UNIVERSITY OF CALIFORNIA,
IRVINE

Computing Betweenness Centrality of Large Colored Graphs based on Data Reduction and Decomposition with a Case Study on Breast Cancer Analytics

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Engineering

by

Chung-Hsien(Bryan) Chou

Dissertation Committee:
Professor Phillip Sheu, Chair
Professor Jean-Luc Gaudiot
Professor Nader Bagherzadeh

2019

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I want to thank my advisor Phillip Sheu. I have been very grateful since the day he chose me to be his student and decided to train me several years ago. It has been an honor to be his Ph.D. student. In the years of my Ph.D. studies, Professor Sheu has taught me a lot, not only academic learning, but also social and ethical competencies. His enthusiasm for research motivates me, and it makes my Ph.D. experience productive and stimulating. I appreciate all the time and ideas he contributes to my dissertation. Words cannot express how thankful I am to have such a supportive advisor.

I am thankful to my labmates. As my senior, Shao-Ting Wang guided me into this field, helped me getting familiar with being a teaching assistant/research assistant, provided me my research ideas, and also set a good example of a PhD researcher. As two of my good friends in the lab, Kyle Li and Jimmy Shih provided valuable ideas on my theorems, and they helped me a lot on polishing my dissertation with constructive suggestions. As a cooperator of our Lab, I am also thankful for the supports from NEC Solution Innovators, Ltd., Japan. Mr. Hiroyuki Shigematsu and Mr. Atsushi Kitazawa gave me many useful suggestions for my research directions. Mr. Masahiro Hayakawa helped me polish my dissertations. I appreciate all their help.

For this dissertation I would also like to thank my committee members: Professor Jean-Luc Gaudiot and Professor Nader Bagherzadeh. With their expert knowledge in this area, they provided many useful comments so that I could refine my dissertation, and they also enlightened me on some future research directions.

In addition, I want to thank the Graduate Coordinator in EECS, Amy Pham, who always helps me with warmth and passion. Without her assistance, I could not be here.

Finally, I would like to thank my father, my mother, my sister, my friends in the Taiwanese society, and especially, Chee-Hann. Without your support and encouragement every day and night, I would not have been able to make this achievement. Thank you.

Bryan Chou
University of California, Irvine
2019

CURRICULUM VITAE

Chung-Hsien (Bryan) Chou

2010	B.S. in Electrical Engineering, National Cheng Kung University, Taiwan
2012	M.S. in Electrical and Control Engineering, National Chiao-Tung University, Taiwan
2012-2013	Military Service, Armored Command Headquarter, Army
2013-2015	Engineer, TSMC Ltd., Taiwan
2015-2016	Software Development Engineer, HTC Co.
2016-2019	Teaching Assistant, University of California, Irvine
2019	Ph.D. in Computer Engineering, University of California, Irvine

FIELD OF STUDY

Decomposition Computation of Combinatorial Algorithms

PUBLICATIONS

Chung-Hsien Chou, et al. “GOLAP: Graph-Based Online Analytical Processing.” International Journal of Semantic Computing, 2018.

ABSTRACT OF THE DISSERTATION

Computing Betweenness Centrality of Large Colored Graphs based on Data Reduction and Decomposition with a Case Study on Breast Cancer Analytics

By

Chung-Hsien(Bryan) Chou

Doctor of Philosophy in Computer Engineering

University of California, Irvine, 2019

Professor Phillip Sheu, Chair

Graph theory has been widely applied to the studies in biomedicine, and graph structural analytics, such as betweenness centrality, has played an important role in finding the most central vertices in graph data. Hence, betweenness centrality has been heavily applied to discover the most important genes with respect to multiple diseases in biomedicine research. However, if the network size is too large, the result of betweenness centrality would be difficult to obtain in a reasonable amount of time. In this research, we describe two complementary approaches to computing betweenness centrality on large graphs: graph decomposition and graph reduction. Graph decomposition for betweenness centrality speeds up the conventional computational approach by cutting the original graph into several subgraphs and computing the global betweenness measures by summing up the results from all the subgraphs. For graph reduction, our hypothesis is that if we allow approximate results, and if the difference between the approximate and the precise results is bounded, a graph may be reduced to a smaller scale to speed up the computation. We use

a co-expression network of breast cancer as a case study to prove our hypothesis for graph decomposition and graph reduction.

Additionally, we may investigate the betweenness centrality of each colored subgraph composed of a specific color, considering color as a property of graph data to represent different categories for the nodes and edges in the graph. However, as investigators may be interested in querying betweenness centrality on multiple combinations of the colored subgraphs, the total execution time on all the subgraphs may be excessively long, considering all the possible combinations. Therefore, we propose another approach to computing betweenness centrality by incorporating node colors and edge colors. We propose that the node with the highest betweenness centrality can be computed for a very large and colored graph by decomposing the graph into colored subgraphs and merging the result from the base cases. Furthermore, we compare our approach with the conventional approaches in the experiments section, and we demonstrate that our scalable approach is more efficient when finding the global backbone node with the highest betweenness centrality.

CHAPTER 1 Introduction

Graph theory has been applied to biomedical researches in many ways. Betweenness centrality has been an important structural analytic on genetic networks, since betweenness centrality is one of the approaches to discover the most important nodes which are considered to be more "central" in a graph data network. For example, scientists have applied betweenness to search for the most important genes in a genetic network [1-3, 4-6]. In GOLAP (graph based OLAP) [7], we extend the space of graph analytics by the incorporation of node types and edge types using colored graphs. In this research we extend the definition of betweenness centrality in colored graphs and propose that betweenness centrality can be computed for a very large and colored graph by decomposing the graph into subgraphs. By obtaining the subgraph-level backbone nodes, we can aggregate the subgraphs to obtain the global backbone node with the highest betweenness centrality.

When detecting the most central genes among all the human disease genes, researchers apply structural analytics such as betweenness centrality [8]. In order to discover the most important genes in the context of multiple diseases, it is likely that researchers have to extensively execute betweenness centrality queries on aggregated graphs composed of different combinations of disease subgraphs. For example, in Sun's work, since there are five main disease categories (cancer, cardiovascular disease, immune system disease, metabolic disease, and nervous system disease) in the disease-gene network, there are as many as $2^5 = 32$ total combinations of graphs. In this thesis, we propose a scalable approach which reduces the overall computational time of the 32 total combinations of disease. We compute the bases cases where all the genes in a base case are related to one disease. We then find the most important genes globally in the whole graph by merging the base cases. Another contribution of this research is that we can partition each colored

subgraph into multiple smaller subgraphs, and hierarchically apply the same approach to obtain the most important genes.

The concept of the proposed scalable approach is a new idea for betweenness centrality computation. It is based on the concept of groups in a graph, in which all nodes or edges share the same label/color so that they can be identified as in the same category. Thus, our approach conquers the base cases which include the betweenness centrality results in each group, and then it aggregates the base cases to find the global solution. In the past, some research propose to find the group-level betweenness centrality for different groups, as if each group is an individual node. However, their work does not address the betweenness centrality of each individual node; as a result, they cannot answer queries such as "Among all the groups, which node is the most important node in the whole graph." One variation of group-based betweenness centrality is developed by Brandes [9]. Based on the group betweenness centrality, they identify the importance of each group in a graph. Another related research, done by Everett [10], is group-level betweenness centrality. They find a betweenness measure for a group based on the betweenness of each node in a group, so they can answer category-based queries such as "How important the group of breast cancer is in the GEO dataset integrated by all human diseases." In addition, Kolaczyk et.al [11] propose another new concept of group betweenness centrality, called co-betweenness, which is based on computing the shortest paths of pair of consecutive nodes instead of every single node. Using co-betweenness, instead of finding the most central individual nodes, they can find the nodes which are important for relaying information and for information control flow. The applications of group betweenness centrality are usually more interested in finding the small groups (i.e., groups with a smaller number of nodes) with higher betweenness values, instead of merely finding the more central group [9]. Different from these related works, instead of finding the group-level

betweenness centrality, our research focuses on finding the global betweenness centrality by decomposing a graph into subgraphs.

In this research, we also apply data reduction to speed up graph algorithms with better execution time and better performance, as data reduction [12] is useful for many applications that involve big data. Existing approaches to graph data reduction may be syntactic or semantic, which may be lossless [13-15] or lossy [16-17]. The difference between syntactical graph reduction and semantical data reduction is that traditional methods are syntactic and rely on data compression, which means they focus on graph structures by serialization or redundancy removal. On the other hand, semantic graph data reduction is query-based that can be applied on top of syntactic data compression to provide additional data reduction. For example, query-based graph data reduction may further reduce a graph by preserving only the information relevant to the queries needed by an application.

Our hypothesis is that further reduction may be possible even on top of semantic reductions. Sometimes the accuracy of query results may not be strictly required (or expected such as with a search engine). Specifically, if we allow approximate results, where the difference between the approximate results and the precise results is bounded, a graph may be further reduced. In the past, to our knowledge, the only work related to this approach is [18] where Bader et al. propose an approximate approach to enhance the performance of betweenness centrality. They speed up the performance by reducing computation for some irrelevant single-source shortest paths. However, based on different graph data, the error rate varies from 10% to 46.7%, which could be considered too high. In this study, we use a co-expression network of breast cancer as a case study to prove our hypothesis. We can demonstrate that our approach not only speeds up the total computational time since we not only reduce the graph size, but also maintains the accuracy with an acceptable

error rate bounded by 10%. Unfortunately, in [18] the authors do not show the exact speedup result, but they only show the percentage of shortest paths that were computed versus shortest paths sampled/ all shortest paths. Our dataset is much larger than theirs (n=17,055 and m=4,109,069 >> n=9,914 and m=36,854), and our betweenness reduction has 100% accuracy which is better than 89.69% as reported in [18].

When a graph becomes very large, even graph reduction will be limited. A graph has to be decomposed so that subgraphs may be processed in batches or in parallel. For most applications, unfortunately, parallel computing remains to be an expensive option. Instead Tang et al. [19] propose a batch processing method that does not acquire parallel computing, but still can speed up the shortest path algorithm on large graphs. The idea of graph decomposition is that a large graph can be decomposed into several smaller subgraphs, so that the individual results could be obtained by running algorithms on each subgraph. The overall result can be computed by aggregating all the individual results on subgraphs. Our hypothesis is that, unlike Tang's approach, we do not need to compute the actual betweenness measures if our interest is to discover the most critical nodes. A simpler approach with a lower computational cost can be taken.

CHAPTER 2 Existing Work

In this section, we summarize the computational problems associated with betweenness centrality. We also discuss the methods we employ to reduce graphs.

2.1. Betweenness Centrality

Betweenness centrality is one of the commonly used structural measure when finding the most important genes in a given genetic graph; and usually the genes found are considered as the backbone and basis that represent a genetic graph [3]. For example, to target the most influential

genes in a cancer-gene graph, finding the betweenness centrality is a common approach for biologists, and the genes with a higher rank could be considered as the most influential genes.

Betweenness centrality has been computed based on the shortest path algorithm [20]. Although the algorithm proposed by Brandes [21] has a polynomial-time complexity which is $O(mn+n^2\log n)$, the computational time is still considered expensive, especially when the graph data is huge. Therefore, scientists have tried to find approaches that can reduce the computational time for betweenness centrality. Bader et al. propose a parallel algorithm for centrality metrics [22], and they also propose an adaptive sampling approximation approach to speed up the execution time of betweenness centrality [18]. In addition, Brandes et al. [23] present another heuristic to compute betweenness centrality. They experiment with different strategies, and find that a random selection of the source nodes has better accuracy than a deterministic strategy. Most of these previous works have applied sampling technique to reduce exhaustive execution on traversing the nodes which are relatively irrelevant. But these works face the same trade-off – to obtain a better performance or to sacrifice the accuracy of betweenness centrality. However, in our approach, we provide a reduction method which has a better performance without sacrificing the accuracy.

2.2. Computing Betweenness Centrality

As discussed in Section 2.1, betweenness centrality is one common approach used to discover the most important nodes in a graph. Given a graph *G*, betweenness centrality is computed by applying an all-pair shortest path algorithm to find the nodes that have the most shortest paths passing through. In this research, without losing generality we assume one variation of the Dijkstra algorithm that includes both endpoints in each path [24] is applied to compute betweenness centrality. The procedure is summarized as follows:

1. Obtain all-pair shortest paths. Find shortest paths from each node to all the other nodes. Repeat this step *n* times so that every node takes turns to be the source.
2. After all-pair shortest paths are obtained, compute the number of shortest paths pass each node. The value is the betweenness centrality of the node.

For example, in the directed graph shown in Figure 1, nodes represent genes (e.g., P1, P2, etc), and edges represent diseases such that two nodes are connected if they are both related to a disease and the weight carried by an edge represents the influence factor. To find the most important gene we can find the node that has the highest betweenness centrality value.

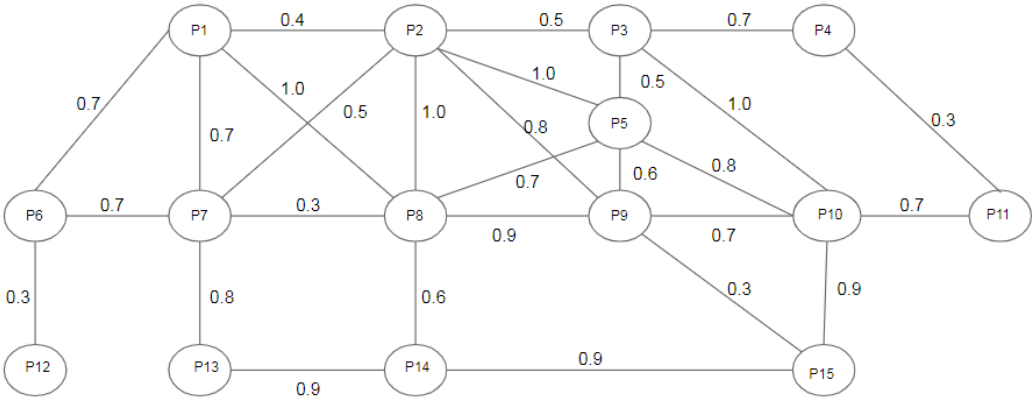


Figure 1. A graph example

CHAPTER 3 Scalable Betweenness Centrality

When the size of a graph becomes very large, the computational time of betweenness centrality becomes longer, and worse, it may not be feasible to compute the betweenness centrality due to the memory limitation. One obvious approach for solving the problem is to apply batch processing. In the past, Tang et al. [19] propose a method to compute the shortest paths using batch processing. The idea of the approach is that the original graph can be manually separated into several subgraphs, so that the all-pair shortest path algorithm can be applied individually first. By aggregating the findings of each subgraph, it obtains the correct results from all the parts.

Specifically, in every iteration, the source is "relaxed" as much as it can. Then, every subgraph initiates the shortest path algorithm. When a node is relaxed to a node in another subgraph, it will initiate the subgraph if it has not been initiated before, or it will trigger a series of updates on the ancestor nodes in the graph if the shortest distance needs to be updated. Overall, Tang's batch processing approach guarantees actual betweenness centrality measures because all pairs of shortest paths in the graph are found completely.

