Distance Landmarks Revisited for Road Graphs

Shuai Ma[1] Kaiyu Feng[1] Haixun Wang[2] Jinpeng Huai[1]
[1] SKLSDE Lab, Beihang University, China [2] Google Research, USA
{mashuai, fengky, huaijp}@buaa.edu.cn haixun@google.com

***Abstract*—Computing shortest distances is one of the fundamental problems on graphs, and remains a *challenging* task today. *Distance landmarks* have been recently studied for shortest distance queries with an auxiliary data structure, referred to as *landmark covers*. This paper studies how to apply distance landmarks for fast *exact* shortest distance query answering on large road graphs. However, the *direct* application of distance landmarks is *impractical* due to the high space and time cost. To rectify this problem, we investigate novel techniques that can be seamlessly combined with distance landmarks. We first propose a notion of *hybrid landmark covers*, a revision of landmark covers. Second, we propose a notion of *agents*, each of which represents a small subgraph and holds good properties for fast distance query answering. We also show that agents can be computed in *linear time*. Third, we introduce graph partitions to deal with the remaining subgraph that cannot be captured by agents. Fourth, we develop a unified framework that seamlessly integrates our proposed techniques and existing optimization techniques, for fast shortest distance query answering. Finally, we experimentally verify that our techniques significantly improve the efficiency of shortest distance queries, using real-life road graphs.**

I. Introduction

We study the *node-to-node shortest distance* problem on large graphs: given a weighted undirected graph $G(V, E)$ with non-negative edge weights and two nodes of G, the source s and the target t, find the shortest distance from s to t in G. We allow the usage of auxiliary structures generated by preprocessing, but restrict them to have a moderate size (compared with the input graph). In this work, we are only interested in *exact* shortest distances on *large* graphs.

Finding shortest distances, a twin problem of *finding shortest paths*, is one of the fundamental problems on graphs, and has found its usage as a building block in various applications, *e.g.,* measuring the closeness of nodes in social networks and Web graphs [18], [24], [28], and finding the distances between physical locations in road networks [34].

Algorithms for shortest distances have been studied since 1950's and still remain an *active* area of research. The classical one is Dijkstra's algorithm [6] due to Edsger Dijkstra. Dijkstra's original algorithm runs in $O(n^2)$ [7], and the enhanced implementation with Fibonacci heaps runs in $O(n \log n + m)$ due to Fredman & Tarjan [10], where n and m denote the numbers of nodes and edges in a graph, respectively. The latter remains asymptotically the fastest known solution on arbitrary undirected graphs with non-negative edge weights [30].

However, computing shortest distances remains a challenging problem, in terms of both time and space cost, for large-scale graphs such as Web graphs, social networks and road networks. The Dijkstra's algorithm [10] is not acceptable on large graphs (*e.g.,* with tens of millions of nodes and edges) for online applications [24]. Therefore, a lot of optimization techniques have been recently developed to speed up the computation [5], [13], [20], [24], [25], [27], [28], [33], [34].

Distance landmarks (*a.k.a. distance oracles*, see Section II-B for details) are data structures that support efficient shortest distance query answering, and have been recently studied in both theory [23], [30] and practice [24], [26], [28]. An $n \times n$ *triangular matrix* of size $n^2/2$ for all-pair shortest distances can be computed in $O(n^2 \log n + mn)$ time, using Dijkstra's algorithm [10], where n and m are the numbers of nodes and edges, respectively. With the distance matrix, shortest distance queries can be answered in $O(1)$ time. This solution, however, is *not practical* on large graphs: the preprocessing time is too long, and even if one is willing to wait that long, the matrix is too large to be stored effectively. For instance, the matrix of a graph with one million nodes needs about $1,862$ GB memory (here the distance entries are stored as 4-byte integers).

Distance landmarks aim at *striking a balance* between the efficiency benefits of answering shortest distance queries and the time and space cost of computing and storing them. And distance landmarks have already been adopted for answering *approximate* shortest distances [24], [26], [28], [30], and for answering *exact* shortest distances on directed graphs [14], [23]. However, how to apply distance landmarks for answering exact shortest distances on undirected graphs is mainly limited to pure theoretical analyses [30].

Contributions & Roadmap. To our knowledge, we are among the first to study the application of distance landmarks for fast exact shortest distance queries on large undirected graphs.

(1) We develop an approximation algorithm with a constant factor 2 to analyze distance landmarks by establishing connections with vertex covers (Section III), based on which we show that the *direct* application of distance landmarks is not practical for large-scale graphs. We then propose *hybrid landmark covers*, a revised notion of traditional landmark covers, to reduce the space cost (Section III).

(2) We propose a notion of *agents* such that each agent represents a small subgraph, referred to as *deterministic routing areas* (DRAs) (Section IV). Then landmarks are only built for agents, instead of the entire graph. Hence, both space and time cost are reduced. We give an analysis of agents and DRAs, based on which we develop a linear time algorithm for computing DRAs along with their maximal agents. As shown in the experimental study, on average about 1/3 nodes of a graph are captured by agents and their DRAs.

(3) We introduce the *bounded graph partitioning* problem (BGP) to deal with the remaining subgraph that cannot be captured by the DRAs of agents, and show that the problem is NP-complete (Section V). We then propose a notion of SUPER graphs that combine graph partitions with hybrid landmark covers to support efficient shortest distance answering. We also build connections between the traditional graph partitioning problem and the BGP problem, and utilize the traditional graph partitioning approaches, *e.g.,* METIS, to solve the problem. As shown by the experiments, METIS works well.

(4) We propose a unified framework DISLAND for fast shortest distance query answering (Section VI), which seamlessly combines distance landmarks with agents, graphs partitions (SUPER graphs), and existing speed-up techniques [32], [34].

(5) Using real-life large road graphs, we conduct an extensive experimental study (Section VII). We find that our DISLAND scales well with large graphs, *e.g.,* it takes 0.28×10^{-3} seconds on graphs with 2.4×10^7 nodes and 5.7×10^7 edges. Moreover, DISLAND is 9.4, 134.9, and $14,540.1$ times faster than CH [13], ARCFLAG [22], and bidirectional Dijkstra [20], respectively. Moreover, the auxiliary structures occupy only a moderate size of space (about $1/2$ of the input graphs), and can be pre-computed efficiently.

Due to the space constraint, we defer all the proofs to [11].

Related work. (1) Algorithms for node-to-node shortest distances have been extensively studied since 1950's, and fall into different categories in terms of different criteria:

- exact distances [4], [5], [7], [10], [13], [14], [20], [23], [23], [25], [27], [29], [32], [34] and approximate distances [24], [26], [28], [30];
- memory-based [7], [10], [13], [20], [23]–[30], [32]–[34] and disk-based algorithms [4], [5];
- for unweighted [24], [28], [33] and weighted graphs [4], [5], [7], [10], [13], [14], [20], [23], [23], [25]–[27], [29], [30], [32], [34]; and
- for directed [14], [23], [29] and undirected graphs [4], [5], [7], [10], [13], [20], [23]–[28], [30], [32]–[34].

In this work, we study the memory-based exact shortest distance problem on weighted undirected large real-world graphs. None of the previous work has *experimentally* studied how to apply distance landmarks for solving this problem.

(2) Distance landmarks have been recently investigated for *approximate* shortest distance queries [24], [26], [28], [30], and for answering *exact* shortest distances on directed graphs [14], [23]. However, how to apply distance landmarks for answering exact shortest distances on undirected graphs is mainly limited to pure theoretical analyses [30]. Nevertheless, in this work, we investigate how to utilize distance landmarks to speed-up shortest distance queries on real-life large road graphs.

(3) There has recently been extensive work on speed-up techniques for shortest distance queries: bidirectional search [20], hierarchical approaches [13], node and edge labeling [22], [27] and shortcuts [25] (see [32], [34] for two recent surveys). These techniques are complementary to our work, and can be incorporated into our approach. We have indeed seamlessly integrated the CH [13] and ARCFLAG [22] techniques with distance landmarks into our framework.

(4) Graph partitioning has been extensively studied since 1970's [16], [17], [35], and has been used in various applications, *e.g.,* circuit placement, parallel computing and scientific simulation [35]. The graph partitioning problem considered in this work differs from the traditional one that it concerns more on the number of nodes with edges across different partitions, instead of the number of edges with endpoints across different partitions. Nevertheless, we build connections between these two problems, and make use of the existing approaches, *e.g.,* METIS [16], to solve the graph partitioning problem considered in this work. It is also worth mentioning that graph partitioning has already been used to speed-up Dijkstra's algorithm [22].

(5) Agents and deterministic routing areas proposed in this study (Section IV) are significantly different (from definitions to analyses to algorithms) from the 1-dominator sets proposed in [29]. Moreover, the latter are for shortest path queries on nearly acyclic directed graphs, which is not appropriate for real-life large graphs, as these graphs typically contain a large strongly connected components [2].

II. PRELIMINARY

In this section, we first present basic notations of graphs. We then introduce the notion of distance landmarks.

A. Graph Notions

We first introduce graphs and the related concepts.

Graphs. A *weighted undirected graph* (or simply a *graph*) is defined as $G(V, E, w)$, where (1) V is a finite set of nodes; (2) $E \subseteq V \times V$ is a finite set of edges, in which (u, v) or (v, u) denotes an undirected edge between nodes u and v; and (3) w is a total weight function that maps each edge in E to a positive rational number.

We will simply denote $G(V, E, w)$ as $G(V, E)$ when it is clear from the context.

Subgraphs. Graph $H(V_s, E_s, w_s)$ is a *subgraph* of graph $G(V, E, w)$ if (1) for each node $u \in V_s$, $u \in V$, and, moreover, (2) for each edge $e \in E_s$, $e \in E$ and $w_s(e) = w(e)$. That is, H contains a subset of nodes and a subset of edges of G.

We also denote subgraph H as $G[V_s]$ if E_s is *exactly* the set of edges appearing in G over V_s.

Paths and cycles. A *simple path* (or simply a *path*) ρ is a sequence of nodes $v_1/\ldots/v_n$ with no repeated nodes, and, moreover, for each $i \in [1, n-1]$, (v_i, v_{i+1}) is an edge in G.

A *simple cycle* (or simply a *cycle*) ρ is a sequence of nodes $v_1/\ldots/v_n$ with $v_1 = v_n$ and no other repeated nodes, and, moreover, for each $i \in [1, n-1]$, (v_i, v_{i+1}) is an edge in G.

The *length* of a path or cycle ρ is the sum of the weights of its constituent edges, *i.e.,* $\sum_{i=1}^{n-1} w(v_i, v_{i+1})$.

We say that v_{i+1} (resp. v_i) is a *neighbor* of v_i (resp. v_{i+1}).

We also say that a node is *reachable* to another one if there exists a path between these two nodes.

Shortest paths and distances. A *shortest path* from one node u to another node v is a path whose length is minimum among all the paths from u to v.

The *shortest distance* between nodes u and v, denoted by $\mathsf{dist}(u, v)$, is the length of a shortest path from u to v.

Connected components. A *connected component* (or simply a CC) of a graph is a subgraph in which any two nodes are connected by a path, and is connected to no additional nodes. A graph is connected if it has exactly one connected component, consisting of the entire graph.

