A Hybrid Index for Distance Queries

Junhu Wang[1(✉)], Shikha Anirban[1], Toshiyuki Amagasa[2], Hiroaki Shiokawa[2], Zhiguo Gong[3], and Md. Saiful Islam[1]

[1] Griffith University, Queensland, Australia
j.wang@griffith.edu.au
[2] University of Tsukuba, Tsukuba, Japan
[3] University of Macau, Zhuhai, China

Abstract. Shortest distance queries not only find important applications in real-word systems, it is also the foundation for numerous other graph analysis problems. State-of-the-art techniques for such queries use 2-hop labeling, in particular, the Pruned Landmark Labeling (PLL) index is among the best performing for small world networks. However, PLL suffers from large label size and index computation time when the graph is large. In this paper, we propose two techniques to address the problem. The first technique is to limit the landmarks to vertices in a minimum vertex cover, and the second is to use a hybrid index by decomposing the graph into a *core* and a forest, and combining the PLL index for the core and a simpler distance labeling for trees. Extensive experiments with real-world graphs verified the effectiveness of these techniques.

1 Introduction

Given a graph G and a pair of vertices s and t, the distance query asks for the distance from s to t, that is, the length of the shortest path from s to t. Distance queries find numerous applications such as in route planning and spatial databases [7,12], community search and influence maximization in location-based social networks [25]. They are also the foundation for several other problems such as graph homomorphism search [8] and distributed subgraph enumeration[21]. Answering such queries instantly is crucial for many of these applications.

Distance queries can be answered using one of two naive methods. The first method is to do a breath-first search[1] from s until we reach t, and the second method is to pre-compute the distances between all vertex pairs and store them as an index. With large graphs nowadays both methods are impractical: the former is too slow, and the latter takes too much space and pre-computation time. Therefore, many indexing techniques have been proposed that tried to strike a balance between index size/index construction time and online query

[1] Or to use Dijkstra's algorithm for weighted graphs.

Z. Huang et al. (Eds.): WISE 2020, LNCS 12342, pp. 227–241, 2020.
https://doi.org/10.1007/978-3-030-62005-9_17

processing time. As noted in several recent studies [16,17], for complex networks the state-of-the-art technique is *pruned landmark labelling* (PLL) [3]. PLL is a type of 2-hop labeling [6], which constructs a label for each vertex u, denoted $label(u)$, consisting of some vertex-distance pairs (v, d) and stores them as index (v is called a *hub* of u if $(v, d) \in label(u)$). At query time, the distance between s and t can be directly answered by examining the common hubs of s and t, and adding the distance from s to the hub and the distance from the hub to t together. Although PLL works very well compared with other techniques, for large graphs the index can be too large, which will not only take up memory but also make the queries slower. Several subsequent works tried to address the problem by better vertex ordering [2,17], graph compression, and elimination of local minimum nodes [16], however, these methods still generate label sets that are unnecessarily large.

In this paper, we propose two novel techniques to tackle the problem. Specifically, (1) we propose to use only vertices in a minimum vertex cover as landmarks; (2) we propose to decompose the graph into a core and a set of trees, and combine two types of labels, one for the core and one for the trees, as the distance index; We conduct extensive experiments with real-world graphs, which demonstrate that the proposed techniques can significantly reduce the index size and index time, and make the queries faster.

Organization. We present our techniques for undirected, unweighted graphs, but most of them can be easily adapted to directed and/or weighted graphs. Section 2 provides the preliminaries, Sect. 3 discusses computing labels using minimum vertex cover. Section 4 presents hybrid labeling by decomposing the graph into core and trees, and our experiments are reported in Sect. 5. Section 6 discusses related work. We conclude the paper in Sect. 7.

2 Preliminaries

A graph $G \equiv (V, E)$ consists of a set V of vertices and a set $E \subseteq V \times V$ of edges. In this paper we implicitly assume all graphs are *simple* graphs, that is, there is at most one edge from one vertex to another. We will focus on undirected and unweighted graphs where the edges have no direction and every edge has a length of 1.

If (u, v) is an edge in an undirected graph, we say u is neighbor of v and v is a neighbor of u, and we use $N(v)$ to denote the set of all neighbors of v. Let u, v be vertices in graph G. A path from u to v is a sequence of neighboring edges (u, v_1), (v_1, v_2), …, (v_k, v), which is also commonly represented as the sequence of vertices $u, v_1, \ldots, v_k, v$. The number of edges in the path is called its *length*. Given a pair of vertices u and v in the graph, a *shortest path* from u to v is a path from u to v with the smallest length, and we call this length the *distance* from u to v, denoted $d_G(u, v)$ or simply $d(u, v)$ when G is clear from the context. Note that for undirected graphs $d(u, v) = d(v, u)$.

