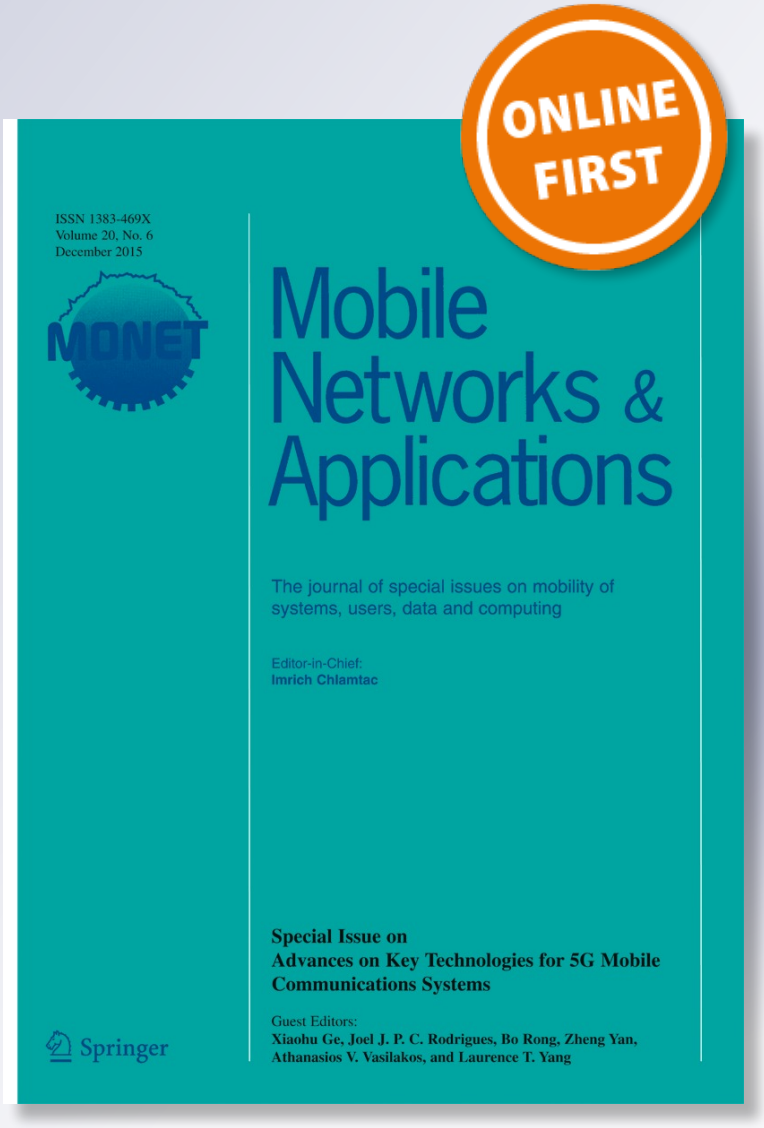
Efficient Estimation of Betweenness Centrality in Wireless Networks

Chan-Myung Kim, Yong-hwan Kim, Youn-Hee Han & Jeong-Hyon Hwang

Mobile Networks and Applications
The Journal of SPECIAL ISSUES on Mobility of Systems, Users, Data and Computing

ISSN 1383-469X

Mobile Netw Appl
DOI 10.1007/s11036-015-0660-x



Springer

Your article is protected by copyright and all rights are held exclusively by Springer Science +Business Media New York. This e-offprint is for personal use only and shall not be self-archived in electronic repositories. If you wish to self-archive your article, please use the accepted manuscript version for posting on your own website. You may further deposit the accepted manuscript version in any repository, provided it is only made publicly available 12 months after official publication or later and provided acknowledgement is given to the original source of publication and a link is inserted to the published article on Springer's website. The link must be accompanied by the following text: "The final publication is available at link.springer.com".

Author's personal copy

Mobile Netw Appl
DOI 10.1007/s11036-015-0660-x

Efficient Estimation of Betweenness Centrality in Wireless Networks

Chan-Myung Kim[1] · Yong-hwan Kim[1] · Youn-Hee Han[1] · Jeong-Hyon Hwang[2]

© Springer Science+Business Media New York 2015

Abstract In wireless networks, the betweenness of a node has been considered an indication of that node's importance in efficiently and reliably delivering messages. In a large wireless network, however, the cost of computing the betweenness of every node is impractically high. In this paper, we introduce a new representation of a node's vicinity, called the *expanded ego network* (shortly, *x-ego network*) of that node. We also propose an approach that calculates the *x-ego betweenness* of a node (i.e., the betweenness of that node in its x-ego network) and use it as an estimate of the true betweenness in the entire network. Furthermore, we develop an algorithm that quickly computes x-ego betweenness by exploiting structural properties of x-ego networks. Our evaluation results show the benefits and effectiveness of the above approach using trace data obtained from real-world wireless networks.

Keywords Ego networks · Betweenness centrality · Wireless networks

✉ Youn-Hee Han
yhhan@koreatech.ac.kr

Chan-Myung Kim
cmdr@koreatech.ac.kr

Yong-hwan Kim
cherish@koreatech.ac.kr

Jeong-Hyon Hwang
jhh@cs.albany.edu

[1] School of Computer Science and Engineering at Korea University of Technology and Education, Cheonan, Korea

[2] Department of Computer Science, University at Albany – State University of New York, Albany, NY, USA

1 Introduction

Recently, researches strove to enhance the performance of wireless networks based on social network analysis since social relationships between wireless nodes tend to influence their mobility patterns and message delivery [1–4]. Among various metrics used in social network analysis, the betweenness of a node indicates the extent to which that node is between all other nodes within the network.

In a wireless network, a node with high betweenness has the capacity to facilitate both direct and indirect communications between nodes [5, 6]. Therefore, researchers have been taking advantage of betweenness for a variety of purposes. For example, Daly et al. [7] and Hui et al. [8] developed techniques that efficiently and reliably forward messages via nodes with high betweenness. Dimokas et al. proposed an approach that caches popular data items at nodes with high betweenness in order to reduce communication overhead [9]. Cuzzocrea et al. provided a protocol that exploits betweenness to construct an energy-efficient network topology [10]. Gupta et al. developed a method that groups nodes into local clusters and selects, for each cluster, a node with high betweenness as the head node, which communicates with a base station on behalf of its cluster [11]. Katsaros et al. proposed an approach that efficiently manages and upgrades a wireless network via a node with high betweenness [4].

Calculating the betweenness of each node, however, requires finding all of the shortest paths between every pair of nodes in the given network. Since carrying out this task in a large wireless network will incur prohibitively expensive network and computational costs, techniques for estimating betweenness have been developed [7, 12–15]. In these techniques, each node identifies its *ego network*, a logical network consisting of that node, its 1-hop neighbors, and

Published online: 10 December 2015

Author's personal copy

all links between these nodes. Then, each node calculates, as an estimate of its betweenness in the entire network, its betweenness only in its ego network, thereby saving both network and computational resources.

In a wireless network, each node can obtain its ego network if every node exchanges its neighbor information with each other so that each node becomes aware of the connections between its neighbors. While the above message exchange allows each node to obtain information about its 2-hop neighbors (i.e., neighbors of its neighbors), the ego network of a node cannot capture that information since the coverage of the ego network is limited up to the 1-hop neighbors of the node. In this paper, we introduce a new type of logical network, called *expanded ego network* (shortly, *x-ego network*), which covers a larger number of nodes and links than the corresponding ego network for the same network cost. For this reason, the betweenness from an x-ego network (*x-ego betweenness*) is in general more similar to the true betweenness than the betweenness from an ego network (*ego betweenness*). We also present four key properties of x-ego networks and an algorithm that quickly computes x-ego betweenness by taking advantage of these properties. Furthermore, through evaluations based on wireless trace data [16], we show that x-ego betweenness more accurately estimates betweenness than ego betweenness and our algorithm more quickly computes x-ego betweenness than the state-of-the-art betweenness computation algorithm.

In this paper, we make the following contributions:

- We introduce x-ego networks, which contain more information than ego networks for the same network cost and therefore lead to more accurate estimation of betweenness.
- We describe properties of x-ego networks, which enable fast x-ego betweenness computation.
- We provide an algorithm that computes x-ego betweenness outperforming existing techniques.
- We present evaluation results that show the benefits of x-ego betweenness and our algorithm for computing x-ego betweenness.

The remainder of this paper is organized as follows. Section 2 presents the definition and properties of the proposed x-ego network. Section 3 describes our algorithm for quickly computing x-ego betweenness. Section 4 evaluates the effectiveness of x-ego betweenness and the efficiency of our x-ego betweenness algorithm. Finally, Section 5 concludes this paper.

