Information Systems 69 (2017) 180–193

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

Shrink: Distance preserving graph compression

Amin Sadri[a,*], Flora D. Salim[a], Yongli Ren[a], Masoomeh Zameni[b], Jeffrey Chan[a], Timos Sellis[c]

[a] School of Sience, RMIT University, Melbourne, Australia
[b] School of Computing and Information Systems, University of Melbourne, Melbourne, Australia
[c] School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Australia

ARTICLE INFO

Article history:
Received 10 January 2017
Revised 19 April 2017
Accepted 1 June 2017
Available online 7 June 2017

Keywords:
Graph compression
Graph simplification
Graph databases
Shortest paths

ABSTRACT

The ever increasing size of graphs makes them difficult to query and store. In this paper, we present *Shrink*, a compression method that reduces the size of the graph while preserving the distances between the nodes. The compression is based on the iterative merging of the nodes. During each merging, a system of linear equations is solved to define new edge weights in a way that the new weights have the least effect on the distances. Merging nodes continues until the desired size for the compressed graph is reached. The compressed graph, also known as the coarse graph, can be queried without decompression. As the complexity of distance-based queries such as shortest path queries is highly dependent on the size of the graph, *Shrink* improves the performance in terms of time and storage. *Shrink* not only provides the length of the shortest path but also identifies the nodes on the path. The approach has been applied to both weighted and unweighted graphs including road network, friendship network, collaboration network, web graph and social network. In the experiment, a road network with more than 2.5 million nodes is reduced to fifth while the average relative error is less than 1%.

1. Introduction

Nowadays, it is increasingly common to find graphs with millions of nodes in various domains. Due to the commonness of large graphs, graph compression is becoming an important research topic. In this process, also known as graph simplification, the complexity of the graph is reduced while certain characteristics of the graph are preserved. Graph compression can be performed by reducing the number of edges, nodes or extracting a high-level abstraction of the graph. A similar problem is graph summarization where summary graph uncovers the underlying topology of the original graph [12,13,24,34].

Graph compression has three main purposes. First, graph compression algorithms produce a simpler graph that can be queried faster than the original graph. This is useful for graph-based mining algorithms and also more complex problems (e.g. measuring similarities between graphs). Distance-based queries such as shortest-path have a great importance [6,7,19,35,42]. Many fundamental tasks in graph mining, such as computing diameter, closeness, centrality, and betweenness centrality, are dependent on computing shortest path distance. It has countless applications in transportation networks [3,23,43], networking [8,40], and databases [33]. Distance preserving graph compression speeds up the shortest path queries because they run on a smaller graph. Second, the compressed graph, also known as the coarse graph, can be stored in less space. Over the past few years, due to increasing the size of graph-structured databases, it becomes challenging and expensive to store the data, and graph compression techniques are deployed to reduce space consumption [10,24,25]. Third, graph compression algorithms help the users to understand and visualize the high-level structure of the graph [4,21,30] [21] [30]. It is almost impossible to understand the information encoded in large graphs with thousands or even millions of nodes by only visual inspection [36]. The coarse graph, produced by compression, is smaller and easier to be visualized [18].

Despite the importance of distance-based queries, only a few compression methods were proposed to preserve the distance [32,37]. Moreover, most of these existing methods are designed to compress unweighted graphs, and are not compatible with weighted graphs [9,10,14]. Furthermore, another challenge is that sometimes, the compression ratio needs to be chosen by the user. In this case, the user is able to control the trade-off between the accuracy and the size of the compressed graph. However, in exist-

* Corresponding author.
E-mail addresses: amin.sadri@rmit.edu.au (A. Sadri), flora.salim@rmit.edu.au (F.D. Salim), yongli.ren@rmit.edu.au (Y. Ren), mzameni@student.unimelb.edu.au (M. Zameni), jeffrey.chan@rmit.edu.au (J. Chan), tsellis@swin.edu.au (T. Sellis).

http://dx.doi.org/10.1016/j.is.2017.06.001

ing methods [16,32], the compression ratio is determined by the method, not the user.

In this paper, we introduce a novel distance preserving compression method *Shrink*, which can be used to query and store both weighted and unweighted graphs, e.g. enhancing shortest-path or other distance-based algorithms. Compressing with *Shrink* has the least effect on the distances between nodes. Specifically, when merging two nodes to a supernode, a system of equations is introduced to minimize the distance variations caused by this merge. The rationale behind this is to keep the mean of caused error equal to zero. These equations determine the edge weights connected to the supernode. After each merging, the number of nodes decreases by one and merging stage, called coarsening stage, continues until the desired size is achieved. The next stage is executing stage where the distance based query runs on the coarse graph. The last stage, refining stage, is optional that provides the path between the queried nodes. We have theoretically proved that for long paths, the error of compression converges to zero if merging errors are independent.

To sum up, *Shrink* possesses the following features: (1) it is linear in the number of nodes when $\sigma \ll |V|$, where $|V|$ is the number of nodes and σ is the average degree (see Section 4.2). This is common in large graphs with thousands or millions of nodes; (2) the larger the original graph is, the more accurate *Shrink* is. The reason is that large graphs usually have long paths and *Shrink* has less effect on the length of the long paths (see Section 3.2 and 6.4); (3) the error rate and the compression ratio are adjustable; (4) it provides not only distances but also the corresponding instances (nodes) of the shortest paths; (5) it is applicable to all types of distance queries, including reachability, single-source shortest-path (SSSP), all-pairs shortest path (APSP), closeness centrality and betweenness centrality. The experiment results show that compressing a two-million-node graph into fifths has the average error less than 1%.

The main contributions of the paper are as follows:

- We propose a compression method *Shrink* that has the least effect on the distances between the nodes.
- *Shrink* is fast and linear time complexity in the number of nodes, $O(|V|)$. Hence, it is applicable to large graphs.
- The proposed method is evaluated on both weighted and unweighted real-world data sets, including road network, friendship network, collaboration network, web graph and social network, etc.

In the next section, we have definitions and problem statement. In Section 3, first, we present the baselines for defining the equations and discuss why the equations are suitable for assignment of the new weights. Then, an overview of *Shrink* is provided. Three stages of *Shrink* are described in section 4 and 5. We evaluate our method in terms of time, accuracy and storage in Section 6. Finally, Section 7 and 8 discuss related work and conclusion, respectively.

2. Problem formulation

In this section, we first introduce the necessary notation to describe the problem formulation, and then we state the problem.

Definition 1. *Original graph* is the input graph which is a triple $G = (V, E, w)$ where V is a set of nodes (or vertices), $E \subset V \times V$ denotes edges, and $w : E \to R^+$ assigns a non-negative weight to each edge $e \in E$.

In this paper, the original graphs can be either unweighted or weighted. For unweighted graphs, the same weight can be assigned to all edges. The notations used in this paper are listed in Table 1.

Definition 2. *Coarse graph*, $G' = (V', E', w')$, is the compressed graph. $V' = \{v_1', \ldots, v_n'\}$ is a partition of V (i.e. $v_i' \subset V$ for all $i, \bigcup_i v_i' = V$, and $v_i' \bigcap v_j' = \emptyset$ for all $v_j \neq v_i$). Namely, each node $v_i' \in V'$, also known as a supernode, may consist of some nodes in G. E' denotes the edges set $E' \subset V' \times V'$, $w' : E' \to R^+$. In contrast to the nodes, there is no mapping between the edges.

Table 1
Definition of the variables.

Variable	Definition
G	Original graph: $G = (V, E, w)$
V	Set of the nodes in the original graph
E	Set of the edges in the original graph: $E \subset V \times V$
w	Weights on E in the original graph:$w : E \to R^+$
x, y	two nodes in the original graph $x \in V, y \in V$
u, v	To be merged nodes in the original graph $u \in V, v \in V$
G'	Coarse graph: $G' = (V', E', w')$
V'	Set of the nodes in the coarse graph
E'	Set of the edges in the coarse graph: $E' \subset V' \times V'$
w'	Weights on E' in the coarse graph: $w' : E' \to R^+$
v'	Supernode in the coarse graph $v' \in V'$
k	Number of v's neighbors
$N(u)$	Set of u's neighbors that are not connected to v
$N(v)$	Set of v's neighbors that are not connected to u
$N(uv)$	Set of Common neighbors of u and v
N	Set of neighbors of u and v: $N = N(u) \cup N(v) \cup N(uv)$
v_i	A neighbour of u or v
w_{vi}	weight of the edge between v_i and v
w_{ui}	weight of the edge between v_i and u
w'_{vi}	weight of the edge between v_i and v'
$l(v_i, v_j)$	Length of the path that connects v_i and v_j and crosses u or v in the original graph
$l'(v_i, v_j)$	Length of the path that connects v_i and v_j and crosses v' in the coarse graph

$$E' = \{(u', v') | u \in u', v \in v', (u, v) \in E\} \quad (1)$$

Specifically, two supernodes are connected if and only if there is a node in one supernode that is connected to a node in the other supernode. Here, the main problem is assigning weights to the new edges. To this end, we define and solve equations to have new weights with the least effects on the distances between nodes.

Definition 3. The *distance* between x and y is $d(x, y)$, which is the length of the shortest path between x and y in the original graph. The shortest path is a path with the lowest total sum of edge weights. Similarly, $d'(x, y)$ denotes the length of the shortest path between the supernodes that contain x and y in the coarse graph.

Definition 4. Error of the compression for nodes x and y, $Err(x, y)$, is the difference between the distance of x and y in the original graph and the distance of the supernodes that x and y belongs to in the coarse graph.

$$Err(x, y) = \left| d(x, y) - d'(x, y) \right| , x \neq y \quad (2)$$

Definition 5. Normalizing $Err(x, y)$ with $d(x, y)$, we have the *relative error* of nodes x and y.

$$RErr(x, y) = \frac{|d(x, y) - d'(x, y)|}{d(x, y)} , x \neq y \quad (3)$$

where $RErr(x, y)$ denotes the relative error.

Definition 6. Given G and G', the compression ratio of the coarsening stage is defined as $CR(G') = \frac{|V'|}{|V|}$. The number of edges is not included in the definition.