Cut-nodes and bi-connected components. A *cut-node* of a graph is a node whose removal increases the number of connected components in the graph.

A *bi-connected component* (or simply a BCC) of a graph is a subgraph consisting of a maximal set of edges such that any two edges in the set must lie on a common simple cycle.

B. Distance Landmarks

We next introduce the notion of distance landmarks [24].

Consider an ordered set of l vertices $D = < x_1, \ldots, x_l >$ such that for each $i \in [1, l]$, x_i is a distinct node in graph G.

We say that D is a *landmark cover* of graph G if and only if for any node pair (u, v) in G with u reachable to v, there exists a *landmark* x_i ($1 \leq i \leq l$) in D such that the shortest distance $\mathsf{dist}(u, v) = \mathsf{dist}(u, x_i) + \mathsf{dist}(x_i, v)$. This is achieved by representing each node in G as a vector of shortest distances to the set of landmarks in D. More specifically, each node $u \in V$ is represented as an l-dimensional vector $\mathsf{distVec}(u)$:

$$\mathsf{distVec}(u) = < \mathsf{dist}(x_1, u), \ldots, \mathsf{dist}(x_l, u) >.$$

The LMC problem is to find a landmark cover with a minimum number of landmarks in a graph. The problem is unfortunately intractable, as shown below.

Proposition 1: *The* LMC *problem is* NP*-complete [24].* □

To reduce its computational complexity, an $O(\log n)$-approximation algorithm was proposed by using the approximation algorithms for the *set cover* (SC) problem [24]. This algorithm, however, runs in cubic time, and cannot be directly used for large graphs, as already been observed in [24].

Remarks. (1) With a landmark cover D, the exact shortest distance $\mathsf{dist}(u, v)$ for any node pair (u, v) can be computed in $O(|D|)$ time, where $|D|$ is the number of landmarks in D. This is obvious as $\mathsf{dist}(u, v) = \mathsf{min}\{\mathsf{dist}(u, x_i) + \mathsf{dist}(x_i, v) \mid x_i \in D\}$. (2) As a landmark cover D occupies $|D|$ ($|V| - 1$) space, its size $|D|$ must be small in order to apply it on large graphs.

III. Distance Landmarks Revisited

In this section, we first show that it is not practical to *directly* utilize landmark covers due to the high space cost. We then propose a notion of *hybrid landmark covers* to alleviate this problem. Here we consider a graph $G(V, E, w)$.

A. Landmark Covers

To give a more accurate estimation of landmark covers, we develop an approximation algorithm with a constant factor 2. Recall that the SC based algorithm (Section II-B, [24]) has an approximation factor of $O(\log n)$. To do this, we first present a notion of redundant-edge-free (REF) graphs. We then build the relationship between the LMC problem and the clasical *vectex cover* (VC) problem on REF graphs, which leads to a 2-approximation algorithm. Finally, we evaluate the cost of landmark covers with the approximation algorithm.

Input: A weighted undirected graph $G(V, E, w)$.
Output: A landmark cover D of G.
1. Remove redundant edges from G;
2. Compute a vertex cover D of G;
3. **return** D.

Figure 1. 2-approximation algorithm for computing landmark covers

A *vertex cover* of a graph is a set of nodes such that each edge of the graph is incident to at least one node of the set. The VC problem is to find a minimum set of vertex covers, a classical optimization problem known to be NP-complete [12].

Graphs often contain redundant edges when distance queries are concerned. Graph G is *redundant-edge-free* (REF) if it contains no *redundant* edges, where an edge (u, v) is redundant if its removal has no effects on the shortest distance $\mathsf{dist}(u, v)$.

By the definition of REF graphs above, it is trivial to see that REF graphs preserve shortest distances, and that a graph may have multiple REF graphs. We next build the relationship between landmark covers and vertex covers, stated as follows.

Theorem 2: *For any* REF *graph G, a set S of nodes is a landmark cover of G iff S is a vertex cover of G.* □

As a consequence, the LMC problem is identical to the VC problem on REF graphs.

Approximation algorithm. It is well-known that the VC problem has a 2-approximation algorithm [31], which basically computes a *maximal matching* of a graph by greedily picking edges and removing all endpoints of the picked edges [6]. Following from Theorem 2, we obtain a 2-approximation algorithm for the LMC problem, presented in Fig. 1.

Given a graph $G(V, E)$, the algorithm first computes an REF graph of G by removing redundant edges (line 1). It then computes a vertex cover D of the REF graph (line 2), and simply returns D as a landmark cover of G (line 3).

Note that testing whether an edge (u, v) is redundant in a graph $G(V, E, w)$ is typically efficient. When computing $\mathsf{dist}(u, v)$ using Dijkstra's algorithm on graph $G(V, E \setminus \{(u, v)\}, w)$, if $\mathsf{dist}(u, v') > w(u, v)$ for any node v' before reaching v, it is easy to verify that (u, v) is not a redundant edge. Moreover, for a large portion of edges (u, v), its weight $w(u, v)$ is exactly the shortest distance $\mathsf{dist}(u, v)$ in real-life graphs such as road networks. Hence, our VC based algorithm is typically much faster than the SC based algorithm [24], though they have the same time complexity.

Remarks. The 2-approximation algorithm allows us to have both lower and upper bounds for the sizes and space cost of landmark covers. If the algorithm returns a landmark cover D, then the lower and upper bounds for the size of the *optimal landmark cover* are $|D|/2$ and $|D|$, respectively.

Findings on landmark covers. We next experimentally test the overhead of landmark covers with our approximation algo-

Table I
OVERHEAD OF LANDMARK COVERS VS. ORIGINAL GRAPHS

Graphs $G(V, E)$		Landmark covers D				
name	size (MB)	$\leq \lvert D\rvert \leq$	$\leq \frac{\lvert D\rvert}{\lvert V\rvert} \leq$ (%)	$\leq$ size $\leq$ (GB)	$\leq \frac{\text{size}(D)}{\text{size}(G)} \leq$	time (s)
CO	9.62	[181,276, 362,552]	[41.6, 83.2]	[588.42, 1,176.83]	$[6.27 \times 10^4,\ 1.25 \times 10^5]$	34.1
FL	24.59	[447,486, 894,972]	[41.8, 83.6]	[3,568.67, 7,137.33]	$[1.49 \times 10^5,\ 2.97 \times 10^5]$	391.8
CA	42.54	[761,662, 1,523,324]	[40.3, 80.6]	[10,730.05, 21,460.09]	$[2.58 \times 10^5,\ 5.17 \times 10^5]$	1,205.6
E-US	80.17	[1,450,115, 2,900,230]	[40.3, 80.6]	[38,880.24, 77,760.48]	$[4.97 \times 10^5,\ 9.93 \times 10^5]$	4,315.3
W-US	139.24	[2,545,995, 5,091,990]	[40.7, 81.3]	[118,786.74, 237,573.47]	$[8.74 \times 10^5,\ 1.75 \times 10^6]$	12,984.3
C-US	312.10	[5,811,428, 1,1622,856]	[41.3, 82.5]	[609,721.69, 1,219,443.38]	$[2.00 \times 10^6,\ 4.00 \times 10^6]$	66,996.9
US	531.63	[9,737,381, 19,474,762]	[40.7, 81.3]	[1,737,359.48, 3,474,718.95]	$[3.35 \times 10^6,\ 6.69 \times 10^6]$	196,194.6

rithm. We tested seven real-life datasets from [8] (please refer to Section VII for details about the datasets and experimental settings). We adopted the adjacency-list representation [6] for graphs when counting their space cost, and assumed that nodes and distances were stored as 4-byte integers.

The experimental results shown in Table 1 tell us that:

(1) The size of an optimal landmark cover is large, and typically 40%–80% of the nodes in a graph are landmarks.

(2) The space cost of a landmark cover is huge, and is typically more than 10^4–10^6 times of the graph itself. For instance, the landmark cover of the US graph with $1/2$ GB space may incur a space cost of more than 1.74×10^6 GB.

(3) Computing landmark covers of large graphs is inefficient. It took our algorithm more than 2 days 6 hours on the US dataset. It is worth mentioning that here we only compute the landmarks nodes, not including computing the shortest distances between graph nodes and landmarks. Furthermore, the directly usage of SC based algorithm [24] is even worse, due to its high space and time cost (it even runs out of memory –16GB– for the smallest CO dataset on our testing machine).

Hence, it suffices to conclude that the direct application of distance landmarks as [24] is impractical for large graphs.

B. Hybrid Landmark Covers

The naive matrix approach stores the pre-computed all-pair shortest distances of a graph $G(V, E)$, and takes $|V|(|V|-1)/2$ space. And the landmark approach was proposed to reduce the space cost to $|V||D|$, where $|D|$ is the size of a landmark cover. One might believe that the landmark approach always incurs less space than the matrix approach. It is, however, not the case as shown by the following example.

Example 1: Consider node x in a landmark cover D that lies on the shortest paths of a set $\{(u_1, u_2), \ldots, (u_{2k-1}, u_{2k})\}$ of k node pairs in a graph, where nodes $u_i \neq u_j$ for any $i \neq j \in [1, 2k]$. Then node x takes k space in the naive approach, by directly adding edges to connect those k node pairs, while it takes $2k$ space in the landmark approach, by adding edges between x and each of the $2k$ nodes. □

This motivates us to propose a *hybrid* approach combining the naive approach with the landmark one. To do this, we first define the following notions.

Consider a node x in a graph G. Let P_x be a set of node pairs such that x lies on their shortest paths, and let N_x be the set of distinct nodes in P_x. For a landmark node x, we only store the shortest distances between x and the node in N_x, instead of all the nodes in the graph as [24]. Hence, the space cost of making x a landmark, denoted by $\mathsf{space}_\mathsf{L}(x)$, is exactly $|N_x|$. Alternatively, the naive approach incurs a space cost of $|P_x|$, denoted by $\mathsf{space}_\mathsf{N}(x)$, by storing the shortest distances for each node pair in P_x.

Consider an ordered set of l vertices $D = < x_1, \ldots, x_l >$ such that (a) for each $i \in [1, l]$, x_i is a node in graph G, and (b) $P_{x_i} \cap P_{x_j} = \emptyset$ for any $i \neq j \in [1, l]$.

Hybrid landmark covers. We say that $\tilde{D} = (D, E_D^-)$ is a *hybrid landmark cover* of graph G if and only if:

(1) for each x_i $(i \in [1, l])$, $\mathsf{space}_\mathsf{L}(x_i) \leq \mathsf{space}_\mathsf{N}(x_i)$,

(2) there exist no other nodes x in G, but $x \notin D$, such that $\mathsf{space}_\mathsf{L}(x) \leq \mathsf{space}_\mathsf{N}(x)$, and

(3) E_D^- is a set edges, denoting all the node pairs of G such that no landmarks in D lie on their shortest paths.

We also call $E_{\tilde{D}} = \{(u, x) \mid u \in N_x, x \in D\} \cup E_D^-$ the set of edges *enforced* by a hybrid landmark cover $\tilde{D}$.

Remark. (1) Essentially, D consists of a maximal set of landmarks such that the space cost of each landmark in the set is not larger than the corresponding naive cost.

