# Minimum Steiner Tree Approximation for Extracting Unknown Information via Avoiding High-Centrality Nodes

Rintaro Nishiyama
*Faculty of Science and Technology*
*Keio University*
Kanagawa, Japan
nishiyama-rintaro@keio.jp

Andrew Shin
*Faculty of Science and Technology*
*Keio University*
Kanagawa, Japan
shin@inl.ics.keio.ac.jp

Naoki Matsumoto
*Faculty of Education*
*University of the Ryukyus*
Okinawa, Japan
naoki_m@edu.u-ryukyu.ac.jp

Kunitake Kaneko
*Faculty of Science and Technology*
*Keio University*
Kanagawa, Japan
kaneko@dmc.keio.ac.jp

***Abstract*—Minimum Steiner tree is frequently used as a means for information retrieval, particularly for search with keyword. While it tends to include high-centrality nodes, the information obtained from such tree may already be accessible to the user, which is not very useful. In this paper, we propose a Steiner tree that avoids high-centrality nodes, in order to increase the likelihood of information being unknown to the user, and thus more useful, while maintaining a high degree of relevance to the keywords. We examine various schemes to select high-centrality nodes to avoid, and compare them in terms of the degree of relevance with the keyword, probability of being unknown to the user, and the pre-computation time. We employ degree centrality and betweenness centrality as our metric for centrality. We find that, our best-performing scheme, despite its simplicity, is able to reduce more than 5% in terms of the number of betweenness centrality nodes compared to the minimum Steiner tree, along with its pre-computation time being about 20% times faster, regardless of our choice of centrality metric.**

***Index Terms*—graph, tree, Steiner tree**

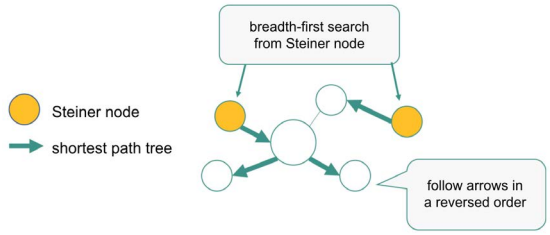
## I. Introduction

In graph theory, trees are frequently used as a means to analyze graphs. Steiner tree, which we are mainly concerned with in this paper, is one of such trees to analyze graphs. Given a node set $Q \subset V$ from an undirected graph $G = (V, E)$ consisting of a set of nodes $V$ and a set of edges $E$, a Steiner tree is a tree containing all nodes contained in $Q$. Also, a minimal Steiner tree is a Steiner tree with the smallest number of edges. Nodes included in $Q$ are called terminals, and non-terminals nodes are called Steiner nodes. Unlike the minimum spanning tree, which connects the given nodes without incorporating any additional nodes, the minimum Steiner tree has the unique capability of accommodating additional nodes within its structure.

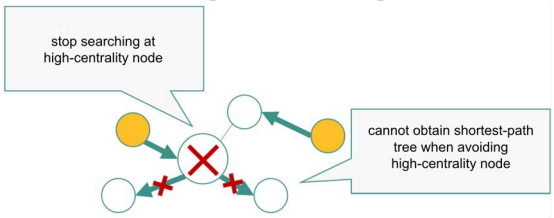
Minimal Steiner trees are used in various applications, including keyword retrieval, which is the primary focus in this paper. In keyword retrieval, the user inputs multiple keywords and the system presents information relevant to the keywords. Specifically, the system finds the minimum Steiner tree on the graph with the keyword as the terminal, and the information is presented to the user based on the Steiner nodes of the minimum Steiner tree.

Since the minimal Steiner tree has the smallest number of edges, the leaves are always terminals, and the distance between terminals on the Steiner tree is generally short. Thus, the Steiner node is close to any terminal and has information with a high degree of relevance to the keyword. In other words, the number of edges of the Steiner tree indicates the degree of relevance to the keyword of the information obtained from the Steiner node, and the smaller the number of edges, the stronger the association, and vice versa. We refer to the number of edges of the Steiner tree as the *size* of the Steiner tree.