Problem: Given the original graph G and a compression ratio CR, $0 < CR < 1$, how to define G' such that the sum of the errors is minimum over all pairs. Specifically, the cost function that should be minimized is as follow:

$$\sum_{x,y \in V} Err(x, y) , x \neq y \quad (4)$$

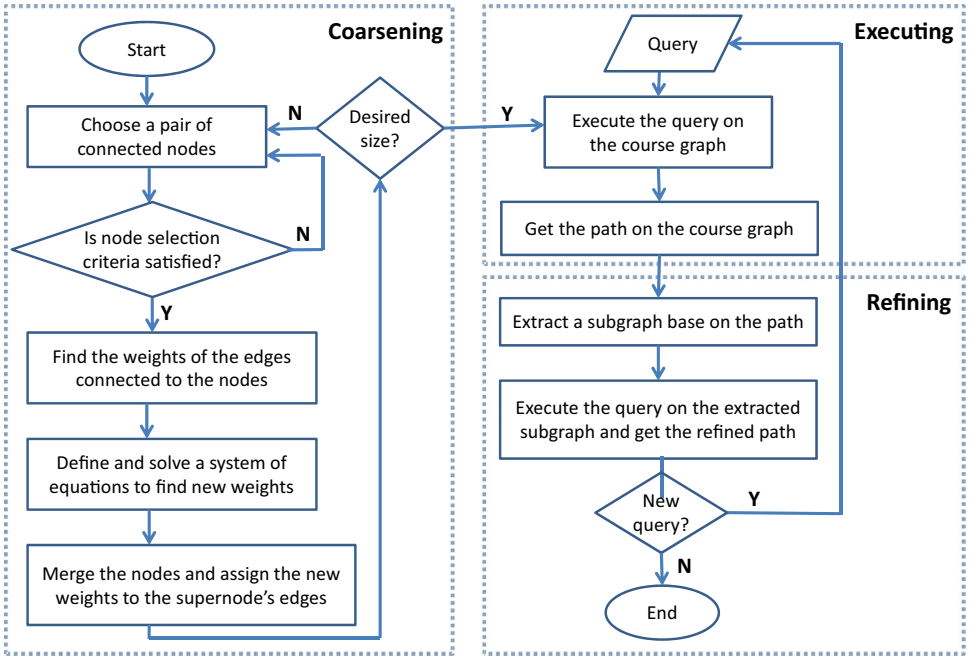

Fig. 1. *Shrink* has three stages: Coarsening, Executing, Refining. Queries are run on the coarse graph produced in the coarsening stage.

3. Shrink method

In this section, we introduce the overview of the proposed method. Then, we present preliminary concepts and discuss the rationale behind the method.

3.1. Overview

Shrink consists of three components: 1) Coarsening stage: This is the first stage in which the coarse graph is constructed by merging pairs of connected nodes iteratively until the desired size is achieved. The main challenge is finding the new edge weights for the supernodes with the least effect on the distances in the graph. We define a system of linear equations to find the new weights. We also propose a speed-up technique to solve the equations. It should be noted that the coarse graph is constructed once and after that, it can be queried for all types of the distance-based queries. 2) Executing stage: In this stage, the query is executed on the coarse graph. As the coarse graph is smaller than the original graph, execution is faster and requires lower heap size. 3) Refining stage: This stage is optional. If only an estimation of the distance is enough, the refining stage is not needed, and the original graph can be discarded. On the other hand, if the laying nodes on the path are needed, the refining stage is necessary. In the refining stage, we extract a subgraph by projecting the output path to the original graph. Then, the path is refined by rerunning the query on the subgraph from the original graph. Fig. 1 demonstrates the flowchart of *Shrink* including the three stages.

Our main focus is on the coarsening stage where we try to compress the graph while preserving the distances. In the next subsection, we discuss our principal for assigning the new weights to edges connected to a supernode.

3.2. Preliminary

Merging a pair of nodes changes the distances because by merging, the structure of the graph is changed. The merging error is consequently defined in terms of the change in the length of a shortest-path caused by the merging.

Definition 7. Assume that v and u are merged into v' (See Fig. 2). pa is a shortest-path between two unknown nodes in the original graph that passes through u or v. As a result, the length of pa will be affected by the merging. The merging error caused by merging u and v on pa ($M_{uv}(pa)$) is defined as follow:

$$M_{uv}(pa) = l(pa) - l'(pa) \tag{5}$$

where $l(pa)$ is the length of the part of pa in the original graph that will be changed after merging. This part of pa consists of edges connected to u and v. $l'(pa)$ is the length of the part of pa in the coarse graph that is changed after merging. This part of pa consists of two edges connected to v'.

The value of the merging error depends on how the path passes through the merged nodes and their neighbors. For example, in Fig. 2, if pa passes through v_i, u, v, and v_j, $l(pa)$ is $w_{ui} + w_{uv} + w_{vj}$. In the coarse graph, $l'(pa)$ is $w'_{vi} + w'_{vj}$. The merging error of u and v on this path is $(w_{ui} + w_{uv} + w_{vj}) - (w'_{vi} + w'_{vj})$. The merging error could be positive or negative depending on v_i, v_j, w'_{vi} and w'_{vj}.

By considering the occurrence of the neighbors of u and v in the shortest paths, we define a random variable regarding the merging error. To have a better understanding of this random variable, assume that someone starts marking the edges on the shortest paths one by one and we can only see u and v neighbors and the rest of the graph is a black box. The random variable presents the merging error when the edges connected to u or v are marked. Specifically, a probability assigned to the random variable, such as p_{ij}, denotes the probability that edges connected to v_i and v_j from the visible part are marked.

The PDF (Probability Density Function) of the random variable shows the impact of the merging on the distances/shortest paths. For example, from the PDF, we can calculate the probability that a random path passing through u or v becomes shorter after the merging. The PDF is computed based on the probability of occurrence of each pair of neighbors. Fig. 3 illustrates a long path in the coarse graph consisting of many supernodes as well as the PDF of the merging errors. The total error, the difference between the length of the path in the coarse and original graphs, is equal to the

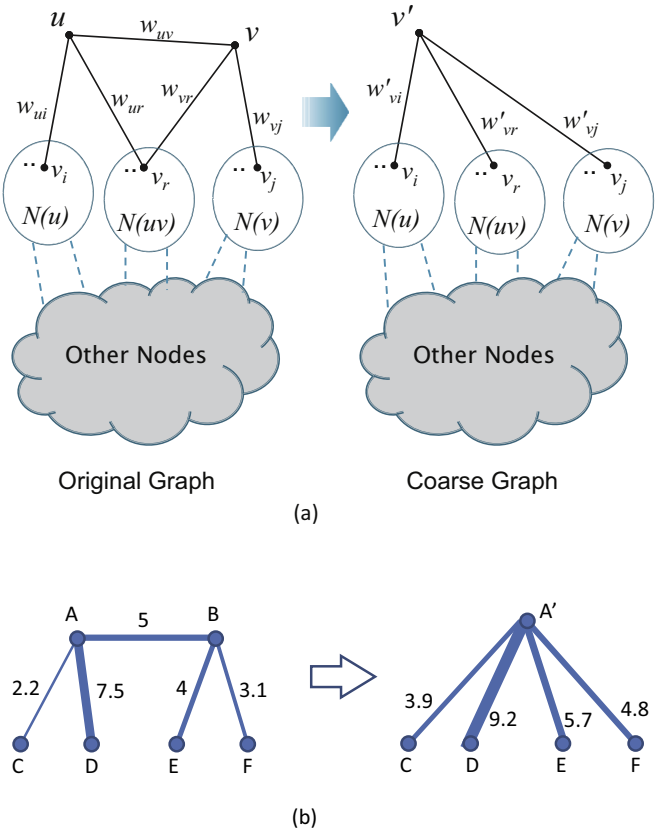

Fig. 2. (a) u and v are merged into v' connected to the neighbors of both u and v. The connectivity pattern of the other parts of the graph remains the same. (b) A simple example where A and B are merged into A′. The weights of the new edges connected to A′ are determined by *Shrink*.

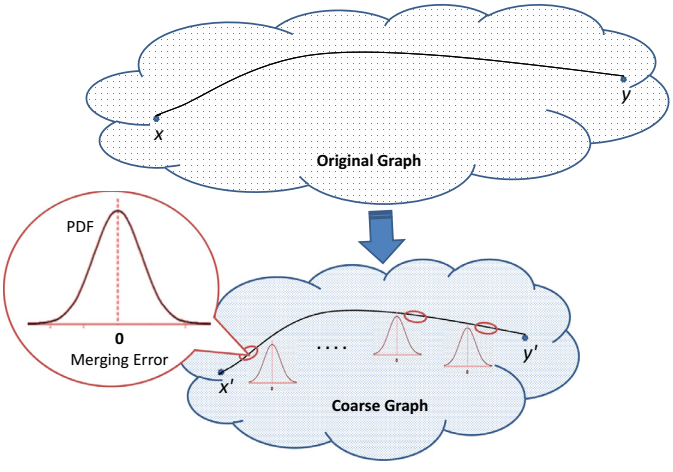

Fig. 3. Each supernode on the path causes merging error that has a PDF. The proposed weight assignment makes every merging error mean equal to zero.

sum of the merging errors of the supernodes laying on the path. The total error is also a random variable and has a PDF.

After each merging, new weights should be assigned to the supernode edges. *Our main principle for defining the new weights is to keep the mean of each merging error equal to zero.* Here, we investigate the effect of this principle on mean and variance of total error.

3.2.1. Mean of error

According to probability theory, the mean of the sum of some random variables is equal to the sum of the random variable means [39]. In our problem, we have

$$E(M(pa)) = E(M_{u_1v_1}(pa)) + \cdots + E(M_{u_mv_m}(pa)) \tag{6}$$

where pa is the shortest path between two random nodes and $M(pa)$ is the total error which is the difference between the length of pa in the original graph and the length of pa in the coarse graph. u_i and v_i is a merged node pair laying on pa. Therefore, the mean of the error of a path is zero regardless of the correlation between merging errors of the supernodes laying on the path because it is equal to the sum of the merging error means that are set to zero. To sum up, if we consider the shortest path between two random nodes, the compression has zero effect on the path length on average.

We also argue that when the average error is zero, Mean Square Error (MSE) is minimum. According to probability theory, if μ and σ^2 are the mean and variance of the error respectively, MSE is equal to $\mu^2 + \sigma^2$ [39]. Based on this theorem, if the variance of error is σ^2, the minimum MSE is when $\mu = 0$. This means our weight assignment, which leads to a zero-mean error, is the *best* one when the variance is fixed.

3.2.2. Variance of error

The average of a sufficiently large number of iterates of independent[1] zero-mean random variables converges to zero, regardless of the underlying distributions (Law of Large Numbers [39]). Merging errors are zero-mean. Based on Law of Large numbers, if they are independent, for long paths, the relative errors are zero and the distances are preserved. Intuitively, some supernodes increase the length of the path while others decrease the path length. Overall, the change is negligible because the increases and decreases compensate for each other.

$$\lim_{m\to\infty} \frac{E(M_{u_1v_1}(pa)) + \cdots + E(M_{u_mv_m}(pa))}{m} = \lim_{m\to\infty} \frac{E(M(pa))}{m} = 0 \tag{7}$$

$$\lim_{m\to\infty} \frac{E(M(pa))}{m \times \overline{l(pa)}} = RErr(pa) = 0 \tag{8}$$

where $\overline{l(pa)}$ is the average value of $l(pa)_{u_iv_i}$ over u_i and v_i. $RErr(pa)$ is the relative error of path pa.