Pruned Landmark Labeling (PLL). The pruned landmark labeling is a type of 2-hop labeling scheme [6] designed for the efficient processing of distance

queries. The general idea of a 2-hop labeling scheme is as follows: Suppose G is an undirected graph. For each vertex $u \in V$, we compute a label $L(u)$ which is a set of $(vertex, distance)$ pairs $(v, d(u, v))$, where v is a vertex in V, and $d(u, v)$ is the distance from u to v. A 2-hop labeling scheme is *complete* for G if it can be used to answer all distance queries in G as follows: let $H(u) = \{v|(v, d(u, v)) \in L(u)\}$. For every pair of vertices s and t in the graph:

$$d(s,t) = \min_{v \in H(s) \cap H(t)} d(s,v) + d(v,t) \tag{1}$$

A naive complete 2-hop label $L(u)$ contains $(v, d(u, v))$ for every vertex $v \in V$. However, it is unnecessary and can be too large and too expensive to compute. Using PLL we can compute a much smaller label set for each u using a pruning strategy which also makes the computation much faster.

Let $v_1, v_2, \ldots, v_n$ be the vertices in V, the PLL algorithm in [3] computes $L(u)$ as follows: First, we choose a fixed order of the nodes. Without loss of generality, we assume the order is $v_1, v_2, \ldots, v_n$. Then, we do BFS from v_1, then from v_2, ..., and finally from v_n, each time we traverse from v_j to u, we will decide whether $(v_j, d(u, v_j))$ should be added into $L(u)$.

1. Initially for every $u \in V$, $L_0(u) = \emptyset$;
2. When we reach u from v_j, we create $L_j(u)$ from $L_{j-1}(u)$ as follows: if there does not exist $i < j$ such that $d(v_j, u) \geq d(v_j, v_i) + d(v_i, u)$,

$$L_j(u) = L_{j-1}(u) \cup \{(v_j, d(v_j, u))\}$$

   Otherwise $L_j(u) = L_{j-1}(u)$, and we stop going further from u.
3. Finally, $L(u) = L_n(u)$.

The index L consists of the labels of all vertices, that is, $L = \{L(u)|u \in V\}$.

Observe that the condition in Step 2 above means that if some shortest path from v_j to u goes through one of the vertices v_i $(i < j)$, then $(v_j, d(v_j, u))$ will not be added into $L(u)$, and otherwise it will.

Due to the smart pruning strategy, PLL usually generates a much smaller label for each vertex u without losing completeness. That is, for only some vertices $v \in V$, the pair $(v, d(u, v))$ is in $L(u)$. For easy explanation and to be consistent with the terminology in [16], we will call v a *hub* of u if $(v, d(u, v)) \in L(u)$. We will also say v_i is ranked higher than v_j if v_i precedes v_j in the vertex ordering when computing PLL labels and denote it by $rank(v_i) > rank(v_j)$.

3 Restricting Landmarks to a Minimum Vertex Cover

Intuitively a landmark is a vertex that many shortest paths pass through. If there is a subset S of vertices that every shortest path in G passes through, then we can use S as landmarks when computing the 2-hop labels. Our key insight here is that the original PLL essentially treats every vertex in the graph as a potential landmark, i.e., every vertex can be a hub of some other vertices. However, this

is usually not necessary. We observe that using vertices in a *vertex cover* C is sufficient (A *vertex cover* is a set C of vertices in G such that every edge in G has at least one end in C). This is because every edge is incident on some vertex in C. Therefore, every non-trivial path, i.e., path of length ≥ 1, passes through a vertex in C. In other words, for any path between s and t ($s \neq t$), there is a vertex $v \in C$ such that $d(s,t) = d(s,v) + d(v,t)$. Therefore, our first intuition is to find a minimum vertex cover C, and limit the BFS in the PLL computation to vertices in C. For convenience, we denote the PLL labels computed using a set S of vertices by $PLL(S)$.

Example 1. Consider the graph G_1 shown in Fig. 1. $C = \{v_0, v_2, v_4, v_6\}$ is a minimum vertex cover. The computation of PLL labels using the vertex cover and not using the vertex cover are shown in Fig. 2. Using the vertex cover, we create a label set that contains a total of 21 labels, and using (an arbitrary ordering $v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7$ of) all vertices, we obtain a label set of 30 labels.

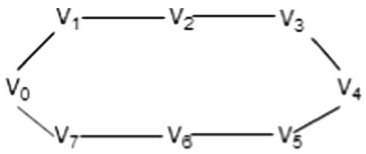


Fig. 1. Graph G_1

	v_0	v_2	v_4	v_6
v_0	0			
v_1	1	1		
v_2	2	0		
v_3	3	1	1	
v_4	4	2	0	
v_5	3	3	1	1
v_6	2		2	0
v_7	1		3	1

(a) $PLL(C)$

	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v_0	0							
v_1	1	0						
v_2	2	1	0					
v_3	3	2	1	0				
v_4	4	3	2	1	0			
v_5	3		3	2	1	0		
v_6	2			3	2	1	0	
v_7	1				3	2	1	0

(b) $PLL(V)$

Fig. 2. PLL label computation using vertex cover $S \equiv \{v_0, v_2, v_4, v_6\}$ and not using the vertex cover for graph G_1. The labels for each vertex are contained in a row. For instance, in $PLL(C)$, $L(v_3) = \{(v_0, 3), (v_2, 1), (v_4, 1)\}$.

One might wonder whether there are smaller landmark sets than the minimum vertex cover that can make the labels complete. The following proposition answers the question.