The centrality index is an index that indicates the importance of nodes on the graph. The Steiner nodes of a minimal Steiner tree tend to have high centrality. Nodes with high centrality have connections with nodes closer to terminals than nodes with low centrality, and can shorten the distance between terminals on the Steiner tree. As such, the nodes with high centrality tend to be selected as the Steiner nodes of the minimum Steiner tree. However, since nodes with high centrality are likely to contain information that is already known to the user, information obtained from the minimum Steiner tree is also likely to be already known to the user. Thus, in order to obtain information that is unknown to the user, yet related to the keyword, a sufficiently minimal Steiner tree, with a small




(a) An example of a shortest path tree.




(b) An example of a shortest path tree that avoids high-centrality nodes.

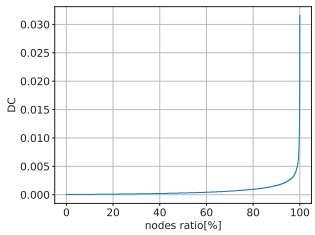
Fig. 1: Comparison of shortest path trees in terms of avoiding high-centrality nodes.

sum of centrality indices from the Steiner nodes, is required.

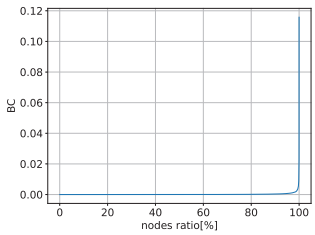
Let us consider an example. Suppose that a university and on-campus clubs are connected on a social network service. A user may perform a keyword search on the club of interest. When the minimum Steiner tree is used, the user is likely to be presented with information that is highly likely to be already known to the user, e.g., the university itself. However, with an approximate minimum Steiner tree whose centrality indices are low, the lesser-known clubs that match the interest are likely to be presented. It is less likely that the users are more familiar with these clubs than the university itself, and the information is thus more useful with higher entropy. As illustrated in this example, by using an approximate minimum Steiner tree with a small total centrality index, the probability of presenting information related to the keywords yet unknown to the user is increased.

SketchLS [7], a computation method for minimal Steiner tree, pre-computes the shortest path from each node to a set of randomly sampled nodes, and uses the shortest paths to compute the minimal Steiner tree. This set of randomly sampled nodes is called the seed nodes. The shortest path is obtained from the shortest path tree starting from the seed nodes, where the shortest path tree is obtained via breadth-first search from the starting point. Figure 1a illustrates an example. The arrows indicate the result of breadth-first search from the seed node in orange. Each node can obtain the shortest path to the seed nodes by following this arrow in a reversed order.

In order to obtain a Steiner tree with a small sum of centrality indices, it is necessary to compute Steiner trees that avoid high-centrality nodes. Also, when calculating a Steiner tree that avoids high-central nodes based on SketchLS, it is necessary to generate a shortest path tree that avoids high-centrality nodes. However, not all nodes can obtain the shortest path




(a) Plots in ascending order of degree centrality




(b) Plots in ascending order of betweenness centrality

Fig. 2: Degree centrality and betweenness centrality.

from the shortest path tree while avoiding high-centrality nodes. Figure 1b illustrates such example. When a high-centrality node is visited, its neighboring nodes are not searched, so the lower-left and lower-right nodes cannot obtain the shortest path to the seed nodes. However, in order to reduce the total centrality index, it is necessary to obtain the shortest path that avoids as many high-centrality nodes as possible. Yet, if we were to find the shortest path tree for every possible combination of high-centrality nodes to be avoided, the pre-computation time will exponentially increase.

The purpose of our work is to find a combination of avoidable high-centrality nodes that can reduce the total centrality index of the Steiner nodes, while suppressing the increase in pre-computation time and keeping the size of the Steiner tree close to the minimum. Since the centrality distribution is concentrated regardless of the metric, as shown in Figure 2a and Figure 2b, the combination is selected by gradually increasing the number of high-centrality nodes to be avoided. We propose various simple schemes to increase the number of high-centrality nodes to avoid. We evaluate the proposed schemes for combinations from the perspectives of 1) the relevance of the information obtained to the keywords, 2) the probability of the information being unknown, 3) and the pre-computation time, and examine how to select the optimal combination among the proposed combination schemes. Our experiments show that it is possible to find a combination scheme that reduces the number of high-centrality nodes compared to the minimum Steiner tree, while suppressing the increase in pre-computation time.

## II. Preliminaries

### A. Centrality Metric

We use two commonly used centrality metrics [11], namely degree centrality and betweenness centrality.