2 Ego networks and betweenness

In this section, we define terms that represent two types of ego networks of a node in a wireless network (Section 2.1) as well as the betweenness of a node in its ego and x-ego networks (Section 2.2). We then present several properties of the x-ego networks (Section 2.3). Our betweenness computation algorithm in Section 3 takes advantage of these properties.

2.1 Definitions

In this paper, we model a wireless network using a graph $G(V, E)$, where V is a set of vertices representing the nodes in the network and E is a set of edges representing social links between nodes (see Fig. 1).[1] Given a pair of nodes that have communicated with each other, a social link between these nodes can be assumed if their communications meet certain criteria (e.g., the accumulated duration of communications [8], the frequency of communications [8, 17], or the elapsed time after the last communication is beyond/under a threshold [18]). In this paper, we assume undirected social links among nodes (see Section 4.1). Furthermore, we define the length of a path between two nodes as the number of social links on that path. In other words, we model a wireless network using an *undirected* and *unweighted* graph.

In the literature [7, 12–14], given a graph $G(V, E)$ and a vertex $v \in V$, the *ego network* of v is defined as the subgraph of G consisting of v and its 1-hop neighbors (i.e., vertices with an edge to v) as well as the edges between these vertices. Using the notation summarized in Table 1, this ego network can be formally defined as follows:

Definition 1 Given a graph $G(V, E)$ and a vertex $v \in V$, the *ego network* of v is defined as $\mathcal{E}_v(V_{\leq 1}(v), E_{\leq 1}(v))$ where $V_{\leq 1}(v)$ is the set of vertices whose shortest distance from v is no longer than 1 (i.e., $\{v\} \cup V_1(v)$) and $E_{\leq 1}(v)$ denotes the set of edges between the vertices in $V_{\leq 1}(v)$.

In Fig. 1, $V_{\leq 1}(v) = V_0(v) \cup V_1(v) = \{v, a, b, c, d\}$ and $E_{\leq 1}(v) = \{\{v, a\}, \{v, b\}, \{v, c\}, \{v, d\}, \{a, b\}, \{c, d\}\}$. The ego network of vertex v, $\mathcal{E}_v(V_{\leq 1}(v), E_{\leq 1}(v))$, is shown in Fig. 2a.

In a wireless network, the ego network of each node (i.e., the ego network of the vertex representing that node) can be obtained if all of the nodes periodically broadcast information about their neighbors. For example, in the wireless network illustrated by Fig. 1, assume that nodes a, b, c, and d broadcast this information and node v has been receiving such information. Then, node v will be able to derive its ego network as in Fig. 2a.

[1] We use the term *node* and *social link* to refer to the devices and their relationships in a wireless network. On the other hand, the graph representing a wireless network consists of *vertices* and *edges* representing nodes and social links, respectively.

Author's personal copy

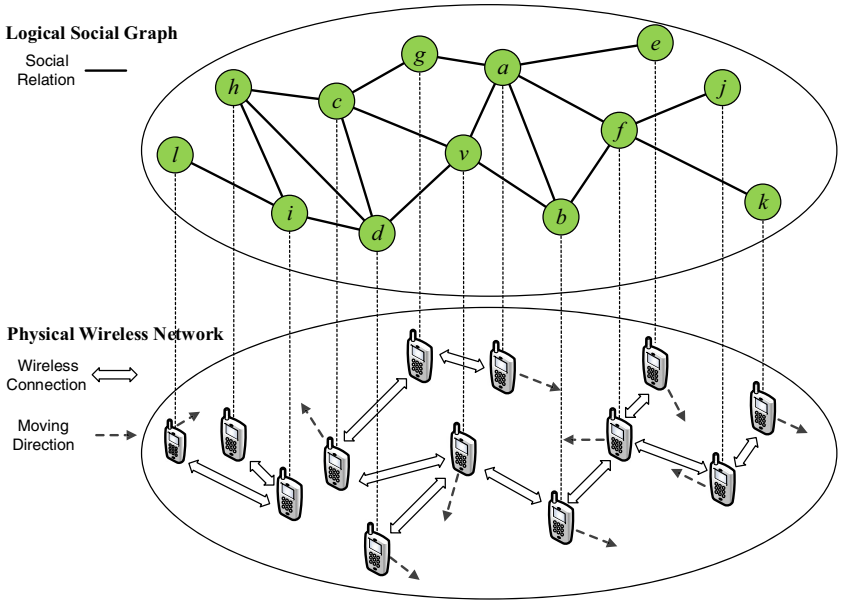


Fig. 1 A wireless network and its social graph

In the context of wireless networks, ego networks well model the relationships/interactions between a node and others. However, ego networks have the limitation that it does not capture a substantial amount of information obtained at the expense of network resources. For example, in Fig. 1, assume that node v received from node a the information about a's 1-hop neighbors (i.e., b, e, f, g, and v). Despite this information, the ego network of v cannot record the social links between a and e, between a and f, and between a and g since it can represent only the social links between v and the 1-hop neighbors of v.

To overcome the above limitation, we introduce the following extension to ego networks:

Definition 2 Given a graph $G(V, E)$ and a vertex $v \in V$, the *extended ego network* (briefly, *x-ego network*) of v is $\mathcal{X}_v(V_{\leq 2}(v), E_{\leq 2}(v) - E_2(v))$, where $V_{\leq 2}(v)$ is the set of vertices that are at most 2 hops away from v, $E_{\leq 2}(v)$ is the set of edges between vertices that are at most 2 hops away from v, and $E_2(v)$ is the set of edges between 2-hop neighbors of v.

Figure 2b shows the x-ego network of v from the graph in Fig. 1. As Fig. 2a and b illustrate, the x-ego network of v is different from the ego network of v in that it contains 2-hop neighbors of v (i.e., $V_{\leq 2}(v) - V_{\leq 1}(v) = V_2(v)$) as well as the edges between a 1-hop neighbor and a 2-hop neighbor of v. Despite this difference, both the ego and x-ego networks of v can be obtained with the same network overhead (since they consume the same messages from 1-hop neighbors of v). The benefits of x-ego networks over ego networks are further verified in Section 4.

2.2 Ego and x-ego betweenness

In the literature [5, 13, 19], given a graph $G(V, E)$, the betweenness $B(v)$ of a vertex v is defined as:

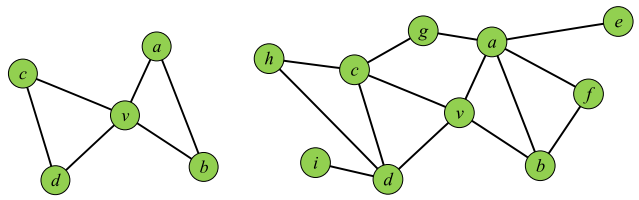
$$B(v) = \frac{\sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}}{(|V| - 1)(|V| - 2)} \quad (1)$$

where σ_{st} is the number of shortest paths from vertex s to vertex t and $\sigma_{st}(v)$ is the number of those shortest paths that pass through vertex v. In the above definition, the denominator represents the total number of pairs of vertices except v. It normalizes $B(v)$ to a value between 0 and 1. Given an undirected graph, $\sigma_{st} = \sigma_{ts}$ and $\sigma_{st}(v) = \sigma_{ts}(v)$ for all vertices s, t, and v. Therefore, it is sufficient to find either σ_{st} or σ_{ts} and either $\sigma_{st}(v)$ or $\sigma_{ts}(v)$.

Table 1 Summary of notation

Symbol	Description
$V_i(v)$	set of i-hop neighbors of vertex v ($V_0(v) = \{v\}$)
$V_{\leq i}(v)$	set of vertices that are at most i hops away from vertex v (i.e., $V_{\leq i}(v) = \cup_{k=0}^{i} V_k(v)$)
$E_i(v)$	set of edges connecting a vertex in $V_i(v)$ and another vertex in $V_i(v)$
$E_{\leq i}(v)$	set of edges connecting a vertex in $V_{\leq i}(v)$ and another vertex in $V_{\leq i}(v)$