To sum up, zero-mean merging error has two advantages. First, the total error of a path becomes a zero-mean random variable that has the least MSE. Second, for the long paths, the total error is nearly zero if merging errors are independent.

4. Coarsening stage

The coarsening stage is the main stage where the size of the graph is reduced. By considering the underlying idea, our method seeks to reduce the graph size by merging pairs of nodes one after another, setting though the mean of merging error to zero. Let u is the closest neighbor of v and they satisfy the node selection criteria described in Section 4.3. It should be noted that u and v could be supernodes that result from the previous steps. The edge weights and the connectivity pattern of u and v are sufficient to define a system of equations and find the new weights.

Fig. 2(a) illustrates the general overview of the problem, focusing on the connectivity of nodes u and v. All of the nodes that

[1] Two random variables are independent when receiving information about one of the two does not change our assessment of the probability distribution of the other.

are connected to either u or v are connected to supernode v' but with different weights. The neighbors of u and v can be divided into three sets: $N(u)$, $N(v)$, and $N(uv)$. Nodes in set $N(u)$ are connected to u but not to v. Set $N(uv)$ consists of the nodes that are connected to both u and v, while $N(v)$ is the mirror of $N(u)$ and consists of the nodes that are connected to v but not u. Suppose N is the set of the neighbors of u and v, $N = N(u) \cup N(v) \cup N(uv)$ and k is the total number of neighbors, $k = |N|$. w_{uv} denotes the weight between u and v. Fig. 2(b) shows a simple example in which A and B are merged.

Given the edge weights in the original graph, the problem is finding k new edge weights between v' and the nodes in $N(u)$, $N(v)$, and $N(uv)$ so that the mean of the merging error becomes zero. For each merging, the equations should be defined and solved, and then new weights should be assigned to the new edges.

4.1. Defining equations

In this section, we define a system of k linear equations in the k variables to find the new weights where k is the total number of the neighbors of u and v. The equations imply that the changes in the length of paths that pass through the merged nodes after and before merging have a zero mean. We only consider parts of the paths that contains u or v and new edges because other parts do not change.

Each of the k equations is based on one of the new edge weights connecting v' to $v_i \in N$. The main principle states that the mean of the difference between the paths in the original and coarse graphs should be zero. Therefore, the mean (or average) value of path lengths that pass $v_i \in N$ and u or v in the original graph should be equal to the mean value of path lengths that pass v_i and v' in the coarse graph.

To calculate the mean value, the probabilities of the occurrence of the neighbors in the shortest paths are needed. Following the principle of indifference [23], it is assumed that all pairs of neighbors have the same probability of occurrence in the shortest paths because there is no information about the other parts of the graph. This assumption may not be true, and it drops accuracy since low weight edges are used more in shortest paths. Future work involves new techniques to estimate the usage probability distribution of the neighbors.

Lemma. *If the sum of $l(v_i, v_j)$ and the sum of $l'(v_i, v_j)$ over v_j are the same, the merging error of u and v for the paths that pass v_i and v' is a zero-mean random variable.*

Proof. We show that if the sum of $l(v_i, v_j)$ and the sum of $l'(v_i, v_j)$ are the same, then the mean of $l(v_i, v_j)$ and the mean of $l'(v_i, v_j)$ become the same. Therefore, the merging error becomes a zero-mean random variable because merging error is the difference between $l(v_i, v_j)$ and $l'(v_i, v_j)$. □

$$\sum_{v_j \in N, j \neq i} l'(v_i, v_j) = \sum_{v_j \in N, j \neq i} l(v_i, v_j) \tag{9}$$

$$\sum_{v_j \in N, j \neq i} p \times l'(v_i, v_j) = \sum_{v_j \in N, j \neq i} p \times l(v_i, v_j) \tag{10}$$

$$E(M_{uv}(v_i)) = \sum_{v_j \in N, j \neq i} p \times l'(v_i, v_j) - \sum_{v_j \in N, j \neq i} p \times l(v_i, v_j) = 0 \tag{11}$$

where p is the probability that v_i and v_j occur on a shortest-path connecting two nodes of the graph. $l(v_i, v_j)$ indicates the length of the part of the paths that includes v_i, v_j, and the edges to be deleted in the original graph. Likewise, $l'(v_i, v_j)$ indicates the length of the part of the paths that includes v_i, v_j, and new edges in the coarse graph. $M_{uv}(v_i)$ is merging error of the shortest paths passing v_i and v'. In these equations, it is not necessary to consider the complete length of the paths since other parts of the graph remain unchanged. Eqs. (9)–(11) focuses on v_i and the sum in these equations is over v_j (v_i is fixed). This means we have one equation for each neighbour. $l'(v_i, v_j)$ is a function of variables to be determined such as w'_{vx}, $w'_{v'y}$, and $w'_{v'z}$ while $l(v_i, v_j)$ is a function of w_{ux}, w_{vy}, and w_{uz} which are known and constant. Eq. (9) is a key equation that clarifies that sum of the part of the shortest paths' lengths that pass through v_i should be the same in the original and coarse graphs. In fact, if we want the merging process to have the minimum change in the shortest paths, the sum of $l'(v_i, v_j)$ and $l(v_i, v_j)$ should be the same. Based on the lemma, k linear equations are defined.

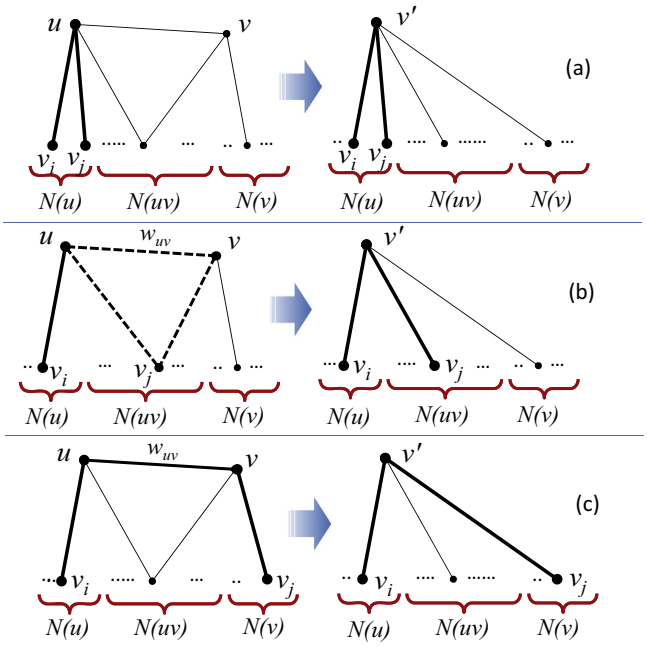

Fig. 4. When v_i is in set $N(u)$, the shortest paths that crosses v_i and v_j can be categorized into three cases: (a) $v_j \in N(u)$, (b) $v_j \in N(uv)$, and (c) $v_j \in N(v)$. (See rows 1–3 in Table 2).

$$\sum_{j \neq i, v_j \in N(u)} l'(v_i, v_j) + \sum_{j \neq i, v_j \in N(v)} l'(v_i, v_j) + \sum_{j \neq i, v_j \in N(uv)} l'(v_i, v_j) = \sum_{j \neq i, v_j \in N(u)} l(v_i, v_j) + \sum_{j \neq i, v_j \in N(v)} l(v_i, v_j) + \sum_{j \neq i, v_j \in N(uv)} l(v_i, v_j) \tag{12}$$

Eq. (12) is the extension of Eq. (9). The left side of the equation contains variables that should be computed while the right side of the equation is constant. The constant part can be determined based on the given edge weights in the original graph. In the rest of this section, $l(v_i, v_j)$ and $l'(v_i, v_j)$ are computed based on whether v_i and v_j are in set $N(u)$, $N(v)$, or $N(uv)$.

4.1.1. Equations for $v_i \in N(u)$

In this subsection, we write an equation for a node $v_i \in N(u)$. $l'(v_i, v_j)$ and $l(v_i, v_j)$ are computed based on whether node v_j belongs to $N(u)$, $N(v)$ or $N(uv)$. Fig. 4 shows three possible situations for v_j that is used to find the length of the part of the shortest paths which contains v_i and merged nodes. This path is shown by bold lines in the new and original graphs. Fig. 4(a) demonstrates a situation in which v_j belongs to set $N(u)$. Therefore, we can deter-

mine the first part of the right and the left side of Eq. (12).

$$\sum_{v_j \neq v_i, v_j \in N(u)} l'(v_i, v_j) \tag{13}$$
$$= \sum_{v_j \neq v_i, v_j \in N(u)} (w'_{vi} + w'_{vj})$$
$$= (|N(u)| - 1)w'_{vi} + \sum_{v_j \neq v_i, v_j \in N(u)} w'_{vj}$$

$$\sum_{v_j \neq v_i, v_j \in N(u)} l(v_i, v_j) = \sum_{v_j \neq v_i, v_j \in N(u)} (w_{ui} + w_{uj}) \tag{14}$$

In other words, if both i and j belong to $N(u)$, Eq. (13) and Eq. (14) indicate the sum of the length of the affected paths in the new and original graph, respectively. Fig. 4(b) is more complex. The exact value of the $l(v_i, v_j)$ depends on the w_{uj}, w_{uv}, and w_{vj}. If $w_{uj} < w_{uv} + w_{vj}$ then $l(v_i, v_j)$ is w_{uj} otherwise it is $w_{uv} + w_{vj}$.

$$\sum_{v_j \in N(uv)} l'(v_i, v_j) = \sum_{v_j \in N(uv)} (w'_{vi} + w'_{vj}) = |N(uv)|w'_{vi} + \sum_{v_j \in N(uv)} w'_{vj} \tag{15}$$

$$\sum_{v_j \in N(uv)} l(v_i, v_j) = \sum_{v_j \in N(uv)} (w_{ui} + min(w_{uj}, w_{uv} + w_{vj})) \tag{16}$$

Finally, if v_j belongs to set $N(v)$, the path that includes v_i, u, v, and v_j could be part of a shortest-path whose length is $w_{ui} + w_{vj} + w_{uv}$. Nodes in set $N(uv)$ cannot be in this path because v is the closest neighbor of u and w_{uv} has the minimum weight among the neighbors of u.