Proposition 1. *Let S be a subset of vertices in G. If S is not a vertex cover of G, then there exist a pair of vertices u, v $(u \neq v)$ such that $d(u, v)$ can not be computed using the PLL labels computed from S.*

Proof. Since S is not a vertex cover, there must be an edge (u, v) such that neither u nor v is in S. Clearly the edge (u, v) is a shortest path from u to v, while any path from u to v that passes through a vertex in S is of length at least 2. Therefore, $d(u, v)$ can not be computed using the PLL labels computed from S.

It was observed that in the original PLL algorithm, v is a hub of u if and only if v has the highest rank on all shortest paths from u to v.

Lemma 1 ([16]). *For any pair of vertices $u, v \in V$, v is a hub of u if and only if v is ranked highest on all shortest paths from u to v.*

Let C be a vertex cover of G. Without loss of generality, let us assume $v_1, \ldots, v_k$ $(k < n)$ are the vertices in C, and $v_{k+1}, \ldots, v_n$ are vertices not in C. Using Lemma 1, we can show

Theorem 1. *If we use the original PLL algorithm to compute the vertex labels in the order $v_1, v_2, \ldots, v_n$, then for every $v \in V - C$, v is the hub of only itself. That is, for any $u \neq v$, v is not a hub of u.*

Proof. Let $v \in V - C$ be any vertex outside the vertex cover C. For every vertex $u \in C$, $rank(u) > rank(v)$, therefore, according to Lemma 1, v is not a hub of u. For every vertex $v' \notin C$ such that $v' \neq v$, any path from v' to v must pass through some vertex $u \in C$ and u is ranked higher than v. Therefore, v is not ranked the highest on any path from v' to v. Thus v is not a hub of v' according to Lemma 1.

Observe that, if we order the vertices in C before those in $V - C$, then all the vertices in $V - C$ are *local minimum vertices* [16] which are vertices whose ranks are lower than that of their neighbours. Hence by Lemma 4.12 of [16], they are hubs of only themselves. This serves as an alternative proof of Theorem 1.

Vertices that are hubs of only themselves are not really helpful for distance queries, hence they can be removed from the label set L. Theorem 1 indicates that if the vertices in V are appropriately ordered, we can find a 2-hop label which is equivalent to the labels we find using a vertex cover. However, our result is still useful since finding a good vertex ordering is not easy, and the approach in [16] is the same as that in [3], that is, to order the vertices by degree and vertex ID, and then find the local minimum set L. In comparison, our method is to order the vertices in the minimum vertex cover before those not in the cover, and by doing this we are likely to get a larger local minimum set.

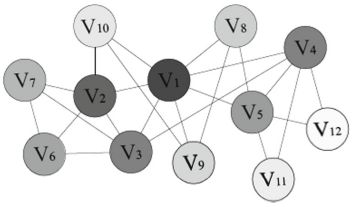


Fig. 3. Example graph in [16]

This is verified in our experiments where the average number of labels of each vertex using our approach is significantly smaller then using the degree-based ordering. To further illustrate this point, we provide the following example.

Example 2. Consider the graph in Fig. 3, which is the graph in the running example of [16]. If the vertices are ordered using degree and vertex ID, then $\{v_7, v_{10}, v_{11}, v_{12}\}$ is the local minimum set (Note that v_8 is not local-minimum). So in computing the PLL labels only the other 8 vertices need to be used as landmarks. For the same graph, we can see that the vertices $v_1, v_2, v_3, v_4, v_5, v_6, v_9$ form a vertex cover. If we rank these vertices before others, then v_8 will be local minimum as well. In our approach we only need to use these 7 vertices as landmarks when computing the PLL labels.

Finding a Minimum Vertex Cover. It is well known that finding a minimum vertex cover is an NP-hard problem. However, we can use the following greedy heuristic algorithm to find an approximate minimum vertex cover.

Algorithm 1: FindMVC

Input: Graph G
Output: A vertex cover C of G, initially $C = \emptyset$
1 **while** *there is edge in* G **do**
2 $u \leftarrow$ a vertex in G with largest degree
3 $C \leftarrow C \cup \{u\}$
4 Remove u and all edges incident on u from G
5 Return C

Suppose we have found a minimum vertex cover C. When computing the PLL labels using C, we arrange the vertices in C according to non-increasing degree.

Removing 0-Distance Labels. To further reduce the number of labels, we note that distance 0 labels can be eliminated if we slightly modify the algorithm for computing the distance. Specifically, when computing the distance $d(s, t)$, we first check whether t is a hub of s or s is a hub of t, if yes, then the distance

$d(s,t)$ is recorded in $L(s)$ or $L(t)$ already, so we can directly retrieve it; otherwise we use formula (1).

For example, we can remove all labels of the form $(v,0)$ from the labels in Example 1. To compute $d(v_4,v_7)$, we find that v_4 is a hub of v_7, that is, $(v_4,3) \in L(v_7)$, therefore, we know $d(v_4,v_7)=3$. However, for $d(v_3,v_6)$, neither v_3 is a hub of v_6, nor v_6 is a hub of v_3. Therefore, we use Eq. (1), and find $d(v_3,v_6)$=$d(v_3,v_4)+d(v_4,v_6)$=$1+2=3$.