*1) Degree Centrality:* A degree of a node refers to the number of nodes connected to that node. Degree centrality (DC) is based on a premise that the more nodes are connected to a node, the more important that node is. Degree centrality of a node $v$ is defined as following [14] [5]:

$$DC(v) = \frac{degree(v)}{N-1} \tag{1}$$

where $degree(v)$ refers to the degree of the node $v$, and $N$ refers to the number of nodes in the graph.

*2) Betweenness Centrality:* Betweenness centrality is based on the idea that the nodes can be considered more central if they frequently appear on the shortest path between other nodes, hence the name betweenness centrality. Betweenness centrality of a node $v$ is defined as following: [5] [1] [4].

$$BC(v) = \sum_{s \in V, s \neq v} \sum_{t \in V, t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}} \quad (2)$$

where $\sigma_{st}$ indicates the number of shortest paths between $s$ and $t$, and $\sigma_{st}(v)$ refers to the number of shortest paths between $s$ and $t$ that pass the node $v$.

### B. Approximation of Minimal Steiner Tree

Minimum Steiner tree approximation methods prior to SketchLS [7] include BANKS [2], Bidirectional [8], and STAR [9]. However, since these methods require operations such as breadth-first search from each terminal, the computational complexity during execution is large. SketchLS determines the search policy at pre-computation time, and performs the search at runtime based on the policy. Thus, there are fewer redundant searches compared to the previous methods above, and the computation during execution is smaller.

During pre-computation, SketchLS finds the shortest path from each node on the graph to each set of seed nodes, and each node stores an array of the shortest paths obtained from pre-computation. This array of shortest paths is called Sketch [3] [6]. During runtime, it reads Sketch from a specified terminal and runs the SketchLS algorithm on Sketch to obtain an approximate minimum Steiner tree.

There is also an enumeration method [10] that, for a given terminal, computes and finds all the minimum Steiner trees that exist on the graph. If we can find the sum of the centrality indices of the Steiner nodes of all the minimum Steiner trees, and identify the minimum Steiner tree whose sum is the smallest using this enumeration method, the goal of this research can be achieved. However, the enumeration of multiple minimum Steiner trees is much more computationally intensive than for a single minimum Steiner tree, and is thus impractical.

## III. Model

### A. SketchLS

*1) Sketch:* SketchLS generates a pre-computed set of shortest paths called Sketch for each node in the graph. Sketch is generated according to the following procedure.

1) Randomly sample the sets of seed nodes $S_1, S_2, ..., S_m$ whose sizes are $1, 2, ..., 2^{m-1}$ from the graph
2) For each set of seed nodes $S_i$, generate a shortest path tree $SPT_i$ starting from a node in $S_i$
3) Obtain the shortest path $sp_i(v)$ between each node $v$ and $S_i$ from the shortest path tree $SPT_i$, and store it as an array

Note that $m = \log|V|$. The outcome of SketchLS is Sketch($v$)=$[sp_1(v), sp_2(v), ..., sp_m(v)]$.

*2) SketchLS Algorithm:* SketchLS obtains an approximate minimum Steiner tree by running the SketchLS algorithm on the terminal set $Q$. It first reads the Sketch for each terminal, which is treated as a tree rooted at the terminal $q_i$, and subsequently performs breadth-first search. Results are then stored in the array $BFS_i$. The following operations are performed for each $BFS_i$:

1) Fetch next element $v$ of $BFS_i$, and add it to array $F_i$
2) If $v$'s neighbor node $n$ is contained in $F_j (j \neq i)$ and the addition of the path $path = (q_i, ..., v, n, ..., q_j)$ does not form a cycle in tree $T$, add $path$ to tree $T$
3) Remove $BFS_i$ from array $BFS$ if there is no next element of $BFS_i$
4) If tree $T$ contains all terminals, return tree $T$ as the Steiner tree and terminate

This is performed in a round-robin manner. In other words, when the operation on the last element of $BFS$ is finished, it moves to the operation on the first element of $BFS$.