(2) A hybrid landmark cover $\tilde{D}$ of a graph can be treated as another graph with the same set of nodes, but with a different set of edges, *i.e.,* the set $E_{\tilde{D}}$ of enforced edges. Similarly, the naive approach transforms a graph into a complete graph. This provides a unified view for these two approaches.

(3) Computing hybrid landmark covers on large graphs remains very challenging. Indeed, they cannot be directly used in practice as well. As will be seen in Section V, we build hybrid landmark covers *w.r.t.* a (small) subset of nodes in graph G. In the following, we will explore techniques to support efficient shortest distance queries on large real-life road graphs.

IV. USING REPRESENTATIVES FOR LANDMARKS

As illustrated and analyzed in Section III, the direct application of distance landmarks is not practical for large graphs. A straightforward approach is to use *representatives*, each of which captures a set of nodes in a graph. The distance landmarks are for the representatives only, instead of the entire graph, which reduces both space and time cost.

The task to find a proper form of representatives is, however, *nontrivial*. Intuitively, we expect representatives to have the following properties. (1) A small number of representatives can represent a large number of nodes in a graph; (2) Shortest distances involved within the set of nodes being represented by the same representative can be answered efficiently; And, moreover, (3) the representatives and the set of nodes being represented can be computed efficiently.

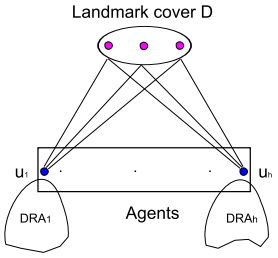


Figure 2. Using agents for landmarks

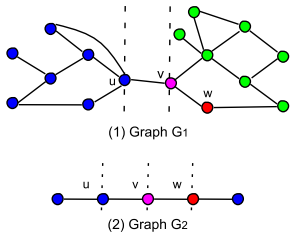


Figure 3. Example agents and DRAs

In this section, we first propose *agents* and *deterministic routing areas* (DRAs) to capture representatives and the set of nodes being represented, respectively. We then give an analysis of the properties of DRAs and their agents, and show that they are indeed what we want. Finally, we present a linear-time algorithm for computing agents and their DRAs. The idea of using agents and DRAs is illustrated in Fig. 2.

We consider a graph $G(V, E, w)$.

A. Agents and Deterministic Routing Areas

We first present agents and their DRAs.

Agents. Given a node u in graph $G(V, E)$, we say that u is an *agent* of a set of nodes, denoted by A_u, if and only if:
(1) node $u \in A_u$ is reachable to any node of A_u in G,
(2) all neighbors of any node $v \in A_u \setminus \{u\}$ are in A_u, and
(3) the size $|A_u|$ of A_u is equal or less than $c \cdot \lfloor\sqrt{|V|}\rfloor$,
where c is a small constant number, such as 2 or 3.

Here condition (1) guarantees the connectivity of subgraph $G[A_u]$, condition (2) implies that not all neighbors of agent u are necessarily in A_u; and condition (3), referred to as *size restriction*, limits the size of A_u of agent u.

Note that a node u may be an agent of multiple sets of nodes $A_u^1, \ldots, A_u^k$ such that $A_u^i \cap A_u^j = \{u\}$ for any $i \neq j \in [1, k]$. And we denote as A_u^+ the union of all the sets of nodes whose agent is u , *i.e.,* $A_u^+ = A_u^1 \cup \ldots \cup A_u^k$.

Maximal agents. We say that an agent u is *maximal* if there exist no other agents u' such that $A_u^+ \subset A_{u'}^+$.

Trivial agents. We say that a maximal agent u is *trivial* if A_u^+ contains itself only, *i.e.,* $A_u^+ = \{u\}$.

Equivalent agents. We say that two agents u and u' are *equivalent*, denoted by $u \equiv u'$, if $A_u^+ = A_{u'}^+$.

Deterministic routing areas (DRAs). We refer to the *subgraph* $G[A_u^+]$ with nodes A_u^+ as a DRA of agent u.

Intuitively, DRA $G[A_u^+]$ is a *maximal* connected subgraph connecting to the rest of graph G through agent u only.

We next illustrate these notions with an example below.

Example 2: First consider graph $G_1(V_1, E_1)$ in Fig. 3, and let $c \cdot \lfloor\sqrt{|V_1|}\rfloor = 2 \cdot \lfloor\sqrt{16}\rfloor = 8$, where $c = 2$ and $|V_1| = 16$.
(1) Node u is an agent, and its DRA is the subgraph in the left hand side of the vertical line across u;
(2) Node v is an agent, and its DRA is the subgraph in the left hand side of the vertical line across v;
(3) Node w is not an agent since it can not find a DRA with size less or equal than 8;
(4) Node v is a maximal agent, while node u is not a maximal agent since $A_u^+ \subset A_v^+$.

We then consider graph $G_2(V_2, E_2)$ in Fig. 3, and let $c \cdot \lfloor\sqrt{|V_2|}\rfloor = 2 \cdot \lfloor\sqrt{5}\rfloor = 4$, where $c = 2$ and $|V_2| = 5$.
(1) Nodes u, v and w are three maximal agents, whose DRAs are all the entire graph G_2, and, hence,
(2) u, v and w are three equivalent agents. □

Remarks. (1) As illustrated by the above examples, a DRA of graph $G(V, E)$ may have a size larger than $c \cdot \lfloor\sqrt{|V|}\rfloor$, and multiple equivalent agents. (2) Trivial agents can only represent themselves. Hence, we are only interested in non-trivial agents (or simply called agents) in the sequel.

B. Properties of Agents and DRAs

We next give an analysis of agents and DRAs, and show that they hold good properties for shortest distance queries.

Proposition 3: *Any agent in a graph has a unique* DRA. □

This shows that agents and DRAs are well defined notions.

Proposition 4: *Without the size restriction, any node u in graph G is a maximal agent, and its* DRA *$G[A_u^+]$ is exactly the connected component (*CC*) to which u belongs.* □

This justifies the necessity of the size restriction for agents. Otherwise, DRAs are simply CCs, and are mostly useless.

Proposition 5: *For any two nodes v, v' in the* DRA *$G[A_u^+]$ of agent u in graph G,*
(1) the shortest distance $\mathsf{dist}(v, v')$ *in* DRA *$G[A_u^+]$ is exactly the one in the entire graph G; and*
(2) it can be computed in linear time in the size of G. □

The size restriction guarantees that the shortest distance computation within a DRA can be evaluated efficiently.

Proposition 6: *Given a node v in the* DRA *$G[A_u^+]$ of agent u in graph G, and another node v' in G, but not in $G[A_u^+]$, the shortest distance* $\mathsf{dist}(v, v') = \mathsf{dist}(v, u) + \mathsf{dist}(u, v')$. □

Propositions 5 and 6 together guarantee that the shortest distances between the nodes in the DRAs of two distinct agents can be answered correctly and efficiently.

Proposition 7: *Any agent in a* CC *$H(V_s, E_s)$ of graph $G(V, E)$ with $|V_s| > c \cdot \lfloor\sqrt{|V|}\rfloor$ must be a cut-node of graph G.* □

This motivates us to identify maximal agents by utilizing the cut-nodes and BCCs, which will be seen immediately.

Proposition 8: *Any node in a bi-connected component (*BCC*) with size larger than $c \cdot \lfloor\sqrt{|V|}\rfloor$ of graph $G(V, E)$ is a trivial agent.* □

As we are interested in non-trivial agents only, those large BCCs could be simply ignored with any side effects.

Theorem 9: *Given any two agents u and u',*
(1) if $u \in A_{u'}^+$, then $A_u^+ \subseteq A_{u'}^+$;
(2) if $u' \in A_u^+$, then $A_{u'}^+ \subseteq A_u^+$; and
(3) $A_u^+ \cap A_{u'}^+ = \emptyset$, otherwise. □

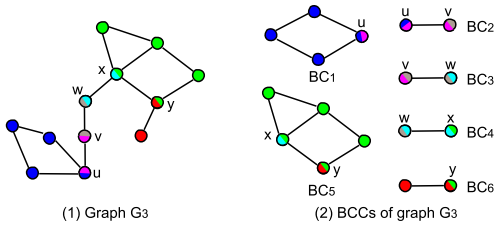


Figure 4. Cut-nodes and bi-connected components

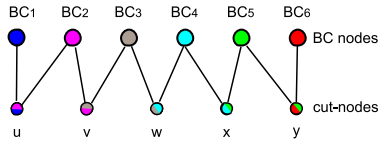


Figure 5. BC-SKETCH graph $\mathbb{G}_3$ of graph G_3

Corollary 10: *Given any two maximal agents u and u', then either $A^+_u = A^+_{u'}$ or $A^+_u \cap A^+_{u'} = \emptyset$ holds.* □

This says when maximal agents are concerned, there exists a unique set of non-overlapping DRAs.

C. Computing DRAs and Maximal Agents

In this section, we first present a notion of BC-SKETCH graphs, based on which we then propose an algorithm for computing DRAs and their maximal agents.

The main result here is stated as follows.

Theorem 11: *Finding all DRAs, each associated with one maximal agent, in a graph can be done in linear time.* □

We shall prove this by providing a linear time algorithm that computes DRAs and maximal agents. We first present BC-SKETCH graphs, a key notion employed by the algorithm.

A BC-SKETCH graph $\mathbb{G}(\mathbb{V}, \mathbb{E}, \omega)$ of a graph $G(V, E)$ is a bipartite graph, in which (1) $\mathbb{V} = \mathbb{V}_c \cup \mathbb{V}_{bc}$ such that $\mathbb{V}_c$ is the set of cut-nodes in G, and $\mathbb{V}_{bc}$ is the set of BCCs in G; (2) for each cut-node $v \in \mathbb{V}_c$ and each BCC $y_b \in \mathbb{V}_{bc}$, there exists an edge $(v, y_b) \in \mathbb{E}$ iff v is a cut-node of BCC y_b; and (3) ω is a weight function such that for each node $y_b \in \mathbb{V}_{bc}$, $\omega(y_b)$ is the number of nodes of G in BCC y_b.

Example 3: Consider graph G_3 in Fig. 4(1), in which labeled nodes u, v, w, x, y are the cut-nodes of G_3, and the corresponding BCCs of G_3 are $BC_1, BC_2, BC_3, BC_4, BC_5$, and BC_6, and are shown in Fig. 4(2).

The BC-SKETCH graph $\mathbb{G}_3(\mathbb{V}, \mathbb{E}, \omega)$ of graph G_3 is shown in Fig. 5, in which $\omega(BC_1) = 4$, $\omega(BC_2) = \omega(BC_3) = \omega(BC_4) = \omega(BC_6) = 2$, and $\omega(BC_5) = 5$. □

One may notice that there are no cycles in the BC-SKETCH graph $\mathbb{G}_3$. This is not a coincidence, as shown below.

Proposition 12: BC-SKETCH *graphs have no cycles, which implies that they are simply trees.* □

Proposition 12 indicates that we can employ the good properties of trees for computing DRAs and maximal agents.

We are now ready to present algorithm compDRAs shown in Fig. 6. It takes as input graph G and constant c, and outputs the DRAs of G, each associated with a maximal agent.