Although the above idea is simple, for large graphs with millions of vertices, the amount of savings by eliminating all 0-distance labels can be significant.

4 Combining Two Types of Labels - a Hybrid Index

Consider a graph that consists of a single path v_0,v_1,v_2,v_3,v_4. Observe that using PLL, even with a minimum vertex cover $\{v_1,v_3\}$, we will create a label where v_1 is a hub of every other vertex, and v_3 is a hub of v_2,v_3,v_4. However, we can regard v_0 as a landmark, and create a label L' for every other vertex as follows: $L'(v_i)=i$ (note $i=d(v_i,v_0)$) for $i\in[0,4]$. To find the distance between v_i and v_j, we can use $d(v_i,v_j)=|d(v_j,v_0)-d(v_i,v_0)|$. Here v_0 is a landmark that can be used differently than the 2-hop labels discussed in Sect. 2. We can also use any other vertex as the landmark. For example, if we use v_3 as the landmark, we can compute the distances as follows: for v_i and v_j, if the two vertices lie on the same side of v_3, we use difference, for instance $d(v_0,v_2)=d(v_0,v_3)-d(v_2,v_3)$; if the two vertices lie on different sides of v_3, we use addition, for instance, $d(v_2,v_4)=d(v_2,v_3)+d(v_3,v_4)$. In other words, we can use either addition or difference to compute the difference between two vertices.

More generally, consider a tree T rooted at vertex r. For each vertex v in T, we only need to record its distance $d(v,r)$ from the root r. Given any two vertices v_i,v_j in T, we can compute their distance as follows: (1) find the lowest common ancestor of u,v, denoted $lca(u,v)$, (2) $d(u,v)=d(u,lca(u,v))+d(v,lca(u,v))$ where $d(u,lca(u,v))=d(u,r)-d(r,lca(u,v))$ and $d(v,lca(u,v))=d(v,r)-d(r,lca(u,v))$. In other words,

$$d(u,v)=d(u,r)+d(v,r)-2d(r,lca(u,v)) \tag{2}$$

Thus for a tree, it is sufficient to use its root as the landmark, and record the distances between each vertex from the root. Note that if u is an ancestor of v, then $lca(u,v)=u$.

Now consider a general graph G. We note that G can be decomposed into a 2-core and a set of trees, where the root of each tree is a vertex in the 2-core [4]. Let V_c be the vertex set of the 2-core, $G(V_c)$ be the 2-core, and $T_1,\ldots,T_k$ be the distinct trees, and r_i be the root of tree T_i. Note that for any pair of vertices within $G(V_c)$, the shortest path will not pass through any non-root vertex in the trees. Based on this observation, we design a hybrid label index for G as follows:

1. Construct the PLL label $L(u)$ for each vertex $u\in V_c$ for the graph $G(V_c)$ as described in Sect. 3, ignoring the vertices not in V_c.

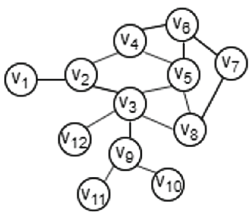
2. Construct a label for each $u \in T_i$, denoted as $L'(u)$, such that $L'(u) = d(r_i, u)$ for $i \in [1, k]$.

When it comes to query processing, we can use the process described in Algorithm 2. The algorithm is self-explanatory. We will use the following following example to further explain it.

Algorithm 2: Computing distance using hybrid landmark labeling

Input: Graph G, label L for $G(V_c)$ and L' for the tree vertices, and two vertices $s, t \in V$
Output: $d(s, t)$

if *both* s *and* t *are in* V_c **then**
  Compute $d(s, t)$ using $L(s)$, $L(t)$ and equation (1)
else if $s \in V_c$ *and* $t \in V(T_i) - \{r_i\}$ **then**
  $d(s, t) = d(s, r_i) + L'(t)$ where $d(s, r_i)$ is computed using equation (1)
else if $s, t \in V(T_i)$ **then**
  Compute $d(s, t)$ using equation (2)
else if $s \in T_i$, $t \in T_j$, *and* $i \neq j$ **then**
  $d(s, t) = L'(s) + d(r_i, r_j) + L'(t)$ where $d(r_i, r_j)$ is computed using equation (1)
Return $d(s, t)$



(a) Graph G_2

	v_5	v_2	v_6	v_8
v_2	2	0		
v_3	1	1		1
v_4	1	1	1	
v_5	0			
v_6	1	2	0	
v_7	2	3	1	1
v_8	1	2		0

(b) Label L

$d(v_2, v_1)=1$
$d(v_3, v_9)=1$
$d(v_3, v_{11})=2$
$d(v_3, v_{10})=2$
$d(v_3, v_{12})=1$

(c) Label L'

Fig. 4. Graph G_2 and its hybrid labels

Example 3. Consider the graph G_2 in Fig. 4 (a). The graph can be decomposed into a core, denoted $G_2[V_c]$, consisting of the vertices v_2 to v_8 and two trees, denoted T_1 and T_2, rooted at v_2 and v_3 respectively. For the core $G_2[V_c]$, we find a minimum cover $\{v_5, v_2, v_6, v_8\}$, and compute the PLL labels as shown in Fig. 4 (b). The tree vertex label L' is as shown in Fig. 4 (c).