### B. Approximation of Minimal Steiner Tree Avoiding High-Centrality Nodes

*1) Generation of Sketch Avoiding High-Centrality Nodes:* Note that our aim is to generate Sketches that avoid top $x\%$ nodes with high-centrality. We thus need to determine a combination of high-centrality nodes to be avoided. First, we determine the sets of high-centrality nodes $A_0, A_1, A_2, ..., A_n$ with increasing size. We then specify one set of high-centrality nodes $A_j$. For the specified $A_j$, Sketch$_j$ that avoids the high-centrality nodes up to the size of $|A_j|$ is generated by the following procedure:

1) Randomly sample the sets of seed nodes $S_1, S_2, ..., S_m$ with sizes $1, 2, ..., 2^{m-1}$ from the graph
2) For each set of seed nodes $S_i$ and for each set of high-centrality nodes $A_l$, find a shortest path tree $SPT_{il}$ starting from nodes in $S_i$ while avoiding nodes in $A_l$
3) The path $p_i(v)$ from each node $v$ to each set of seed nodes $S_i$ is determined by fetching the routes sequentially from $SPT_{i0}$ to $SPT_{il}$. If the route cannot be obtained, set the route most recently obtained as $p_i(v)$. If the route can be obtained from $SPT_{ij}$, set the route as $p_i(v)$.

where $A_0$ is the set of seed nodes with size 0, and $SPT_{i0}$ is the shortest path tree. The size of $A_n$ is $x\%$ of the total number of nodes. As a result, we end up obtaining Sketch$_j(v) = [p_1(v), p_2(v), ..., p_m(v)]$ that avoids up to $|A_j|$ high-centrality nodes at most. Also, we generate Sketch for each possible number of nodes in the sets, *i.e.*, Sketch$_0$, Sketch$_1$, ..., Sketch$_n$. We modify Dijkstra's algorithm to obtain $SPT_{il}$ as described in Algorithm1.

The array $prev$ obtained by Algorithm1 records the previously visited node in the breadth-first search starting from the source node set $S_i$. By going back to the node visited before the specified node, the shortest path to the source node set $S_i$, avoiding the set of high-centrality nodes $A_l$, is obtained. In

**Algorithm 1** modified-Dijkstra

**Require:** $G, S_i, A_l$
**Ensure:** $prev$ -the list storing previous nodes

```
n ← |V|
dist ← new vector(n)
prev ← new vector(n)
que ← new queue
for v ∈ V do
    dist[v] ← INF
    prev[v] ← -2
end for
for s ∈ S_i do
    dist[s] ← 0
    prev[s] ← −1
    add s to que
end for
while !que.empty do
    from ← que.pop()
    for to ← Neighbor(from) do
        if dist[to] ≠ INF then
            continue
        end if
        if to in A_l then
            continue
        end if
        dist[to] ← dist[from] + 1
        prev[to] ← from
        que.push(to)
    end for
end while
return prev
```

other words, $prev$ is $SPT_{il}$. However, since we avoid the sets of high-centrality nodes, there may be nodes that cannot be reached from the source node set. Thus, we set $p_i(v)$ as the shortest path that avoids the largest possible number of high-centrality nodes under $A_l$. In case no high-centrality nodes are avoided, $p_i(v)$ is the shortest path.

*2) Selecting Combination of High-Centrality Nodes to Avoid:* We examine 4 simple schemes to select the sets of high-centrality nodes to be avoided with increasing size, $A_0, A_1, ..., A_n$. Table I describes how each of the 4 schemes increases the size of the combination, where $x$ is a number of nodes to avoid. The number of required shortest path trees in the last column is based on the number of shortest path trees required to generate the original Sketch. Since the pre-computation time increases as the number of required shortest path trees increases, it is the longest for *increment* and the shortest for *all or nothing*.

Intuitively, the probability of obtaining the shortest path while avoiding high-centrality nodes is the highest for *increment*, and the lowest for *all or nothing*. Let us consider a scenario shown in Figure 3. Suppose a shortest path from an arbitrary node $v$ to an arbitrary set of seed nodes $S_i$ cannot be obtained when $x$ high-centrality nodes are to be avoided, but can be obtained if $x-1$ high-centrality nodes are avoided. In this case, using *all or nothing*, the shortest path avoiding 0 high-centrality nodes, i.e, the usual shortest route, is obtained, while using *increment*, it is possible to obtain the shortest path avoiding $x-1$ high-centrality nodes. *Increment*, however, does not turn out to be the most effective scheme as will be shown in our experiments.