Author's personal copy



(a) Ego network, $\mathcal{E}_v$ (b) X-ego network, $\mathcal{X}_v$

Fig. 2 Ego and x-ego networks of vertex v in Fig. 1

In a large wireless network, obtaining the betweenness of each node (i.e., the betweenness of the vertex representing that node) is costly since all nodes that have limited memory and energy must exchange and consume a substantial number of messages to identify the shortest paths between them. On the other hand, each node can obtain its ego and x-ego networks with much lower overhead since each node needs to broadcast information about its 1-hop neighbors (Section 2.1). In this paper, we consider the situations where each node computes its betweenness using either its ego network or x-ego network and then uses the result as an estimate of its true betweenness in the entire network. We refer to the betweenness of v computed from the ego network and x-ego network of v as the *ego betweenness* and *x-ego betweenness* of v (denoted $B^{\mathcal{E}}(v)$ and $B^{\mathcal{X}}(v)$), respectively. Table 2 shows, for every vertex v in Fig. 1, the betweenness ($B(v)$), ego betweenness ($B^{\mathcal{E}}(v)$), and x-ego betweenness ($B^{\mathcal{X}}(v)$). In this table, for most of the vertices, x-ego betweenness is closer to betweenness than ego-betweenness mainly because it is derived from a larger number of vertices and edges. For this reason, the correlation coefficient (also known as the Pearson correlation coefficient) of x-ego betweenness and betweenness (0.9) is higher than that of ego-betweenness and betweenness (0.63). The advantage of x-ego betweenness over ego betweenness can also be observed in terms of Spearman's rank correlation, which indicates, given two series $\mathcal{X} = (X_1, X_2, \cdots, X_N)$ and $\mathcal{Y} = (Y_1, Y_2, \cdots, Y_N)$, the correlation between $(r_{\mathcal{X}}(X_1), r_{\mathcal{X}}(X_2), \cdots, r_{\mathcal{X}}(X_N))$ and $(r_{\mathcal{Y}}(Y_1), r_{\mathcal{Y}}(Y_2), \cdots, r_{\mathcal{Y}}(Y_N))$, where $r_{\mathcal{X}}(X_i)$ and $r_{\mathcal{Y}}(Y_i)$ represent the rank of X_i in $\mathcal{X}$ and that of Y_i in $\mathcal{Y}$, respectively. In Table 2, Spearman's correlation is 0.93 between x-ego betweenness and betweenness and 0.79 between ego betweenness and betweenness. In Section 4, we further demonstrate the benefit of x-ego betweenness using wireless trace data.

2.3 Properties of x-ego networks

In this section, we present four properties of x-ego networks. These properties enable efficient x-ego betweenness computation (Section 3). As in Brandes' work [19], we denote the *dependency* of vertices s and t on vertex v as $\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}$. Then, the betweenness of v (Eq. 1) can be computed by adding the dependency values for all pairs of vertices excluding v. To quickly calculate such dependency values in x-ego networks, we have identified the properties explained below.

Theorem 1 *Assume a vertex v, its x-ego network $\mathcal{X}_v$, and its two different 1-hop neighbors s and t (i.e., $s, t \in V_1(v)$ and $s \neq t$). Then, $\delta_{st}(v) = 0$ if there is an edge between s and t (i.e., $\{s, t\} \in E_1(v)$).*

Proof Since $\{s, t\} \in E_1(v)$, $d(s, t) = 1$. Furthermore, $s, t \in V_1(v)$, meaning that $d(s, v) + d(v, t) = 1 + 1 = 2 > d(s, t) = 1$ (i.e., no shortest path from s to t passes through v). Therefore, $\delta_{st}(v) = \sigma_{st}(v)/\sigma_{st} = 0/\sigma_{st} = 0$. □

Example 1 In Fig. 2b, $\delta_{ab}(v) = 0$ since $a, b \in V_1(v)$ and there is an edge between a and b.

Theorem 1 allows us to quickly compute dependencies particularly when x-ego networks exhibit a strong community structure (i.e., 1-hop neighbors of a node tend to have direct social links between them). In Section 4, we show this benefit using wireless trace data.

When Theorem 1 cannot be applied to v's 1-hop neighbors s and t (i.e., there is no edge between s and t), the

Table 2 Comparison of betweenness, ego betweenness, and x-ego betweenness for the graph shown in Fig. 1 (the Pearson correlation is 0.63 between $B(v)$ and $B^{\mathcal{E}}(v)$ and 0.90 between $B(v)$ and $B^{\mathcal{X}}(v)$, and the Spearman correlation is 0.79 between $B(v)$ and $B^{\mathcal{E}}(v)$ and 0.93 between $B(v)$ and $B^{\mathcal{X}}(v)$)

Nodes		v	a	b	c	d	e	f	g	h	i	j	k	l
$B(v)$	Value	0.405	0.383	0.124	0.131	0.286	0.000	0.318	0.049	0.030	0.167	0.000	0.000	0.000
	Rank	1	2	7	6	4	10	3	8	9	5	10	10	10
$B^{\mathcal{E}}(v)$	Value	0.667	0.750	0.167	0.583	0.333	0.000	0.833	1.000	0.167	0.667	0.000	0.000	0.000
	Rank	4	3	8	6	7	10	2	1	8	4	10	10	10
$B^{\mathcal{X}}(v)$	Value	0.383	0.500	0.125	0.269	0.339	0.000	0.524	0.214	0.133	0.400	0.000	0.000	0.000
	Rank	4	2	9	6	5	10	1	7	8	3	10	10	10

Author's personal copy

dependency of s and t on v can be computed using the following theorem.

Theorem 2 *Assume a vertex v, its x-ego network $\mathcal{X}_v$, and its two different 1-hop neighbors s and t (i.e., $s, t \in V_1(v)$ and $s \neq t$). Then, $\delta_{st}(v) = 1/|V_1(s) \cap V_1(t)|$ if there is no edge between s and t (i.e., $\{s, t\} \notin E_1(v)$).*

Proof Since $\{s, t\} \notin E_1(v)$, $d(s, t) > 1$. In this case, $d(s, t) = 2$ due to the path $s - v - t$. Furthermore, (i) the number of shortest paths from s to t (i.e., paths whose length is 2) can be expressed as $\sigma_{st} = |V_1(s) \cap V_1(t)|$. On the other hand, (ii) the path $s - v - t$ is the only shortest path from s to t that passes through v (i.e., $\sigma_{st}(v) = 1$) since the length of that path is 2, which must be smaller than the length of any other path from s to t that passes through v. By (i) and (ii), $\delta_{st}(v) = \sigma_{st}(v)/\sigma_{st} = 1/|V_1(s) \cap V_1(t)|$. □

Example 2 In Fig. 2b, vertices a and c are 1-hop neighbors of v and there is no edge between a and c. Furthermore, $V_1(a) = \{b, e, f, g, v\}$ and $V_1(c) = \{d, g, h, v\}$. Therefore, $\delta_{ac}(v) = 1/|V_1(a) \cap V_1(c)| = 1/|\{g, v\}| = 1/2$.

Just like Theorem 1, the following theorem quickly computes the dependency of two vertices on a vertex v. While the former applies to a pair of 1-hop neighbors of v, the latter applies to a pair of a 1-hop or 2-hop neighbor and a 2 hop-neighbor of v.

Theorem 3 *Assume a vertex v, its x-ego network $\mathcal{X}_v$, a vertex $s \in V_{\leq 2}(v)$, and another vertex $t \in V_2(v)$ such that $s \neq t$. Then, $\delta_{st}(v) = 0$ if $\delta_{sn}(v) = 0$ for a 1-hop neighbor vertex n of vertex t (i.e., for $n \in V_1(t)$).*

Proof Since $\delta_{sn}(v) = 0$ (i.e., no shortest path from s to n passes through v), (i) $d(s, n) < d(s, v) + d(v, n)$. Then, (ii) $d(s, t) \leq d(s, n) + d(n, t) < d(s, v) + d(v, n) + d(n, t)$ by (i). Furthermore, $n \in V_1(t)$ and $t \in V_2(v)$, meaning that vertex n is either a 1-hop or a 3-hop neighbor of v. In $\mathcal{X}_v$, however, any vertex including n is at most 2 hops away from v. For this reason, n is a 1-hop neighbor of v. Since $d(v, n) = d(n, t) = 1$ and $d(v, t) = 2$, (iii) $d(v, n) + d(n, t) = d(v, t)$. By (ii) and (iii), $d(s, t) < d(s, v) + d(v, t)$ (i.e., no shortest path from s to t passes through v). Therefore, $\delta_{st}(v) = \sigma_{st}(v)/\sigma_{st} = 0/\sigma_{st} = 0$. □

Example 3 In Fig. 2b, $b \in V_{\leq 2}(v)$ and $g \in V_2(v)$. Furthermore, for a 1-hop neighbor a of g, $\delta_{ba}(v) = 0$ (Theorem 1). Therefore, by Theorem 3, $\delta_{bg}(v) = 0$.

Given vertices $s \in V_{\leq 2}(v)$ and $t \in V_2(v)$ such that $s \neq t$, Theorem 3 cannot be applied if $\delta_{sn}(v) > 0$ for every 1-hop neighbor n of t. In this case, the dependency of s and t on v can be obtained using the following theorem.