Consider the distance between vertices v_6 and v_{11}. We have $d(v_{11}, v_6) = d(v_1 1, v_3) + d(v_3, v_6) = 2 + (1+1) = 4$. Similarly $d(v_1, v_{10}) = d(v_2, v_1) + d(v_2, v_3) + d(v_3, v_{10}) = 1 + 1 + 2 = 4$

It is observed that many real-world graphs are of a core-periphery structure [22], where the core consists of a central, densely connected set of vertices, and the periphery is the set of sparsely connected vertices that are also linked to the core. The periphery part of such graphs is likely to contain many tree vertices. For those graphs that do contain a large forest, our hybrid label index can significantly reduce the index size. For example, for the `DBpedia`, `Email-enron`, and `DBLP` data sets, using core-forest decomposition and our hybrid index can reduce the average number of labels per vertex by 45.5%, 22% and 14% respectively, as shown in Table 2 of our experiments.

Finding a Core-Forest Decomposition. To find the core, we start with a vertex of degree one, remove it and all incident edges, and repeat this process until all vertices have a degree of 2 or more. This will obtain the core. We can then identify the root vertices of the forest, which are vertices in the core that are neighbors of at least one vertex not in the core. The forest (trees) can then be obtained using BFS traversal over vertices not in the core.

Complexity. Finding the core-forest Sdecomposition can be done in time $O(|V| + |E|)$. Building the label L' for a tree T can be done by a single BFS from the root of the tree, hence it takes $O(|T|)$ where $|T|$ represents the tree size.

5 Experiments

In this section, we report our experiments with the proposed methods (i.e., the vertex cover as landmarks and the hybrid-index with core-forest decomposition). We compare the label set size and query time of the original PLL where the vertices are ordered according to decreasing degree, with the following methods:

PLL$^+$. PLL using landmarks in an approximate minimal vertex cover.
HLL. The hybrid landmark labeling with the index for core computed using an approximate minimal vertex cover of the core.

Datasets. We use 16 real-world graphs as listed in Table 1. The size of each data set is also shown in the table. `Wiki-croc`, `wiki-charm` and `wiki-squi` are Wikipedia page-page networks on specific topics in December 2018. `Github` is the GitHub author-follower network collected in June 2019. `Ro` and `Hu` are friendship networks of users from 2 European countries collected from the music streaming service Deezer in November 2017. `DBLP` is an co-authorship network where two authors are connected if and only if they publish at least one paper together. `Email-enron` is an email communication network of Enron. `DBpedia` is a snapshot of the DBpedia knowledge graph downloaded from http://pan.baidu.com/s/1c00Jq5E. The remaining datasets are the blue verified Facebook page

networks of different categories collected in November 2017. All data sets, except `DBpedia`, are downloaded from the Stanford Large Network Data Collection (https://snap.stanford.edu/data/). We would like to stress that, although most of these data sets are not large, they can still provide a good indication of the ratio of index size/time reduction by using our techniques.

Implementation Environment. All algorithms are implemented in C++ and compiled with gcc 7.4.0 compiler. The experiments are done in a machine with Intel Core i7-7700 with 3.60 GHz CPU, 32 GB memory and Linux (Ubuntu 18.04.3 LTS) operating system.

Table 1 also lists the number of vertices in the approximate minimum vertex cover ($|C_G|$), the number of vertices in the core $|V_{core}|$, the percentage of tree vertices, and the average tree size (number of vertices in the tree), as well as the number of vertices in the approximate minimum vertex cover of the core $|C_{core}|$. It can be seen that the percentage of minimal vertex cover vertices ranges from 0.26% to 0.75%. The percentages of tree vertices is about 35%, 31.1%, and 19% for `DBpedia`, `Email-enron`, `Company` respectively, while they are less than 10% for several other data sets. The average sizes of the tree, shown as $|t|$, are all small except for email-enron wiki-croc, facebook, and DBpedia.

Table 1. Datasets

Dataset	$\|V\|$	$\|E\|$	$\|C\|$	Tree node ratio	Avg. tree size	$\|V_{core}\|$	$\|C_{core}\|$
Wiki-croc	11631	180020	3037	0.044	5.72	11116	3033
Wiki-cham	2277	36101	810	0.041	3.2	2184	806
Wiki-squi	5201	217073	2147	0.027	2.42	5063	2143
Github	37700	289003	15511	0.139	1.76	32464	14775
RO	41773	125826	22726	0.153	1.33	35402	20831
HU	47538	222887	29535	0.06	1.13	44669	28902
Politician	5908	41729	3015	0.109	1.68	5262	2917
Athletes	13866	86858	6889	0.092	1.53	12590	6764
Company	14113	52310	7078	0.188	1.68	11457	6515
TVshow	3892	17262	1994	0.179	1.73	3194	1849
New-sites	27917	206259	15862	0.082	1.42	25626	15460
Government	7057	89455	4271	0.053	1.33	6681	4210
Facebook	4039	88234	3038	0.019	7.5	3964	3038
DBLP	317080	1049866	165229	0.143	1.77	271646	160824
Email-enron	36692	183831	14480	0.311	16.18	25286	13504
DBpedia	3365623	7989191	700756	0.35	4	2188839	457547