TABLE I: Proposed combination schemes

| scheme | node selection | # shortest-path trees |
|---|---|---|
| increment | increment by 1 | $\times(x+2)$ |
| add | add 10% of $x$ | $\times 11$ |
| multiply | double the number (1 follows 0) | $\times(\lfloor \log_2 x \rfloor + 3)$ |
| all or nothing | 0 or $x$ | $\times 2$ |

## IV. Experiments

### A. Setting

Extending a minimum Steiner tree computation method SketchLS, we propose a computational method for approximate minimum Steiner trees that avoids high-centrality nodes, and further examine 4 basic schemes of selecting the combination of high-centrality nodes to be avoided.

In order to investigate how to select combinations that are likely to be unknown to the user, while maintaining a high degree of relevance to the keywords of interest, we compare the size of the Steiner tree based on the minimum Steiner tree obtained by SketchLS against the sum of the centrality indices of the Steiner nodes. In addition, in order to investigate how to select a combination that minimizes the overhead in pre-computation time, we also compare the size of the minimum Steiner tree against the pre-computation time.

We used musae-facebook [15], which is a web graph that connects Facebook pages that "like" each other, and ego-facebook [13], consisting of friends lists, both from Stanford Large Network Dataset Collection [12]. musae-facebook contains 22,470 nodes and 171,002 edges, while ego-facebook contains 4,039 nodes and 88,234 edges. Both are undirected graphs. We experiment with 1,000 sets of randomly sampled terminals, and report the average values. We performed our experiments in Ubuntu 20.04.3 LTS with Intel(R) Xeon(R) CPU E5-2643 v2 @ 3.50GHz, with 94GiB RAM.

Let us first consider the ratio of the sum of the centrality indices of the Steiner nodes with respect to the size of the Steiner tree as the number of terminals changes. We initially varied the number of terminals ranging from 3 to 7. From Figure 4a and Figure 4b, it can be seen that, regardless of whether we avoid the high-centrality nodes in terms of the degree centrality or betweenness centrality, the overall trend of the ratio of the sum of the centrality indices decreasing is not significantly affected by the number of terminals. We thus fix the number of terminals to 5 in our subsequent evaluations.

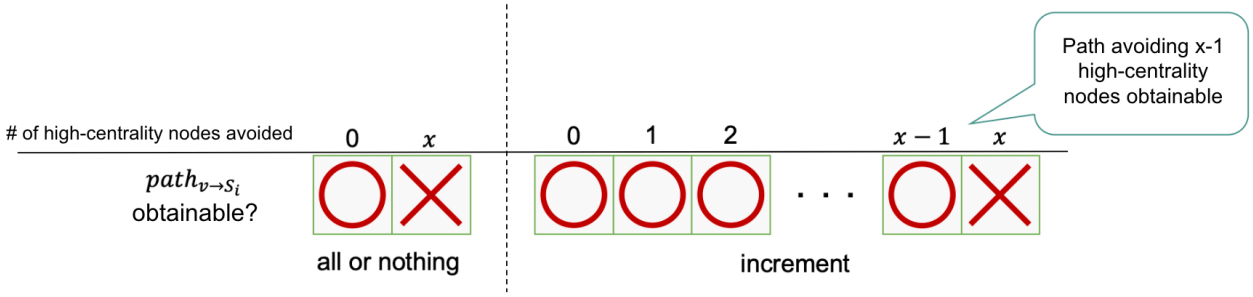
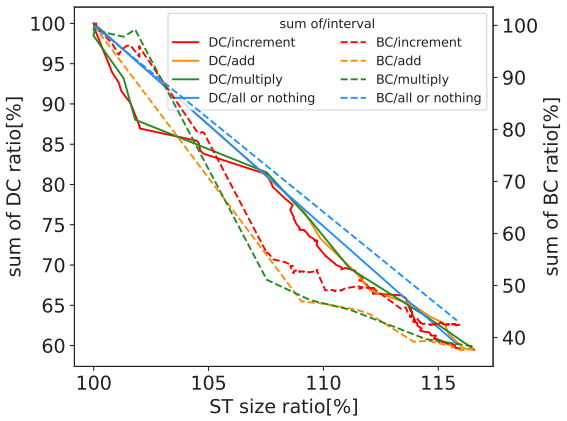
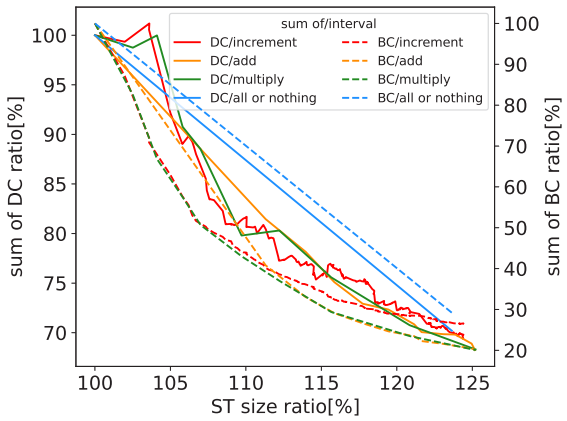