Theorem 4 *Assume a vertex v, its x-ego network $\mathcal{X}_v$, a vertex $s \in V_{\leq 2}(v)$, and another vertex $t \in V_2(v)$ such that $s \neq t$. Then, $\delta_{st}(v) = \bar{H}(\{\delta_{sn}(v) : n \in V_1(t)\})$ if $\delta_{sn}(v) > 0$ for every 1-hop neighbor n of vertex t, where $\bar{H}(\{\delta_{sn}(v) : n \in V_1(t)\})$ denotes the harmonic mean computed over $\{\delta_{sn}(v) : n \in V_1(t)\}$.*

Proof We prove this theorem considering the following two cases.

[Case I. s is a 1-hop neighbor of v (i.e., $s \in V_1(v)$)] Let n_i ($i = 1, 2, \cdots, m$) denote the ith 1-hop neighbor of t. Then, each n_i is a 1-hop neighbor of v since, in $\mathcal{X}_v$, any 2-hop neighbor (including t) of v can be connected only to a 1-hop neighbor of v (Definition 1). Thus, by Theorem 2, $\delta_{sn_i}(v) = 1/|V_1(s) \cap V_1(n_i)|$. Since $|V_1(s) \cap V_1(n_i)|$ represents the number of shortest paths from s to n_i (i.e., σ_{sn_i}) and each n_i has an edge to t, the total number of shortest paths from s to t can be expressed as (i) $\sigma_{st} = \sum_{i=1}^{m} |V_1(s) \cap V_1(n_i)| = \sum_{i=1}^{m} 1/\delta_{sn_i}(v)$. On the other hand, the path $s - v - n_i - t$ is the only shortest path from s to t via both v and n_i since the length of the path is 3 and any other path from s to t via both v and n_i must be longer. For this reason, (ii) there are m shortest paths from s to t via v (i.e., $\sigma_{st}(v) = m$). By (i) and (ii), $\delta_{st}(v) = \frac{m}{\sum_{i=1}^{m} 1/\delta_{sn_i}(v)} = \bar{H}(\delta_{sn}(v))$.

[Case II. s is a 2-hop neighbor of v (i.e., $s \in V_2(v)$)] Let n_i ($i = 1, 2, \cdots, m$) denote the ith 1-hop neighbor of t. Let also p_j ($j = 1, 2, \cdots, k$) denote the jth 1-hop neighbor of s. In $\mathcal{X}_v$, 2-hop neighbors (including s and t) of v can be connected only to a 1-hop neighbor of v. For this reason, any shortest path from s to t must contain a shortest path from p_j to n_i for some j and i (i.e., $\sigma_{st} = \sum_{i=1}^{m} \sum_{j=1}^{k} |V_1(n_i) \cap V_1(p_j)|$). For a pair of i and j, on the other hand, the path $n_i - v - p_j$ is the only shortest path from n_i to p_j via v and so is the path $t - n_i - v - p_j - s$ (i.e., $\sigma_{st}(v) = mk$). Therefore,

$$
\begin{aligned}
\delta_{st}(v) &= \frac{mk}{\sum_{i=1}^{m}\sum_{j=1}^{k}|V_1(n_i) \cap V_1(p_j)|} \\
&= \frac{mk}{\sum_{i=1}^{m}\sum_{j=1}^{k} 1/\delta_{n_i p_j}(v)} \\
&= \frac{m}{\sum_{i=1}^{m} \frac{\sum_{j=1}^{k} 1/\delta_{n_i p_j}(v)}{k}} \\
&= \frac{m}{\sum_{i=1}^{m} 1/\delta_{n_i s}(v)} \\
&= \frac{m}{\sum_{i=1}^{m} 1/\delta_{sn_i}(v)} \\
&= \bar{H}(\delta_{sn}(v)).
\end{aligned}
$$

□

Author's personal copy

Example 4 In Fig. 2b, $V_1(h) = \{c, d\}$. By Theorem 2, $\delta_{ac}(v) = \frac{1}{2}$ and $\delta_{ad}(v) = 1$. Therefore, by Theorem 4, $\delta_{ah}(v) = \bar{H}(\{\delta_{an}(v) : n \in V_1(h)\}) = \bar{H}(\{\delta_{ac}(v), \delta_{ad}(v)\}) = \frac{2}{\frac{1}{\frac{1}{2}}+\frac{1}{1}} = \frac{2}{3}$.

3 X-ego betweenness computation

This section provides an overview of previous work on betweenness computation (Section 3.1), presents our algorithm for quickly computing x-ego betweenness (Section 3.2), and analytically shows the benefits of our algorithm (Section 3.3).

3.1 Previous work on betweenness computation

To the best of our knowledge, the fastest algorithm for computing the betweenness of every vertex in a given graph is developed by Brandes [19]. Given an unweighted graph, this algorithm performs breadth first search from each vertex to compute the number of shortest paths from that vertex (i.e., the root of search) to each visited vertex. The Brandes algorithm then derives the betweenness of each vertex by aggregating the previously computed count values backwards along the edges. Given an unweighted graph $G(V, E)$, this Brandes algorithm takes $O(|V||E|)$ time.

The x-ego betweenness of a vertex can be found by obtaining the betweenness of that single vertex in the corresponding x-ego network. Applying the Brandes algorithm to the x-ego network, however, results in finding the betweenness of every vertex in the x-ego network (i.e., the Brandes algorithm incurs overhead to find unneeded information). Section 3.2 presents our algorithm that quickly computes x-ego betweenness by computing the betweenness of only the target vertex and by skipping computations according to the properties of x-ego networks explained in Section 2.3.

3.2 Our x-ego betweenness algorithm

Given a vertex v and its x-ego network $\mathcal{X}_v$, let u be the number of 1-hop neighbors of v (i.e., $u = |V_1(v)|$). Furthermore, let w be the number of 2-hop neighbors of v (i.e., $w = |V_2(v)|$). Then, consider an one-to-one mapping I from $V_{\leq 2}(v) = \{v\} \cup V_1(v) \cup V_2(v)$ to $\{0, 1, 2, \cdots, u+w\}$, which maps (1) v to 0, (2) each vertex in $V_1(v)$ to a distinct number in $\{1, 2, \cdots, u\}$, and (3) each vertex in $V_2(v)$ to a distinct number in $\{u+1, u+2, \cdots, u+w\}$. In this case, we can assume a table which stores, for each s and t in $V_{\leq 2}(v)$, $\delta_{st}(v)$ (i.e., the dependency of a pair s and t on vertex v) as the element at the $I(s)$-th row and $I(t)$-th column. Table 3 shows such a *dependency table* for the vertex v in Fig. 2(b). For example, in the dependency table, the element at the 1st row and 3rd column is $1/2$ since $\delta_{ac}(v) = 1/2$ (Example 2) and $I(a) = 1$ and $I(c) = 3$.

Algorithm 1 describes our approach for quickly computing $B^{\mathcal{X}}(v)$. This approach uses a 1-dimensional array $N[0, 1, ..., u+w]$ whose p-th element, $N[p]$, is a collection storing $I(n)$ for each 1-hop neighbor n of vertex s such that $I(s) = p$. For example, in Fig. 2(b) and Table 3, $V_1(v) = \{a, b, c, d\}$ and $I(v) = 0, I(a) = 1, I(b) = 2, I(c) = 3$, and $I(d) = 4$. Therefore, $N[0] = \{I(n) : n \in V_1(v)\} = \{I(a), I(b), I(c), I(d)\} = \{1, 2, 3, 4\}$. Algorithm 1 builds a 2-dimensional array $D[1, 2, ..., u+w][1, 2, ..., u+w]$ to store and then quickly retrieve dependency values. As further explained below, for vertices s and t such that $I(s) < I(t)$, Algorithm 1 stores $\delta_{st}(v)$ at $D[I(s)][I(t)]$ (lines 7, 11, and 17). Since $\delta_{st}(v) = \delta_{ts}(v)$ for all vertices s and t (Section 2.2), $\delta_{ts}(v)$ can also be obtained from $D[I(s)][I(t)]$. As a result, Algorithm 1 stores dependency values only in $D[p][q]$ for all $p, q \in \{1, 2, \cdots, u+w\}$ such

Table 3 Dependency table for vertex v in Fig. 2b

Vertex group			$V_1(v)$				$V_2(v)$				
Mapping I	vertex		a	b	c	d	e	f	g	h	i
	index		1	2	3	4	5	6	7	8	9
$V_1(v)$	a	1	0	**0**	**1/2**	**1/1**	**0**	**0**	**0**	**2/3**	**1/1**
	b	2	0	0	**1/1**	**1/1**	**0**	**0**	**0**	**2/2**	**1/1**
	c	3	1/2	1/1	0	**0**	**1/2**	**2/3**	**0**	**0**	**0**
	d	4	1/1	1/1	0	0	**1/1**	**2/2**	**0**	**0**	**0**
$V_2(v)$	e	5	0	0	1/2	1/1	0	**0**	**0**	**2/3**	**1/1**
	f	6	0	0	2/3	2/2	0	0	**0**	**4/5**	**2/2**
	g	7	0	0	0	0	0	0	0	**0**	**0**
	h	8	2/3	2/2	0	0	2/3	4/5	0	0	**0**
	i	9	1/1	1/1	0	0	1/1	2/2	0	0	0

Author's personal copy

that $p < q$ (see the dependency values in boldface above the diagonal in Table 3).