Different from Tang's approach, we propose that the backbone node with the highest global betweenness centrality can be computed without finding the accurate overall result of shortest paths across multiple subgraphs. Instead, we only need the betweenness centrality of each subgraph as if each subgraph is isolated.

Specifically, we can find the candidate nodes which include the backbone node, denoted as BBN, that has the highest betweenness centrality of a graph by integrating the betweenness centralities of its subgraphs. This means in order to find the most important node in the original graph, we do not need to completely compute all-pair shortest paths passing through the nodes in the original graph. Instead, we can separate the original graph into several subgraphs, and then compute the betweenness centralities for every node in each subgraph and find the local BBN in each subgraph. After the local BBNs are found, we propose the following two major steps to compute the global BBN:

1. Add all the potential nodes for the global BBN into a candidate set. These include: 1) the global root, 2) the local BBNs, and 3) the nodes on the shortest paths from the global root to all the local BBNs. All other nodes can be excluded. As illustrated in Figure 2(a), the yellow node is the global root, and the blue nodes are the local BBNs. Figure 2(b) shows

the nodes on the shortest paths (with color yellow) from the global root to all the local BBNs.

2. Compute the global betweenness centrality of each node u in the candidate set mentioned above, using a simple formula, $\text{Btw}(u, G) = \text{In\_Btw}(u, G_k) + \text{Ex\_Btw}(G_k)$, where G is the whole graph and G_k is the subgraph where u exists.

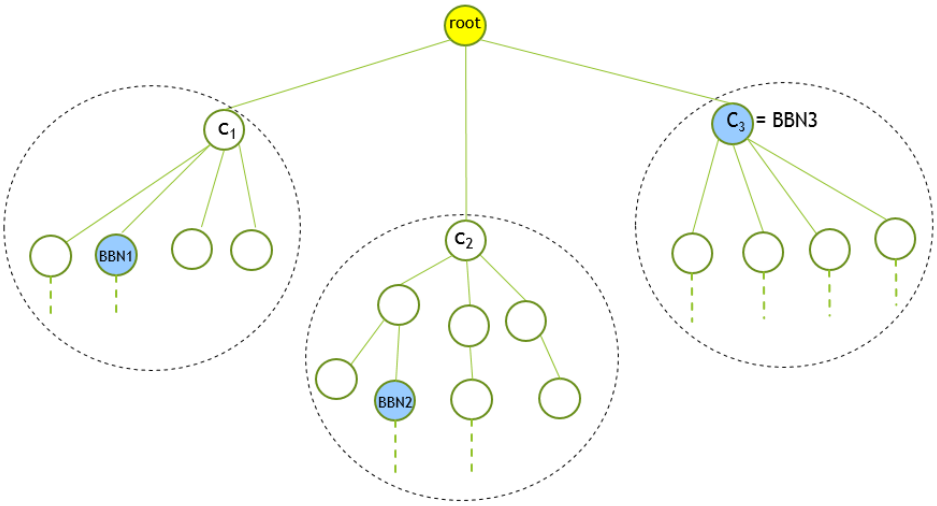


Figure 2.(a). A tree graph G

After we compute the global betweenness of each of the candidate nodes, we can rank the global betweenness centralities of the candidate nodes. Therefore, we can obtain the global BBN with the highest global betweenness without computing the all-pair shortest paths in the graph by decomposing the graph into subgraphs.

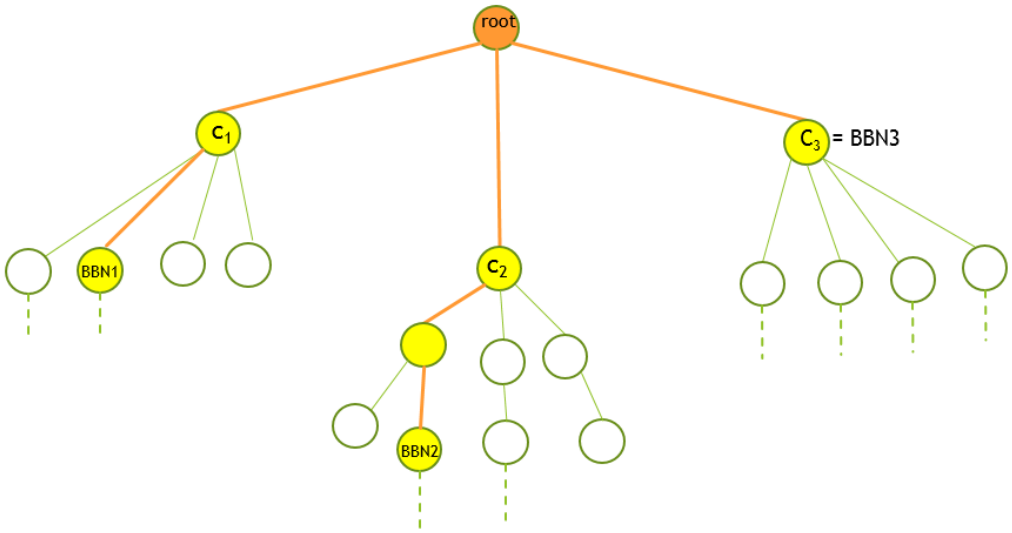


Figure 2(b). Nodes on the shortest paths

3.1. Definitions

Below we present the terms that would be used in our scalable betweenness centrality algorithm (SBC).

- **BBN(*G*):** Backbone node of graph *G*, which is the node that has the highest betweenness centrality in *G*.
- **Btw(*u*, *G*):** Betweenness centrality value of the node *u* in graph *G*.
- **Internal shortest paths:** Shortest paths of which the source node and target node of the path are both in the same subgraph.
- **External shortest paths:** Shortest paths of which the source node and target node of the path are in different subgraphs.
- **In_Btw(*u*, G_k):** The number of internal shortest paths passing through node *u* in G_k.
- **Ex_Btw(G_k):** The number of external shortest paths passing through subgraph G_k.
- **Connecting node in G_k:** The node inside G_K which has outgoing edges connected to any other subgraph.
- ***w*(G_k)**: Number of nodes in G_k.
- **Shortest_path(BBN(G_i), BBN(G_j)):** The shortest path which starts from BBN(G_i) to BBN(G_j).
- **Behind(*u*, *G*):** The set of nodes behind the node *u* in graph *G*, which represents the nodes isolated from graph *G*, if the connecting node *u* is disconnected from *G*. For example, consider the graph *G* shown in Figure 3(a) which is a tree graph. The node c_1 in this

example is the node *u* in the definition. If the edge (root, c_1) in Figure 3(b) is removed from the graph, c_1 will be disconnected from graph *G*. Then, the set of nodes circled in Figure 3(b) will be isolated from the graph and considered as Behind(c_1, *G*).

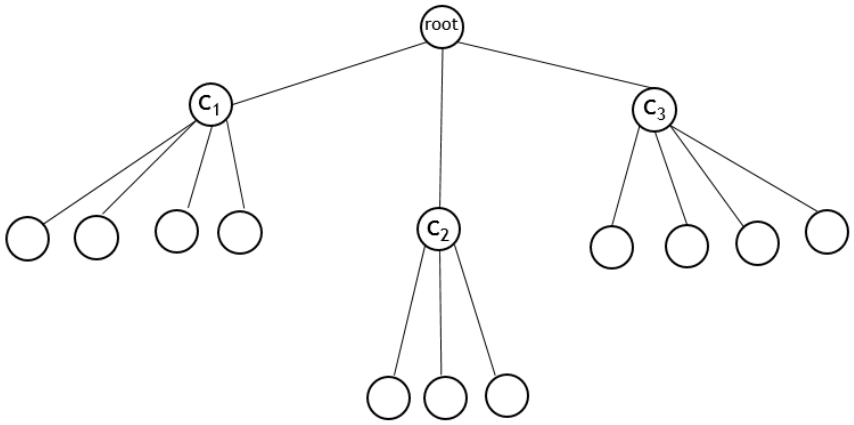


Figure 3(a). Example of Behind nodes

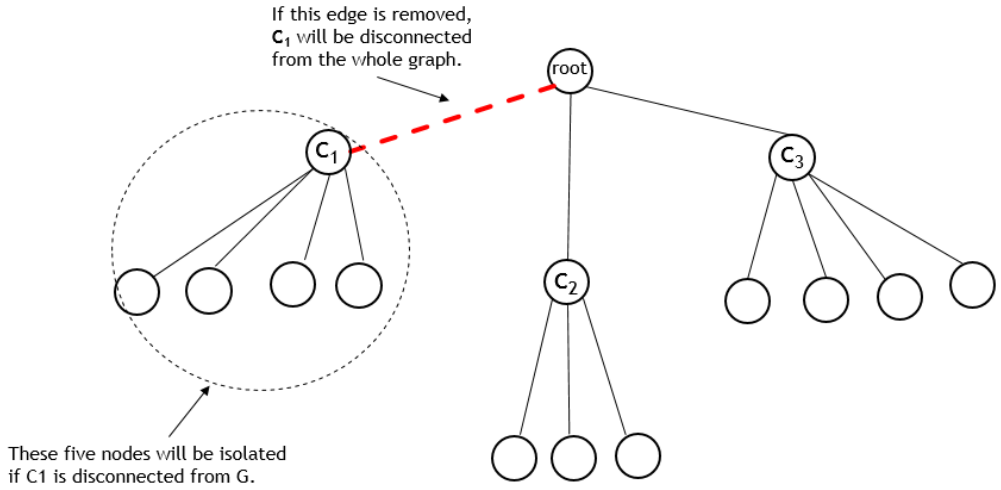


Figure 3(b). If the node c_1 is removed from graph *G*, the set of nodes that will be isolated is the behind(c_1, *G*)

3.2. The SBC Algorithm

The first step of the algorithm discovers all the cycles in the graph and transforms the cycles into abstract coarse nodes. Accordingly, the result graph is a tree structure containing physical nodes and coarse nodes. Although a minimum spanning tree (MST) algorithm [25] finds a tree structure of the graph, we do not apply MST to convert the graph into a tree structure. This is

because MST finds a tree structure which connects all the nodes in the graph with the minimum costs, but MST does not retain the all-pair shortest paths in the original graph, which is the fundamental information for betweenness centrality. Instead, we find the tree structure of a graph by preserving all the cycles and forming them into coarse nodes. Hence, there are several consequences of tree property that would support our approach. One consequence of a tree structure is that the root node is the only connecting node among its subtrees. Since the root node of a tree has edges connected to its subtrees, if there is another connecting node, there will be a cycle introduced. As shown in Figure 4, if the subtree $T(c_3)$ has two connecting nodes, c_3 and u, there will be a cycle formed by the root, c_3 and u. Another consequence is that the shortest path between two nodes is exactly the path between the two nodes because in a tree there is a unique path between any two nodes. Accordingly, the weights of edges in the tree do not affect the computation of betweenness centrality since the shortest path is the only path between every two nodes. However, this approach can still handle weighted graphs because we will examine the nodes inside a coarse node if it is chosen as the global BBN, and the weights inside the coarse node will be applied to the computation of betweenness centrality.

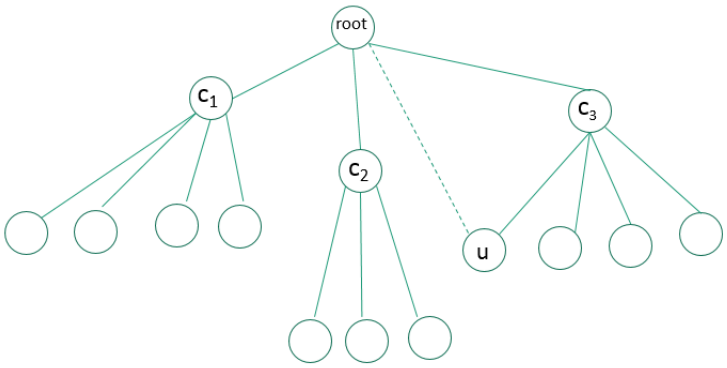


Figure 4. A demonstration of connecting nodes on trees

In this approach, we decompose the graph (tree) into subgraphs (subtrees), and we compute the global betweenness based on the internal betweenness information in each subgraph and the external betweenness across subgraphs. The external betweenness of a subgraph can be computed based on the number of nodes in the subgraph. Consider a graph G which is a tree containing subtrees, G_1, G_2, … G_k as shown in Figure 5, and that the number of nodes in each subgraph is denoted as $w(G_j)$, j = 1 to k. Since a tree is a connected graph, all nodes in graph G are connected. Hence, every node in G_1 can create a path to each node in G_2. As shown in Figure 5, the number of external shortest paths passing between two subgraphs G_1 and G_2 is therefore $w(G_1) * w(G_2)$. Besides, the number of external shortest paths passing between G_1 and the root is $w(G_1) * 1 = w(G_1)$. Therefore, the external betweenness of both subgraphs and the root is $Ex\_Btw(G_1) = w(G_1) * (w(G_2)+1)$, $Ex\_Btw(G_2) = (w(G_1)+1) * w(G_2)$, and $Ex\_Btw(root) = w(G_1) + w(G_2) + w(G_1) * w(G_2)$. Furthermore, we can observe that the external betweenness of a subtree $T(c)$ is proportional to the number of nodes behind the root of the subtree, which is behind(c, $T(c)$).

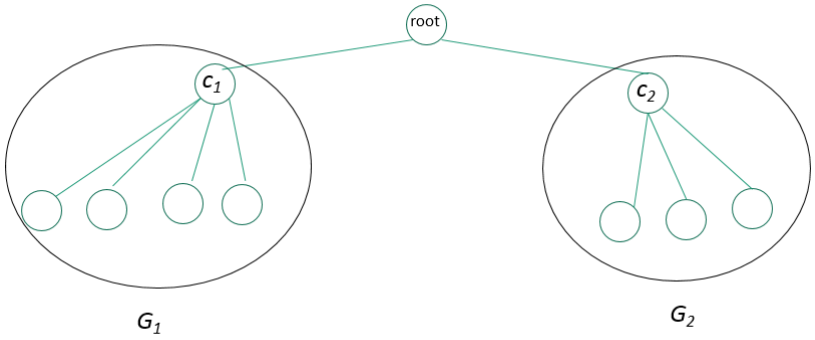


Figure 5. Demonstration of external betweenness

One of our major concepts is that the global betweenness centrality of a tree node can be computed with the sum of internal betweenness and external betweenness using the formula, $Btw(u, G) = In\_Btw(u, G_j) + Ex\_Btw(G_j)$, where u is the only connecting node in the subgraph G_j.

Consider a graph G which is composed of subgraphs, G_1, G_2, *…, and* G_k. The global betweenness is the total number of shortest paths passing through u in G, and these shortest paths can be categorized into two types: 1) internal shortest paths in which both source and target nodes are inside subgraph G_j, and 2) external shortest paths in which neither source nor target is inside G_j, or either one of them is inside G_j. We denote these two types as the first part and the second part below.

1. In_Btw(u, G_j): Since In_Btw(u, G_j) is the number of internal shortest paths passing through u in G_j, it is obvious that In_Btw(u, G_j) is identical to the first part of the Btw(u, G) which is the number of all the internal shortest paths passing through u in G_j.
2. Ex_Btw(G_j): It is the number of all the possible external shortest paths in G_j. Since node u is the only connecting node, all these external paths pass through u.