$$\sum_{v_j \in N(v)} l'(v_i, v_j) = \sum_{v_j \in N(v)} (w'_{vi} + w'_{vj}) = |N(v)|w'_{vi} + \sum_{v_j \in N(v)} w'_{vj} \tag{17}$$

$$\sum_{v_j \in N(v)} l(v_i, v_j) = \sum_{v_j \in N(v)} (w_{ui} + w_{uv} + w_{vj}) \tag{18}$$

Identifying $l(v_i, v_j)$ and $l'(v_i, v_j)$, it is possible to write the equation for v_i which is in set $N(u)$. Based on the main principle, replacing u and v with v' should not change the mean length of the shortest path. The idea results in the lemma that implies that the summation of the shortest path lengths that contain v_i should remain the same after merging. From Eqs. (14) to (18) we have

$$\sum_{v_j \neq v_i, v_j \in N(u)} l'(v_i, v_j) + \sum_{v_j \in N(v)} l'(v_i, v_j) + \sum_{v_j \in N(uv)} l'(v_i, v_j)$$
$$= (|N(u)| - 1)w'_{vx} + |N(uv)|w'_{vx} + |N(v)|w'_{vx}$$
$$+ \sum_{v_j \neq v_i, v_j \in N(u)} w'_{vj} + \sum_{v_j \in N(uv)} w'_{vj} + \sum_{v_j \in N(v)} w'_{vj}$$
$$= (k-2)w'_{vi} + \sum_{v_j \in N} w'_{vj} \tag{19}$$

This equation presents the left side of Eq. (12) that contains to be determined variables such as w'_{vi}. On the other hand, the right side of Eq. (12) is constant and can be calculated based on the edge weights in the original graph. To simplify the equations, let C_i be the constant value in the equation of node v_i:

$$C_i = \sum_{v_j \neq v_i, v_j \in N(u)} l(v_i, v_j) + \sum_{v_j \in N(uv)} l(v_i, v_j) + \sum_{v_j \in N(v)} l(v_i, v_j) \tag{20}$$

Based on Eqs. (12), (19), and (20), we have:

$$(k-2)w'_{vi} + \sum_{v_j \in N(u)} w'_{vj} + \sum_{v_j \in N} w'_{vj} = C_i \tag{21}$$

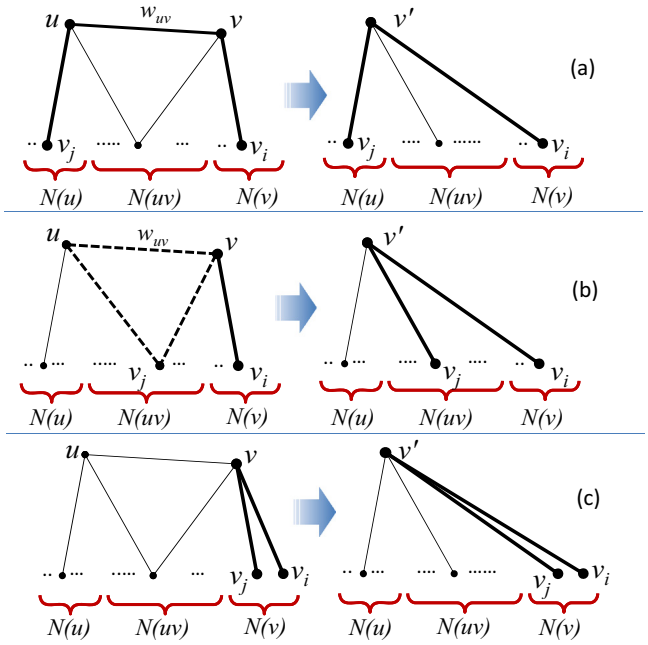

Fig. 5. When $v_i \in N(v)$, the shortest paths that crosses v_i and v_j can be categorized into three cases: (a) $v_j \in N(u)$, (b) $v_j \in N(uv)$, and (c) $v_j \in N(v)$. (See rows 4–6 in Table 2).

To sum up, for each node such as v_i, if the new weights are defined in a way that satisfy Eq. (21), the mean value of the shortest path lengths crossing through v_i does not change in the coarse graph.

4.1.2. Equations for $v_i \in N(v)$

Let v_i belong to set $N(v)$ that is connected to v not u. Similar to the previous subsection, we want to define right value for w'_{vi} and keep the average shortest path lengths that cross through v_i. Based on Fig. 5, $l'(v_i, v_j)$ and $l(v_i, v_j)$ are computed and reported in Table 2. Based on lemma and Eq. (12), we have the following equation for node $v_i \in N(v)$.

$$(k-2)w'_{vi} + \sum_{v_j \in N} w'_{vj} = C_i \tag{22}$$

where C_i is constant and equal to the sum of $l(v_i, v_j)$ when $v_j \in N(v)$.

4.1.3. Equations for $i \in N(uv)$

The formulas for $v_i \in N(uv)$ are shown in the last three rows in Table 2 (Fig. 6). Based on the main principle and the lemma, we have:

$$(k-2)w'_{vi} + \sum_{v_j \in N} w'_{vj} = C_i \tag{23}$$

Similarly, C_i is the constant and it is equal to the sum of $l(v_i, v_j)$ when $v_j \in N(uv)$.

Eqs. (21), (22), and (23) are similar. However, for each equation, the constant part of the equations should be computed separately.

4.2. Solving the equations

In the previous section, a system of k linear equations is defined to find the new weights. Specifically, for merging a couple of nodes, a system of linear equations should be solved which is a costly process. The complexity of solving a system of k linear equations scales with k^3, making the approach impractical for very large graphs. Fortunately, the proposed system of equations is not a

Table 2
$l(v_i, v_j)$ and $l'(v_i, v_j)$ in different cases. All the sums are over v_j. (v_i is fixed).

	v_i	v_j	$\sum_{v_j \neq v_i} l(v_i, v_j)$	$\sum_{v_j \neq v_i} l'(v_i, v_j)$
1	$N(u)$	$N(u)$	$\sum_{v_j \neq v_i}(w_{ui} + w_{uj})$	$\lvert N(u)\rvert w'_{vi} - w'_{vi} + \sum_{v_j \neq v_i} w'_{vj}$
2	$N(u)$	$N(uv)$	$\sum(w_{ui} + \min(w_{uj}, w_{uv} + w_{vj}))$	$\lvert N(uv)\rvert w'_{vi} + \sum w'_{vj}$
3	$N(u)$	$N(v)$	$\sum(w_{ui} + w_{uv} + w_{vj})$	$\lvert N(v)\rvert w'_{vi} + \sum w'_{vj}$
4	$N(v)$	$N(u)$	$\sum(w_{uj} + w_{uv} + w_{vi})$	$\lvert N(u)\rvert w'_{vi} + \sum w'_{vj}$
5	$N(v)$	$N(uv)$	$\sum(w_{vi} + \min(w_{vj}, w_{uv} + w_{uj}))$	$\lvert N(uv)\rvert w'_{vi} + \sum w'_{vj}$
6	$N(v)$	$N(v)$	$\sum_{v_j \neq v_i}(w_{vi} + w_{vj})$	$\lvert N(v)\rvert w'_{vi} - w'_{vi} + \sum_{v_j \neq v_i} w'_{vj}$
7	$N(uv)$	$N(u)$	$\sum(w_{uj} + \min(w_{ui}, w_{uv} + w_{vi}))$	$\lvert N(u)\rvert w'_{vi} + \sum w'_{vj}$
8	$N(uv)$	$N(uv)$	$\sum_{v_j \neq v_i} \min(w_{vj} + w_{vi}, w_{uj} + w_{ui}, w_{vj} + w_{uv} + w_{ui}, w_{vi} + w_{uv} + w_{uj})$	$\lvert N(uv)\rvert w'_{vi} - w'_{vi} + \sum_{v_j \neq v_i} w'_{vj}$
9	$N(uv)$	$N(v)$	$\sum(w_{vj} + \min(w_{vi}, w_{uv} + w_{ui}))$	$\lvert N(v)\rvert w'_{vi} + \sum w'_{vj}$

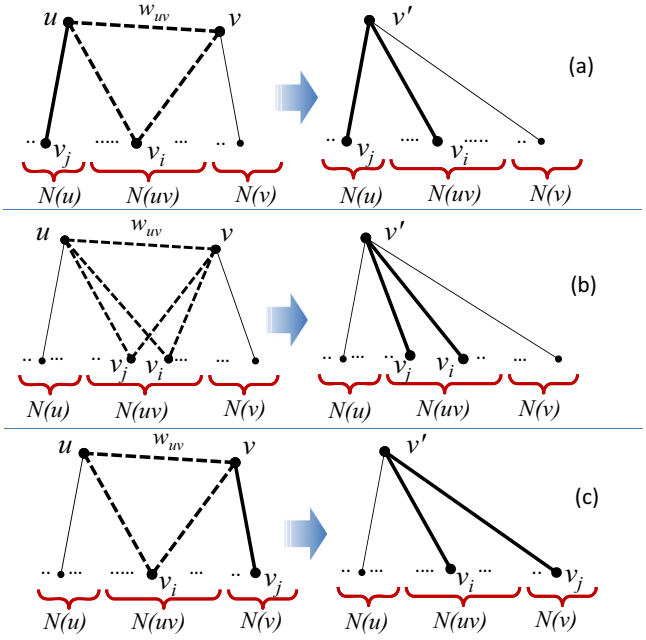

Fig. 6. When $v_i \in N(uv)$, the shortest paths that crosses v_i and v_j can be categorized into three cases: (a) $v_j \in N(u)$, (b) $v_j \in N(uv)$, and (c) $v_j \in N(v)$. (See rows 7–9 in Table 2).

general one and in this section, we demonstrate a straightforward way to solve the equations with the complexity of $O(k)$. Hence, there is no need to solve a general linear system of equations for each merging which is a time-consuming task.