For $p \in \{1, 2, \cdots, u\}$ and $q \in \{p+1, 2, \cdots, u\}$ (lines 5 - 9 in Algorithm 1), by the definition of mapping I, there exist $s \in V_1(v)$ and $t \in V_1(v)$ such that $p = I(s)$ and $q = I(t)$. If there is an edge between s and t (i.e., $t \in V_1(s)$ and, equivalently, $q \in N[p]$), $\delta_{st}(v) = 0$ (Theorem 2). Otherwise, $\delta_{st}(v) = \frac{1}{|V_1(s) \cap V_1(t)|} = \frac{1}{|N[p] \cap N[q]|}$ (Theorem 2). Given p and q as well as array N, Algorithm 2 computes $\delta_{st}(v)$ as mentioned above. The resulting value is saved at $D[p][q]$ (line 7 in Algorithm 1) and incorporated into the sum of calculated dependency values (line 8).

Algorithm 1 *x-ego_betweenness*(v, N)

```
1: Input: a vertex v and an array N[0, 1, ..., u + w]
2: Output: B^X(v)
3: create an array D[1, 2, ..., u + w][1, 2, ..., u + w]
4: sum ← 0
5: for p : 1 to u do
6:     for q : p + 1 to u do
7:         D[p][q] ← dependency1(p, q, N)
8:         sum ← sum + D[p][q]
9:     end for
10:    for q : u + 1 to u + w do
11:        D[p][q] ← dependency2(p, q, N, D)
12:        sum ← sum + D[p][q]
13:    end for
14: end for
15: for p : u + 1 to u + w do
16:    for q : p + 1 to u + w do
17:        D[p][q] ← dependency2(p, q, N, D)
18:        sum ← sum + D[p][q]
19:    end for
20: end for
21: return 2·sum / ((u+w)(u+w−1))
```

Algorithm 2 *dependency1*(p, q, N)

```
1: Input: p, q, N[0, 1, ..., u + w]
2: if q ∈ N[p] then
3:     return 0                                  ▷ by Theorem 1
4: else
5:     return 1/| N[p] ∩ N[q] |                  ▷ by Theorem 2
6: end if
```

Algorithm 3 *dependency2*(p, q, N, D)

```
1: Input: p, q, N[0, 1, ..., u+w], D[1, 2, ..., u+w][1, 2, ..., u+w]
2: C ← []
3: for each r ∈ N[q] do
4:     if p < r then
5:         τ = D[p][r]
6:     else
7:         τ = D[r][p]
8:     end if
9:     if τ == 0 then
10:        return 0                              ▷ by Theorem 3
11:    else
12:        append τ to C
13:    end if
14: end for
15: return H̄(C)                                  ▷ by Theorem 4
```

For $p \in \{1, 2, \cdots, u\}$ and $q \in \{u+1, u+2, \cdots, u+w\}$ (lines 10 - 13 in Algorithm 1), there exist $s \in V_1(v)$ and $t \in V_2(v)$ such that $p = I(s)$ and $q = I(t)$. If $\delta_{sn}(v) = 0$ for some $n \in V_1(t)$, then $\delta_{st}(v) = 0$ (Theorem 3). Otherwise, $\delta_{st}(v) = \bar{H}(\delta_{sn}(v))$, where $\bar{H}(\delta_{sn}(v))$ denotes the *harmonic mean* computed over $\{\delta_{sn}(v) : n \in V_1(t)\}$ (Theorem 4). Given p and q and arrays N and D, Algorithm 3 computes $\delta_{st}(v)$ for s and t such that $p = I(s)$ and $q = I(t)$ according to the above theorems. For each $r \in N[q]$ (line 3), there exists a vertex n such that $n \in V_1(t)$ and $I(n) = r$. At this point, $\delta_{sn}(v)$ must have been stored at either $D[p][r]$ (if $p < r$) or $D[r][p]$ (otherwise) since n is a 1-hop neighbor[2] of v (i.e., $r \in \{1, 2, \cdots, u\}$ and therefore has been handled by lines 6-9 in Algorithm 1). Algorithm 3 retrieves that dependency value (lines 4 - 8). When Algorithm 3 returns $\delta_{st}(v)$ (lines 10 and 15), that value is saved at $D[p][q]$ and incorporated into the sum of calculated dependency values (lines 11 and 12 in Algorithm 1). For $p \in \{u+1, u+2, \cdots, u+w\}$ and $q \in \{p+1, p+2, \cdots, u+w\}$ (lines 15-20 in Algorithm 1), our approach computes dependency values in a way similar to the above case. Next, it computes $B^{\mathcal{X}}(v)$ (line 21) according to Eq. 1. Note that $|V_{\leq 2}(v)| = |V_0(v) \cup V_1(v) \cup V_2(v)| = 1 + u + w$ and $2 \cdot \sum_{p=1}^{u+w} \sum_{q=p+1}^{u+w} D[p][q] = \sum_{s \neq v \neq t \in V} \delta_{st}(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$.

3.3 Algorithm complexity

Assume a vertex v and its x-ego network $\mathcal{X}_v$ where v has u 1-hop neighbors (i.e., $u = |V_1(v)|$) and w 2-hop neighbors (i.e., $w = |V_2(v)|$). Furthermore, assume that $\mathcal{X}_v$ contains z edges. Given $\mathcal{X}_v$, the Brandes algorithm takes $O((u+w)z)$ time (Section 3.1). In this case, our x-ego betweenness algorithm also takes $O((u+w)z)$ time as follows:

Theorem 5 *Given a vertex v and its x-ego network $\mathcal{X}_v$ where there are z edges and v has u 1-hop neighbors and w 2-hop neighbors, our x-ego betweenness algorithm (Algorithm 1) takes $O((u+w)z)$ time.*

Proof Algorithm 2 takes $O(|N[p]| + |N[q]|)$ time since the set intersection operation on line 5 can be completed in $O(|N[p]| + |N[q]|)$ time (e.g., by inserting the vertex indices from $N[p]$ into an array or a hash table and then finding each vertex from $N[q]$ in that array or hash table) and all other operations can be completed in $O(1)$ time. Algorithm 3 takes $O(|N[q]|)$ time since it examines every

[2] In $\mathcal{X}_v$, all vertices including n are at most 2 hops away from v. $n \neq v$ cannot be a 2-hop neighbor of v since it is a 1-hop neighbor of t, which is a 2-hop neighbor of v. Therefore, n can only be a 1-hop neighbor of v.

Author's personal copy

vertex index from $N[q]$ (lines 3-14) and may compute the harmonic mean of $|N[q]|$ values (line 15). Therefore, the overall time complexity of Algorithm 1 can be expressed as

$$O\Big(\sum_{p=1}^{u}\Big(\sum_{q=p+1}^{u}(|N[p]|+|N[q]|)+\sum_{q=u+1}^{u+w}|N[q]|\Big)+\sum_{p=u+1}^{u+w}\sum_{q=p+1}^{u+w}|N[q]|\Big)$$
$$=O\Big(\sum_{p=1}^{u}\sum_{q=p+1}^{u}|N[p]|+\sum_{p=1}^{u+w}\sum_{q=p+1}^{u+w}|N[q]|\Big). \quad (2)$$

Since the sum of the number of 1-hop neighbors for each vertex in $\mathcal{X}_v$ is $2z$ (that is, $\sum_{p=0}^{u+w}|N[p]|=2z$), we have

$$\sum_{p=1}^{u}\sum_{q=p+1}^{u}|N[p]| \le \sum_{p=1}^{u}\sum_{q=1}^{u}|N[p]| = u\sum_{p=1}^{u}|N[p]| \le u\sum_{p=0}^{u+w}|N[p]| = 2uz$$

and

$$\sum_{p=1}^{u+w}\sum_{q=p+1}^{u+w}|N[q]| \le \sum_{p=1}^{u+w}\sum_{q=0}^{u+w}|N[q]| = \sum_{p=1}^{u+w}(2z) = 2(u+w)z.$$

Therefore, the above complexity (Eq. 2) can be expressed as $O(uz+(u+w)z)=O((u+w)z)$. □

While the Brandes algorithm and our algorithm have comparable worst case time complexity, the evaluation results in Section 4.3 show that our algorithm outperforms the Brandes algorithm. One main reason for this phenomenon is that our algorithm can skip dependency computations according to Theorems 1 and 3. In the ideal situation where the dependency computations are skipped (i.e., the conditions on line 2 in Algorithm 2 and on line 9 in Algorithm 2 are met) as early as possible, both Algorithms 2 and 3 will terminate in $O(1)$ time, thereby reducing the time complexity of Algorithm 1 from $O((u+w)z)$ to $O((u+w)^2)$. The rate at which dependency computations are skipped increases as the *clustering coefficient* of each node (i.e., the number of links between a node's neighbors divided by the total number of possible links among them [20]) increases. Further details of that rate and its impact on x-ego betweenness computation are discussed in Section 4.3.