Therefore, The global betweenness, Btw(u, G) = In_Btw(u, G_j) + Ex_Btw(G_j). Additionally, if G_j is equal to $T(u)$ which is the subtree of node u, then we can replace G_j in the formula with $T(u)$, such that Btw(u, G) = In_Btw(u, G_j) + Ex_Btw($T(u)$).

The other major concept in our approach is that we can prune the graph data to have a smaller set of candidates which can be the global BBN. We start the pruning process by acquiring nodes on the paths starting from the global root to the local BBNs. In addition, these nodes are also the parents or ancestors of the local BBN, which are exemplified by the nodes on the blue paths shown in Figure 6, and node u is the local BBN. Other than the candidate nodes on the paths, nodes on another branch of u's parent or any other ancestors may also have a greater global betweenness than the local BBN, such as node v in the figure. They have a chance to attract more external paths than the local BBN if they have more behind nodes than the local BBN. After we include the nodes with a greater behind set than that of u, we filter out the irrelevant nodes because

they have less internal betweenness and fewer behind nodes than *u*. Furthermore, the nodes which are behind *u* would not have a greater global betweenness centrality than *u*, since 1) they have less internal betweenness than the local BBN and 2) all the external shortest paths coming through these nodes must pass through the local BBN first.

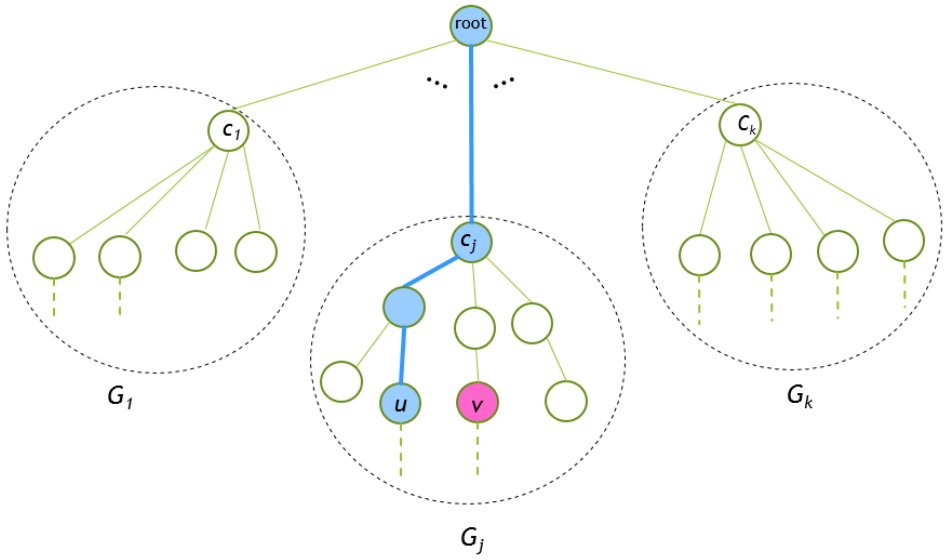


Figure 6. A sample of candidate nodes

Below is a summary of the SBC algorithm:

The SBC Algorithm

Step 1: Apply Depth-first search (DFS) to detect cycles in the graph *G*. Create abstract coarse nodes for cycles. The result graph *G'* is a tree.

Step 2: Add weights *w*(*c*) to coarse nodes, where *c* is a coarse node and *w*(*c*) is the number of physical nodes inside the coarse node.

Step 3: Randomly select a root node which has the highest degree *k* to decompose the graph *G'*. Find *k* subtrees of the global root node as the *k* subgraphs.

Step 4: For each subgraph G_j, j = 1 to k, calculate In_Btw for each node and find the local BBN(G_j). When calculating the internal betweenness for a coarse node, add estimated weight $w(c)*(w(c) - 1)/2$ to the internal betweenness,

$$\text{In\_Btw}(c, G_j)' = \text{In\_Btw}(c, G_j) + w(c)*(w(c) - 1)/2, \quad (1)$$

where c is a coarse node.

Step 5: In order to find the global BBN, add all the candidate nodes into the candidate set. The candidate set includes the following nodes: 1) the global root node, 2) the k local BBNs, 3) all the nodes on the shortest paths from the global root node to all the local BBNs, and 4) the nodes with more behind nodes than that of the local BBN.

Step 6: Compute the global betweenness centrality of each node u in the candidate set, using the formula,

$$\text{Btw}(u, G') = \text{In\_Btw}(u, G_j) + \text{Ex\_Btw}(G_j), \quad (2)$$

where G' is the whole graph and G_j is the subgraph where u exists. Find the global BBN with the highest Betweenness centrality.

3.3. Pruning

The major concept of our approach is to find the global BBN by searching all the candidate nodes and computing their global betweenness. There are three types of candidates: 1) the global root, 2) the local BBNs, and 3) nodes on the paths starting from the root to all the local BBNs. In addition, the global betweenness of the candidates can be computed with the formula, $\text{Btw}(u, G) = \text{In\_Btw}(u, G_k) + \text{Ex\_Btw}(G_k)$, where G is the whole graph and G_k is the subgraph where u exists.

However, from the example in Figure 6, we can observe that a node which does not belong to any type of the candidates mentioned above can possibly have a greater global betweenness than

the local BBN. This situation occurs when the node has a greater set of behind nodes than that of local BBN, so that it can have a higher global betweenness as it has more external paths than the local BBN. Furthermore, there could be multiple abstract coarse nodes representing cycles on the tree graph. To solve this problem we add weights to the internal betweenness of such nodes in order to estimate the maximal internal betweenness of nodes inside the coarse nodes. If a coarse node is decided to be the global BBN, we need to retrieve the betweenness centrality for all the nodes inside the coarse node and rank them by their global betweenness with all the other candidates. The followings are the key steps of the pruning process in the SBC algorithm:

1. When computing the internal betweenness of nodes in each of the subgraphs, we add weights to the internal betweenness for coarse nodes in order to estimate the maximal internal betweenness of nodes inside the coarse nodes. Hence, no potential candidate nodes inside the coarse nodes will be left out.
2. Find the local BBNs after computing the internal betweenness. A local BBN cannot be a coarse node because coarse nodes are abstract. In this case we estimate their internal betweenness. When a coarse node is ranked the highest, find the next physical node with the second highest internal betweenness and include the coarse node into the candidate set.
3. Include more nodes inside the subgraph into the candidate set. If a node has more behind nodes than that of the local BBN, the node can be a candidate. This could be done by running depth-first search. Figure 7 demonstrates searching more nodes from all the nodes on the shortest paths, and the search stops when a node's behind is less than or equal to that of the local BBN.
4. If the global BBN computed is a coarse node, we compute the global betweenness of each node inside the coarse node by summing up their internal betweenness and external

betweenness. We can then rank all the nodes in the candidate set including all the nodes inside the coarse node based on their global betweenness. We do not need to compute the betweenness for the nodes in the other coarse nodes in the candidate set, as they cannot be the global BBN.

In the next sections, we demonstrate the detailed steps of the SBC algorithm with examples, based on the major concept of our approach as well as the pruning process. Additionally, we will discuss and prove the correctness of the SBC algorithm including the pruning process.

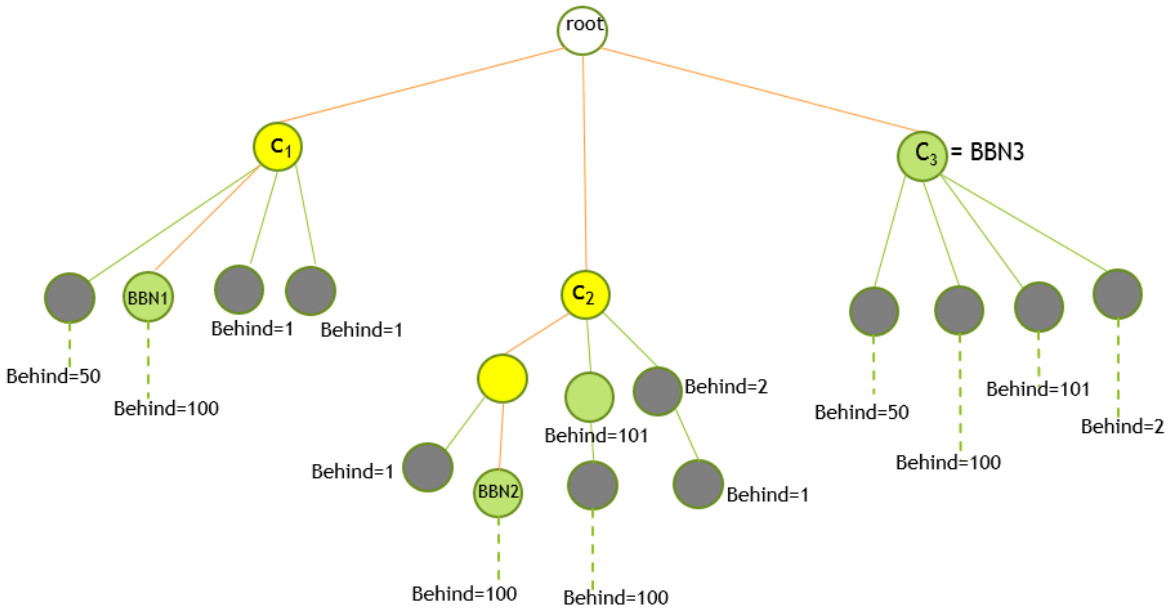


Figure 7. Nodes with more behind nodes in a subgraph

3.4. Examples

In the following, we show how to apply the proposed SBC to find the BBN of a graph. First, as shown in Figure 8, we run DFS from each node to detect cycles in the graph, and each cycle is merged into a course node. The nodes circled in color yellow in Figure 8 are coarse nodes.

After this step, the graph becomes a tree that is free of cycles. We demonstrate three examples below starting from a tree structure. The first example is shown in Figure 9(a). We can decompose

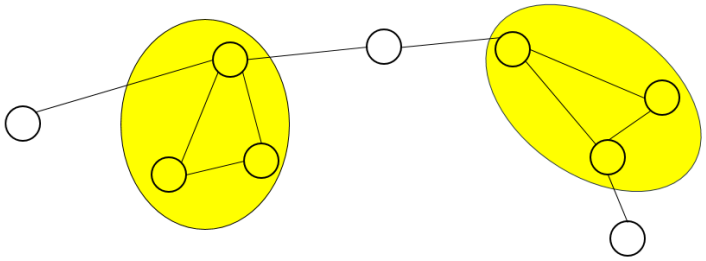


Figure 8. Course nodes representing cycles.

the graph into one root node and subtrees $T(c_1)$, $T(c_2)$, and $T(c_3)$. In Figure 9(b), we partition the graph G into subgraphs which are the root, $T(c_1)$, $T(c_2)$, and $T(c_3)$. We compute the internal betweenness of each node locally in each subgraph, and add the local BBNs into the candidate set. However, when adding the local BBN u, we have to add c_1 also because c_1 is on the shortest path from the global root to u. In the next step, for every candidate, we label its weight to be the number of nodes in the corresponding subgraph, as shown in Figure 9(c), in order to compute external betweenness centrality as shown in Figure 9(d). Especially, for node u, the weight of its subgraph is the weight of its own subtree, which is $w(T(u))$.

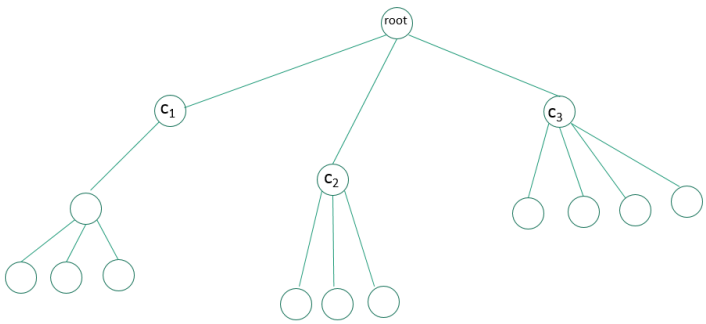


Figure 9(a)

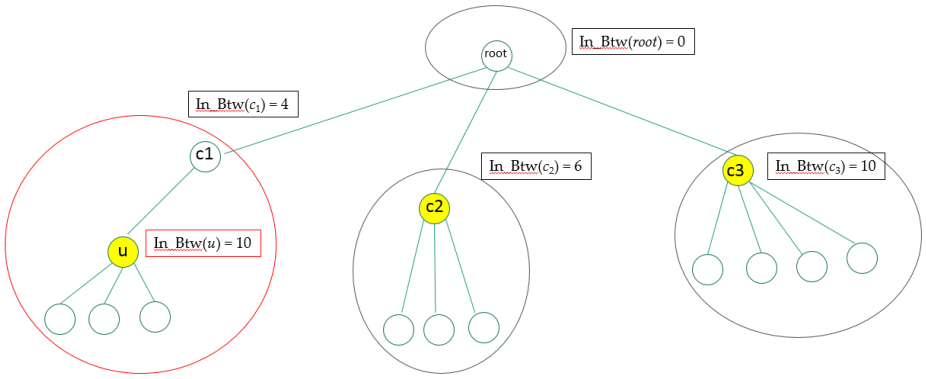


Figure 9(b)

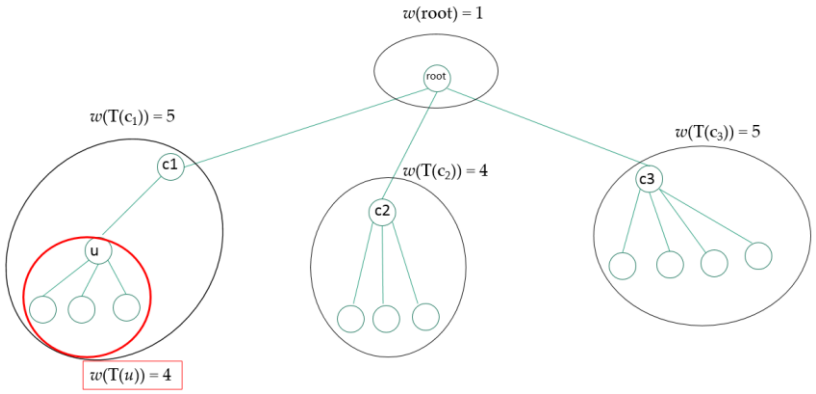


Figure 9(c)

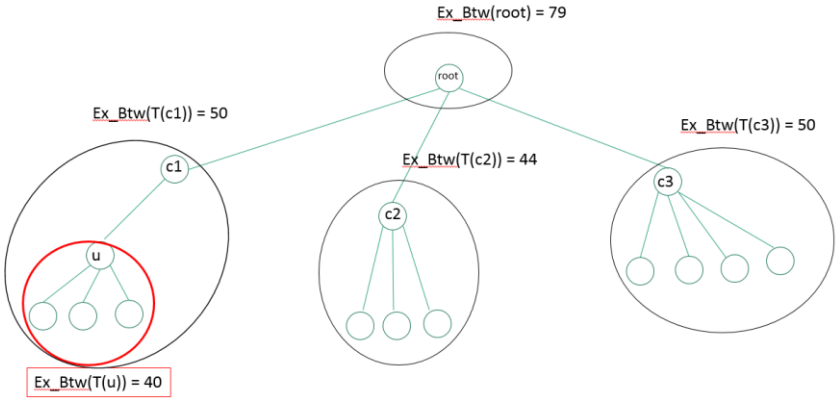


Figure 9(d)

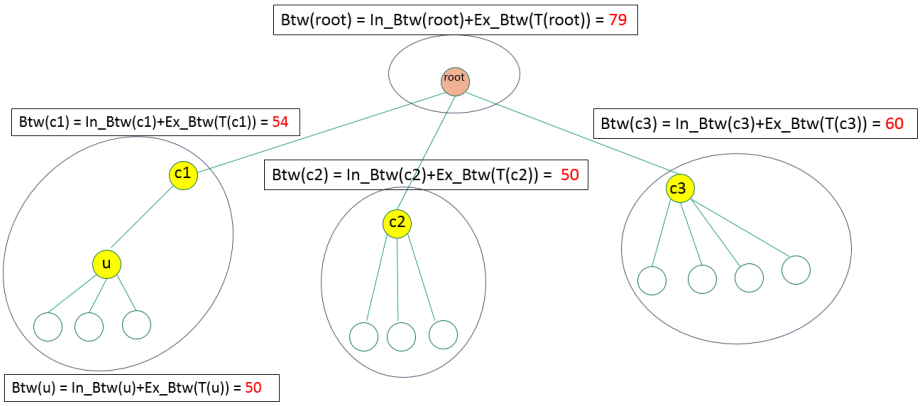


Figure 9(e)

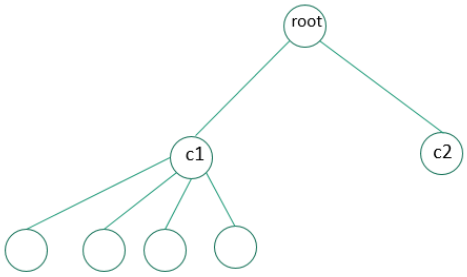
Figure 9. (a)(b)(c)(d)(e) Example 1.