To solve the equations, we define S as the sum of the weights of the new edges.

$$S = \sum_{v_j \in N} w'_{vj} \tag{24}$$

By replacing the sum of the new weights with S in Eqs. (21), (22), and (23), we have the system of k linear equations.

$$\begin{cases} (k-2)w'_{v1} + S = C_1 \\ \vdots \\ (k-2)w'_{vi} + S = C_i \\ \vdots \\ (k-2)w'_{vk} + S = C_k \end{cases} \tag{25}$$

S can be computed by summing up all the k equations.

$$(k-2)\left(\sum_{v_i \in N(u)} w'_{vi} + \sum_{v_i \in N(uv)} w'_{vi} + \sum_{v_i \in N(v)} w'_{vi}\right) + kS = C \tag{26}$$

$$(k-2)S + kS = C \tag{27}$$

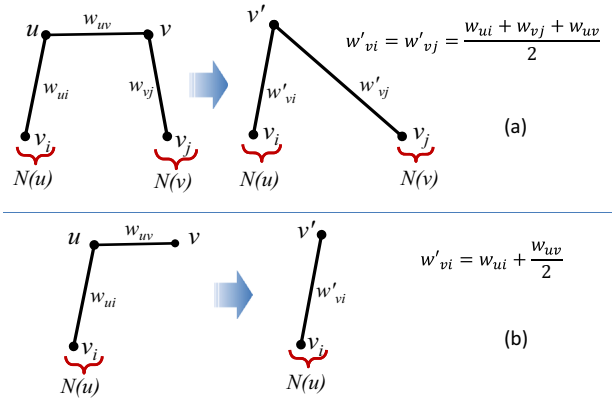

Fig. 7. $k = 2$, $|N(u)| = 1$, $|N(v)| = 1$. The sum of new weights should be $p_1 + q_1 + w_{uv}$. (b) $k = 1$, $|N(u)| = 1$. The figure shows u, v, and v' when $|N(u)| = 1$.

$$S = \frac{C}{2k-2} \tag{28}$$

where C denotes the sum of the constant values in Eq. (25). After identifying S, the new weights can be calculated. Eq. (29) provides new weights if $k > 2$.

$$w'_{vi} = \frac{C_i - S}{k-2} \tag{29}$$

Thus, the system of linear equations is solved in $O(k)$. Defining the equations requires the complexity of $O(k^2)$ because each of the constant values in Eq. (25) requires $O(k)$ to be determined. Therefore, the complexity of our method to merge one pair of nodes is $O(\sigma^2)$. The number of mergings is $(1 - CR)|V|$ which makes the total complexity equal to $O(\sigma^2|V|)$. In large graphs, it is common that $\sigma \ll |V|$. In this case, the total complexity is $O(|V|)$.

4.2.1. Special cases($k \leq 2$)

If $k = 2$, then both of the equations become the same. In this case, the system of equations has infinite solutions because there are two variables and only one equation. Specifically, every set of values for w'_{vi} and w'_{vj} that satisfy the following equations lead to a zero-mean error.

$$w'_{vi} + w'_{vj} = S = C_i \tag{30}$$

$$w'_{vi} = w'_{vj} = C_i/2 \tag{31}$$

$\tilde{q}_i$, or $\tilde{r}_i$. Fig. 7(a) illustrates the situation when $|N(u)| = |N(v)| = 1$.

If $k = 1$, then u and v cannot be part of a shortest-path unless they are the source or destination nodes. Here, for finding the new weight, we apply the same fact used for *Shrink*: for the least mean square error, the mean of error has to be zero. In fact, the w'_{vi} should be equal to the mean of the paths passing through v_i.

Similarly, the probabilities of starting the paths from v and u are assumed to be the same (i.e 1/2). Fig. 7(b) illustrates the situation when $|N(u)| = 1$. The following equations are applied to assign the new weights for $k = 1$.

$$|N(u)| = 1,\ w'_{vi} = \frac{w_{ui}}{2} + \frac{w_{ui} + w_{uv}}{2} = w_{ui} + \frac{w_{uv}}{2} \tag{32}$$

$$|N(uv)| = 1,\ w'_{vi} = \frac{w_{ui}}{2} + \frac{w_{vi}}{2} = \frac{w_{ui} + w_{vi}}{2} \tag{33}$$

$$|N(v)| = 1,\ w'_{vi} = \frac{w_{vi}}{2} + \frac{w_{uv} + w_{vi}}{2} = w_{vi} + \frac{w_{uv}}{2} \tag{34}$$

To sum up, if $k = 2$ or $k = 1$, Eqs. (31)–(34) determine the new weights. If $k = 0$ then we do not need to define new weights and v' will be a single node.

4.3. Node selection criteria

To minimize the error, we define the following rule for the nodes that are going to be merged (i.e. v and u): *the node pair is desirable for merging if it has a small number of neighbors*. Specifically, given two nodes u and v, we define the multiplication of the number of the neighbors of the nodes, as a way for node selection. Hence, if the result is small enough, the selected nodes are suitable for merging. The simple way is setting a constant threshold, Θ, on the multiplication result that has two drawbacks. First, it makes the method parametric. This means the threshold should be defined beforehand and the appropriate threshold is different for each graph. Second, by using threshold, the algorithm may fall into an infinite loop. After merging v and u, the average degree increases because the degree of v' is often bigger than the degree of v and u. Therefore, after plenty of merging, there may be no pair of nodes that satisfies the threshold condition.

To solve this problem, we update the threshold after each selection. The update is based on whether the current pair satisfies the threshold condition or not. Specifically, after picking a pair of nodes, if $N(u) \times N(v) > \Theta$ then theta increases by one otherwise it decreases by one. Based on this criteria, the candidate pairs are mainly from those with the smaller degree. Algorithm 1 shows

Algorithm 1: Node selection algorithm.

Input: v, u, Θ
$-v$, u: candidate pair for merging
$-\Theta$: threshold
Output: Acceptance/Rejection of the pair, updated Θ
if $(N(u) + N(uv)) \times (N(v) + N(uv)) < \Theta$ **then**
  $\Theta = \Theta - 1$;
  return true;
  `/* the pair is accepted */`
else
  $\Theta = \Theta + 1$;
  return false;
  `/* the pair is rejected */`
end

the proposed method for node selection. The algorithm starts with $\Theta = 1$.

The proposed node selection criteria improve the accuracy of *Shrink*. The improvement results from the probability distribution of occurring the neighbors of v and u in shortest paths. In our model, the uniform probability distribution function is used which is closer to reality when merging nodes with few neighbors.

5. Executing and refining stages

The output of the coarsening stage is the coarse graph in which distances between nodes are almost the same as those in the original graph. As each node in the coarse graph is a supernode containing some nodes, there is a mapping between the nodes of the coarse and original graphs. Therefore, instead of querying a pair of nodes in the original graph, we can query the equivalent pair in the coarse graph.

Distance-based algorithms run faster on the coarse graph because it is smaller. The smaller the coarse graph is, the faster a query executes. However, the accuracy drops by decreasing the size of the coarse graph. As one node is eliminated in each merging, the number of nodes in the coarse graph is equal to the number of nodes in the original graph minus the number of mergings. The reduction in the number of edges in each merging depends on the number of the common neighbors of the merged nodes. Specifically, the number of edges is reduced by $|N(uv)| + 1$ in each merging.

Integrating with any distance-based algorithms, *Shrink* can improve the performance. To investigate the effect of *Shrink* on the performance, the complexity of the distance-based algorithms should be taken into account that depends on the number of nodes and edges. For example, if the complexity of an algorithm is $O(|V|^2)$ and $CR = 50\%$, it runs four times faster on the coarse graph.

The output of the executing stage is a path in the coarse graph that consists of the source and destination nodes. However, the actual shortest path is still unknown because a supernode on the path is representative of several nodes. In the refining stage, the path is projected from the coarse graph to the original graph. As a result, a small subgraph of the original graph is obtained. Each node in this subgraph belongs to one of the supernodes of the extracted path. Rerunning the algorithm on the extracted subgraph specifies the laying nodes on the shortest path in the original graph. Furthermore, it provides a better estimation of the distance. It should be noted that rerunning the algorithm on the subgraph is much faster than running the algorithm on the original or coarse graph because the subgraph is very smaller than the whole graph.

The refining stage is an optional stage. In some applications, just a fast estimation of the length of the shortest path is needed. Hence, the refining stage is not necessary. Furthermore, sometimes there is not enough space to store the original graph. In these cases, we can discard the original graph and work on the coarse graph instead. For example, maintaining APSP results on disk for quick lookups requires $O(|V|^2)$ space which is unfeasible in many applications. In this case, *Shrink* reduces the storage size by CR^2 by processing the coarse graph.

6. Experiments

In this section, we evaluate *Shrink* in terms of time complexity and accuracy. Furthermore, we investigate the impact of the graph size, compression ratio, graph type, and node selection criteria. We compare *Shrink* performance with one of the state-of-the-art graph compression method introduced in [37]. We choose this method as a baseline because it is compatible with weighted graphs. Furthermore, it aims to preserve the query over all node pairs and hence it is comparable with our method.

6.1. Experiment setting

Most of the experiments run on New York City road network (NY road network), an undirected weighted graph with 264,346 nodes and 733,846 edges [1]. In this graph, a node represents an intersection or a road endpoint and the weight of an edge represents the length of the corresponding road segment. Further-

more, to investigate the performance of our approach on different datasets, we examine graphs from Stanford network data collections including friendship network, collaboration network, web graph and social network in Section 6.6 [2]. The experiment runs on PC with the configuration of Intel(R) Core i7, 3.4GHZ and 8G RAM. The PC runs Windows 7 and JDK 7 that runs on NetBeans 7.4. The algorithm has been implemented using the free Java graph library JGraphT, which includes mathematical graph-theory objects and algorithms [28].

The actual distances in the original graph are compared to the outputs of *Shrink*. The following parameters are measured in the evaluation procedure: 1) coarsening time (t_c): the required time to produce coarse graph 2) original graph query time (t_{q1}): the query time for 100 random pairs before compression 3) *Shrink* query time (t_{q2}): the query time for the same 100 pairs using *Shrink* (4) relative error: the average relative error over the 100 pairs.

6.2. Primary results

In this set of experiments, the effect of our compression method on NY road network is investigated while the compression ratio is 50%. Specifically, we apply Dijkstra algorithm on NY road network and its coarse graph and evaluate the impact on the distances. We run this experiment 5 times, and each time only a part of the graph is loaded. For covering graphs with different sizes, the number of loaded nodes ranges from 50,000 to 250,000 by a step of 50,000.

Fig. 8(a) shows the running time for the coarsening stage, t_c. It takes only 20 seconds to compress the whole graph to half. When applying node selection criteria, t_c increases to 31 seconds because some selected pairs are not merged and are discarded. The trend is linear because coarsening time has a linear relationship with the number of merged nodes. Fig. 8(b) shows the relative error with and without the refining stage and node selection criteria. Node selection criteria have a bigger effect on the relative error compared to applying refining stage. However, considering node selection criteria, we need more time to compress the graph. Fig. 8(c) shows the improvement in the query time caused by *Shrink* with and without the refining stage.

6.3. Impact of compression ratio

The compression ratio affects the speed, required storage, and accuracy of *Shrink*. When the compression ratio increases, the new graph has fewer nodes, and hence the shortest-path algorithms run faster but with lower accuracy. Therefore, the performance is adjustable which means for a given graph, the compression ratio should be set based on the desired speed and storage and acceptable error rate. Specifically, we can continue merging until the desired performance is achieved.