4 Evaluation

This section presents evaluation results that show the effectiveness of x-ego betweenness and the efficiency of our algorithm for computing x-ego betweenness. Section 4.1 describes the wireless trace data sets and the methods for constructing x-ego networks in our evaluations. Our evaluation results demonstrate that x-ego betweenness is a more accurate approximation of betweenness than ego betweenness (Section 4.2) and our algorithm more quickly computes x-ego betweenness than the Brandes algorithm (Section 4.3).

4.1 Data sets and x-ego network construction

Our evaluations used the CRAWDAD trace data sets [16] that are referred to as *Infocom05*, *Infocom06*, *Cambridge* and *Intel*. Table 4 provides details of these data sets. Each of these data sets includes a number of traces of Bluetooth sightings by groups of users carrying small devices (iMotes) for a number of days. These data sets cover a variety of network scales (from small to large number of nodes) and contact densities (from sparse to dense contacts). For each data set, Table 4 shows the mean and median values of accumulated contact duration per node pair. The disparity between these values indicates that most node pairs make contacts for a short amount of time and there are a few outliers representing long contact durations. Further details of the data sets can be found from the CRAWDAD website [21]. In our evaluations, a social link between two nodes was assumed if the accumulated contact duration between the two nodes was above a given threshold θ. For each data set, we determined nine duration threshold values that correspond to the link addition ratio of 10 %, 20 %, $\cdots$, 90 %. More specifically, for the link addition ratio of 100 %, we set the contact duration threshold (θ) to 0 seconds. In this case, any contact between two nodes led to the addition of a social link between the nodes. Then, we determined each duration threshold value (e.g., $\theta = 141, 468, \cdots, 5921$ in the case of *Infocom05*) so that 90 %, 80 %, $\cdots$, 10 % of all possible social links would be added (Table 4). It should be noted that the contact duration threshold value corresponding to the link addition ratio of 50 % is the median of accumulated contact duration.

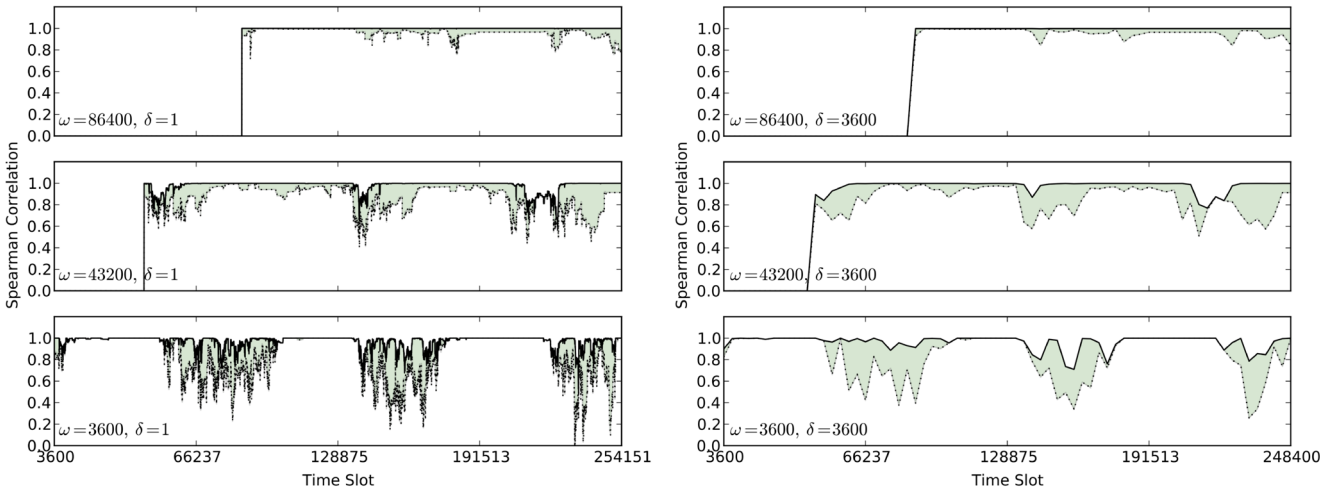
To obtain the contact durations between nodes, our evaluations assumed 1 second time slots with the start time set to 0 and the end time π set to the duration of data collection (e.g., $\pi = 254{,}151$ seconds in the case of the *Infocom05* data set). Our evaluations also assumed a *time window*, a set of consecutive ω time slots, which repeatedly advanced by δ time slots (ω and δ are called the *window size* and

Author's personal copy

Table 4 Data Sets

		Infocom05	Infocom06	Cambridge	Intel
Number of nodes		41 (mobile: 41, fixed: 0)	98 (mobile: 78, fixed: 20)	12 (mobile: 12, fixed: 0)	9 (mobile: 11, fixed: 1)
Number of contacts between nodes		22,459	170,601	4,228	1,364
Average number of contacts per node pair		14.3	20.2	32.0	18.9
Mean of accumulated contact duration per node pair (sec.)		3,348	3,356	9,961	12,495
Median of accumulated contact duration per node pair (sec.)		1,609	1,253	2,684	4,936
Link addition ratio (%) and Corresponding link addition threshold, θ (sec.)	10 %	5,921	6,381	19,606	24,329
	20 %	3,818	3,883	7,786	16,725
	30 %	2,905	2,568	4,470	10,915
	40 %	2,167	1,815	3,246	6,572
	50 %	1,609	1,253	2,684	4,936
	60 %	1,214	862	2,187	4,122
	70 %	846	506	1,806	2,211
	80 %	468	251	1,566	1,200
	90 %	141	10	1,318	9
Data collection period, π (sec.)		254,151 (2.9 days)	337,419 (3.9 days)	455,610 (5.3 days)	359,191 (4.2 days)

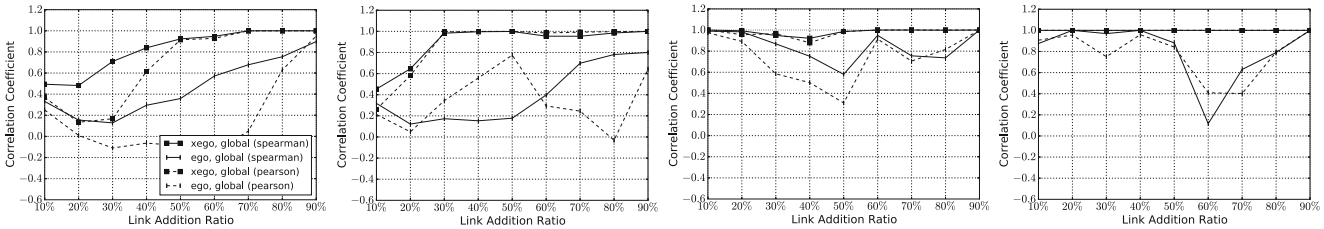
window step size, respectively). For an integer i such that $0 \leq i \leq \lfloor \frac{\pi-\omega+1}{\delta} \rfloor$, the i-th time window represents the time period $[t_{\delta \cdot i}, t_{\delta \cdot i+\omega-1}]$, where $t_j (0 \leq j \leq \pi - 1)$ denotes the j-th time slot. Given a data set, our evaluations identified, for each time window T_i, social links between nodes. In this case, any pair of nodes that communicated with each other for $\theta \cdot \omega/\pi$ time slots or more were assumed to have a social link between them. On the network consisting of the above nodes and social links, our evaluations then computed the betweenness, ego-betweenness, and x-ego betweenness of each node using the Brandes algorithm [19]. The x-ego betweenness of each node was also computed using our



(a) *Step size* $\delta = 1$ *second* (b) *Step size* $\delta = 3600$ *seconds* $(= 1$ *hour*$)$

Fig. 3 Change of correlation between different betweenness metrics over time (the solid line represents the correlation between x-ego betweenness and betweenness whereas the dotted line represents the correlation between ego betweenness and betweenness)

Author's personal copy



(a) *Infocom05* (b) *Infocom06* (c) *Cambridge* (d) *Intel*

Fig. 4 Correlation between different betweenness metrics ($\omega = \pi$)

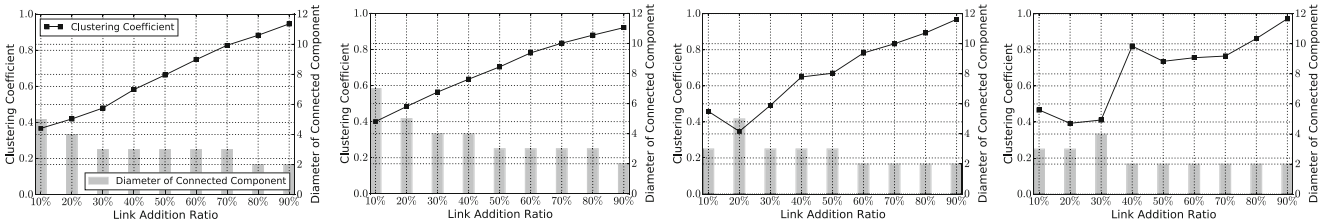
algorithm (Section 3) to experimentally verify the benefit of that algorithm over the Brandes algorithm.