Input: Graph $G(V, E)$ and constant c.
Output: The DRAs associated with their maximal agents.
1. Find all cut-nodes $\mathbb{V}_c$ and BCC nodes $\mathbb{V}_{bc}$ of G;
2. Build the BC-SKETCH graph $\mathbb{G}(\mathbb{V}, \mathbb{E}, \omega)$ with $\mathbb{V} = \mathbb{V}_c \cup \mathbb{V}_{bc}$;
3. Identify and return the DRAs and their maximal agents of G.

Procedure extractDRAs
Input: BC-SKETCH graph $\mathbb{G}(\mathbb{V}, \mathbb{E}, \omega)$ of graph G and constant c.
Output: The DRAs and their maximal agents of G.
1. **let** F be the set of cut-nodes with leaf neighbors in $\mathbb{G}$;
   /* note that a leaf node must be a BCC node */
2. **while** F is **not** empty **do**
3. pick a cut-node v from F; **let** X be the neighbors of v;
   /* note that there is at most one non-leaf node in X */
4. **let** $\alpha := \sum_{y' \in X} \omega(y')$ - $|X|$ + 1;
5. **if** $\alpha \leq c \cdot \lfloor\sqrt{|V|}\rfloor$ **then**
6. merge all BCC nodes in X and v into one BCC node y_n;
7. **let** $\omega(y_n) := \alpha$;
8. **if** there is a non-leaf node in X **then** replace it with y_n;
9. $F := F \setminus \{v\}$;
10. **let** F' be the set of new cut-nodes with leaf neighbors;
11. **for** each cut-node v in F' **do**
12. **let** X' be a set of leaf neighbors of v' such that
13. for each $y' \in X'$, $\omega(y') \leq c \cdot \lfloor\sqrt{|V|}\rfloor$;
14. mark X' as the DRA $A^+_{v'}$ of agent v';
15. **return** all DRAs with their maximal agents.

Figure 6. Computing DRAs and maximal agents

(1) Finding cut-nodes and BCC*s*. The algorithm starts with computing all cut-nodes and bi-connected components (line 1), by using the linear-time algorithm developed by John Hopcroft and Robert Tarjan [6], [15].

(2) Constructing BC-SKETCH *graphs*. After all the cut-nodes and BCCs are identified, the BC-SKETCH graph $\mathbb{G}(\mathbb{V}, \mathbb{E}, \omega)$ can be easily built (line 2). To see this can be done in linear time, the key observation is that the number $|\mathbb{E}|$ of edges in $\mathbb{G}$ is exactly $|\mathbb{V}| - 1$ since $\mathbb{G}$ is a tree.

(3) Identifying DRA*s and their maximal agents*. Finally, the algorithm identifies and returns the DRAs and their maximal agents (line 3), using Procedure extractDRAs in Fig. 6.

Procedure extractDRAs takes as input the BC-SKETCH graph $\mathbb{G}$ of graph G and constant c, and outputs the DRAs and their maximal agents, by repeatedly merging BCCs with size less than $c \cdot \lfloor\sqrt{|V|}\rfloor$. More specifically, the procedure starts with the set F of cut-nodes with leaf neighbors (line 1). It then recursively merges the neighboring BCC nodes of cut-nodes to generate new BCC nodes (lines 2-9). For a node $v \in F$ with neighbors X, if $\sum_{y' \in X} \omega(y')$ - $|X|$ + 1 $\leq c \cdot \lfloor\sqrt{|V|}\rfloor$, they can be merged into a new BCC node (lines 3-8). Intuitively, this says cut-node v is not a maximal agent, and it is combined into the DRAs of maximal agents. A key observation here is that there is at most one non-leaf node in X. If there is such a non-leaf neighbor, then it is replaced by the new BCC node y_n (line 8), by which the merging processing is made possible. Once a cut-node is considered, it is never considered again (line 9). After no merging can be made, we have found all maximal agents, *i.e.,* all the cut-nodes in the updated BC-SKETCH graph. We then identify DRAs for these maximal agents (lines 10-14). For any leaf neighbor y' of a cut-node v', if $\omega(y') \leq c \cdot \lfloor\sqrt{|V|}\rfloor$, then y' is an $A_{v'}$ of agent v'. All these

together constitute the $A^+_{v'}$ of agent v' (lines 12-14). Finally, all DRAs with their maximal agents are returned (line 15).

We now explain the algorithm with an example as follows.

Example 4: Consider graph G_3 in Fig. 4(1) again. Here we let $c = 2$, and $c \cdot \lfloor\sqrt{|V|}\rfloor = 6$. Firstly, cut-nodes and BCCs are computed as shown in Fig. 4(2). Secondly, the BC-SKETCH graph $\mathbb{G}_3$ of G_3 is constructed as shown in Fig. 5. After the merging step stops, the updated BC-SKETCH graph consists of three BCC nodes: $BC'_1 = \{BC_1, BC_2, BC_3\}$, BC_4, $BC'_2 = \{BC_5, BC_6\}$ and two cut-nodes: w and x. Finally, the DRAs and their maximal agents are identified: agent w with DRA BC'_1 and agent x with DRA BC'_2. □

Correctness & Complexity. The correctness of algorithm compDRAs can be readily verified based on the analyses in Section IV-B. To show that algorithm compDRAs runs in linear time, it suffices to show that procedure extractDRAs can be done in linear time. It is easy to see that each node in the BC-SKETCH graph is visited at most twice in procedure extractDRAs, and hence the procedure runs in linear time.

This completes the proof of Theorem 11.

Summary. (1) We have proposed a notion of agents and DRAs aiming at reducing the size of graphs such that landmarks are only for agents, instead of the entire graph. (2) We have given a theoretical analysis of agents and DRAs, based on which we have developed a linear time algorithm for computing DRAs and their maximal agents. (3) As shown in our experimental study, on average about 1/3 nodes of a graph are captured by non-trivial agents and their DRAs.

V. INTRODUCING GRAPH PARTITIONS FOR LANDMARKS

Web graphs contain a large strongly connected components [2], and, similarly, there is usually a large BCC in real-life graphs such as the collaboration and social networks [9], [19]. As pointed out in Section IV, for the BCCs in a graph $G(V, E)$ with a size larger than $\lfloor\sqrt{|V|}\rfloor$, each node in those BCCs is a trivial agent that can only represent itself. This motivates us to introduce the graph partitioning techniques for distance landmarks, based on which we use a small set of nodes, instead of a single agent node, to represent a large set of nodes.

In this section, we first introduce graph partitions. We then propose a notion of SUPER graphs which combine graph partitions with hybrid landmark covers. We finally present the bounded graph partition problem and its solution.

We consider a graph $G(V, E)$.

A. Graph Partitions and Super Graphs

We first introduce graph partitions and SUPER graphs.

Graph partitions. We say that $(V_1, \ldots, V_k)$ is a *partition* of graph $G(V, E)$ if and only if (1) $\bigcup_{i=1}^{k} V_i = V$, and (2) for any $i \neq j \in [1, k]$, $V_i \cap V_j = \emptyset$, in which we refer to a V_i ($i \in [1, k]$) as a *fragment* of the partition.

We also say that node u in V_i ($1 \leq i \leq k$) is a *boundary* node if there exists an edge (u, v) in G from nodes u to v such that $v \in V_j$ and $j \neq i$ ($1 \leq j \leq k$).

SUPER graphs. We next introduce SUPER graphs that combine graph partitions with hybrid landmark covers.

Consider a partition $(V_1, \ldots, V_k)$ of graph G. For each fragment V_i ($i \in [1, k]$), let (1) B_i be the set of boundary nodes of V_i, and (2) $\tilde{D}_i = (D_i, E^-_{\tilde{D}_i})$ be a hybrid landmark cover for the set B_i of *boundary* nodes of V_i.

The SUPER graph of graph partition $(V_1, \ldots, V_k)$ is a weighted undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \Upsilon)$ such that:

(1) $\mathcal{V} = B_1 \cup \ldots \cup B_k \cup D_1 \cup \ldots \cup D_k$, *i.e.,* the union of all boundary nodes and distance landmarks on each fragment;

(2) $E = E_B \cup E_{\tilde{D}_1} \cup \ldots \cup E_{\tilde{D}_k}$, where $E_B \subseteq E$ is the set of edges with both endpoints belonging to $B_1 \cup \ldots \cup B_k$, and for each $i \in [1, k]$, $E_{\tilde{D}_i}$ is the set of edges enforced by the hybrid landmark cover $\tilde{D}_i$; and

(3) For each edge $(u, v) \in E_B$, $\Upsilon(u, v)$ is exactly equal to the edge weight $w(u, v)$ in graph G, and for each edge $(u, v) \in E_{\tilde{D}_i}$ ($i \in [1, k]$), $\Upsilon(u, v)$ is the local shortest distance between u and v in the fragment V_i only.

That is, a SUPER graph $\mathcal{G}$ of graph $G(V, E)$ only consists of the landmarks and boundary nodes. Hence, the size of $\mathcal{G}$ is typically much smaller than graph G. Intuitively, SUPER graphs use a small set of nodes in a fragment, *i.e.,* the boundary nodes and distance landmarks, to represent a large number of nodes, *i.e.,* all the nodes in the fragment.

B. Bounded Graph Decompositions

As the landmarks are for the boundary nodes, the number of boundary nodes has a key impact on the size of SUPER graphs. In addition, the size of a fragment should be bounded in order to efficiently compute its hybrid landmark cover.

This motivates us to study the following problem.

The bounded graph partitioning problem is to find a partition $(V_1, \ldots, V_k)$ of graph $G(V, E)$, denoted by BGP, such that (1) $|V_i| \leq \Gamma$ for each fragment V_i ($i \in [1, k]$), and (2) $|B| \leq \epsilon \cdot |V|$, where $\Gamma \leq |V|$ is a positive integer, $\epsilon \in [0.0, 1.0]$ is a rational number, and $|B|$ is the total number of boundary nodes.

The problem is, however, nontrivial, as expected.

Proposition 13: *The* BGP *problem is* NP*-complete.* □

Traditional graph partitioning is to find a partition (V_1, …, V_k) of a graph such that (1) the k fragments have a roughly equal number of nodes, and (2) the number of edges connecting nodes in different fragments is minimized. The problem has been extensively studied since 1970's [16], [17], [35], and has been used in various applications, *e.g.,* circuit placement, parallel computing and scientific simulation [35].

Large-scale graph partitioning tools are available such as the best-known METIS [16]. Hence, this study is not to propose a new graph partitioning algorithm. Instead, it builds relationships between the BGP problem and the traditional graph partitioning problem, and makes use of existing approaches for solving the BGP problem.

Key observations. For any partition (V_1, …, V_k), the set B of boundary nodes with edges across different fragments and the set E_B of all edges connecting nodes in different fragments satisfy: $|B| \leq 2|E_B|$.

This is, minimizing $|E_B|$ essentially reduces the upper bound of $|B|$. Moreover, those edges in E_B are part of the SUPER graph. Hence, minimizing $|E_B|$ also reduces the size of the SUPER graph. This observation inspires us to adopt existing approaches, *e.g.,* METIS [16], to partition graphs and generate SUPER graphs. As will be seen in in our experiments, smaller SUPER graphs help answer shortest distance queries.