In this experiment, the original graph is NY road dataset (with 264,346 nodes) and the compression ratio changes from 10% to 100%. Results are reported in Fig. 9. The coarsening stage running time for different compression ratios is shown in Fig. 9(a). The trend is not quite linear because the average degree has an increase after compressing the graph. As discussed, the average degree in the graph affects coarsening time. Fig. 9(b) shows how the query time changes when compression ratio increases. Without using *Shrink*, the query time for the original graph lasts for 13 s. When the number of nodes in the coarse graph decreases, the queries run faster. Fig. 9(c) demonstrates the effect of the compression ratio on the average relative error for *Shrink*. It can be inferred that the more the nodes are merged, the less accurate the estimated lengths are. Fig. 9 validates the effectiveness of the proposed approach. As an illustration, if the size of the graph three to 20%, SSSP algorithms runs three times faster while the average error is only 3%. In this case, by using the refining stage, the nodes on the shortest path are also provided.

6.4. Impact of graph size

The evaluation in this subsection is different from that in other subsections. Here, we assess the performance of *Shrink* using All-Pair Shortest Path (APSP) method while in other experiments we use single-source shortest-path (SSSP) method and compare the distances between 100 pair of nodes in the original and coarse graphs. For APSP, the average error over all distances is calculated, but it cannot be applied to a graph with millions of nodes. The Floyd Warshall, a classic APSP algorithm, computes all shortest paths between each pair of nodes with $O(n^3)$ comparisons. Considering the heap size and computational cost, it is not possible to apply Floyd Warshall algorithm on the whole NY dataset. Thus, in this experiment, the number of nodes ranges from 1000 to 5000 by a step of 1000. Fig. 10(a) has two trends. The first one shows the Floyd Warshall running time for the original graph and the second one shows *Shrink* total running time including the coarsening, executing and refining stages. Fig. 10(b) shows the average relative error over all shortest paths. It can be seen that for improving the speed of the algorithm up to 6 times, we only lose less than 3% of the accuracy. For the larger graph, the estimated distance is more accurate because the paths are long and according to Section 3.2, the effect of our compression method on the long paths is little.

The required storage and heap size is another challenge while working with the large graph. As the coarse graph has less node and edges, it can be stored in less space. In our experiments, the maximum heap size of the virtual machine is 512MB which can handle only 12,500 nodes to run Floyd-Warshall algorithm. However, if the graph is reduced by half, it can handle up to 24,000 nodes.

6.5. Comparison with a state-of-the-art method

Here we compare *Shrink* with Compression of Weighted Graph (CWG) method introduce by Toivonen et al. [37]. CWG is based on merging nodes and edges to supernodes and superedges. Toivonen et al. define the difference between the edge weights and the superedges weights as the distance between the original graph and the coarse graph. They claim that the distance is minimized when the superedge weight is the mean of the original edge weights. Therefore, the main focus is on the common neighbors of u and v (i.e $N(vu)$). They also consider the number of the nodes that u and v include. Based on our notations, Eq. (35) gives the new weights for CWG.

$$w'_{vi} = \frac{|u|w_{ui} + |v|w_{vi}}{|u| + |v|}, \quad vi \in N(uv) \tag{35}$$

where $|v|$ is the number of original nodes in supernode v.

To make the comparison fair, we run *Shrink* without refining stage. We also do not consider node selection criteria and the same set of node pairs are merged in both algorithms. Fig. 11(a) shows that it takes approximately the same time to compress the graph with either of methods. The reason is that in each merging both algorithms focus on one pair of nodes and spend the same time to find new edge weights. However, it can be seen from Fig. 11(b) and 11(c) that our method outperforms CWG. While the average relative error is around 2% for our method with $CR = 50\%$, this value is around 18% for CWG. The reason is that *Shrink* takes more parameters into account compared to CWG. For assigning a new weight to the edge between two nodes, *Shrink* considers not only the edge weights between the nodes in the original graph but also the edge weights of their neighbors.

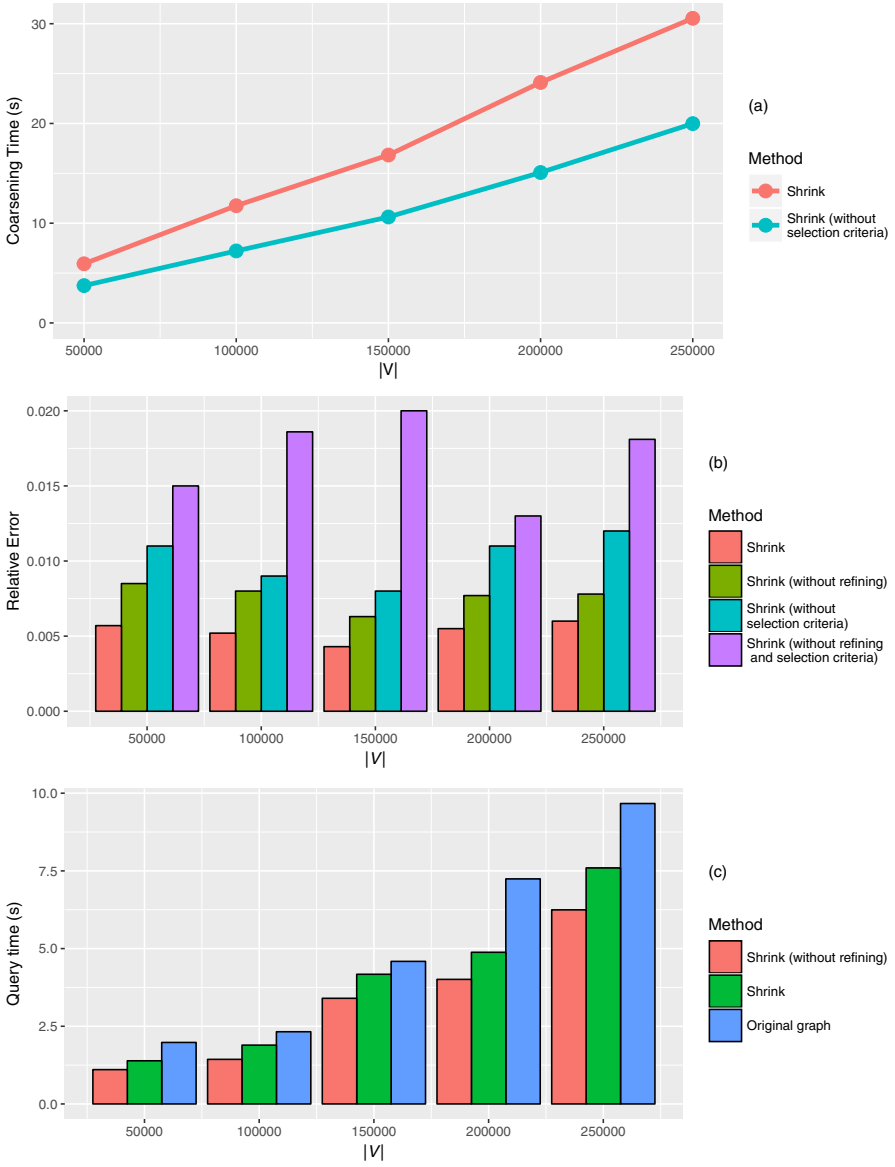

Fig. 8. Results for the subgraphs of NY road network with $CR = 50\%$ (a) coarsening time, t_c (b) average relative error for different setups (c) *Shrink* query time with and without refining stage, t_{q2}, and the query time on the original graph t_{q1}.

6.6. Impact of graph type and graph density

In addition to road networks, we apply our method to different types of graphs including friendship network, collaboration network, web graph and social network. These datasets can be downloaded from Stanford network data collections [2]. Since the graphs are unweighted, all of the edges are assigned the same weight and the direction of the edges are not considered. We also apply *Shrink* to larger road network including Florida and Great Lakes road network with 1 and 2.7 million nodes, respectively. In this experiment, the reduction ratio is set to 20%.

Table 3 shows the results. As expected, when the input graph is dense, the accuracy of our method drops. For example, Brightkite friendship network is a very dense graph because the diameter (longest shortest path) is just 14. As in our model, we assume that the input graph is large and has long paths, *Shrink* provides less accurate compression compared to NY road network, which is a quasi-planner sparse graph. Furthermore, in Brightkite network, more than 40% of the nodes have only one neighbour that drops the accuracy. The reason is that our model tries to minimize the effect on the paths that pass through the supernodes while no path passes through a one-degree node (unless the node is the start or end node). *Shrink*s compresses larger graphs better because they usually have longer shortest paths. According to Table 3, *Shrink* is practical for large graphs as it can compresses a graph with 2.7 million nodes into fifth in 10 minutes.

7. Related work

Graph compression methods can be categorized into two groups: general compression and query-friendly compression. General compression methods preserve the information of the entire graph and answer all types of queries. However, these methods highly depend on the graph type, coding mechanism, and extrinsic information. Furthermore, these methods need decompression before querying the graph [15]. Specifically, the graph should be restored first to answer even simple queries. Some works has been studied graph compression for Web graph and social networks [9,15,31]. Similar to our work, query-friendly compression approaches target specific classes of queries, such as neighborhood