4.2 Accuracy of x-ego betweenness

Figure 3 compares the accuracy of x-ego betweenness and ego betweenness in terms of their similarity to betweenness. Each subfigure in Fig. 3 shows the variation of the Spearman correlation between x-ego betweenness and betweenness (the dotted line) and that between ego betweenness and betweenness (the solid line). The results illustrated in the figures were obtained from the *Infocom05* data set with the time window size ω set to 24 hours (86,400 seconds), 12 hours (43,200 seconds), or 1 hour (3,600 seconds) and the step size δ set to 1 second or 3,600 seconds. Furthermore, the link addition ratio was set to 50 % (equivalently, the contact duration threshold was set to 1,609 seconds). For this reason, when $\omega = 86{,}400$ and $\delta = 3{,}600$, a node assumed a social link with another node if the accumulated contact duration between these nodes was longer than $1{,}609 \cdot 86{,}400/254{,}151 (= 546.9)$ seconds.

Figure 3 shows that x-ego betweenness is more strongly correlated to betweenness than ego betweenness (i.e., x-ego betweenness is more accurate than ego betweenness) across different time window sizes and step sizes. As the window step size (δ) increases from 1 second (Fig. 3 (a)) to 3,600 seconds (Fig. 3 (b)), the betweenness of each vertex is computed/updated less frequently, thereby saving communication and computation resources at the expense of timeliness of betweenness values. When the window step size (δ) is fixed, as the time window size (ω) increases, the variation of correlation between x-ego betweenness and betweenness decreases since the overlap between consecutive windows increases. Given a larger time window, the correlation between x-ego betweenness and betweenness also increases since the x-ego network of each node may include more nodes (i.e., the accuracy of x-ego betweenness may improve). On the other hand, even with small time windows, the overall correlation between x-go betweenness and betweenness remains high. For example, in Fig. 3, only 2.8 % of correlation values between x-ego betweenness and betweenness are smaller than 0.8 when $\omega = 3{,}600$ and $\delta = 1$, and only 7.5 % when $\omega = 3{,}600$ and $\delta = 3{,}600$.

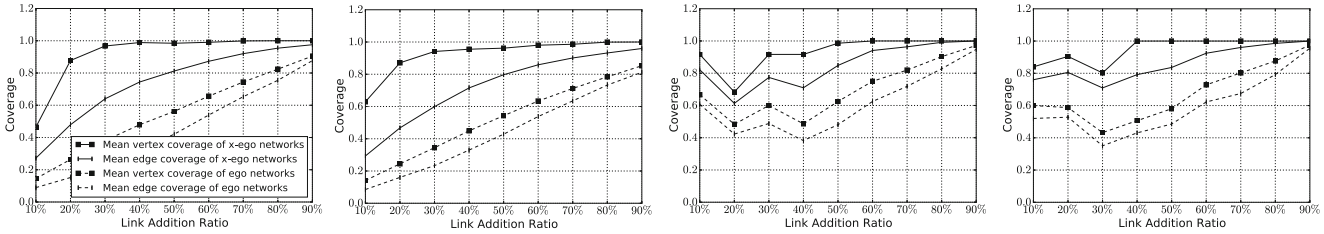
Figure 4 shows the correlation between x-ego betweenness and betweenness as well as between ego betweenness and betweenness across diverse link addition ratios and data sets. This result was obtained by considering all contact information from each data set using a time window that corresponds to the data collection period (i.e., $\omega = 2.9, 3.9, 5.3$ and 4.2 days for *Infocom05*, *Infocom06*, *Cambridge* and *Intel*, respectively). For each combination of data set and link addition ratio, we used the contact duration threshold shown in Table 4.



(a) *Infocom05* (b) *Infocom06* (c) *Cambridge* (d) *Intel*

Fig. 5 Clustering coefficient and diameter of connected component ($\omega = \pi$)

Author's personal copy



(a) *Infocom05* (b) *Infocom06* (c) *Cambridge* (d) *Intel*

Fig. 6 Coverage of x-ego and ego networks over the global network ($\omega = \pi$)

Figure 4 illustrates that:

1) X-ego betweenness is more strongly correlated to betweenness than ego betweenness (i.e., x-ego betweenness is more accurate than ego betweenness).
2) The correlation between x-ego betweenness and betweenness is almost one in most cases. Furthermore, in the cases of *Cambridge* and *Intel*, where a small number of nodes constitute a network, the correlation between x-ego betweenness and betweenness is very close to one across all link addition ratios.

In contrast to the high correlation between x-ego betweenness and betweenness, the correlation between ego betweenness and betweenness is noticeably low in many cases. For example, it is mostly below 0.8 for all of the data sets and also mostly below 0.6 for *Infocom05* and *Infocom06*. The above results indicate the superiority of x-ego betweenness over ego betweenness in wireless networks.

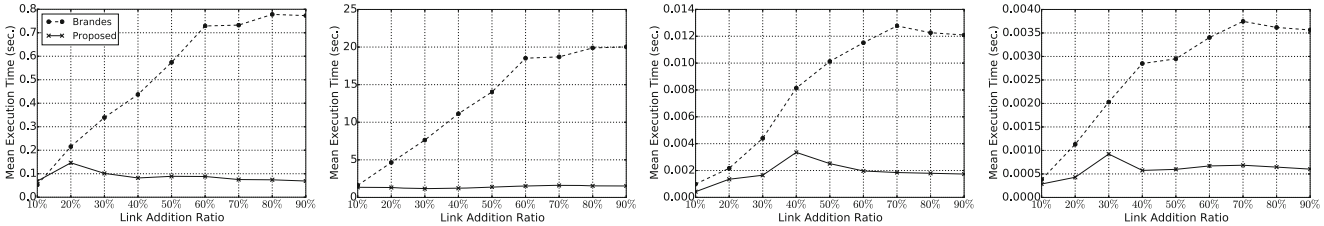
In Fig. 4, while the same network overhead is required for both x-ego betweenness and ego betweenness (Section 2.1), the correlation between x-ego betweenness and betweenness is consistently higher than the correlation between ego betweenness and betweenness. There are, however, link addition ratios (e.g., 10 % and 20 % in Fig. 4 (a) and (b)) that lead to relatively low correlation between x-ego betweenness and betweenness. To understand these situations, we obtained further evaluation results that are depicted in Figs. 5 and 6.

Figure 5 shows, for each data set, the impact of link addition ratio on the clustering coefficient and the average diameter of the connected components in the corresponding network. As more links are added, the 1-hop neighbors of each node tend to have more links between them (i.e., the clustering coefficient increases) and the shortest path length between nodes tends to decrease (i.e., the diameter of the connected component containing the node decreases).[3] Figs. 4 and 5 show that x-ego betweenness and betweenness are strongly correlated when the average diameter of the connected components is less than 4. In such cases, there is a large overlap between the x-ego network of a node and the connected component containing that node (i.e., x-betweenness is an accurate estimation of betweenness).

Figure 6 shows the size of x-ego and ego networks compared to the global network. For this result, we found, for each node v, the number of vertices in the x-ego network of v (denoted as $|\mathcal{X}_v(V)|$) and that in the ego network of v (denoted as $|\mathcal{E}_v(V)|$) as well as the number of edges in the x-ego network of v (denoted as $|\mathcal{X}_v(E)|$) and that in the ego network of v (denoted as $|\mathcal{E}_v(E)|$). We also obtained, for each node v, the number of vertices and that of edges in the connected component containing v (denoted as $|\mathcal{C}_v(V)|$ and $|\mathcal{C}_v(E)|$, respectively) in the global network. Then, the "mean vertex coverage of x-ego networks" and "mean edge coverage of x-ego networks" were computed as $\frac{\sum_{v\in V}\frac{|\mathcal{X}_v(V)|}{|\mathcal{C}_v(V)|}}{|V|}$ and $\frac{\sum_{v\in V}\frac{|\mathcal{X}_v(E)|}{|\mathcal{C}_v(E)|}}{|V|}$, respectively. Furthermore, the "mean vertex coverage of ego networks" and "mean edge coverage of ego networks" were computed as $\frac{\sum_{v\in V}\frac{|\mathcal{E}_v(V)|}{|\mathcal{C}_v(V)|}}{|V|}$ and $\frac{\sum_{v\in V}\frac{|\mathcal{E}_v(E)|}{|\mathcal{C}_v(E)|}}{|V|}$, respectively. Figure 6 shows that, while ego and x-ego networks require the same network overhead (Section 2.1), x-ego networks contain a substantially larger number of vertices and edges than ego networks (i.e., x-ego betweenness is a more accurate estimation of betweenness than ego betweenness).

