, u)$. Hence, any of the landmarks ℓ_i is also either closer to v than u or at the same distance but preceding.

We have shown that if $u \notin L(v)$ there must be r other landmarks closer to v than u. It remains to show that after algorithm completes, $|L(v)| = r$ for all nodes v.

Assume that for some v it is not the case, i.e. $|L(v)| < r$. Then the condition on line 18 was never false for v. Hence, if any landmark u was ever associated with a neighbor w of v, it must have been also associated with v, i.e. $L(w) \subseteq L(v)$. But then $|L(w)| < r$ and we may repeat this logic recursively, ultimately concluding that for any other node w in the same connected component, $L(w) \subseteq L(v)$. But then also $\cup_w L(w) \subseteq L(v)$. The set $\cup_w L(w)$, however, contains all the landmarks from the connected component. We assumed there to be more than r of them, hence $r < |\cup_w L(w)| \leq |L(v)|$ which is a contradiction.