Summary. (1) We have introduced a notion of SUPER graphs that combine graph partitions with distance landmarks. (2) We have proposed the BGP problem, and shown it is NP-complete. (3) We have also built connections between the BGP problem and the traditional graph partitioning problem, which makes it possible to use the existing approaches, *e.g.,* METIS [16], to solve our problem. As will be seen in our experiments, METIS works well for the BGP problem, and the produced SUPER graphs are typical small, which only have 2–4% nodes and 10–15% edges compared with the original graphs.

VI. A Unified Framework for Answering Shortest Distance Queries

In this section, we propose a unified framework, referred to as DISLAND, for fast shortest distance query answering, which consists of two modules: *preprocessing* and *query answering*. We combine distance landmarks with agents and graph partitions (SUPER graphs), and seamlessly integrate existing speed-up techniques [13], [22] into the framework.

Consider a graph $G(V, E)$ with non-negative edge weights.

A. Preprocessing for Query Answering

We first present the preprocessing module.

Given graph $G(V, E)$, the module seamlessly combines agents and graph partitions with hybrid landmark covers, and it produces (a) maximal agents along with their DRAs, (b) graph partitions, and (c) a SUPER graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$.

More specifically, given graph $G(V, E)$, the module executes the following processes:

(1) It first computes the DRAs and their maximal agents, using algorithm compDRAs proposed in Section IV-C.

(2) For each DRA with a non-trivial maximal agent u, it further (a) computes all the shortest distances $\mathsf{dist}(u, v)$ for all nodes v in its DRA, and (b) adds an edge (u, v) with weight $\mathsf{dist}(u, v)$ for each node v in the DRA.

(3) It then generates a *shrink graph*, the subgraph $G[A]$ of G in which A is the set of agent nodes, including both trivial and non-trivial agents. For each DRA with a maximal agent u, only u is kept in $G[A]$.

(4) It next calls METIS [16] to produce a graph partition $(V_1, \ldots, V_k)$ for the shrink graph $G[A]$ such that for each $i \in [1, k]$, $|V_i|$ is roughly equal to $c \cdot \lfloor \sqrt{|V|} \rfloor$. Here c is a small constant number, such as 2 or 3.

(5) For each fragment V_i ($i \in [1, k]$), it computes a (local) hybrid landmark cover $\tilde{D}_i$ for the *boundary nodes* of V_i only, by calling the SC based algorithm (Section II-B, [24]). Note that here we did not use the VC based algorithm, which was proposed for estimating of the size of landmark covers only.

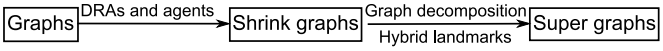


Figure 7. The preprocessing module

(6) Finally, it builds a SUPER graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \Upsilon)$ of graph G.

The entire process is illustrated in Fig. 7.

B. A Bi-level Query Answering Approach

We next present the query answering module.

Given a source node s and a target node t, this module finds the shortest distance from s to t, by making use of the auxiliary structures produced by the preprocessing module.

More specifically, given nodes s and t, the query answering module executes the following processes:

(1) When nodes s and t belong to the same DRA $G[A_u^+]$ with agent u such that $A_u^+ = A_u^1 \cup \ldots A_u^h$.

If s and t further fall into the same A_u^i, then it invokes Dijkstra's algorithm on the subgraph $G[A_u^i]$. Otherwise, it simply returns $w(s, u) + w(u, t)$ in constant time.

(2) When s and t belong to two DRAs $G[A_{u_s}^+]$ and $G[A_{u_t}^+]$ with agents u_s and u_t, respectively. As $\mathsf{dist}(s, t) = \mathsf{dist}(s, u_s) + \mathsf{dist}(u_s, u_t) + \mathsf{dist}(u_t, t)$, in which $\mathsf{dist}(s, u_s)$ and $\mathsf{dist}(u_t, t)$ are already known, we only need to compute $\mathsf{dist}(u_s, u_t)$.

Let V_s and V_t be the fragments to which agents u_s and u_t belong, respectively. As observed in [4], fragments V_s and V_t and the SUPER graph together suffice to answer exact shortest distance queries. Hence, the algorithm invokes the Dijkstra's algorithm on the union of subgraphs $G[V_s]$, $G[V_t]$ and the SUPER graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \Upsilon)$ to compute $\mathsf{dist}(u_s, u_t)$.

Following the analysis above, we have the following.

Proposition 14: *Framework* DISLAND *correctly answers shortest distance queries.* □

C. Optimization Techniques

There exist quite a few speed-up techniques for shortest distance computations [32], [34]. DISLAND is very flexible such that most of these techniques, if not all, can be seamlessly incorporated to further speed-up shortest distance query answering. In this study we have adopted bidirectional search [20], contraction hierarchies (CH) [13], and Arc-Flags (ARCFLAG) [22] due to their effectiveness and generality.

We first introduce the three optimization techniques.

(1) Bidirectional search (BSEARCH, [20]) simultaneously performs two searches: forward and backward, starting at the source and target nodes, respectively [20], [32]. It invokes two instances of the Dijkstra's algorithm simultaneously, and has the same time complexity as the (single directional) Dijkstra's algorithm. However, BSEARCH is usually more efficient than the Dijkstra's algorithm in practice.

(2) Contraction hierarchies (CH, [13]) first imposes a total order $\mathcal{O}$ on the nodes of a graph, in ascending order of their relative 'importance', and then constructs a *hierarchy* by contracting all the nodes in this order. A node v is contracted by removing it from the graph such that shortest paths in the remaining graph are preserved, achieved by replacing paths of the form $u/v/w$ by a *shortcut* edge (u, w). Note that the

shortcut (u, w) is *only* required if $u/v/w$ is the *only* shortest path from u to w. After all the nodes are contracted, all the shortcuts are appended into the graph.

CH uses BSEARCH with minor revisions for query answering. Give two nodes u and w with $\mathcal{O}(u) < \mathcal{O}(w)$, CH only visits two kinds of paths $u/\cdots/v_i/\cdots/w$ in the process: (a) $\mathcal{O}(u) < \cdots < \mathcal{O}(w)$ or (b) there is a unique v_i with $\mathcal{O}(u) < \cdots < \mathcal{O}(v_i)$ and $\mathcal{O}(v_i)$ ¿ $\cdots > \mathcal{O}(w)$. In this way, CH avoids visiting the nodes with an order lower than u and w in the forward and backward searches, respectively, which makes it much more efficient than BSEARCH alone in practice.

(3) Arc-Flags (ARCFLAG, [22]) is a partition-based edge labeling approach, and it divides a graph $G(V, E)$ into partitions $(V_1, \ldots, V_k)$ and gathers information for each edge $e \in E$ and for each fragment V_i ($i \in [1, k]$) on whether the edge e lies on a shortest path into the fragment V_i. To do this, each edge e is associated with a flag vector f_e with k bits (the number of fragments) such that the vector f_e contains a flag 1 or 0 for V_i indicating whether or not e is useful for a shortest path query to nodes in V_i. It is easy to verify that ARCFLAG incurs $k|E|$ bits of extra space.

We next show how to seamlessly incorporate these three optimization techniques into our framework DISLAND.

(1) The shrink graph $G[A]$ of graph G is appended with shortcuts, by using the CH approach.

(2) We build a hybrid landmark cover for each fragment, by incorporating the CH searching process.

We only consider the shortest paths $\rho = u/\cdots/v_i/\cdots/w$ such that (a) $\mathcal{O}(u) < \cdots < \mathcal{O}(w)$, in which case ρ is called *order rising*, or (b) $\mathcal{O}(u)$ ¡ $\cdots < \mathcal{O}(v_i)$ and $\mathcal{O}(v_i)$ ¿ $\cdots > \mathcal{O}(w)$, in which case ρ is called *order turning*. When computing landmarks for a fragment, we cover a node pair (u, v) only if (1) there exists an order rising or turning path between u and v, and (2) their (local) shortest distance in the fragment is equal to their (global) shortest distance in the entire shrink graph. Moreover, (a) for these node pairs (u, w) connected by order turning paths, we select the nodes with *highest order* as landmarks; and (b) for these remaining node pairs (u, w) connected by order rising paths, we use the cost model to greedily select landmarks or build direct edges, following the hybrid landmark approach. As the searching space is reduced, this both improves the efficiency of computing hybrid landmark covers, and, of course, the query answering. Moreover, we adopt the query answering approach for CH [13], instead of the bidirectional Dijkstra's algorithm, in the query answering module of DISLAND.

(3) We compute edge labeling, by using the ARCFLAG approach. To do this, we further call METIS to do a second level partition of the SUPER graph, where each fragment is treated as a single node, and the edge weight between fragments are the number of edges connecting them. When building Arc-Flags, we again incorporate CH, by considering order rising or turning shortest paths only, to speed-up the processing.

Extra space analysis. This module produces two kinds of auxiliary structures: the non-trivial maximal agents along with their DRAs and the SUPER graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \Upsilon)$.

(1) Let $U = \{u_1, \ldots, u_h\}$ be the set of non-trivial maximal agents identified. The extra space of U and their DRAs is the extra edges from those agents to the set of nodes in their DRAs, which is exactly equal to $\sum_{i=1}^{h} |A_{u_i}^{+}|$ - h.

(2) Each fragment in the partition $(V_1, \ldots, V_k)$ roughly has the same size of $c \cdot \lfloor\sqrt{|V|}\rfloor$. We set c = 2 or 3 in practice. Hence, the number of fragments is less than $\lfloor\sqrt{|V|}\rfloor$.

For each fragment V_i ($i \in [1, k]$), let $E_{\tilde{D}_i}$ be the set of edges enforced by the hybrid landmark cover $\tilde{D}_i$ for the *boundary* nodes of V_i. Hence, the number of extra edges in the SUPER graph $\mathcal{G}$ is bounded by $\sum_{i=1}^{k} |E_{\tilde{D}_i}|$.

(3) The remaining extra space is incurred by the shortcuts added by CH and the Arc-Flags added by ARCFLAG.

As will be shown in our experiments, all these auxiliary structures only incur a small space cost, and the entire preprocessing can be finished in a reasonably fast way.

Table II
REAL-WORLD GRAPHS

Name	Regions	# of Nodes	# of Edges
CO	Colorado	435,666	1,042,400
FL	Florida	1,070,376	2,687,902
CA	California & Nevada	1,890,815	4,630,444
E-US	Eastern US	3,598,623	8,708,058
W-US	Western US	6,262,104	1,5119,284
C-US	Central US	14,081,816	33,866,826
US	Entire US	23,947,347	57,708,624

VII. EXPERIMENTAL STUDY

We next present an extensive experimental study of the DISLAND framework for shortest distance query answering. Using real-life road networks, we conducted five sets of experiments to evaluate: (1) the impacts of agents, graph partitions, and hybrid landmark covers; (2) the preprocessing time and space overhead of bidirectional Dijkstra [20], CH [13], ARCFLAG [22], their counterparts using agents (Agent + Dijkstra, Agent + CH, Agent + ARCFLAG), and DISLAND; and (3) the performance of all these approaches.

A. Experimental Settings

We first introduce the settings of our experimental study.