Table 2. Index size and index time

Dataset	PLL			PLL^+			HLL		
	Index size(KB)	Avg. $\|L(v)\|$	Index time(s)	Index size(KB)	Avg. $\|L(v)\|$	Index time(s)	Index size(KB)	Avg. $\|L(v)\|$	Index time(s)
Wiki-croc	1739.8	51.1	11	1274.9	37.4	8	1244	36.5	8
Wiki-cham	225.3	33.8	<1	177.4	26.6	<1	172.84	25.9	<1
Wiki-squi	1351.8	88.7	18	1032.8	67.8	11	1014.45	66.6	11
Github	6166.8	55.8	18	5117.5	46.3	76	4536.9	41	71
RO	72830	595.1	3722	49320.6	403	2592	41722.3	340.9	2255
HU	174318	1252	25262	110914	796.4	15515	103714.5	744.7	14808
Politician	1695.3	97.94	11	903.9	52.2	5	828.5	47.9	5
Athlete	5959.6	146.7	100	3987.8	98.2	66	3666.9	90.3	64
Company	5253.8	127.1	49	3557.5	86	35	2946.9	71.3	31
TVshow	951.4	83.4	3	548.2	48.1	1	456	40	1
New-sites	18699.9	228.6	555	11987	146.6	354	11085.5	135.5	343
Government	2544.4	123.1	36	1819	88	26	1720.1	83.2	25
Facebook	330.41	27.9	1	292	24.7	1	291.5	24.6	1
DBLP	672804.1	724.3	34795	469598.4	505.5	25973	402638.7	433.4	24826
Email-enron	4490.1	62.16	65	3255.1	44.9	51	2547.6	35.1	40
DBpeida	69568	10.1	526	45255	6.4	524.6	260.8	3.5	217.8

5.1 Index Size and Index Time

Table 2 shows the index size in both space consumption (in KB) and average number of labels per vertex (avg. $|L(v)|$). The index time, which includes the time for finding the approximate minimum vertex cover and the core-forest decomposition, is also shown in the table[2]. As can be seen, using the minimal vertex cover as landmarks alone can significantly reduce the index size, with the *Politician* and *TVshow* datasets achieving 46.7% and 42.4% reduction in the index size respectively. Using core-forest decomposition further reduces the index sizes for all of the data sets, with the `DBpedia` and `Email-enron` datasets achieving 45.5% and 21.8% reduction respectively from using the vertex cover alone. Note that the `DBpedia` and the `Email-enron` datasets have relatively large percentage of tree vertices. The index size of HLL is significantly smaller than that of PLL for all data sets, for instance, the index size of HLL is 37.4% of that of PLL for the `DBpedia` dataset. The index time of HLL is also much shorter than that of PLL for most datasets (for some of the small datasets, the index time of all methods are similar).

5.2 Query Time

To test the query time, we randomly generated 1 million vertex pairs for each data set. The average query time for each data set is shown in Table 3. As can bee seen from the table, the average query time for PLL^+ is significantly shorter than that of the original PLL for all data sets. However, the query time for PLL^+ and HLL are similar. This could be because of the extra query processing steps used in HLL offset the gain obtained by the smaller index size.

[2] For most datasets, the time spent on finding the minimum vertex cover and core-forest decomposition is small compared with that of PLL. One exception in our experiments is the `DBpedia`, for which computing the vertex cover too significant amount of time.

Table 3. Query time

Dataset	PLL	PLL$^+$	HLL
Wiki-croc	154	109.11	113.87
Wiki-cham	11.43	9.18	9.52
Wiki-squi	92.29	67.13	68.93
Github	545.74	453.78	464.92
RO	7740.05	5434.42	5271.85
HU	21678.9	14090.3	12547.2
Politician	140.58	76.21	78.38
Athletes	553.63	374.85	374.69
Company	493.42	331.45	331.21
TVshow	75.63	43.8	46.26
New-sites	1823.71	1161.12	1142.11
Government	200.57	150.33	151.6
Facebook	18.11	14.21	15.08
DBLP	87079.9	59721.2	58832.6
Email-enron	696.3	505	498.6
DBPedia	6978	5890.2	5875.4

6 Related Work

The problem of finding point-to-point shortest distances in graphs is one of the most basic, and most studied, problems in graph databases. Algorithms for the problem can be divided into exact (e.g., [3,5,9,10,15,16,26]) and approximate (e.g., [11,18,19,23,24]), or grouped by the targeted graph types such as road-networks [1,2,28,29], small-world networks [3,16] and general graphs [5,26]. The exact algorithms can be further divided into tree-decomposition based [26,27], 2-hop index based [3] or multi-hop index based [5,26]. There are also works that focus on dynamic graphs and incremental maintenance of indexes [13,14,20]. As noted in recent studies [16,17], overall the pruned landmark labeling (PLL) [3] and its variants (such as [2] and [17], to be discussed in some detail below) are the best-performing.