As a result, we can find the global BBN by adding their internal betweenness centrality and external betweenness centrality and finding the one with the highest sum, as shown in Figure 9(e). If a coarse node has the highest BBN, we can drill down and compute the betweenness centrality of each node in the coarse node. In this example, the root node is found to have the highest global betweenness centrality. The detailed values of betweenness centrality and the rank of the nodes are listed in Table 1.

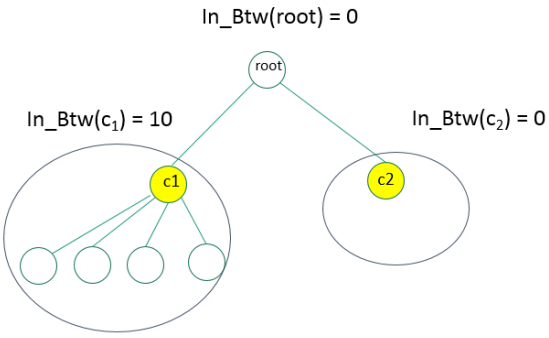
Table 1. Results of betweenness centrality in Figure 9

Candidate Nodes	Internal Betweenness	External Betweenness	Global Betweenness = In_Btw + Ex_Btw	Rank of Betweenness Centrality
node root	0	79	79	1st
node c_1	4	50	54	3rd
node u	10	40	50	4th
node c_2	6	44	50	4th
node c_3	10	50	60	2nd

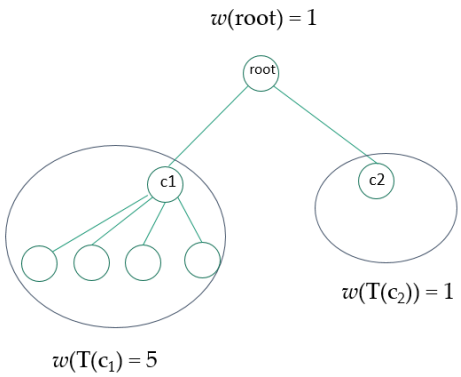
Another example is shown in Figure 10(a). In order to find the global BBN in this graph, as shown in Figure 10(b), we compute the betweenness centrality of each node and find the local BBNs in both of the subgraphs, and then we add the global root and the local BBNs into the candidate set. In Figure 10(c), we compute the external betweenness of each subgraph using their weights shown in Figure 10(d). Since we know that c_1, c_2, and the global root are in the candidate set, we compute their global betweenness centrality which is the sum of the internal betweenness and the external betweenness. As a result, the global betweenness centralities are computed and shown in Figure 10(e). The detailed values of betweenness centrality and the rank of the nodes are shown in Table 2.



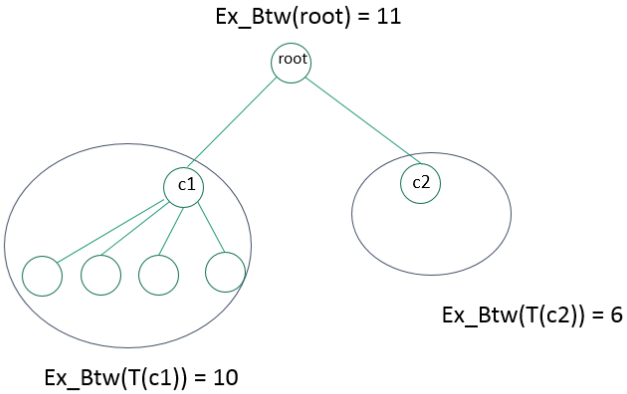
(a)



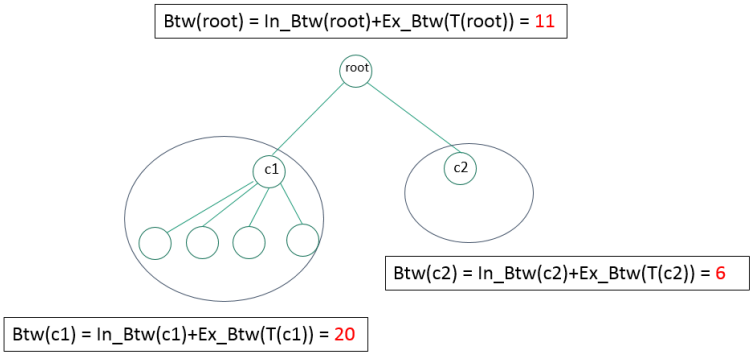
(b)



(c)



(d)



(e)

Figure 10. (a)(b)(c)(d)(e) Example 2.

Table 2. Results of betweenness centrality in Figure 10

Candidate Nodes	Internal Betweenness	External Betweenness	Global Betweenness = In_Btw + Ex_Btw	Rank of Betweenness Centrality
node root	0	11	11	2nd
node c_1	10	10	20	1st
node c_2	0	6	6	3rd

Another example in Figure 11(a) follows the same procedure to compute the global betweenness of all the candidate nodes. However, in this example, we show that we need to include more nodes which are 1) the nodes on the shortest paths from the global root to all the local BBNs, and 2) the nodes on the DFS routes starting from the nodes on the shortest paths. After we compute the internal betweenness centrality of each node and find the local BBNs, we add the global root, the local BBNs, and all the nodes on the shortest path from the global root to all the local BBNs as shown in Figure 11(b).

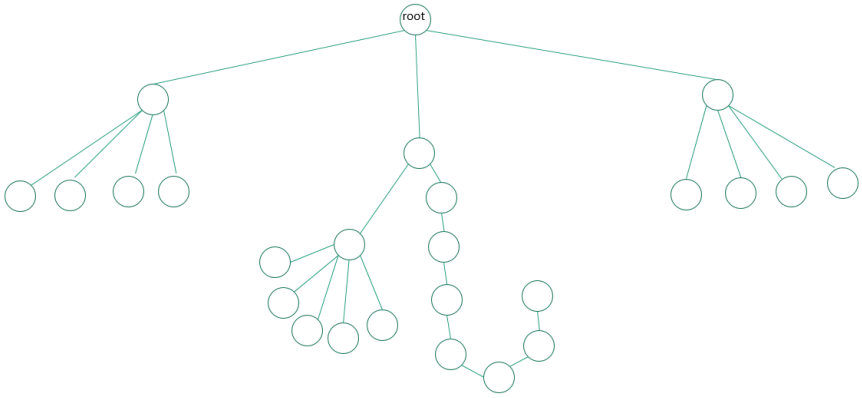


Figure 11(a)

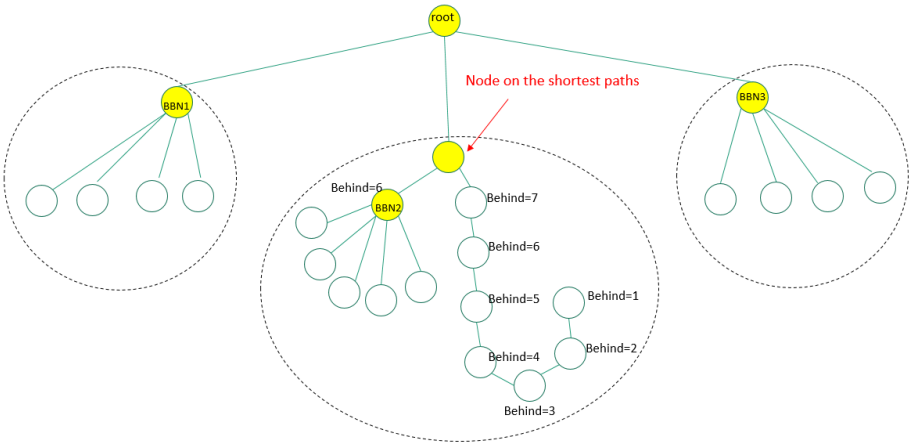


Figure 11 (b)

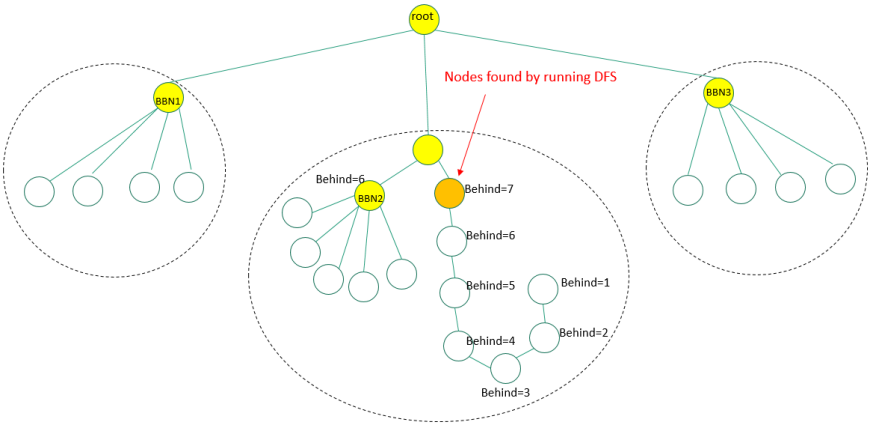


Figure 11(c)

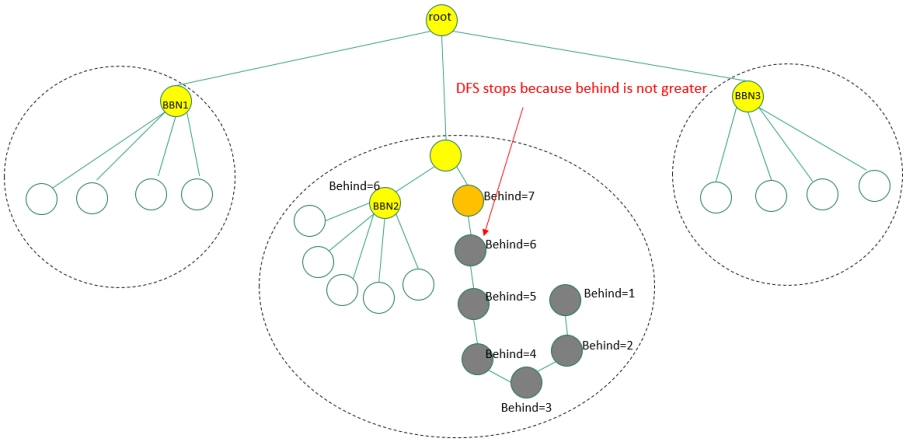


Figure 11 (d)

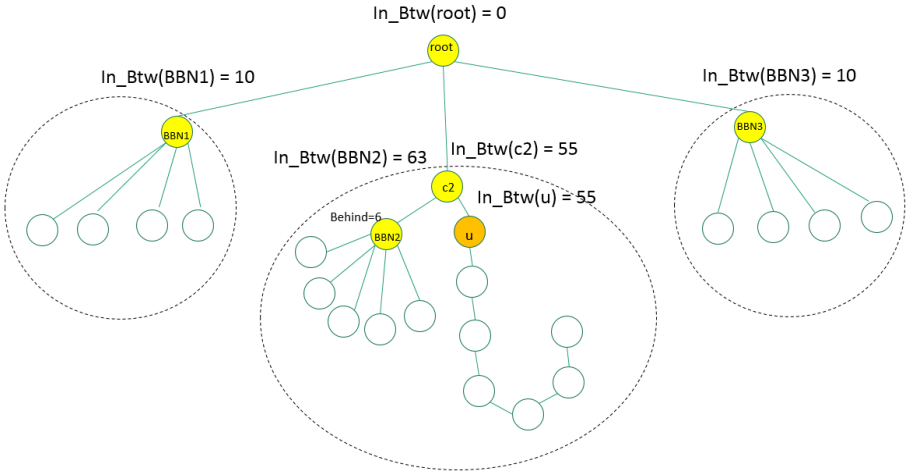


Figure 11 (e)

As shown in Figure 11(c) and (d), we run DFS from each node on a shortest path to search more candidate nodes, and we stop searching when the behind of a node is less than or equal to that of the local BBN. We follow the same procedure to compute the global betweenness centrality of each candidate node with their internal betweenness computed earlier (Figure 11(e)) and the external betweenness computed using the weights of the subtrees (Figure 11(f) and (g)). Figure 11(h) shows that node *u* has a greater global betweenness centrality than that of *BBN2* which is the local BBN of the subgraph and node c_2 is the global BBN. The detailed results are shown in Table 3.