Real-life graphs. We chose seven datasets of various sizes from the Ninth DIMACS Implementation Challenge [8], shown in Table 2. Each dataset is an undirected graph that represents a part of the road network in the United States (US), where each edge weight is the distance (integers) required to travel between the two endpoints of the edge.

Distance queries. We adopted the query generator in [34]. Our distance queries were generated as following. On each road network, we generated eight sets Q_1, Q_2, ... , Q_8 of queries. (1) We first imposed a 256×256 grid on the road network and computed the side length ℓ of each grid cell. (2) We then randomly chose ten thousand node pairs from the road network to compose $Q_i (i \in [1, 8])$, such that the grid distance of all node pairs in Q_i is in $[2^{i-1} \cdot \ell, 2^i \cdot \ell)$. Note that the grid distance of two nodes u, v in a query set is the distance of the cells into which u and v fall, respectively. Moreover, the grid distance of any node pair in Q_i is larger than the grid distance of all

Table III
EFFECTIVENESS OF AGENTS AND DRAS

Graphs	Agents (#, %)	Nodes (#, %) in DRAs	time (s)
CO	(56,277, 12.9%)	(156,329, 35.9%)	1.1
FL	(140,379, 13.1%)	(378,937, 35.4%)	3.7
CA	(273,191, 14.4%)	(623,811, 33.0%)	11.3
E-US	(546,481, 15.2%)	(1,228,876, 34.1%)	34.3
W-US	(869,904, 13.9%)	(2,116,339, 33.8%)	100.4
C-US	(2,034,358, 14.4%)	(4,583,413, 32.5%)	402.4
US	(3,452,222, 14.4%)	(7,927,453, 33.1%)	1153.7

Table IV
EFFECTIVENESS OF GRAPH PARTITIONS

Shrink graphs	fragments (#)	avg # of nodes	avg (#, %) of boundary nodes	time (s)
CO	220	1,269.7	(76.1, 5.99%)	1.1
FL	340	2,033.6	(92.5, 4.55%)	3.1
CA	470	2,695.8	(114.9, 4.26%)	6.6
E-US	630	3,761.5	(156.4, 4.16%)	13.8
W-US	840	4,935.4	(151.9, 3.08%)	26.2
C-US	1,280	7,420.6	(241.4, 3.25%)	85.5
US	1,650	9,709.0	(260.2, 2.68%)	126.7

node pairs in Q_{i-1}. For each query set Q_i ($i \in [1, 8]$), we report the average running time of over all the ten thousand queries in the set.

Algorithms. We adopted the latest version 5.0.2 of METIS [21], implemented with ANSI C. We also re-implemented the original CH [3] from its inventors of using Microsoft Visual C++. Bidirectional Dijkstra, ARCFLAG and their counterparts using agents were also written in Microsoft Visual C++. All these algorithms used common data structures and procedures, borrowed from CH [3], for similar tasks.

All experiments were run on a PC with an Intel Core i5-2400 CPU@3.10GHz and 16GB of memory. Each test was repeated over 5 times, and the average is reported here. We compare algorithms running on general commercial PCs with a 16GB memory limitation, and hence, algorithms using larger memory, *e.g.,* [1], are not in our consideration.

B. Experimental Results

We next present our findings. In all experiments, we tested the datasets in Table 2, and fixed the constant $c = 2$ when computing agents and graph partitions on graphs $G(V, E)$.

Exp-1: Impacts of agents. In the first set of experiments, we evaluated (1) the number of non-trivial agents, (2) the number and percentage of the nodes represented by the agents (excluding the agents themselves from DRAs), and (3) the efficiency of our algorithm compDRAs for computing agents and their DRAs. The results are reported in Table 3.

There are around $1/7$ nodes are non-trivial agents, and about $1/3$ nodes are captured by agents in these graphs, which means basically the shrink graph is only about $2/3$ of the input graph. Moreover, although the size restriction is $\leq 2 \cdot \lfloor \sqrt{|V|} \rfloor$, DRAs are typically small in these graphs, and each agent represents 2 or 3 other nodes on average. Algorithm compDRAs also scales well, and it can be done in less than half an hour for the largest graph in the preprocessing.

As will be seen in the following experiments, this makes agents a light-weight optimization techniques, which benefits most, if not all, existing shortest distance algorithms.

Table V
EFFECTIVENESS OF HYBRID LANDMARK COVERS

Graph fragments	With cost model			Without cost model		
	$\lvert\tilde{D}\rvert$	$\lvert E_{\tilde{D}}\rvert$	time(s)	$\lvert D\rvert$	$\lvert E_D\rvert$	time(s)
CO	32.1	537.8	0.1	49.8	549.4	0.1
FL	39.5	689.3	0.2	61.7	705.7	0.2
CA	51.3	1,021.9	0.4	78.4	1,045.1	0.4
E-US	71.1	1617.1	0.9	107.0	1651.8	0.8
W-US	68.9	1,541.6	0.9	104.6	1,576.8	0.9
C-US	116.4	3,251.3	4.0	169.4	3,329.8	3.9
US	124.9	3,584.3	4.9	183.1	3,673.4	4.8

Table VI
SIZES OF SUPER GRAPHS

$\mathcal{G}$	CO	FL	CA	E-US	W-US	C-US	US
$\lvert\mathcal{V}_c\rvert/\lvert V\rvert$	3.9%	3.0%	2.9%	2.8%	2.1%	2.3%	1.8%
$\lvert\mathcal{E}_c\rvert/\lvert E\rvert$	14.5%	10.9%	12.7%	14.2%	10.3%	14.5%	12.0%
$\lvert\mathcal{V}\rvert/\lvert V\rvert$	3.9%	3.0%	2.9%	2.8%	2.1%	2.3%	1.8%
$\lvert\mathcal{E}\rvert/\lvert E\rvert$	14.8%	11.1%	13.0%	14.5%	10.5%	14.5%	12.3%

Exp-2: Impacts of graph partitions. In the second set of experiments, we justified that the BGP problem could be solved well by METIS, originally for traditional graph partitioning problems. Using the shrink graphs generated at Exp-1, we evaluated the effectiveness and efficiency of METIS. To ensure the query efficiency of DISLAND, each fragment has at most $c \cdot \lfloor \sqrt{|V|} \rfloor$ number of nodes. We used the multilevel bisection method of METIS with the balance factor fixed to 1.003. The results are reported in Table 4.

The results tell us that there are only about (up to) 6% of nodes are boundary nodes, and the largest graph can be finished in 127 seconds. This clearly justified our analysis and choice to attack the BGP problem by using existing approaches to traditional graph partitioning problems.

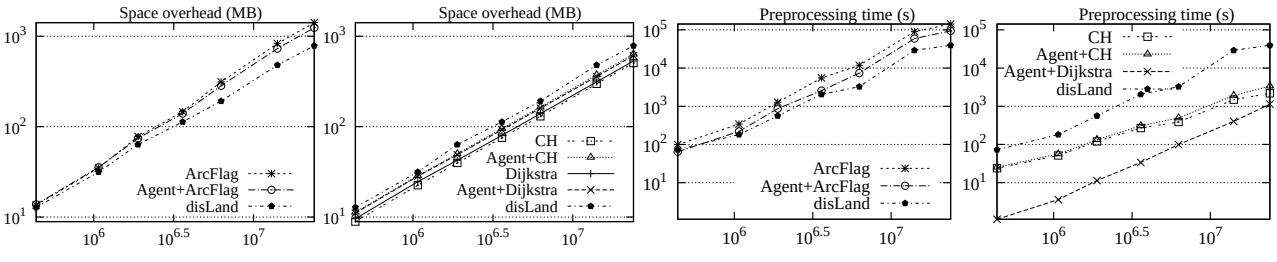
Exp-3: Impacts of hybrid landmark covers. In the third set of experiments, using the graph fragments generated at Exp-2, we evaluated (1) the average number of nodes and edges enforced by the hybrid landmarks covers with or without the cost model, and (2) their average efficiency on a single fragment. The results are reported in Table 5.

The results tell us that the usage of the cost model both reduces the number of landmarks and enforced edges, moreover, it only incurs little extra time cost.

We also report the SUPER graphs in Table 6. The SUPER graphs $\mathcal{G}$ are quite small, typically have 2–4% nodes and 10–15% edges compared with the original graphs $G(V, E)$. Using hybrid landmark covers with the cost model, the SUPER graphs $\mathcal{G}(\mathcal{V}_c, \mathcal{E}_c)$ further reduce 0.2–0.3% edges. This justified the effectiveness of agents and graph partitions, and the introduction of the cost model for hybrid landmark covers.

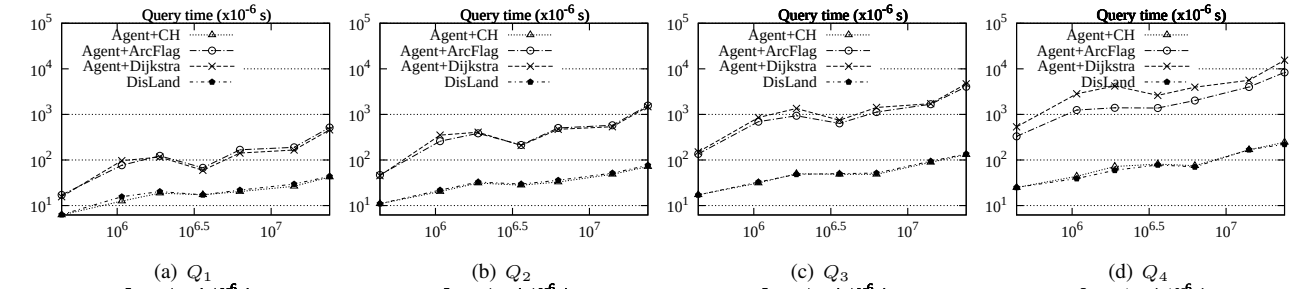
Exp-4: Preprocessing time and space overhead. In the fourth set of experiments, we tested the space cost and preprocessing time of Dijkstra, Agent + Dijkstra, CH, Agents + CH, ARCFLAG, Agents + ARCFLAG, and DISLAND. For DISLAND, we did a second level partition on the SUPER graphs into k fragments, determined as follows: $k = \lfloor \frac{m}{1000} \rfloor \cdot 100$ if m ¿ 1000, and $k = \lfloor \frac{m}{100} \rfloor \cdot 10$, otherwise, where m is the number of fragments of the shrink graphs, shown in Table 4. ARCFLAG called METIS to partition the graphs into k fragments as well. The results are reported in Figure 8.