It was noted in [3] that vertex ordering plays an important role in the label size. However, other than the heuristic degree-based ordering, no specific vertex ordering method was given. The degree-based labeling heuristic sometimes generates labels that are too large, especially for graphs that have "highway" structures that resemble those in real-world road networks. Based on this observation [2] proposed *pruned highway labeling* which first partitions the vertices into $\{P_1, P_2, \ldots, P_N\}$ where each P_i is a shortest path $p_{i,1}, p_{i,2}, \ldots, p_{i,l_i}$ between $p_{i,1}$ and p_{i,l_i} (intuitively these shortest paths represent the "highways"), and then the labels of node v, $label(v)$, are changed to triples of the form

$(i, d(p_{i,1}, p_{i,j}), d(v, p_{i,j}))$, and the way to compute the distance between s and t is also slightly changed accordingly. For each vertex v, it computes $label(v)$ using BFS from the vertices $p_{i,j}$ one by one, with similar prune strategy as that in [3]. To get a good ordering of vertices in $\{P_1, P_2, \ldots, P_N\}$, it first computes a shortest path tree rooted at a randomly selected vertex, then starting from the root, the next vertex is selected as the child with the largest number of descendants. If v and its child w are both selected, and the number of descendants of v and w are similar, then it will *skip* v and only do BFS from w. The authors of [17] proposes an idea similar to [2] for ordering the vertices in PLL, specifically it uses the multiplication of vertex degree and difference in descendant size to rank the vertices. Experiments conducted in [17] demonstrate the vertex ordering strategies in [2] and [17] can lead to good improvement in label size and query time over degree-based labeling. [9] proposes a highway cover labelling where a set H of landmarks (called highway nodes) is chosen, the pairwise distance between the vertices in H are pre-computed, and given any two vertices s and t outside H, the distance of the shortest path between s and t that pass through some vertex in S can be computed directly using the labels of s, and this distance is used as an upper bound of $d(s, t)$. Although the index size is much smaller than PLL, the actual shortest distance needs to be computed using a two-way BFS over the graph obtained from $G - H$, guided by the upper bound. More recently, [16] proposes a way to parallelize the PLL label computation by doing BFS from multiple vertices simultaneously, based on the observation that PLL label computation can be ordered by distance instead of vertices. To make the label size smaller, it proposes to skip local-minimum nodes (which themselves are based on ranking the vertices in non-increasing degrees). [15] proposes a 2-hop labeling technique that is I/O efficient.

To the best of our knowledge, none of the previous works has considered using a minimum vertex cover as potential landmarks, and none of them has used core-forest decomposition and a combination of two types of labels. The vertex ordering techniques proposed in [2] and [17] may be combined with our techniques in computing the PLL labels in the core (by ordering the vertices in the vertex cover using the techniques in [17]) in order to further reduce the index size.

7 Conclusion

In this paper, we first proposed to use the vertex cover as the landmarks, which can significantly reduce the index size of PLL. This method can be viewed as a method to order the vertices in the computation of PLL labels, that is, to order the vertices in a minimum vertex cover before those that are not in the cover. Query time is also significantly shorter due to the smaller label size. We then proposed a hybrid index based on core-forest decomposition, where vertices in the core use the PLL labels while the vertices in the forest use a simpler labeling scheme. For datasets where there are large percentages of forest vertices, the hybrid index can further significantly reduce index size, without compromising query performance. These

techniques not only works for undirected and unweighted graphs, they can also be applied to directed graphs and weighted graphs.

Acknowledgement. This work is partly supported by The Science and Technology Development Fund, Macau SAR, with the code FDCT-SKL-IOTSC-2018–2020 and FDCT/0045/2019/A1.