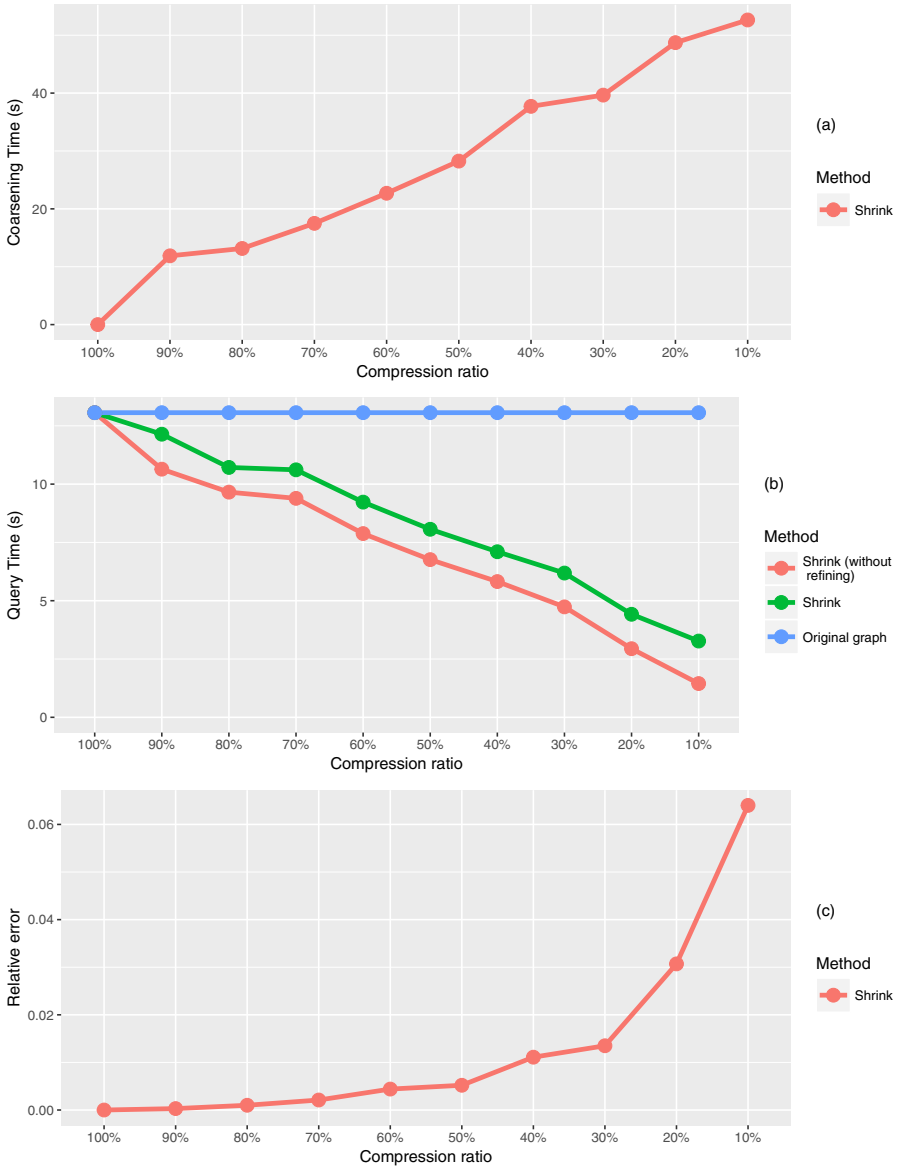

Fig. 9. Results for the subgraphs of NY road network. (a) coarsening time, t_c (b) Query times t_{q2} and t_{q1} (c) Average relative error.

Table 3
Experiment results for different datasets with $CR = 20\%$.

Input graph				Results			
Name (type)[a]	$\|V\|$	$\|E\|$	Diameter	t_c (s)	t_{q1} (s)	t_{q2} (s)	RE (%)
DBLP (collaboration)	317,080	1,049,866	21	62	31	10	19
Brightkite (friendship)	58,228	214,078	14	12	7	3	9
Epinions (social)	75,879	508,837	14	27	13	9	9
New York (road)	264,346	365,050	1589	49	13	6	3
Notre Dame (web)	325,729	1,497,134	46	97	20	8	2
Florida (road)	1,070,376	1,343,951	2957	181	52	17	0.5
Great Lakes (road)	2,758,119	3,397,404	4145	629	151	54	0.4

[a] t_c: Coarsening time, t_{q1}: Query time for the original graph, t_{q2}: *Shrink* query time, RE: Relative Error

[10,26,29], reachability [16,17,27,38], path [11], pattern queries [16], connectivity [41], and distance-based queries [32,41]. For example, Aho et al. reduced graphs by substituting a simple cycle for each strongly connected component to speed up reachability queries [5]. Brisaboa et al. focus on compression of web graphs to reduce the space requirements while running queries such as successors and predecessors extraction is possible on the compressed graph [10]. Some researches address the similar problem as a graph simplification problem [32,41] and graph summarization [12,24,34].

Shrink differs from the current compression methods in the following: (1) *Shrink* is developed for reachability and distance-based queries; (2) most of the methods are designed for unweighted graphs while *Shrink* can be applied to both unweighted and weighted graphs [9,16,26]; (3) *Shrink* provides compressed data structure, the coarse graph, that can be directly queried with-

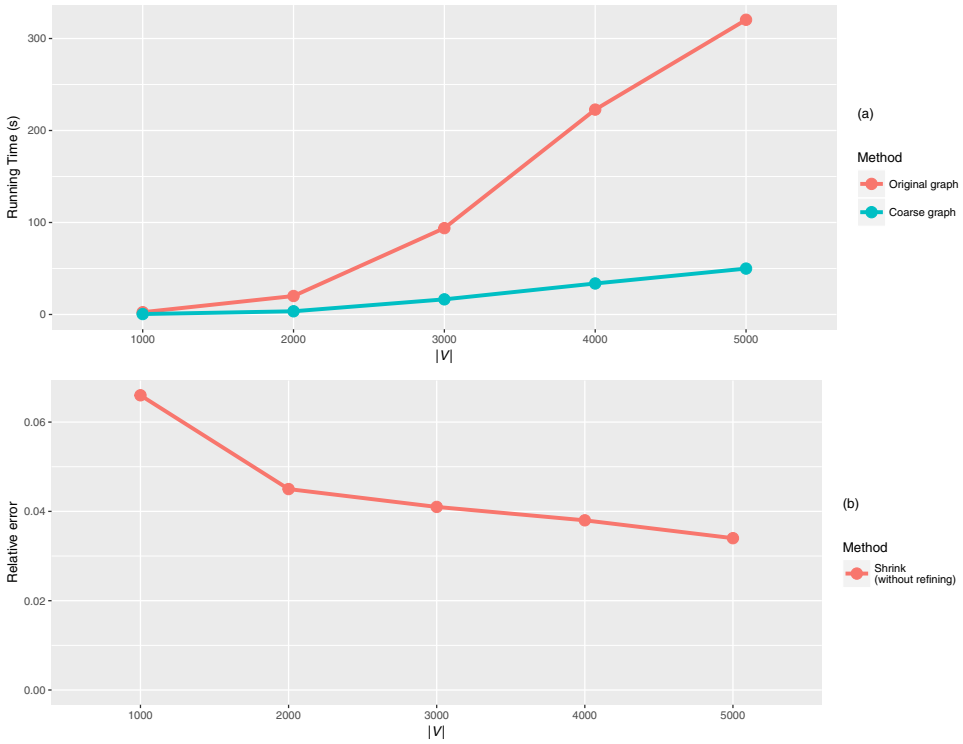

Fig. 10. Computing APSP for the subgraphs of NY road network with $CR = 50\%$. (a) required time for the original and coarse graphs (b) the average relative error over all pairs .

Table 4
Feature comparison between *RPC*[16], *DPS*[32], *CWG*[37] and *Shrink*.

Properties	RPC	DPS	CWG	*Shrink*
Target Graphs	Directed Unweighted	Undirected Unweighted	Undirected Un/Weighted	Undirected Un/Weighted
Query Type	Reachability	Distance-based	Distance-based	Distance-based
Adjustable CR	×	✓	✓	✓
Flexible Running Time	×	×	✓	✓
Provide the Path	×	×	×	✓
Compression works for	all pairs	non-local pairs	all pairs	all pairs
Required Parameters	Nonparametric	ϵ	λ	Nonparametric
Largest examined graph for distance-based queries	–	62K (nodes)	200K (nodes)	2.7M (nodes)

* σ: average degree in the graph.

out decompression [15,29]; (4) Some of the compression methods only reduce the number of the edges while *Shrink* reduce the number of nodes and edges [17,27,38]; (5) *Shrink* can be performed incrementally for temporal graphs while most of the above methods are batch algorithms, requiring to decompress the whole graph to do the change such as edge insertion or node removal [32,38]. (6) *Shrink* not only specifies the distance between the two nodes but also provides the actual path containing laying nodes [20].

Fan et al. extract a coarse graph from a directed unweighted graph to speed up reachability queries [16]. *Shrink* has several advantages compared to Reachability Preserving Compression (RPC). First, RPC only preserves reachability while *Shrink* is advantageous for all types of distance-based queries. Second, *Shrink* compresses the graph much faster. The complexity of compression is linear in the number of nodes for *Shrink* while it is quadratic for RPC (i.e. $O(\sigma|V|^2)$ where σ denotes the average degree in the graph). Third, unlike RPC, *Shrink* grantees the reachability with any compression ratio. RPC has a limit for compression ratio to preserve the reachability ranging from 0.02% to 14.70%. Fourth, the error and compression ratio is flexible in *Shrink*. Therefore, it is possible to terminate the coarsening process at any time while having a semi-compressed graph.

Ruan et al. propose a Distance Preserving Graph Simplification (DPS) for unweighted graphs [32]. In the first stage, some nodes are selected from the original graph and then the selected nodes are connected to each other with weighted edges. This method is designed to preserve the distance between every non-local pair ($\forall x, y\ d(x, y) \leq \epsilon$). The output is a weighted coarse graph that cannot be simplified again. DPS is costly and the largest examined graph contains less than 63,000 nodes. The compression ration is also fixed. Another distance preserving compression method is Compression of Weighted Graphs (CWG) which is compared with *Shrink* [37]. Table 4 summarizes the features of each method.

8. Conclusion and future work

In this paper, we introduce *Shrink* as a new distance preserving graph compression method. In the first stage, the graph size is reduced by replacing node pairs with supernodes. The queries run more efficiently on the reduced graph, called coarse graph. To find the exact shortest path including the laying nodes on the path, we rerun the query on a subgraph of the original graph. *Shrink* is a generic compression method that can be applied to different types of the graph. However, the performance depends on the graph type, graph size, and compression ratio. In the experiment