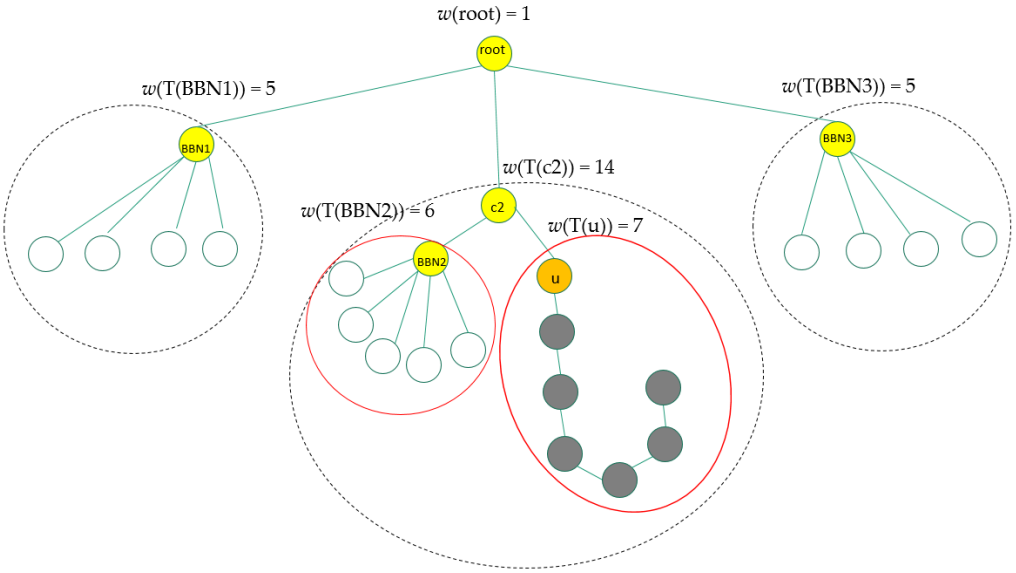


Figure 11 (f)

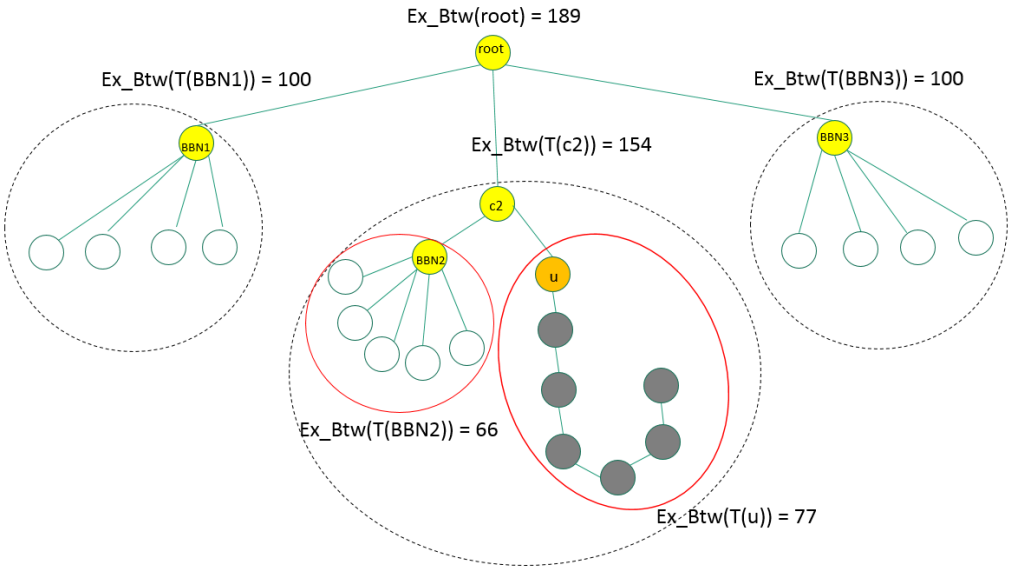


Figure 11 (g)

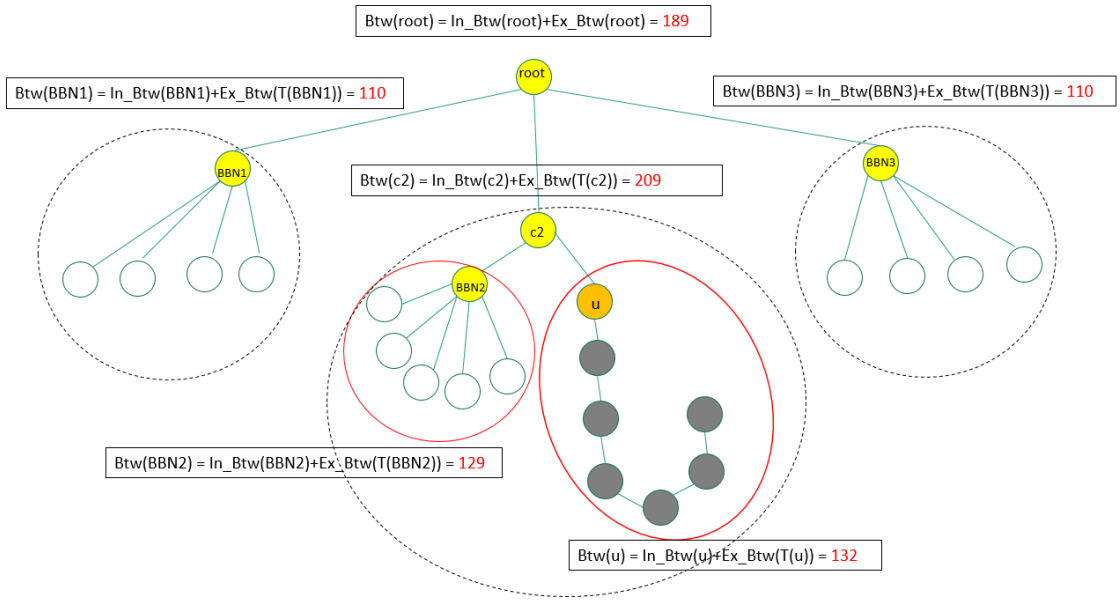


Figure 11 (h)

Figure 11. (a)(b)(c)(d)(e)(f)(g)(h) Example 3

Table 3. Results of betweenness centrality in Figure 11

Candidate Nodes	Internal Betweenness	External Betweenness	Global Betweenness = In_Btw + Ex_Btw	Rank of Betweenness Centrality
node *root*	0	189	189	2nd
node *BBN1*	10	100	110	5th
node *BBN2*	63	66	129	4th
node c_2	55	154	209	1st
node *u*	55	77	132	3rd
node *BBN3*	10	100	110	5th

The last example in Figure 12 shows how we consider the coarse nodes in Figure 12(b) in the graph and how to compute their betweenness centrality. When computing the internal betweenness

centrality of coarse nodes in each subgraph as shown in Figure 12(c), we consider the number of nodes in a coarse node as the weight *w*.

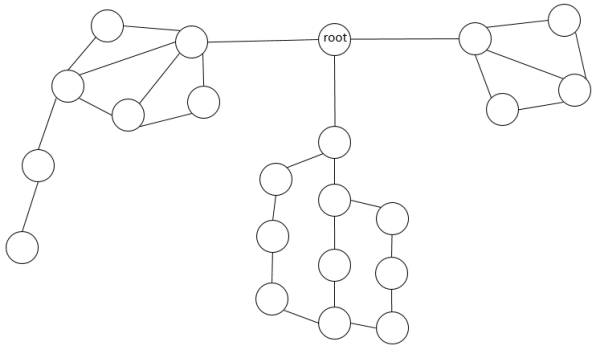


Figure 12(a)

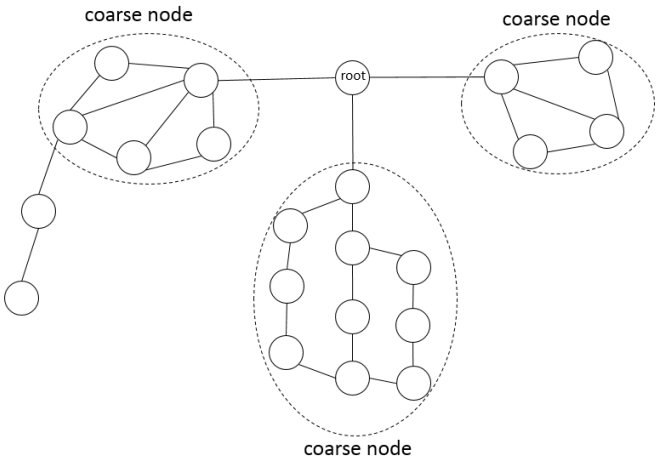


Figure 12(b)

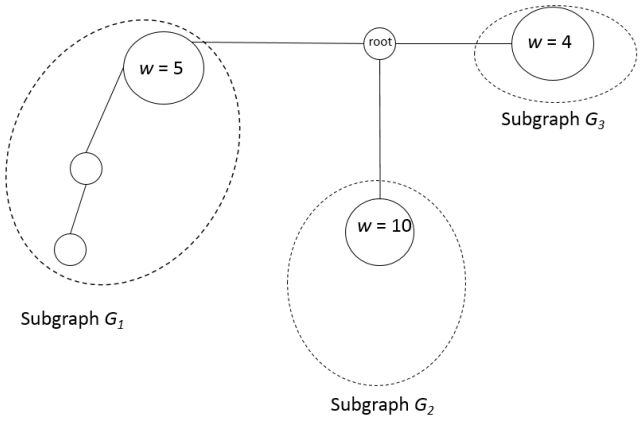


Figure 12(c)

In addition, we add an estimated weight to the internal betweenness, which is w*(w-1)/2. After we compute the internal betweenness centrality of each node and find the local BBNs, we add the global root, the local BBNs, and all the nodes on the shortest path from the global root to all the local BBNs as shown in Figure 12(d). Moreover, we follow the same procedure to compute the global betweenness centrality of each candidate node with their internal betweenness computed earlier (Figure 12(d)) and the external betweenness computed using the weights of the subtrees (Figure 12(e)). Figure 12(f) shows that coarse node c_2 has the highest global betweenness centrality, so we need to compute the global betweenness centrality for all the physical nodes inside the coarse node c_2 to find the global BBN. The detailed results are shown in Table 4.

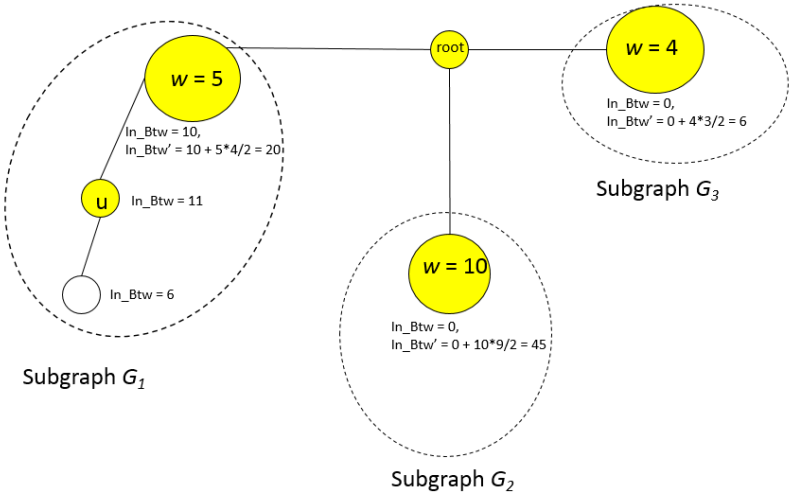


Figure 12(d)

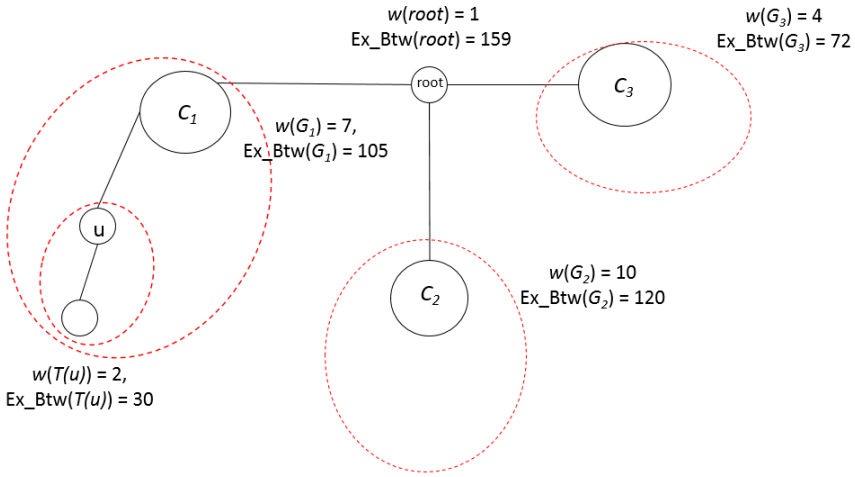


Figure 12(e)

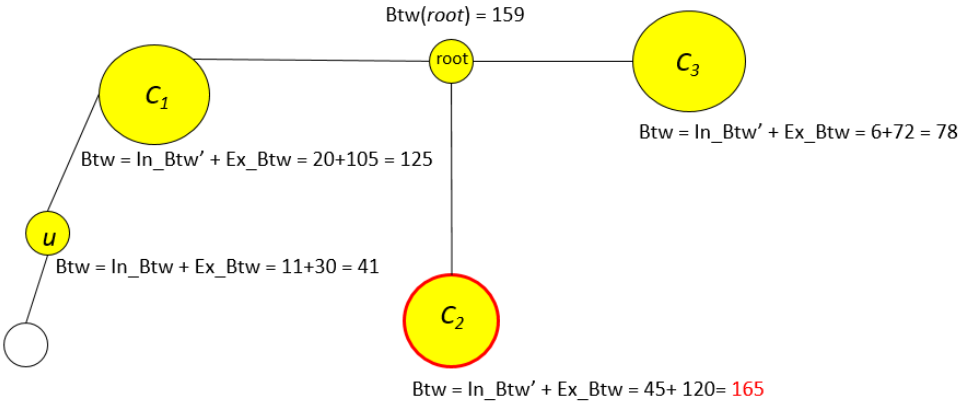


Figure 12(f)

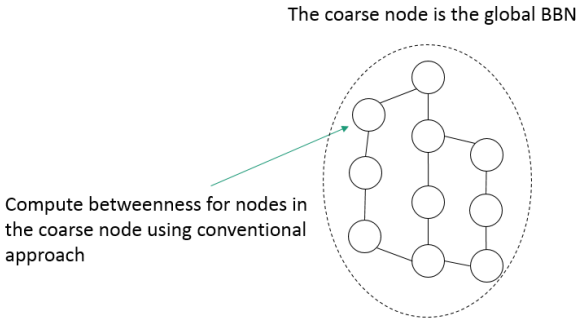


Figure 12(g)

Figure 12. (a)(b)(c)(d)(e)(f)(g) Example 4.

Table 4. Results of betweenness centrality in Figure 12

Candidate Nodes	Internal Betweenness	External Betweenness	Global Betweenness = In_Btw + Ex_Btw	Rank of Betweenness Centrality
node root	0	159	159	2nd
node c_1	20	105	125	3rd
node c_2	45	120	165	1st
node c_3	6	72	78	4th
node u	11	30	41	5th

3.5. Correctness SCB Algorithm

In the following, we will provide the theoretical foundation to justify the correctness of the SBC algorithm. Especially, we will justify the details of the pruning process.

Before we finish the pruning process and compute the global betweenness of all candidates, we have to assure that no candidate is left out during the process. As we introduce the idea of coarse node which is an abstract structure, we have to consider the nodes inside the coarse nodes. Hence, when a coarse node is excluded by the pruning process, we are certain that the physical nodes inside the coarse node will not have a chance to be the global BBN. This is discussed below.