The results tell us that (1) the space cost follows the order:

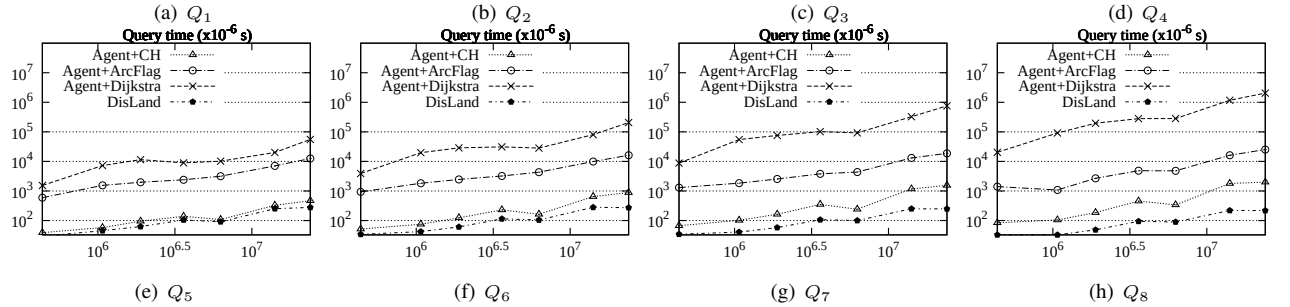


(a) Vary # of nodes (b) Vary # of nodes (c) Vary # of nodes (d) Vary # of nodes

Figure 8. Space overhead and preprocessing time

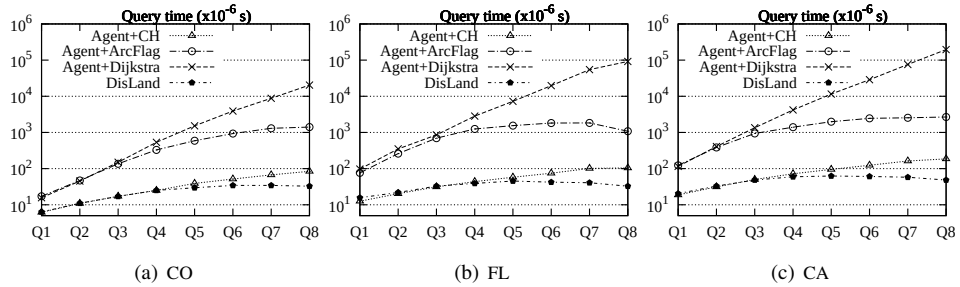


(a) Q_1 (b) Q_2 (c) Q_3 (d) Q_4

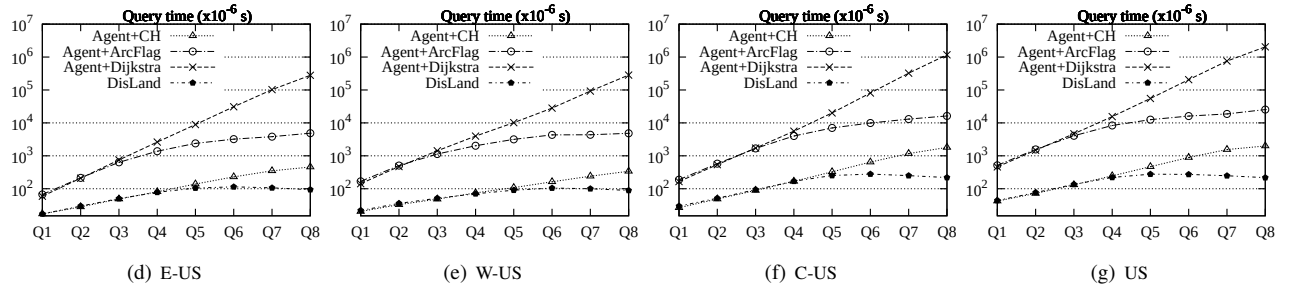


(e) Q_5 (f) Q_6 (g) Q_7 (h) Q_8

Figure 9. Performance evaluation *w.r.t.* graph sizes



(a) CO (b) FL (c) CA



(d) E-US (e) W-US (f) C-US (g) US

Figure 10. Performance evaluation *w.r.t.* distance queries

ARCFLAG > Agents + ARCFLAG > DISLAND > Agents + CH > Agent + Dijkstra > Dijkstra > CH; and (2) the preprocessing time follows the order: ARCFLAG > Agents + ARCFLAG > DISLAND > Agents + CH > CH > Agent + Dijkstra. In particular, CH even uses less space than the original graphs, and DISLAND uses about $1/2$ time extra space, while Agent + ARCFLAG and ARCFLAG use 1.66 and 1.24 times extra space, respectively. While CH and DISLAND could finish the preprocessing in less than 0.5 and 11 hours, repectively, it took Agent + ARCFLAG and ARCFLAG 26 and 40 hours, respectively. Thus all approaches, except ARCFLAG and Agents + ARCFLAG, produce auxiliary structures with a small space cost and in a reasonably fast way.

Exp-5: Efficiency of shortest distance queries. In the last set of experiments, using the 8 sets $Q_1, \ldots, Q_8$ of distance queries, we tested the efficiency of Dijkstra, Agent + Di-

jkstra, CH, Agents + CH, ARCFLAG, Agents + ARCFLAG, and DISLAND on the 7 datasets with corresponding generated auxiliary structures. The results are reported in Figures 9 and 10. As for all algorithms, their counterparts with agents were always faster, we omitted their running time for clarity.

The results tell us that (1) all algorithms scale well *w.r.t.* the graph sizes and *w.r.t.* the distance queries, and (2) the efficiency of the algorithms follows the order: DISLAND, Agent + CH $>$ Agent + ARCFLAG $>$ Agent + Dijkstra. For the distance queries $(Q_1, \ldots, Q_4)$ with relative close distance node pairs, the running time of DISLAND and Agent + CH is comparable. However, for the distance queries $(Q_5, \ldots, Q_8)$ with relative long distance node pairs, DISLAND is apparently faster than Agent + CH. Indeed, for Q_8 on the US dataset, DISLAND is $14,540.1$, $9,430.2$, 134.9, 116.5 , 9.4 and 9.1 times faster than Dijkstra, Agent + Dijkstra, ARCFLAG, Agent + ARCFLAG, CH, and Agent + CH, respectively.

Summary. From these experimental results, we find the following. (1) DISLAND scales well on large road graphs, *e.g.,* it takes only 0.28×10^{-3} seconds on graphs with 2.4×10^7 nodes and 5.7×10^7 edges. (2) Agents and their DRAs are a light-weight preprocessing technique, which benefits almost all shortest distance algorithms. (3) Agents, graph partitions and hybrid landmark covers together provide a good solution to produce small SUPER graphs, which typically have 2–4% nodes and 10–15% edges compared with the original graphs. (4) DISLAND produces auxiliary structures with a small space cost (about $1/2$ of the input graphs), and their preprocessing could be finished in a reasonably fast way. (5) DISLAND provides a good solution for shortest distance query answering, especially for far node pairs on large graphs. For Q_8 on the US dataset, it is even 9.1 times faster than Agent + CH, where CH is the best approach without using extra information, *e.g.,* longitude and latitude, tested in [34]. Finally, (6) hybrid landmark covers play a central role that makes our proposed techniques (*e.g.,* agents and graph partitions) and the existing techniques (*e.g.,* CH and ARCFLAG) seamlessly integrate into a unified framework – DISLAND.

VIII. CONCLUSION

We have studied how to apply distance landmarks for fast exact shortest distance query answering on large weighted undirected road graphs. To our knowledge, we are among the first to settle this problem. We have shown that the direct application of distance landmarks is impractical due to their high space and time cost. To rectify these problems, we have proposed: hybrid landmark covers, agents and DRAs, bounded graph partitions, SUPER graphs and framework DISLAND. We have also verified, both analytically and experimentally, that hybrid landmark covers, together with these techniques, significantly improve efficiency of shortest distance queries.

Several topics are targeted for future work. We are to extend our techniques for other types real-life datasets that could be modeled as weighted undirected graphs, *e.g.,* social networks. We are also to explore the possibility of applying distance landmarks for other classes of graph queries, *e.g.,* reachability.

REFERENCES

[1] I. Abraham, D. Delling, A. V. Goldberg, and R. F. F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In *SEA*, 2011.

[2] A. Z. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. L. Wiener. Graph structure in the web. *Computer Networks*, 33(1-6):309–320, 2000.

[3] CH. *http://algo2.iti.kit.edu/english/routeplanning.php.*

[4] E. P. F. Chan and H. Lim. Optimization and evaluation of shortest path queries. *VLDB J.*, 16(3):343–369, 2007.

[5] J. Cheng, Y. Ke, S. Chu, and C. Cheng. Efficient processing of distance queries in large graphs: a vertex cover approach. In *SIGMOD*, 2012.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2001.

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

[8] DIMACS. *http://www.dis.uniroma1.it/challenge9.*

[9] M. Franceschet. Collaboration in computer science: A network science approach. *JASIST*, 62(10):1992–2012, 2011.

[10] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *FOCS*, 1984.

[11] Full version. http://mashuai.buaa.edu.cn/full.pdf.

[12] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[13] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *WEA*, 2008.

[14] A. V. Goldberg and C. Harrelson. Computing the shortest path: *A* search meets graph theory. In *SODA*, 2005.

[15] J. E. Hopcroft and R. E. Tarjan. Efficient algorithms for graph manipulation [h] (algorithm 447). *Commun. ACM*, 16(6):372–378, 1973.

[16] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SISC*, 20(1):359–392, 1998.

[17] B. W. Kernighan and S. Lin. An efficientheuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(1):13–21, 1970.

[18] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *KDD*, 2009.

[19] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Statistical properties of community structure in large social and information networks. In *WWW*, 2008.

[20] M. Luby and P. Ragde. A bidirectional shortest-path algorithm with good average-case behavior. *Algorithmica*, 4(4):551–567, 1989.

[21] Metis. *http://glaros.dtc.umn.edu/gkhome/views/metis.*

[22] R. H. Möhring, H. Schilling, B. Schütz, D. Wagner, and T. Willhalm. Partitioning graphs to speedup Dijkstra's algorithm. *ACM Journal of EA*, 11, 2006.

[23] S. Mozes and C. Sommer. Exact distance oracles for planar graphs. In *SODA*, 2012.

[24] M. Potamias, F. Bonchi, C. Castillo, and A. Gionis. Fast shortest path distance estimation in large networks. In *CIKM*, 2009.

[25] P. Sanders and D. Schultes. Highway hierarchies hasten exact shortest path queries. In *ESA*, 2005.

[26] J. Sankaranarayanan and H. Samet. Query processing using distance oracles for spatial networks. *TKDE*, 22(8):1158–1175, 2010.

[27] J. Sankaranarayanan, H. Samet, and H. Alborzi. Path oracles for spatial networks. *PVLDB*, 2(1), 2009.

[28] A. D. Sarma, S. Gollapudi, M. Najork, and R. Panigrahy. A sketch-based distance oracle for web-scale graphs. In *WSDM*, 2010.

[29] S. Saunders and T. Takaoka. Solving shortest paths efficiently on nearly acyclic directed graphs. *TCS*, 370(1-3):94–109, 2007.

[30] M. Thorup and U. Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005.

[31] V. V. Vazirani. *Approximation Algorithms*. Springer, 2003.

[32] D. Wagner and T. Willhalm. Speed-up techniques for shortest-path computations. In *STACS*, 2007.

[33] F. Wei. Tedi: efficient shortest path query answering on graphs. In *SIGMOD*, 2010.

[34] L. Wu, X. Xiao, D. Deng, G. Cong, A. D. Zhu, and S. Zhou. Shortest path and distance queries on road networks: An experimental evaluation. *PVLDB*, 5(5), 2012.

[35] S. Yang, X. Yan, B. Zong, and A. Khan. Towards effective partition management for large graphs. In *SIGMOD*, 2012.