Fig. 3: An example of obtaining the shortest path avoiding $x - 1$ high-centrality nodes




(a) Comparison of terminals when avoiding high-centrality nodes based on degree centrality.




(b) Comparison of terminals when avoiding high-centrality nodes based on betweenness centrality.

Fig. 4: Comparison of terminals.

### B. Results

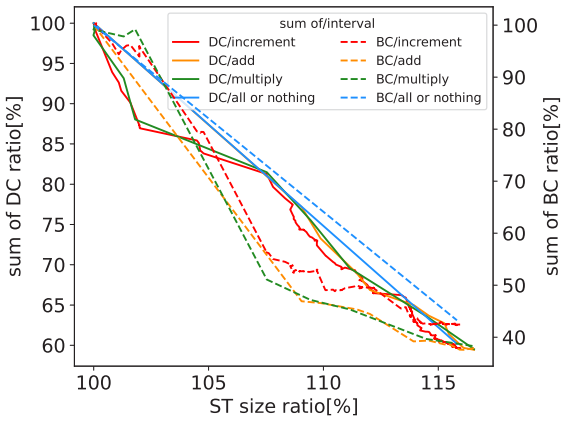
We plot our results in Figure 5a and Figure 5b. Vertical axes indicate the sum of degree centrality indices (left), and the begween centrality indices (right). It can be seen that, regardless of the centrality metric, the ratio of the sums with higher degree centrality decreases and is not strongly dependent on selection scheme. Yet, it can be seen that the ratio of total betweenness centrality can be reduced by about 5% or more with *add* and *multiply* than with *increment* and *all or nothing*. This suggests that *add* and *multiply* may be more appropriate ways to select a combination that increases the possibility of the information obtained from the Steiner tree being unknown to the user, while maintaining a high degree of relevance to the keyword.

From Figure 6a and Figure 6b, it can be seen that, regardless of whether we avoid the high-centrality nodes in terms of the degree centrality or betweenness centrality, the pre-computation time gets longer in the order of *increment*, *add*, *multiply*, *all or nothing*. Comparing *add* and *multiply*, which were superior in terms of centrality, *multiply* is about 300ms shorter, or about 0.8 times faster.

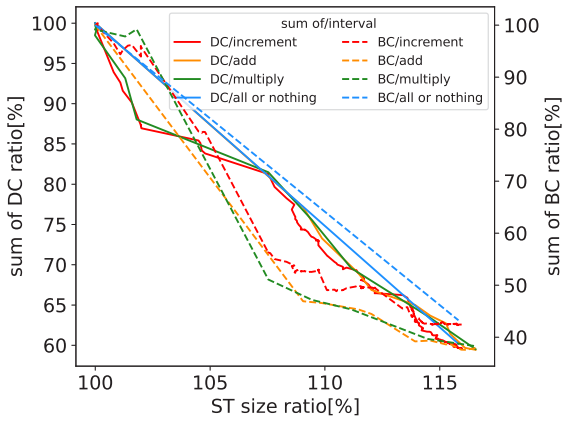
In summary, of the selection schemes we examined, *add* and *multiply* turn out to be reliable schemes for selecting a combination that increases the probability of obtaining unknown information from the Steiner tree, while maintaining a high degree of relevance to the keyword. Also, *multiply* demonstrated faster pre-computation time than *add*. Summing up the results from different perspectives, including the degree of relevance, the probability of obtaining unknown information, and the pre-computation time, *multiply* turns out to be the optimal choice out of the selection schemes examined in this paper.