Consider graph G which contains physical nodes and coarse nodes. The nodes inside any coarse node that is excluded by the candidate set has no chance to be the global BBN. This is because the internal betweenness of the coarse node is computed based on the upper bound $w(c)*(w(c) - 1)/2$ where $w(c)$ is the number of nodes in the coarse node, therefore the internal betweenness of any node inside the coarse node has to be less. On the other hand, any physical node excluded by the candidate set also cannot be the global BBN. Accordingly, the nodes

excluded by the candidate set must have less internal betweenness and fewer behind nodes than the local BBN (which is not a coarse node). Thus, their external betweenness must be less than the local BBN. These physical nodes do not have a chance to have a greater global betweenness than the local BBN, and they also cannot be the global BBN.

Accordingly, we can compute the global betweenness of each node in the candidate set. After all the irrelevant nodes are excluded, we apply Formula (2) to compute the global betweenness of the candidates. If the global BBN computed is a coarse node, we compute the global betweenness of each node inside the coarse node by summing up their internal betweenness and external betweenness. Subsequently, we can rank all the nodes in the candidate set including all the nodes inside a coarse node based on their global betweenness. If a coarse node is not chosen to be the global BBN, we do not need to examine the nodes inside the coarse node since these nodes do not have a chance to be the global BBN. We will prove this in the theorem below.

Theorem 1. *Consider graph G which contains physical nodes and coarse nodes. If a coarse node is a candidate but not the global BBN, the nodes inside the coarse node have no chance to be the global BBN, and the global betweenness of the nodes inside the coarse node does not have to be computed.*

Proof of Theorem 1: *Assume that a physical node v is the common BBN that was found by the SCB algorithm and that u is a physical node inside the coarse node c. Node c is a candidate but not the common BBN. According to the assumption, c is a candidate but not the common BBN, so v has higher betweenness centrality than c,*

$$Btw(c, G)' < Btw(v, G) \quad (1)$$

For the left-hand side of (1),

$$Btw(c,G)' = In\_Btw(c,G_j) + w(c)*(w(c)-1)/2 + Ex\_Btw(G_j),$$

because c is a coarse node, and we applied an upper bound for the physical nodes inside c when calculating the local BBNs, which is w(c)(w(c) - 1)/2, where w(c) is the number of nodes in c. Hence,*

$$Btw(c, G)' = In\_Btw(c, Gj) + w(c)*(w(c) - 1)/2 + Ex\_Btw(Gj) = In\_Btw(c, Gj)' + Ex\_Btw(Gj)$$

In addition, u is a physical node inside c, and In_Btw(c, Gj)' is the upper bound for the internal betweenness of u in Gj,

$$Btw(u, G) = In\_Btw(u, Gj) + Ex\_Btw(u, Gj) <= In\_Btw(c, Gj)' + Ex\_Btw(Gj) = Btw(c, G)'$$

Therefore, we have the following equation,

$$Btw(u, G) <= Btw(c, G)' \qquad (2)$$

Based on (1) and (2),

$$Btw(u, G) <= Btw(c, G)' < Btw(v, G),$$

$$Hence, Btw(u, G) < Btw(v, G) \blacksquare$$

In conclusion, all the nodes (physical and coarse nodes) excluded by the candidate set cannot be the global BBN. In addition, the global betweenness of all the candidates can be computed by Formula (2). Therefore, the physical node ranked the highest in the candidate set is indeed the global BBN.

3.6. Complexity and Hierarchical SBC

The time complexity of the conventional approach as discussed in Section 2.2 is $O(mn+n^2\log n)$ based on Dijkstra's version of all-pair shortest path algorithm, where n is the number of nodes, and m is the number of edges. For the subgraph-level betweenness approach we proposed, the time

complexity can be reduced. In the best case in which the number of edges in each subgraph is equal, and the whole graph is a sparse graph, the whole graph can be considered as a tree since all the cycles are transformed into coarse nodes. In this case, the weights on the edges are not necessary when finding the BBNs, and these weights only affect the computation when computing betweenness centrality for the nodes inside the coarse nodes which are the BBNs by the traditional approach. For the worst case where the graph is a complete graph or the whole graph is a cycle, the time complexity is $O(mn+n^2\log n)$, since our approach has to apply the traditional approach to compute the betweenness centrality based on the edge weights inside the coarse nodes when the coarse nodes are the BBNs. In addition, if the number of edges in each subgraph is extremely unbalanced, the time complexity approximately becomes the same as the conventional approach. It is observed that when k is greater, the computational time speeds up more evidently. However, since the factor k is restricted by the maximum degree of nodes in the graph, we might not be able to partition the graph as much as desired. Therefore we introduce the idea which is running SBC hierarchically. It allows us to find the global BBN by merging the results of the next-level BBNs, and the next-level BBNs can be found based on the results at the deeper levels. The hierarchical SBC algorithm is the SBC algorithm with one additional step:

Step 7: If the graph was decomposed as single-layered structure, stop the algorithm when finishing Step 6; otherwise, repeat Step 3 to Step 6 until the global BBN is found, as the graph is a multi-layered structure. If the global BBN computed is a coarse node, we compute the global betweenness of each node inside the coarse node by summing up its internal betweenness and external betweenness. Hence, we can rank all the nodes in the candidate set including all the nodes inside coarse nodes based on their global betweenness.

We discuss the time complexity of both single-layered SBC and hierarchical SBC in the following and the speedups of both strategies in comparison with the traditional approach which takes $O(mn+n^2\log n)$.

1. Single-layered structure:

   Assume that the number of subgraphs is k, so this structure takes $O((mn+n^2\log n)/k^2)$ as the computational time in each subgraph, so it takes $O((mn+n^2\log n)/k)$ to process k subgraphs. In addition, it requires one-time BFS and DFS to merge the results from the local BBNs, so the total computational time is $O((mn+n^2\log n)/k + (n+m))$. Hence, the speedup is about $O(k)$ as the running time of DFS is minor enough to be ignored.

2. Hierarchical structure:

   Assume that the number of subgraphs is k. It takes the same computational time in each subgraph as the single-layered approach. The only difference is the time for merging. It requires $O(l*(n+m))$ to run DFS in each layer of the structure in the process of merging, where $l = \log_d k$ is the number of layers, and d is the maximum outgoing degree of each layer. Therefore, the total time required is $O((mn+n^2\log n)/k + l*(n+m))$, so the speedup is less than that of the single-layered structure.

As a result, if a graph is not too large the single-layered structure has a better performance than the hierarchical structure, and both of them can reduce the computational time with respect to k. However, the maximum degree of nodes in the graph is limited, so the number of subgraphs k has to be restricted as well. For very large graphs, we can have a combination of both types of the structures to have a better performance.

CHAPTER 4 Color Graphs

In this section, we introduce colors to revise the definition of betweenness centrality. We show the merits when nodes and edges in a graph are colored. We then introduce a new model to compute the betweenness measures on graphs with colored edges.

4.1. Colorless Graphs

In a colorless graph, when finding all-pair shortest paths, all the nodes are treated equally so that we can exhaustively go through each node in the graph as the source node and select all the other nodes as the target nodes. After all the shortest paths are computed, we compute and find the nodes that have the most shortest paths passing through.

For example, in a directed graph shown in Figure 13, nodes represent genes (e.g., P1, P2), and edges represents diseases such that two nodes are linked if they are both related in a disease with an influence factor. To find the most important gene that has the highest betweenness centrality value, we can first apply an all-pair shortest path algorithm; we then compute the number of shortest paths passing through each node, and the node with the most shortest paths passing through is considered the most influential node.

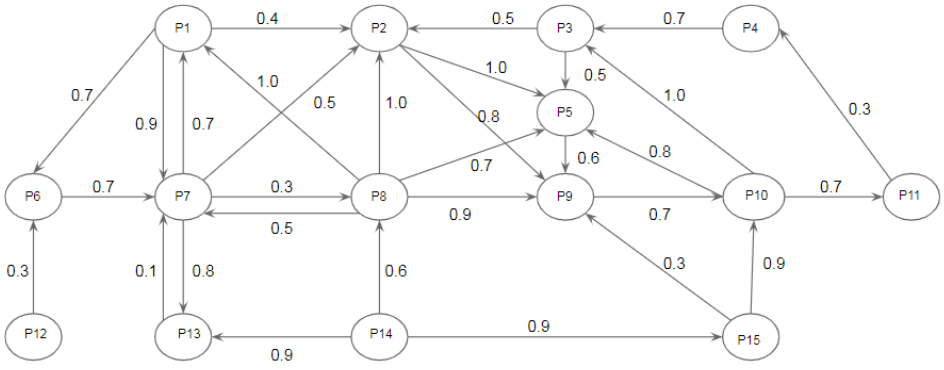


Figure 13. A colorless graph

4.2. Colored-Node Graphs

In a colored graph, all the nodes are categorized into different groups expressed in different colors. Before computing betweenness centrality, we select the nodes with desired colors and drop the other nodes with different colors. We can then compute all the shortest paths in the extracted colored subgraph. After the shortest paths are computed, we apply betweenness centrality and find the nodes with desired colors that have the most number of shortest paths passing through.

For example, in a directed and colored graph shown in Figure 14, each node represents a disease and a color represents a disease group. Two disease nodes are linked if one is impacted by the other with an influence factor. In Figure 14, yellow means human cancers, red represents cardiovascular diseases, and blue stands for immune system diseases. An investigator may be interested in the query "Find the most important disease in the context of the cardiovascular disease group and the immune system related disease group."

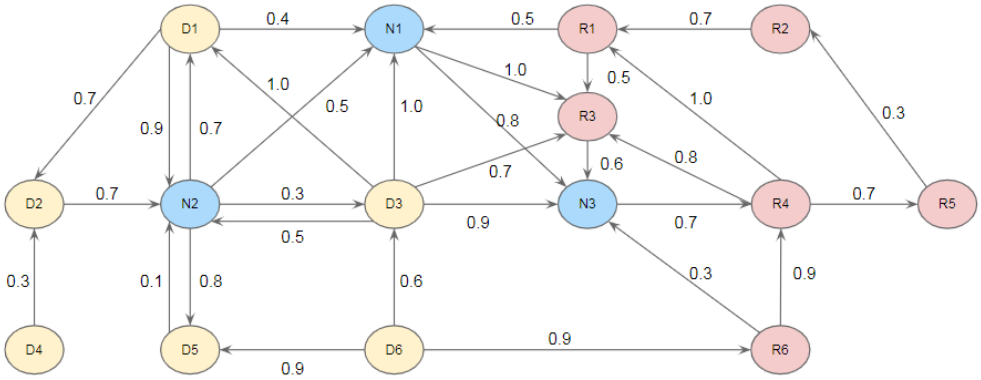


Figure 14. A colored graph

4.3. Colored-Edge Graphs

Given a graph with colorless nodes and colored edges, there could be multiple edges with different colors between two nodes as shown in Figure 15, where the nodes are genes and two

nodes are connected if they are correlated in a disease. If two genes are correlated in multiple diseases, a set of links are created where each link has a different color and has an assigned edge-weight. As discussed in Assumption 1 below, if there are multiple edges between any pair of two nodes, when calculating the global betweenness centrality, we assume the selected edge-weight is the smallest edge-weight among the multiple edges. Betweenness centrality is based on all-pair shortest paths on the graph, and since there are multiple options on the weights between each pair of nodes, choosing the weight for each pair of nodes may influence the computation of the all-pair shortest paths. In addition, the computation of global betweenness centrality in the whole graph can be considered as computing the betweenness centrality in the colorless graph because all the colors are considered when computing the global betweenness. Therefore, for any pair of nodes containing multiple colorless edges, the edge with the minimum weight will be selected when finding the shortest path passing through the pair of nodes. An investigator may be interested in finding the most important genes that are correlated in a given set of diseases (e.g., blue represents breast cancer and red represents colon cancer). Before computing betweenness centrality, we can extract multiple subgraphs based on the colors (e.g., the breast cancer subgraph and the colon cancer subgraph), and then calculate the betweenness of every node in each subgraph.

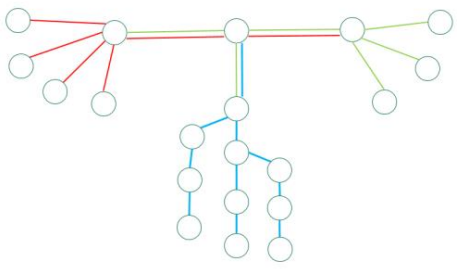


Figure 15. A colored-edge graph.

Assumption 1. *Consider a graph G containing two subgraphs, G_{red} and G_{green}, with different colored edges. In the graph G, there may be more than one edge between any pair of two nodes,*

which belong to G_{red} or G_{green}. In the aggregated graph G, we assume the edge weight between any pair of two nodes u and v to be:

$$weight(u,v) = \underset{Gi = Gred\,|Ggreen}{\operatorname{argmin}} \{weight(u,v) \mid (u,v) \in Edge(Gi)\}.$$

4.4. Colored-Node-Edge Graphs

Given a graph with colored nodes and colored edges, all the nodes are categorized into different colors, and there could be multiple edges with different colors between any pair of two nodes. The nodes are genes and two nodes are connected if they are correlated in a disease. As two genes may be correlated in multiple diseases, a set of links can be created between the two genes where each link has a different color and has an assigned edge-weight. Additionally, colors on nodes represent different organs, so that all the genes are colored with the same color when they are related to the specific organ. The problem of betweenness centrality for this type of graph may include "find the most important gene related to brains among three types of diseases, such as brain cancer, cardiovascular disease and neurological disorder." Before computing betweenness centrality, we select the nodes with desired colors and drop the other colored nodes. Then, we extract the desired subgraph by dropping the other irrelevant colored edges. Hence, we can compute the all-pair shortest paths in the extracted colored subgraph. After the shortest paths are computed, we compute betweenness centrality and find the nodes that have the highest betweenness centrality.

For most of the time, we apply the betweenness centrality to find the most important node which is the backbone node (denoted as BBN) in a graph composed of several colors. There are several types of colored graphs as mentioned earlier in this section. Additionally, we would be interested in the backbone nodes in an integrated graph composed of several colors, or in the whole

graph considering all the colors. To compute the betweenness centrality and find the BBN in any integrated graph composed of certain specific colors using the conventional approach proposed by Brandes, we need to execute the betweenness centrality algorithm once for each combination of disease subgraphs, while all the possible combination of subgraphs is 2^k, and k is the number of colors. When the size of the whole graph is large, the computational time of the traditional betweenness centrality is excessively long, and it is even worse when we need to execute the algorithm 2^k times for all the possible combinations.

In the next section we propose an algorithm called SCB (Scalable Colored Betweenness). Since we are usually concerned about the BBNs in each colored subgraph or the whole graph, the SCB is an efficient approach to finding the global BBN for the whole graph based on the local BBNs in each colored subgraph. SCB is a scalable approach for dividing the graph data into subgraphs and conquering the aggregated betweenness centrality results by merging the local betweenness centrality results. computed from the betweenness in the subgraphs.

CHAPTER 5 Scalable Colored Betweenness (SCB) Centrality

We have defined different types of colored graphs, and we have discussed different scenarios of using betweenness centrality on each type of graphs in the previous section. Accordingly, the purpose of this research is to propose different scalable approaches based on each type of the graphs, i.e., colored-node graphs, colored-edge graphs, and colored-node-edge graphs, which will be discussed in this section. We start the discussion of colored-edge graphs in the following paragraphs.