[3] For the link addition ratio of 10% in Fig. 5 (c) and the same of 10% and 20 % in Fig. 5 (d), the network diameter is unusually small since, due to a very small number of links in the underlying network, each node has only a few reachable nodes and the latter are within 2 hops away from the former.

Author's personal copy



(a) *Infocom05* (b) *Infocom06* (c) *Cambridge* (d) *Intel*

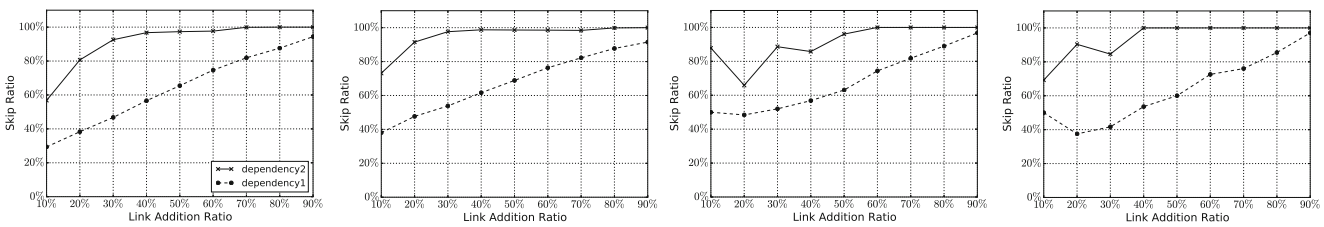
Fig. 7 Comparison of algorithm execution speed to get x-ego betweenness

4.3 Efficiency of Our x-ego betweenness algorithm

This section evaluates the performance benefit of our x-ego betweenness algorithm (Section 3). To the best of our knowledge, no other x-ego betweenness algorithms have been proposed in the literature. For this reason, we compare the performance of our algorithm to that of the Brandes algorithm [19], the state-of-the-art method for computing betweenness, while applying the latter to the x-ego network of each node. Figure 7 shows the mean execution time of these algorithms (averaged over all nodes). For each data set, this evaluation result was obtained by using a time window that corresponded to the data collection period (i.e., $\omega = \pi$). As in Fig. 4, the link addition ratio was varied from 10% (sparse) to 90 % (dense). Figure 7 shows that our algorithm is substantially faster than the Brandes algorithm. In the figure, the benefit of our algorithm over the Brandes algorithm is more evident in dense networks (i.e., when the link addition ratio is high). As the link addition ratio increases (i.e., the network density increases), the execution time of the Brandes algorithm increases. On the other hand, the execution time of our algorithm may decrease in such cases.

To understand the above benefits of our algorithm, we further analyzed our evaluation results with a focus on $\frac{NUM_X}{NUM_C}$, which we call the *skip ratio*. When *dependency1* (Algorithm 2) is considered, NUM_C denotes the number of dependency computations to which Theorems 1 or 2 are applied and NUM_X denotes the number of dependency computations to which only Theorem 1 is applied (both NUM_C and NUM_X are determined while the x-ego betweenness of every node is calculated). When *dependency2* (Algorithm 3) is considered, NUM_C denotes the number of dependency computations to which Theorems 3 or 4 are applied and NUM_X denotes the number of dependency computations to which only Theorem 3 is applied. As NUM_X increases (i.e., Theorem 1 or Theorem 3 are more frequently applied), more expensive set operations (line 5 of Algorithm 2) and harmonic mean computations (line 15 of Algorithm 3) are skipped, thereby reducing the overall x-ego betweenness computation time.

Figure 8 shows that the skip ratio for *dependency1* is in general higher than 80 % across all link addition ratios and is almost 100 % (i.e., the set intersection operation on line 5 of Algorithm 2 is very rarely performed) when the link addition ratio is higher than 50 %. In a dense x-ego network, the probability that two arbitrary 1-hop neighbors of a node share an edge is high (i.e., the skip ratio is high since Theorem 1 applies frequently). In this case, the skip ratio for *dependency2* is also high as Theorem 3 applies frequently. In Fig. 8, the skip ratio for *dependency2* is higher than 60 % when the link addition ratio is higher than 50 %. These high skip ratios account for the performance benefit of our x-ego betweenness algorithm in dense networks.



(a) *Infocom05* (b) *Infocom06* (c) *Cambridge* (d) *Intel*

Fig. 8 Skip ratio in the proposed algorithm

Author's personal copy

5 Conclusion

In wireless networks, relaying messages via nodes with high betweenness centrality improves both the efficiency and reliability of message delivery. Computing the betweenness of each node, however, incurs impractically high network and computational overhead. In our approach, each node constructs its *x-ego network*, a logical network capturing the social links in its vicinity, and then computes its *x-ego betweenness* (i.e., its betweenness in its x-ego network) as an estimate of its true betweenness in the wireless network. By taking advantage of structural properties of x-ego networks, our algorithm quickly computes x-ego betweenness. Our evaluation results demonstrate that x-ego betweenness accurately estimates betweenness and our algorithm more quickly computes x-ego betweenness than the state-of-the-art betweenness computation algorithm.

Acknowledgements This work has been supported by the National Science Foundation under CAREER award IIS-1149372, and also supported by Basic Science Research Program through the National Research Foundation (NRF) of Korea (No. NRF-2013R1A1A2010050).

References

1. Zhang Y, Pan E, Song L, Saad M, Dawy Z, Han Z (2014) IEEE Trans Wirel Commun 14(1):177
2. Schurgot MR, Comaniciu C, Jaffres-Runser K (2012) IEEE Commun Mag 50(7):155
3. Papadimitriou A, Katsaros D, Manolopoulos Y (2010) Next Generation Society. Technological and Legal Issues 26:411
4. Katsaros D, Dimokas N, Tassiulas L (2010) IEEE Netw 24(6):23
5. Freeman LC (1979) Social Networks 1:215
6. Jain A, Reddy BVR (2013) In: Proc. of 2013 IEEE 3rd International Advance Computing Conference (IACC), pp 127–131
7. Daly EM, Haahr M (2009) IEEE Trans Mobile Comput 8(5):606
8. Hui P, Crowcroft J, Yoneki E (2008) In: IEEE Trans. Mobile Comput., vol 10, pp 1576–1589
9. Dimokas N, Katsaros D, Manolopoulos Y (2007) In: Proc. of the 3rd International Conference on Mobile Multimedia Communications, MobiMedia 2007, pp 377–382
10. Cuzzocreaa A, Papadimitrioub A, Katsarosc D, Manolopoulosb Y (2012) J Netw Comput Appl 35(4):1210
11. Gupta I, Riordan D, Sampalli S (2005) In: 3rd Annual Conference on Communication Networks and Services Research (CNSR 2005), pp 255–260
12. Marsden PV (2002) In: Social Networks, vol 24, pp 407–422
13. Everett M, Borgatti SP (2005) Soc Networks 27(1):31
14. Nanda S, Kotz D (2008) In: Proc of IEEE ICCN
15. Pantazopoulos P, Karaliopoulos M, Stavrakakis I (2013) In: 7th Int'l Workshop on Self-Organizing Systems (IFIP IWSOS'13)
16. Scott J, Gass R, Crowcroft J, Hui P, Diot C, Chaintreau A (2009) CRAWDAD data set cambridge/haggle (v. 2009-05-29). downloaded from http://crawdad.cs.dartmouth.edu/cambridge/haggle
17. Burgess J, Gallagher B, Jensen D, Levine B (2006) In: Proceedings of 25th IEEE INFOCOM, pp 1–11
18. Dubois-ferriere H, Grossglauser M, Vetterli M (2003) In: Proceedings of ACM MobiHoc, pp 257–266
19. Brandes U (2001) Journal of Mathematical Sociology 25:163
20. Tang L, Liu H (2010) Synthesis Lectures on Data Mining and Knowledge Discovery 2(1)
21. A list of papers that use CRAWDAD data and tools. http://www.citeulike.org/group/5303/library