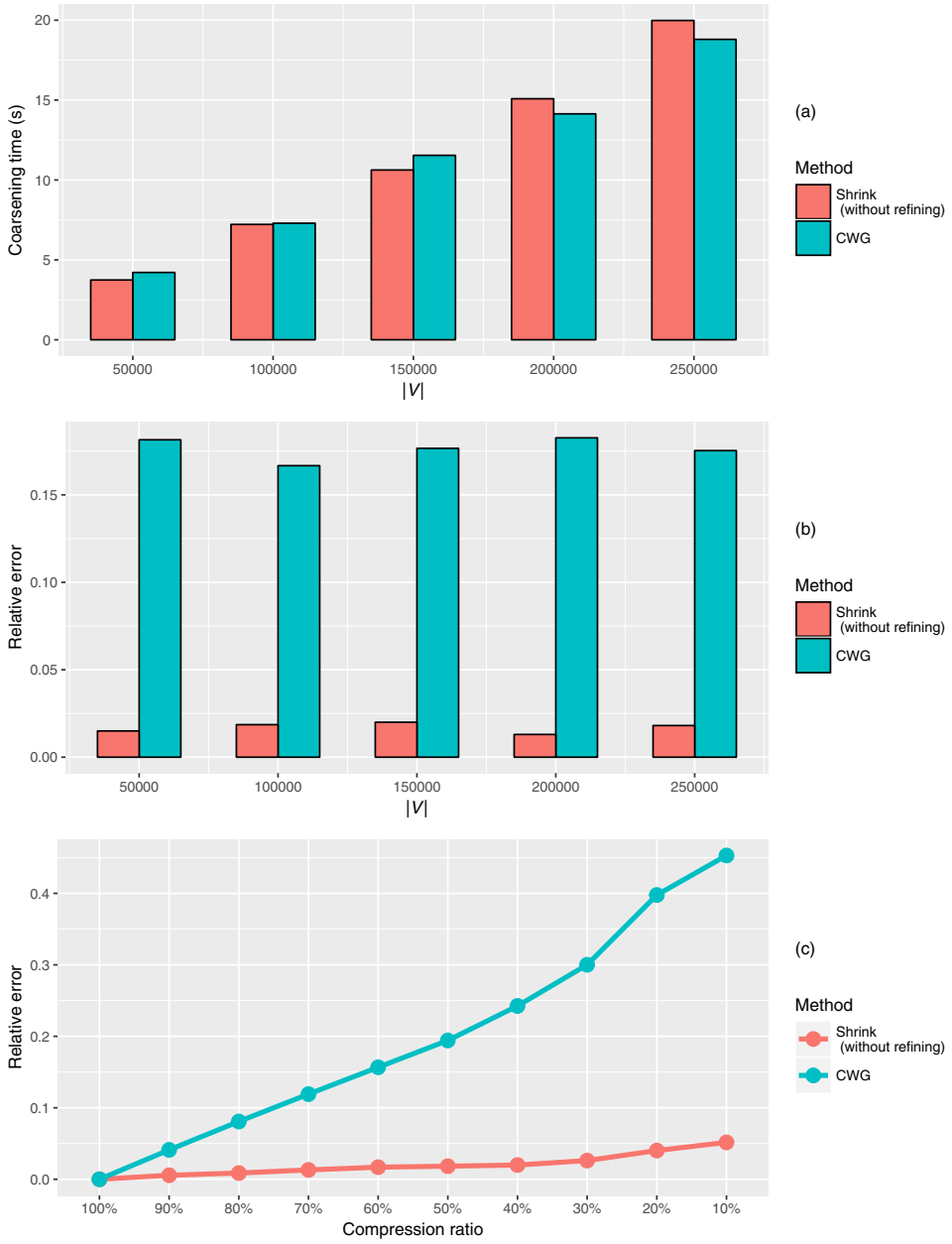

Fig. 11. Comparison between Shrink and CWG (a) t_c for NY road network subgraphs with different size ($CR = 50\%$) (b) average relative error for NY road network subgraphs with different size ($CR = 50\%$) (c) average relative error.

section, we investigate the impact of each parameter on the performance. *Shrink* has several valuable features discussed in the paper. These features make *Shrink* practical for real-world applications.

Due to the novelty of the proposed method, many new ideas arise. A generalized problem could be extending the proposed idea for merging three nodes or even a subgraph in the same way. In this case, a subgraph is merged into a supernode and the problem would be finding weights for the new edges connected to the supernode. *Shrink* can also be applied in a distributed manner because it works on the graph locally. Future work involves new techniques to run efficiently *Shrink* in a distributed system.

Acknowledgement

This project is funded by RMIT Sustainable Urban Precinct Project (SUPP) “Online Infrastructure for iCO2mmunity”.

References

[1] http://www.dis.uniroma1.it/challenge9/download.shtml.
[2] https://snap.stanford.edu/data/.
[3] I. Abraham, D. Delling, A.V. Goldberg, R.F. Werneck, A Hub-Based Labeling Algorithm for Shortest Paths in Road Networks, Springer, pp. 230–241.
[4] C.C. Aggarwal, H. Wang, A Survey of Clustering Algorithms for Graph Data, in: Managing and mining graph data, Springer, 2010, pp. 275–301.
[5] A.V. Aho, M.R. Garey, J.D. Ullman, The transitive reduction of a directed graph, SIAM J. Comput. 1 (2) (1972) 131–137.
[6] T. Akiba, T. Hayashi, N. Nori, Y. Iwata, Y. Yoshida, Efficient top-k shortest-path distance queries on large networks by pruned landmark labeling., in: AAAI, 2015, pp. 2–8.
[7] T. Akiba, Y. Iwata, Y. Yoshida, Fast exact shortest-path distance queries on large networks by pruned landmark labeling, in: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, ACM, 2013, pp. 349–360.
[8] I. Althöfer, G. Das, D. Dobkin, D. Joseph, J. Soares, On sparse spanners of weighted graphs, Discrete Comput. Geom. 9 (1) (1993) 81–100.
[9] P. Boldi, M. Rosa, M. Santini, S. Vigna, Layered label propagation: a multiresolution coordinate-free ordering for compressing social networks, in: Proceed-

ings of the 20th international conference on World Wide Web, ACM, 2011, pp. 587–596.

[10] N.R. Brisaboa, S. Ladra, G. Navarro, Compact representation of web graphs with extended functionality, Inf. Syst. 39 (2014) 152–174, doi:10.1016/j.is.2013.08.003.

[11] P. Buneman, M. Grohe, C. Koch, Path queries on compressed xml, in: Proceedings of the 29th International Conference on Very Large Data bases-Volume 29, VLDB Endowment, 2003, pp. 141–152.

[12] Š. Čebirić, F. Goasdoué, I. Manolescu, Query-oriented summarization of rdf graphs, Proc. VLDB Endow. 8 (12) (2015) 2012–2015.

[13] J. Chan, W. Liu, A. Kan, C. Leckie, J. Bailey, K. Ramamohanarao, Discovering latent blockmodels in sparse and noisy graphs using non-negative matrix factorisation, in: Proceedings of the 22nd ACM International Conference on Information & Knowledge Management, ACM, 2013, pp. 811–816.

[14] C. Chen, X. Yan, F. Zhu, J. Han, S.Y. Philip, Graph olap: towards online analytical processing on graphs, in: 2008 Eighth IEEE International Conference on Data Mining, IEEE, 2008, pp. 103–112.

[15] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, P. Raghavan, On compressing social networks, in: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2009, pp. 219–228.

[16] W. Fan, J. Li, X. Wang, Y. Wu, Query preserving graph compression, in: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, in: SIGMOD '12, ACM, New York, NY, USA, 2012, pp. 157–168.

[17] T. Feder, R. Motwani, Clique partitions, graph compression and speeding-up algorithms, J. Comput. Syst. Sci. 51 (2) (1995) 261–272.

[18] P.-O. Fjällström, Algorithms for Graph Partitioning: A Survey, 3, Linköping University Electronic Press Linköping, 1998.

[19] J. Gao, R. Jin, J. Zhou, J.X. Yu, X. Jiang, T. Wang, Relational approach for shortest path discovery over large graphs, Proc. VLDB Endow. 5 (4) (2011) 358–369.

[20] A. Gubichev, S. Bedathur, S. Seufert, G. Weikum, Fast and accurate estimation of shortest paths in large graphs, in: Proceedings of the 19th ACM International Conference on Information and Knowledge Management, CIKM '10, ACM, 2010, pp. 499–508.

[21] D. Hennessey, D. Brooks, A. Fridman, D. Breen, A simplification algorithm for visualizing the structure of complex graphs, in: 12th International Conference Information Visualisation, 2008. IV'08., IEEE, 2008, pp. 616–625.

[22] J. Hong, K. Park, Y. Han, M.K. Rasel, D. Vonvou, Y.-K. Lee, Disk-based shortest path discovery using distance index over large dynamic graphs, Inf. Sci. (Ny) 382–383 (2017) 201–215, doi:10.1016/j.ins.2016.12.013.

[23] E.T. Jaynes, Probability Theory: The Logic of Science, Cambridge University Press, 2003.

[24] K. LeFevre, E. Terzi, Grass: graph structure summarization, in: SDM, SIAM, 2010, pp. 454–465.

[25] M. Lohrey, S. Maneth, R. Mennicke, Xml tree structure compression using repair, Inf. Syst. 38 (8) (2013) 1150–1167, doi:10.1016/j.is.2013.06.006.

[26] H. Maserrat, J. Pei, Neighbor query friendly compression of social networks, in: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2010, pp. 533–542.

[27] D.M. Moyles, G.L. Thompson, An algorithm for finding a minimum equivalent graph of a digraph, J. ACM (JACM) 16 (3) (1969) 455–460.

[28] B. Naveh, Jgrapht, 2008, http://jgrapht.sourceforge.net.

[29] S. Navlakha, R. Rastogi, N. Shrivastava, Graph summarization with bounded error, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, ACM, 2008, pp. 419–432.

[30] D. Rafiei, Effectively visualizing large networks through sampling, in: Visualization, 2005. VIS 05. IEEE, IEEE, 2005, pp. 375–382.

[31] S. Raghavan, H. Garcia-Molina, Representing web graphs, in: Proceedings 19th International Conference on Data Engineering, 2003., IEEE, 2003, pp. 405–416.

[32] N. Ruan, R. Jin, Y. Huang, Distance preserving graph simplification, in: 2011 IEEE 11th International Conference on Data Mining (ICDM), IEEE, 2011, pp. 1200–1205.

[33] R. Schenkel, A. Theobald, G. Weikum, HOPI: An Efficient Connection Index for Complex XML Document Collections, Springer, pp. 237–255.

[34] B.-S. Seah, S.S. Bhowmick, C.F. Dewey, H. Yu, Fuse: a profit maximization approach for functional summarization of biological networks, BMC Bioinform. 13 (3) (2012) 1.

[35] C. Sommer, Shortest-path queries in static networks, ACM Comput. Surv. (CSUR) 46 (4) (2014) 45.

[36] Y. Tian, R.A. Hankins, J.M. Patel, Efficient aggregation for graph summarization, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, ACM, pp. 567–580.

[37] H. Toivonen, F. Zhou, A. Hartikainen, A. Hinkka, Compression of weighted graphs, in: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2011, pp. 965–973.

[38] S.J. van Schaik, O. de Moor, A memory efficient reachability data structure through bit vector compression, in: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, ACM, 2011, pp. 913–924.

[39] D. Wackerly, W. Mendenhall, R. Scheaffer, Mathematical Statistics with Applications, Cengage Learning, 2007.

[40] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, Y. Xu, Path problems in temporal graphs, Proc. VLDB Endow. 7 (9) (2014) 721–732.

[41] F. Zhou, S. Malher, H. Toivonen, Network simplification with minimal loss of connectivity, in: 2010 IEEE 10th International Conference on Data Mining (ICDM), IEEE, 2010, pp. 659–668.

[42] A.D. Zhu, X. Xiao, S. Wang, W. Lin, Efficient single-source shortest path and distance queries on large graphs, in: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2013, pp. 998–1006.

[43] C.J. Zhu, K.-Y. Lam, S. Han, Approximate path searching for supporting shortest path queries on road networks, Inf. Sci. (Ny) 325 (2015) 409–428, doi:10.1016/j.ins.2015.06.045.