The definition of SCB for colored-edge graphs is to find the node which has the highest number of shortest paths passing through, and those shortest paths have to be composed of a

specific color. Different from the traditional approach, we can find the backbone node, the global BBN, which has the highest betweenness centrality in a graph by integrating the information of local BBNs in the subgraphs, where the subgraphs have different colors. This means in order to find the most important node in the original graph, we do not need to completely compute all-pair shortest paths passing through the nodes in the original graph. Instead, we can partition the original graph into several subgraphs where each subgraph includes the edges of one color, and then compute betweenness centralities for every node in each colored subgraph and find the local BBN in each colored subgraph. Subsequently we apply the divide-and-conquer design paradigm [26] and find the common BBN for two subgraphs at a time. After the local BBNs of both subgraphs are found, we propose the following two major steps to compute the common BBN.

1. Add all the potential nodes for the common BBN into a candidate set. These include: 1) the local BBNs in both of the colored subgraphs, and 2) all the nodes on the shortest path between the two local BBNs. As illustrated in Figure 16(a), the blue nodes are the local BBNs. Figure 16(b) shows the nodes on the shortest paths (with color yellow) between the two local BBNs.

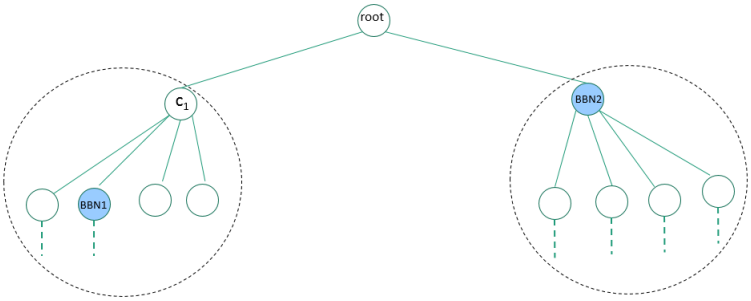


Figure 16(a). Local BBNs.

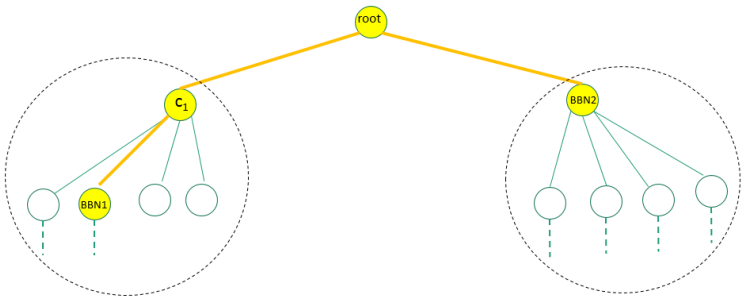


Figure 16 (b). Yellow nodes are the nodes on the shortest paths between every two local BBNs.

2. Compute the betweenness centrality of each node *u* in the candidate set based on internal betweenness and external betweenness, using a simple formula, Btw(*u*, *G*) = In_Btw(*u*, *T*(*u*)) + Ex_Btw(*T*(*u*)), where *G* is the graph integrated from two colored subgraphs.

Since the total number of colored subgraphs might be more than two, we recursively find the common BBN for two subgraphs, and we merge the results for every two subgraphs until the global BBN is found. After the global betweenness of each of the candidate nodes is computed, we can rank the global betweenness centralities of the candidate nodes based on their global betweenness. Therefore, we can obtain the global BBN among all the colored subgraphs without computing the all-pair shortest paths in the whole graph by decomposing the graph into colored subgraphs.

5.1. Definitions

Below we present the terms that would be used frequently in our scalable colored betweenness centrality algorithm (SCB).

- **BBN(*G*)*:*** The Backbone node of graph *G*, which is the node that has highest betweenness centrality in *G*.
- **Btw(*u*, *G*):** The Betweenness centrality value of the node *u* in graph *G*.

- **Internal shortest paths:** The source node and target node of the path are both in the same subgraph.
- **External shortest paths:** The source node and target node of the path are in different subgraphs.
- **In_Btw(u, G_k):** The number of internal shortest paths passing through node u in G_k.
- **Ex_Btw(G_k):** The number of external shortest paths passing through the subgraph G_k.
- **Shortest_path(BBN(G_1), BBN(G_2)):** The shortest path which starts from BBN(G_1) to BBN(G_2).
- ***w*(BBN(G)):** Number of nodes behind the BBN node in graph G.
- **Behind(BBN(G), G):** The set of nodes behind the node BBN(G) in graph G. This represents the nodes isolated from graph G if the node BBN(G) is disconnected from G. For example, consider the graph G shown in Figure 17(a) which is a tree graph. The node c_1 in this example is the node u in the definition. If the edge (root, c_1) in Figure 17(b) is removed from graph, c_1 will be disconnected from graph G. Then, the set of nodes circled in Figure 17(b) will be isolated from the graph and considered as Behind(c_1, G).

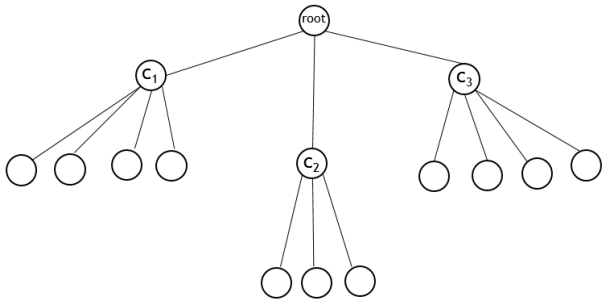


Figure 17(a). A tree graph G.

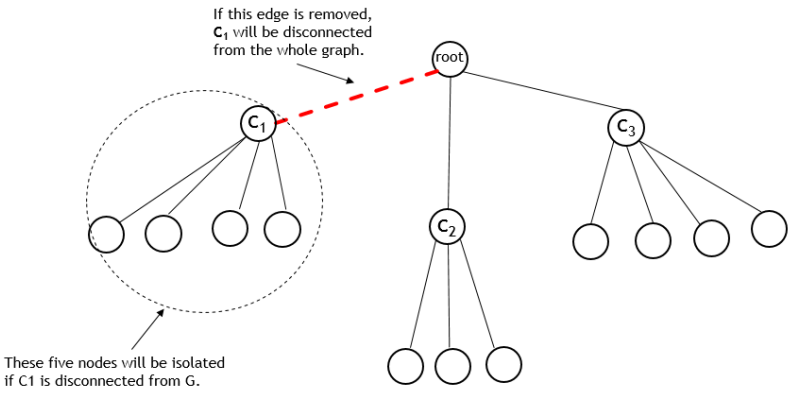


Figure 17 (b). Demonstration of behind set

5.2. The SCB (Scalable Colored Betweenness) Algorithm

The first step of the algorithm discovers all the cycles in the graph and transforms the cycles into abstract coarse nodes. As discussed earlier in another approach proposed by us [27], the result graph is a tree structure containing physical nodes and coarse nodes. Although a minimum spanning tree (MST) algorithm [25] finds a tree structure of the graph, we do not need to apply MST to convert the graph into a tree structure. This is because MST finds a tree structure which connects all the nodes in the graph with the minimum costs, but MST does not retain the all-pair shortest paths in the original graph, which is the fundamental information for betweenness centrality. Instead, we find the tree structure of a graph by preserving all the cycles and grouping them into coarse nodes. Hence, there are several consequences of the tree property that would support our approach. One consequence of a tree structure is that the root node is the only connecting node among its subtrees. Since the root node of a tree has edges connected to its subtrees, if there is another connecting node, there will be a cycle introduced. As shown in Figure 18, if the subtree $T(c_3)$ has two connecting nodes, c_3 and u, there will be a cycle formed by the root, c_3 and u. Another consequence is that the shortest path between two nodes is exactly the path between the two nodes because in a tree there is a unique path between any two nodes. Accordingly, the weights of edges in the tree do not affect the computation of betweenness centrality. However, our approach still needs to handle weighted graphs because we will examine the nodes inside the coarse node if a coarse node is the global BBN, and the weights inside the coarse node will be applied to the computation of betweenness centrality.

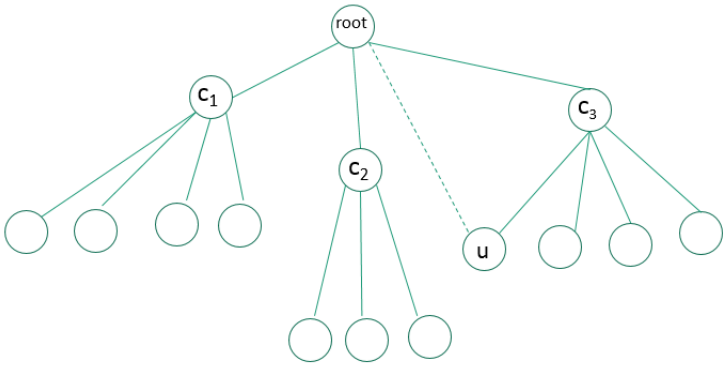


Figure 18. Connecting nodes

In this approach, we decompose the graph (tree) into subgraphs (subtrees), and we compute the global betweenness based on the internal betweenness information in each subgraph and the external betweenness across subgraphs. The external betweenness of a subgraph can be computed based on the number of nodes in the subgraph. Consider a graph G which is a tree containing subtrees, G_1, G_2, … G_k as shown in Figure 19, and that the number of nodes in each subgraph is denoted as $w(G_j)$, $j = 1$ to k. Since a tree is a connected graph, all nodes in graph G are connected. Hence, every node in G_1 can create a path to each node in G_2.

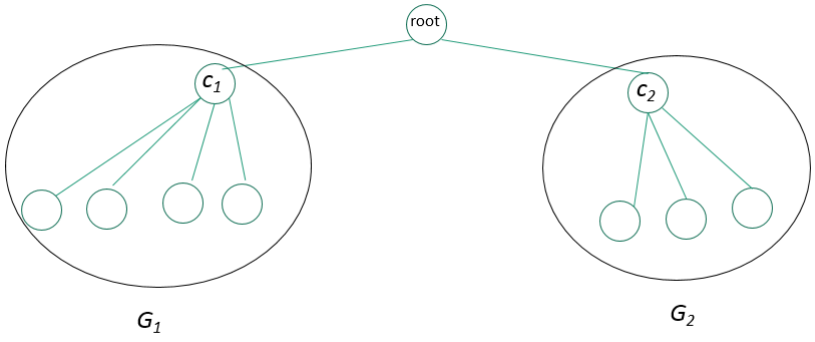


Figure 19. External betweenness

As shown in Figure 19, the number of external shortest paths passing between two subgraphs G_1 and G_2 is therefore $w(G_1) * w(G_2)$. Besides, the number of external shortest paths passing between G_1 and the root is $w(G_1) * 1 = w(G_1)$. Therefore, the external betweenness of both subgraphs and the root is $\text{Ex\_Btw}(G_1) = w(G_1) * (w(G_2)+1)$, $\text{Ex\_Btw}(G_2) = (w(G_1)+1) * w(G_2)$, and $\text{Ex\_Btw}(root) = w(G_1) + w(G_2) + w(G_1) * w(G_2)$. Furthermore, we can observe that the external betweenness of a subtree $T(c)$ is proportional to the number of nodes behind the root of the subtree, which is behind(c, $T(c)$).

One of our major concepts is that the global betweenness centrality of a tree node can be computed with the sum of internal betweenness and external betweenness using the formula, $\text{Btw}(u, G) = \text{In\_Btw}(u, G_j) + \text{Ex\_Btw}(G_j)$, where u is the only connecting node in the subgraph G_j. Consider a graph G which is composed of subgraphs, G_1, G_2, …, and G_k. The global betweenness is the total number of shortest paths passing through u in G, and these shortest paths can be categorized into two types: 1) internal shortest paths in which both source and target nodes are inside subgraph G_j, and 2) external shortest paths in which neither source nor target is inside G_j, or either one of them is inside G_j. We denote these two types as the first part and the second part below.

1. **$\text{In\_Btw}(u, G_j)$**: Since $\text{In\_Btw}(u, G_j)$ is the number of internal shortest paths passing through u in G_j, it is obvious that $\text{In\_Btw}(u, G_j)$ is identical to the first part of the $\text{Btw}(u, G)$ which is the number of all the internal shortest paths passing through u in G_j.
2. **$\text{Ex\_Btw}(G_j)$**: It is the number of all the possible external shortest paths in G_j. Since node u is the only connecting node, all these external paths pass through u.

Therefore, The global betweenness, Btw(u, G) = In_Btw(u, G_j) + Ex_Btw(G_j). Additionally, if G_j is equal to $T(u)$ which is the subtree of node u, then we can replace G_j in the formula with $T(u)$, such that Btw(u, G) = In_Btw(u, $T(u)$) + Ex_Btw($T(u)$).

The other major concept in our approach is that we can prune the graph data to have a smaller set of candidates which can be the common BBN between two colored subgraphs. We start the pruning process by combining two colored subgraphs. Since the cycles were transformed into coarse nodes and the result graph is a tree structure, there are several possibilities for combining the subgraphs as listed below:

1. **The two subgraphs completely overlap**: We do not need additional computation to find the common BBN. Since the two local BBNs are exactly the same physical node, the common BBN is the same as the local BBNs.
2. **The two subgraphs partially overlap**: We need to compute and find the common BBN on the combined graph which is a tree.
3. **The two subgraphs are isolated**: Since the two subgraphs are isolated, there is no external path passing through the two subgraphs. Hence, both the two local BBNs can be the candidates of the common BBN, and the number of common BBN can be more than one.

Accordingly, we need to select a root node on the combined graph which is a tree structure before we find all the potential nodes in the candidate set. We randomly select a node on the path between two local BBNs as the root node. Then, we continue acquiring all nodes on the path between the local BBNs. In addition, these nodes are also the parents or ancestors of the local BBNs, which are exemplified by the nodes on the blue paths shown in Figure 20, and node u is the local BBN in G_2. Other than the candidate nodes on the paths, nodes on another branch of u's

parent or any other ancestors may also have a greater global betweenness than the local BBN, such as node *v* in the figure. They have a chance to attract more external paths than the local BBN if they have more behind nodes than the local BBN. After we include the nodes with a greater behind set than that of *u*, we filter out the irrelevant nodes because they have less internal betweenness and fewer behind nodes than *u*. Furthermore, the nodes which are behind *u* would not have a greater global betweenness centrality than *u*, since 1) they have less internal betweenness than the local BBN and 2) all the external shortest paths coming through these nodes must pass through the local BBN first.

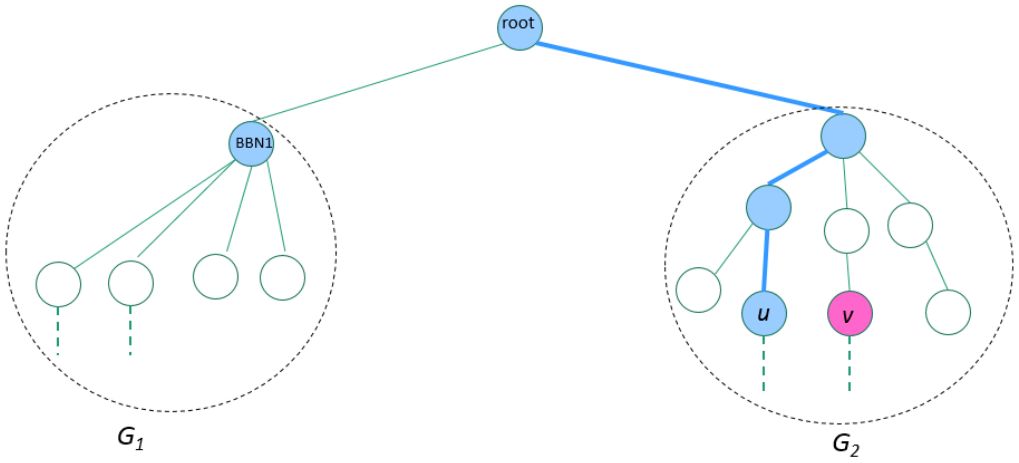


Figure 20. Candidates of common BBN.