References

1. Abraham, I., Delling, D., Goldberg, A.V., Werneck, R.F.: A Hub-based labeling algorithm for shortest paths in road networks. In: Pardalos, P.M., Rebennack, S. (eds.) SEA 2011. LNCS, vol. 6630, pp. 230–241. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20662-7_20
2. Akiba, T., Iwata, Y., Kawarabayashi, K., Kawata, Y.: Fast shortest-path distance queries on road networks by pruned highway labeling. In: 2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2014, Portland, Oregon, USA, 5 January 2014, pp. 147–154 (2014)
3. Akiba, T., Iwata, Y., Yoshida, Y.: Fast exact shortest-path distance queries on large networks by pruned landmark labeling. Proc. ACM SIGMOD Int. Conf. Manag. Data **2013**, 349–360 (2013)
4. Bi, F., Chang, L., Lin, X., Qin, L., Zhang, W.: Efficient subgraph matching by postponing cartesian products. In: Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26–July 01 2016, pp. 1199–1214 (2016)
5. Chang, L., Yu, J.X., Qin, L., Cheng, H., Qiao, M.: The exact distance to destination in undirected world. VLDB J. **21**(6), 869–888 (2012). https://doi.org/10.1007/s00778-012-0274-x
6. Cohen, E., Halperin, E., Kaplan, H., Zwick, U.: Reachability and distance queries via 2-hop labels. SIAM J. Comput. **32**(5), 1338–1355 (2003)
7. Delling, D.: Route planning in transportation networks: from research to practice. In: Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2018, Seattle, WA, USA, 06–09 November 2018, p. 2. ACM (2018)
8. Fan, W., Li, J., Ma, S., Wang, H., Wu, Y.: Graph homomorphism revisited for graph matching. PVLDB **3**(1), 1161–1172 (2010)
9. Farhan, M., Wang, Q., Lin, Y., McKay, B.D.: A highly scalable labelling approach for exact distance queries in complex networks. In: Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, 26–29 March 2019, pp. 13–24 (2019)
10. Fu, A.W., Wu, H., Cheng, J., Wong, R.C.: IS-LABEL: an independent-set based labeling scheme for point-to-point distance querying. PVLDB **6**(6), 457–468 (2013)
11. Gubichev, A., Bedathur, S.J., Seufert, S., Weikum, G.: Fast and accurate estimation of shortest paths in large graphs. In: Proceedings of the 19th ACM Conference on Information and Knowledge Management, CIKM 2010, pp. 499–508 (2010)
12. Haryanto, A.A., Islam, M.S., Taniar, D., Cheema, M.A.: Ig-tree: an efficient spatial keyword index for planning best path queries on road networks. World Wide Web **22**(4), 1359–1399 (2019)

13. Hassan, M.S., Aref, W.G., Aly, A.M.: Graph indexing for shortest-path finding over dynamic sub-graphs. In: Özcan, F., Koutrika, G., Madden, S. (eds.) Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, June 26 - July 01 June 2016, pp. 1183–1197. ACM (2016)
14. Hayashi, T., Akiba, T., Kawarabayashi, K.: Fully dynamic shortest-path distance query acceleration on massive networks. In: Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, 24–28 October 2016, pp. 1533–1542. ACM (2016)
15. Jiang, M., Fu, A.W., Wong, R.C., Xu, Y.: Hop doubling label indexing for point-to-point distance querying on scale-free networks. PVLDB **7**(12), 1203–1214 (2014)
16. Li, W., Qiao, M., Qin, L., Zhang, Y., Chang, L., Lin, X.: Scaling distance labeling on small-world networks. In: Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, 30 June - 5 July 2019, pp. 1060–1077 (2019)
17. Li, Y., U, L.H., Yiu, M.L., Kou, N.M.: An experimental study on hub labeling based shortest path algorithms. PVLDB **11**(4), 445–457 (2017)
18. Potamias, M., Bonchi, F., Castillo, C., Gionis, A.: Fast shortest path distance estimation in large networks. In: Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, 2–6 November 2009, pp. 867–876. ACM (2009)
19. Qi, Z., Xiao, Y., Shao, B., Wang, H.: Toward a distance oracle for billion-node graphs. PVLDB **7**(1), 61–72 (2013)
20. Qin, Y., Sheng, Q.Z., Falkner, N.J.G., Yao, L., Parkinson, S.: Efficient computation of distance labeling for decremental updates in large dynamic graphs. World Wide Web **20**(5), 915–937 (2016). https://doi.org/10.1007/s11280-016-0421-1
21. Ren, X., Wang, J., Han, W., Yu, J.X.: Fast and robust distributed subgraph enumeration. PVLDB **12**(11), 1344–1356 (2019)
22. Rombach, P., Porter, A.A., Fowler, J.H., Mucha, P.J.: Core-periphery structure in networks (revisited). SIAM review **59**(3), 619–646 (2014)
23. Sadri, A., Salim, F.D., Ren, Y., Zameni, M., Chan, J., Sellis, T.: Shrink: Distance preserving graph compression. Inf. Syst. **69**, 180–193 (2017)
24. Sarma, A.D., Gollapudi, S., Najork, M., Panigrahy, R.: A sketch-based distance oracle for web-scale graphs. In: Proceedings of the Third International Conference on Web Search and Web Data Mining, WSDM 2010, pp. 401–410. ACM (2010)
25. Wang, X., Zhang, Y., Zhang, W., Lin, X.: Efficient distance-aware influence maximization in geo-social networks. IEEE Trans. Knowl. Data Eng. **29**(3), 599–612 (2017)
26. Wei, Fang: Efficient graph reachability query answering using tree decomposition. In: Kučera, Antonín, Potapov, Igor (eds.) RP 2010. LNCS, vol. 6227, pp. 183–197. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15349-5_13
27. Wei-Kleiner, F.: Tree decomposition-based indexing for efficient shortest path and nearest neighbors query answering on graphs. J. Comput. Syst. Sci. **82**(1), 23–44 (2016)
28. Wu, L., Xiao, X., Deng, D., Cong, G., Zhu, A.D., Zhou, S.: Shortest path and distance queries on road networks: an experimental evaluation. PVLDB **5**(5), 406–417 (2012)
29. Zhu, A.D., Ma, H., Xiao, X., Luo, S., Tang, Y., Zhou, S.: Shortest path and distance queries on road networks: towards bridging theory and practice. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, 22–27 June 2013, pp. 857–868. ACM (2013)