APPENDIX: Proofs

A. Proof of Theorem 2

Given a graph $G(V, E)$ and a set $L \subseteq V$ of nodes, (1) L is said to be an LMC of G iff for any two nodes $v, v' \in V$, there exists at least one node in L that lies in a shortest path from v to v'; and (2) L is said to be a VC of G iff for any edge $(v, v') \in E$, either v or v' is in L.

Lemma 15: *Given a redundant-edge-free graph $G(V, E)$ and a set $L \subseteq V$ of nodes, if L is a* VC *of G, then L is an* LMC *of G as well.* □

Proof: Consider a shortest path $< v_1, v_2, \ldots, v_{k-1}, v_k >$ between nodes v_1 and v_k, in which for each $i \in [1, k-1]$, (v_i, v_{i+1}) is an edge in G. Since L is a VC of G, $L \cap \{v_i, v_{i+1}\} \neq \emptyset$ for any $1 \leq i \leq k-1$. Hence, L is an LMC of G. ■

Lemma 16: *Given a redundant-edge-free graph $G(V, E)$ and a set $L \subseteq V$ of nodes, if L is a* LMC *of G, then L is an* VC *of G as well.* □

Proof: Consider an edge (v, v') in G. By the definition of redundant-edge-free graphs, (v, v') is the *only* shortest path between v and v'. Since L is an LMC of G, $L \cap \{v, v'\} \neq \emptyset$. Hence, L is a VC of G. ■

By Lemmas 15 and 16, we have the conclusion. □

B. Proof of Proposition 3

We show this by contradiction.

Assume first that there exists an agent u such that there are two distinct DRAs: $G[A^+_{u,1}]$ and $G[A^+_{u,2}]$. Then it is trivial to verify the following:

(1) $A^+_{u,1} \not\subset A^+_{u,2}$;
(2) $A^+_{u,2} \not\subset A^+_{u,1}$; and
(3) $A^+_{u,1} \cap A^+_{u,2}$ contains a set of at least 2 nodes, one among which is u.

By the definition of agents, u is an agent of all the set of nodes in $A^+_{u,1} \cup A^+_{u,2}$. Hence, the DRA of u should be $G[A^+_{u,1} \cup A^+_{u,2}]$. That is, neither $G[A^+_{u,1}]$ nor $G[A^+_{u,2}]$ is a DRA of u, which contradicts to our previous assumption. □

C. Proof of Proposition 4

Given any node u in a graph G, all nodes reachable to u are in A_u. All nodes reachable to u are exactly those nodes lie in the CC to which u belongs. From this, it is easy to have the conclusion. □

D. Proof of Proposition 5

Consider an agents u in a graph, and two nodes v and v' in the DRA $G[A^+_u]$.

Let G_s be the subgraph of G with the removal of u, and let cc_1, ..., cc_h be the CCs whose sizes are equal or less than $c \cdot \lfloor\sqrt{|V|}\rfloor - 1$. Note that $G[A^+_u]$ is simply the union of cc_1, ..., cc_h, together with u.

Case (1) when v and v' are in a single CC cc_j ($1 \leq j \leq h$).

This is easy.

Case (2) when v and v' are two distinct CCs cc_i and cc_j ($1 \leq i \neq j \leq h$).

dist (v,v') = dist (v,u) + dist (u,v')

E. Proof of Proposition 6

This is easy.

F. Proof of Proposition 7

Consider a connected graph $G(V, E)$ with $|V| > c \cdot \lfloor\sqrt{|V|}\rfloor$. We show this by contradiction.

Assume first that an agent u of graph G is not a cut-node. Let $G \setminus \{u\}$ be the subgraph of G by removing node u from G. Note that $G \setminus \{u\}$ remains connected. From the definition of agents, at least one neighbor v of u must belong to A_u. All nodes in $G \setminus \{u\}$ are reachable to v.

From these, it is easy to see that $G[A_u]$ is exactly G. Note that the size of G is larger than $c \cdot \lfloor\sqrt{|V|}\rfloor$. Hence, u is not an agent of G, which contradicts to our assumption. □

G. Proof of Proposition 8

This is easy.

H. Proof of Theorem 9

We first prove a more general conclusion, shown below.

Lemma 17: *Given two agents u and u',*
(1) if $u \in A^+_{u'}$, then $A^+_u \subseteq A^+_{u'}$;
(2) if $u' \in A^+_u$, then $A^+_{u'} \subseteq A^+_u$; and
(3) $A^+_u \cap A^+_{u'} = \emptyset$, otherwise. □

Consider two agents u and u' in a graph.

Let G_s be the subgraph of G with the removal of u, and let $cc_{u,1}$, ..., $cc_{u,h}$ be the CCs whose sizes are equal or less than $c \cdot \lfloor\sqrt{|V|}\rfloor - 1$. Note that $G[A^+_u]$ is simply the union of $cc_{u,1}$, ..., $cc_{u,h}$, together with u;

Let G'_s be the subgraph of G with the removal of u', and let $cc_{u',1}$, ..., $cc_{u',k}$ be the CCs whose sizes are equal or less than $c \cdot \lfloor\sqrt{|V|}\rfloor - 1$. Note that $G[A^+_{u'}]$ is simply the union of $cc_{u',1}$, ..., $cc_{u',k}$, together with u'.

(I) To show (1) and (2), it suffices to prove (1) only. By symmetry we have (2) when (1) is true.

We show this by contradiction. Assume first $A_u^+ \not\subseteq A_{u'}^+$ when $u \in A_{u'}^+$. Then there exists a node $w \in A_u^+$, but $w \notin A_{u'}^+$.

- Since $u \in A_{u'}^+$, u must belong to a CC in subgraph G'_s whose size is equal or less than $c \cdot \lfloor\sqrt{|V|}\rfloor - 1$.
- Since $w \notin A_{u'}^+$, w must belong to a CC in subgraph G'_s whose size is larger than $c \cdot \lfloor\sqrt{|V|}\rfloor - 1$.

From above, we know that u' and w must belong to a same CC in subgraph G_s after the removal of u. Hence, the size of the CC is larger than $c \cdot \lfloor\sqrt{|V|}\rfloor$, which implies that w is not in A_u^+. This contradicts to our assumption.

(II) Consider the case when $u \notin A_{u'}^+$ and $u' \notin A_u^+$. This is easy based on the analysis of (I).

By Lemma 17, Theorem 9 follows immediately. □

I. Proof of Theorem 11

This is based on the properties of agents and DRAs in Section IV-A, Proposition 12, and the correctness and complexity of algorithm compDRAs .

J. Proof of Proposition 12

We show this by contradiction. Assume first there is a cycle $B_1, v_1, B_2, v_2, \ldots, B_k, v_k, B_1$ in a sketch graph $\mathbb{G}$ of graph G, where $B_1, \ldots, B_k$ are BCCs and $v_1, \ldots, v_k$ are cut-nodes. That is, the removal of any v_j $j \in [1, k]$ does not increase the number of CCs in graph G.

However, by the definition of cut-nodes, the removal of any nodes in $v_1, \ldots, v_k$ will increase the number of CCs in G. This is a contradiction.

Putting these together, we conclude that there exist no cycles in sketch graphs. □

K. Proof of Proposition 13

Upper bound. We first show that the (Γ, ϵ)-BGP problem is in NP. An NP algorithm is given as follows: first guess a partition $(V_1, \ldots, V_k)$ of G, and then check whether $|V_i| \leq \Gamma$ for each $i \in [1, k]$, and (2) $|B| \leq \epsilon \cdot |V|$, in which $|B|$ is the total number of boundary nodes.

Lower bound. We next show that this problem is NP-hard by reduction from the (K, J) graph partitioning problem, which is NP-complete (cf. [12]).

Given graph $G(V, E)$, the (K, J) graph partitioning problem is to find a partition $(V_1, \ldots, V_h)$ of G such that $|V_i| \leq \Gamma$ for each $i \in [1, h]$ and the total number of boundary edges, whose endpoints belong to distinct V_j $j \in [1, k]$, is less than J.

Given a graph $G(V, E)$ of the (K, J) graph partitioning problem, we construct another graph $G'(V', E')$ such that there is a solution of G' for the (Γ, ϵ)-BGP problem iff there is a solution of G for the (K, J) graph partitioning problem

(1) We first construct the instance of the (Γ, ϵ)-BGP problem.

(a) The $G' = (V', E')$ is defined as follows:

- Let $V_u = \{x_{u,1}, \ldots, x_{u,2|V|^2+1}\}$ for each node $u \in V$;
- Let $E_u = \{(x, y) \mid x \neq y$ **and** $x, y \in V_u\}$ for each node $u \in V$;
- Let $E_e = \{(x_{u,i}, x_{v,j})\}$ for each edge $(u, v) \in E$ such that there exist no edges $(x_{u,i}, x_{v,j_1})$ and $(x_{u,i}, x_{v,j_2})$ with $u \neq v$.
- Let $V' = \bigcup_{u \in V} V_u$ and $E' = \bigcup_{u \in V} E_u \cup E_e$.

Note that here (1) $|V'| = |V|(|V|^2 + 1)$ and (2) $|E'| = |E| + |V|^3(|V|^2 + 1)/2$.

(b) Let $\Gamma = K(|V|^2 + 1)$ and let $\epsilon = 2J/|V'|$.

(2) We then show that there is a solution of G' for the (Γ, ϵ)-BGP problem iff there is a solution of G for the (K, J) graph partitioning problem

(a) First assume that $(V_1, \ldots, V_h)$ is a solution of the (K, J) graph partitioning problem We next construct a solution $(V'_1, \ldots, V'_h)$ for the (Γ, ϵ)-BGP problem.

Let $V'_1 = \{V_u \mid u \in V'_1\}$;

$\ldots$;

Let $V'_h = \{V_u \mid u \in V'_h\}$.

It is easy to know that (1) $|V'_i| \leq \Gamma$, and (2) the number of boundary nodes of $(V'_1, \ldots, V'_h)$ is equal or less $2J$ Hence $(V'_1, \ldots, V'_h)$ above is indeed a solution for the (Γ, ϵ)-BGP problem.

(b) Conversely, assume that $(V'_1, \ldots, V'_h)$ is a solution for the (Γ, ϵ)-BGP problem. We next construct a solution $(V'_1, \ldots, V'_h)$ for the (Γ, ϵ)-BGP problem.

Let $V_1 = \{u \mid V_{u,i} \in V'_1\}$;

$\ldots$;

Let $V_h = \{u \mid V_{u,i} \in V'_h\}$.

The first observation is that for any $u \in V$ of graph G, the set V_u of all nodes in G' must belong to a single V'_j $(1 \leq j \leq h)$ of the solution. Otherwise, the number of boundary nodes of the solution is $\geq 2|V|^2 + 1$. Note that $J < |V|^2$, and $\epsilon \leq 2|V|^2/|V'|$. That is, the number of boundary nodes of any solution of the (Γ, ϵ)-BGP problem is $\leq 2|V|^2$.

From these, we know that $(V_1, \ldots, V_h)$ is a partition of the nodes V of graph G. It is also easy to know that (1) $|V_i| \leq K$, and (2) the number of boundary edges of $(V_1, \ldots, V_h)$ is equal or less J. Hence, $(V_1, \ldots, V_h)$ above is indeed a solution for the (K, J) graph partitioning problem.