## V. Conclusion

In this paper, in order to increase the probability of presenting unknown and thus more useful information to the user, while maintaining a high degree of relevance to the keyword, we proposed a computation method for Steiner trees, where nodes with high centrality are avoided as much as possible. We also examined different schemes to select combinations of high-centrality nodes to be avoided, and evaluated them from the viewpoints of the degree of relevance to the keywords, the probability of being unknown, and the pre-computation time. Of the proposed selection schemes, *add* and *multiply* reduced the ratio of the total betweenness centrality of the Steiner nodes by about 5% or more within the range of about 1.25 times the size of the Steiner tree, demonstrating that they can select a combination that increases the possibility of the information obtained from the Steiner tree being unknown to the user, while maintaining a high degree of relevance to the keyword.

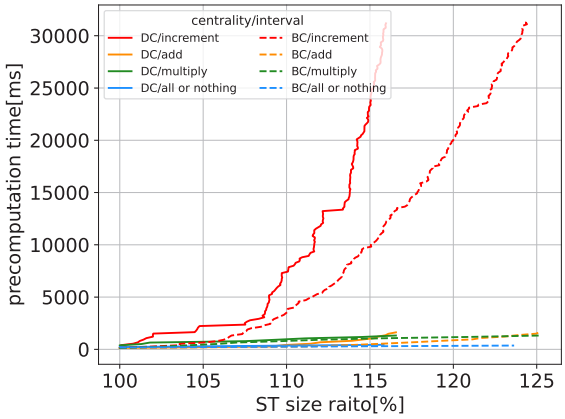



(a) Comparison of selection schemes based on degree centrality.

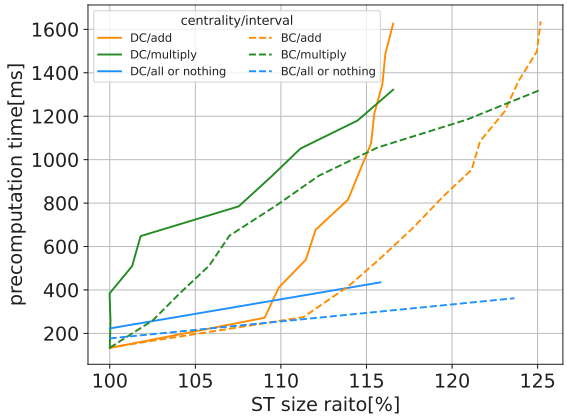



(b) Comparison of selection schemes based on betweenness centrality.

Fig. 5: Comparison of selection schemes.




(a) Comparison of *increment*, *add*, *multiply*, *all or nothing*




(b) Comparison of *add*, *multiply*, *all or nothing*

Fig. 6: Comparison of selection schemes.

## References


[1] Jac M. Anthonisse. The rush in a directed graph. *Journal of Computational Physics*, pages 1–10, 1971.

[2] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *Proceedings 18th International Conference on Data Engineering*, pages 431–440, 2002.

[3] Atish Das Sarma, Sreenivas Gollapudi, Marc Najork, and Rina Panigrahy. A sketch-based distance oracle for web-scale graphs. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, page 401–410, New York, NY, USA, 2010. Association for Computing Machinery.

[4] Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.

[5] Linton C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1(3):215–239, 1978.

[6] Andrey Gubichev, Srikanta Bedathur, Stephan Seufert, and Gerhard 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, page 499–508, New York, NY, USA, 2010. Association for Computing Machinery.

[7] Andrey Gubichev and Thomas Neumann. Fast approximation of steiner trees in large graphs. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, page 1497–1501, New York, NY, USA, 2012. Association for Computing Machinery.

[8] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S. Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. Bidirectional expansion for keyword search on graph databases. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, page 505–516. VLDB Endowment, 2005.

[9] Gjergji Kasneci, Maya Ramanath, Mauro Sozio, Fabian M. Suchanek, and Gerhard Weikum. Star: Steiner-tree approximation in relationship graphs. In *2009 IEEE 25th International Conference on Data Engineering*, pages 868–879, 2009.

[10] Yasuaki Kobayashi, Kazuhiro Kurita, and Kunihiro Wasa. Linear-delay enumeration for minimal steiner problems, 2020.

[11] Vito Latora and Massimo Marchiori. A measure of centrality based on the network efficiency, 2004.

[12] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection, June 2014.

[13] Julian McAuley and Jure Leskovec. Learning to discover social circles in ego networks. In *NIPS*, 2012.

[14] JUHANI NIEMINEN. On the centrality in a graph. *Scandinavian Journal of Psychology*, 15(1):332–336, 1974.

[15] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding, 2019.