Below is a summary of the SCB algorithm:

The SCB Algorithm

Step 1: Apply Depth-first search(DFS) to detect cycles in the graph *G*. Create abstract coarse nodes for cycles.

Step 2: Apply the divide-and-conquer design paradigm. Partition graph *G* into two subgraphs recursively until there are *k* subgraphs, and each subgraph has a unique color.

Step 3: For each subgraph G_j, j = 1 to k, calculate In_Btw for each node and find the local BBN(G_j). When calculating the internal betweenness for a coarse node, add estimated weight $w(c)*(w(c) - 1)/2$ to the internal betweenness,

$$\text{In\_Btw}(c, G_j)' = \text{In\_Btw}(c, G_j) + w(c)*(w(c) - 1)/2, \quad (3)$$

where c is a coarse node.

Step 4: Find the common BBN for two subgraphs at a time. In order to find the common BBN for two subgraphs (i.e., G_1 and G_2), add all the potential nodes into a candidate set. These include: 1) both the local BBNs, 2) all the nodes on the shortest path between the two local BBNs, and 3) the nodes with more behind nodes than that of the local BBNs.

Step 5: Randomly select a node on the path between the two local BBNs to be the root node.

Step 6: Compute betweenness centrality of each node u in the candidate set, using the formula,

$$\text{Btw}(u, G') = \text{In\_Btw}(u, T(u)) + \text{Ex\_Btw}(T(u)), \quad (4)$$

where G' is a combination of G_1 and G_2, and $T(u)$ is the subtree of u. Find the BBN(G') with the highest betweenness centrality.

Repeat Step 4 to Step 5 until G' is identical to G, and then the global BBN is found.

5.3. Pruning

The major concept of our approach is to find the global BBN by searching all the candidate nodes and computing their global betweenness. There are two types of candidates: 1) the local BBNs, and 2) the nodes on the paths between the local BBNs. In addition, the global betweenness of the candidates can be computed with Formula (4).

However, from the example in Figure 20, we can observe that a node which does not belong to any type of the candidates mentioned above can possibly have a greater global betweenness than the local BBN. This situation occurs when the node has a greater set of behind nodes than that of a local BBN, so that it can have a higher global betweenness as it has more external paths than the local BBN. Furthermore, there could be multiple abstract coarse nodes representing cycles in the tree graph. To solve this problem we add weights to the internal betweenness of such nodes in order to estimate the maximal internal betweenness of the nodes inside the coarse nodes. If a coarse node is decided to be the global BBN, we need to retrieve the betweenness centrality for all the nodes inside the coarse node and rank them by their global betweenness with all the other candidates. The followings are the key steps of the pruning process in the SCB algorithm:

1 When computing the internal betweenness of the nodes in each of the subgraphs, we add weights to the internal betweenness of each coarse node in order to estimate the maximal internal betweenness of the nodes inside the coarse node. With such, no potential candidate nodes inside a coarse node can be left out.

2 Find the local BBNs after computing the internal betweenness. When a coarse node is ranked the highest, find the physical node with the second highest internal betweenness, and include the coarse node into the candidate set.

3 Include more nodes inside the subgraph into the candidate set. If a node has more behind nodes than that of the local BBN, the node can be a candidate. This could be done by running depth-first search. In Figure 21, it demonstrates searching more nodes from all the nodes on the path between local BBNs, and the search stops when a node's behind is less than or equal to that of the local BBN.

4 If the global BBN computed is a coarse node, we compute the global betweenness of each node inside the coarse node by summing up their internal betweenness and external betweenness. We can then rank all the nodes in the candidate set including all the nodes inside the coarse node based on their global betweenness. We do not need to compute the betweenness for the nodes in the other coarse nodes in the candidate set, as they cannot be the global BBN.

In the next sections, we demonstrate the detailed steps of the SCB algorithm with examples, based on the major concept of our approach as well as the pruning process. Additionally, we will discuss and prove the correctness of the SCB algorithm including the pruning process.

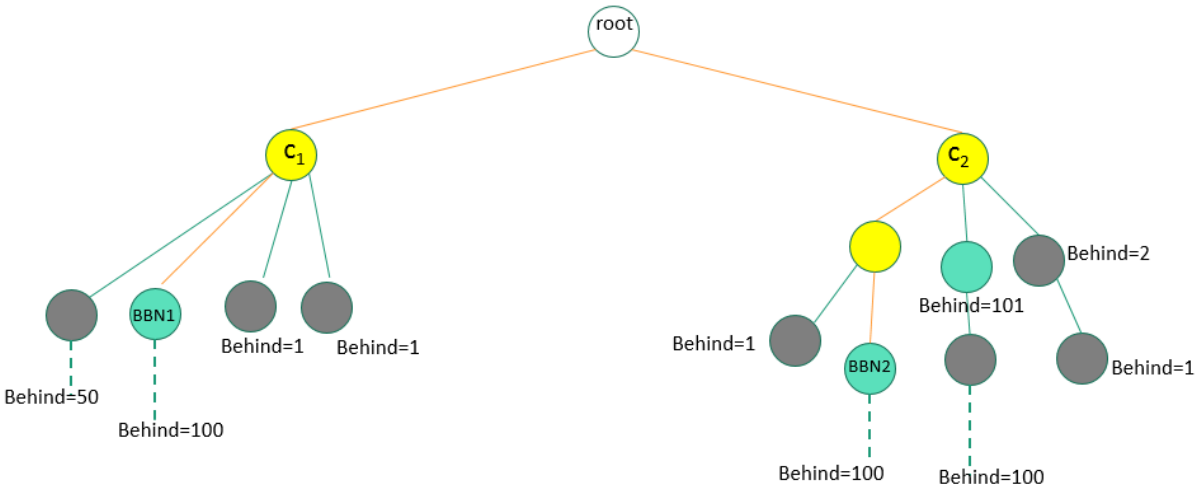
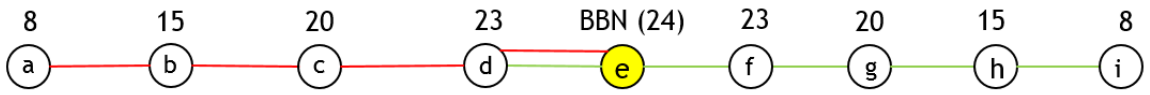


Figure 21. Nodes with greater behind set

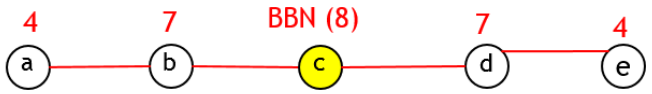
5.4. Examples

In this section, we will demonstrate the idea of SCB to find the global BBN node of a graph *G*. First, a graph composed of multiple colors can be partitioned into subgraphs according to color. Hence, SCB allows sequential or parallel processing on each colored subgraph, so that the local result of betweenness centrality can be computed. For example, the graph *G* shown in Figure 22(a) can be decomposed into two subgraphs, G_{red} and G_{green} shown in Figures 22(b) and 22(c)

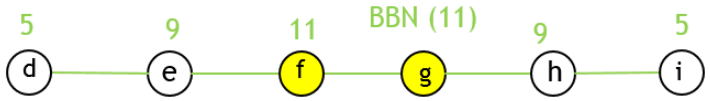
respectively. The local betweenness centrality are shown in Figures 22(b) and 22(c), and BBN(G_{red}) is node *c* and BBN(G_{green}) is node *f* or *g*, as they both have the highest betweenness centrality values.



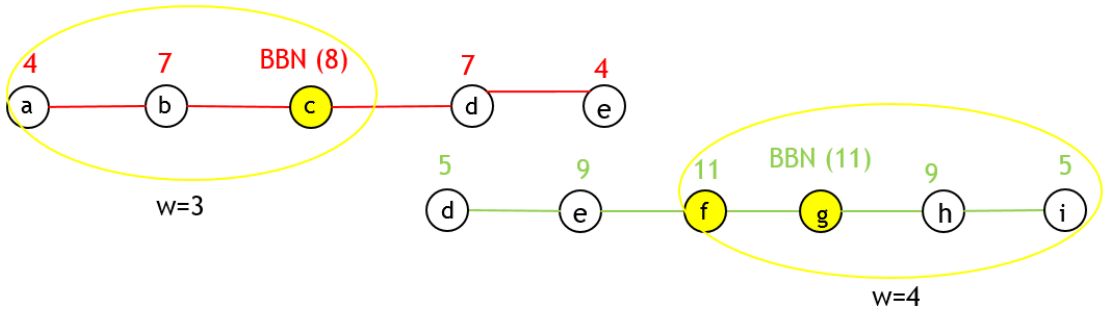
(a) Original graph G composed of color red and color green.



(b) G_{red} that only contains red edges.



(c) G_{green} that only contains green edges.



(d) Assign weights of group of nodes behind the BBNs, and calculate the external betweenness.

Figure 22. Example 5

After combining the two subgraphs G_{red} and G_{green}, the aggregated BBN(*G*) is located on the shortest path starting from BBN(G_{red}) and BBN(G_{green}). In addition, the global BBN(*G*) can be found by calculating the external betweenness of every node on the shortest path. As shown in

Figure 22(d), assign weights of BBN(G_{red}) and BBN(G_{green}) to be number of nodes behind BBN(G_{red}) and BBN(G_{green}) respectively, and find the global BBN by calculating the external betweenness. Therefore, we can find Btw(d)=23 and Btw(e)= 24, and the global BBN(G) is node e.

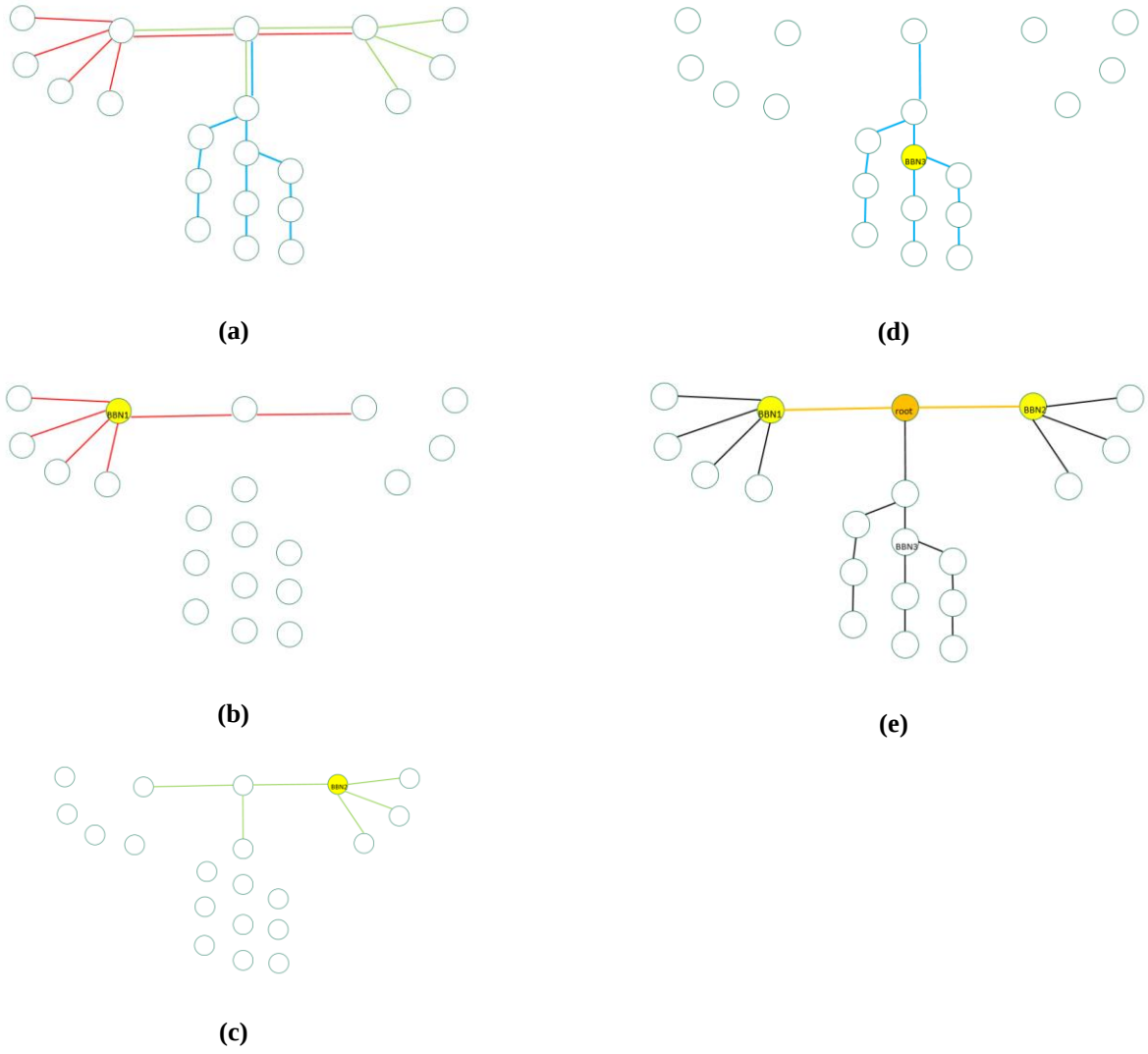


Figure 23. Example 6

We demonstrate another example graph shown in Figure 23(a) with three colors: red, green, and blue. In this example, we show that we need to include more nodes which are the nodes on the

DFS (depth-first search) routes starting from the nodes on the shortest paths. All the colored subgraphs are shown in Figure 23(b), (c), and (d).

In the following steps, we combine two subgraphs every time and we repeat the same steps until we find the global BBN. For colors red and green, after we compute the internal betweenness centrality of each node and find the local BBNs, we select a node in the intersection part as the root and find all the shortest paths between all the local BBNs, such as the orange paths shown in Figure 23(e). Then, we add all the nodes on the shortest paths to a candidate set. Furthermore, we run DFS from the nodes on the shortest paths in each subgraph, and the DFS stops when the behind of the node is less than that of the local BBN. Moreover, in Figure 23(f), we compute the global betweenness centrality of each candidate node by the sum of the internal betweenness and external betweenness. The internal betweenness is computed in the subtree $T(n)$, while the external betweenness is computed based on the external shortest paths passing through the subtree $T(n)$, where n is one of the candidate nodes. In addition, the external betweenness is computed based on the number of nodes behind the nodes n, $w(n)$, and it is equal to the product of $w(n)$ and the number of all the other nodes in the graph.

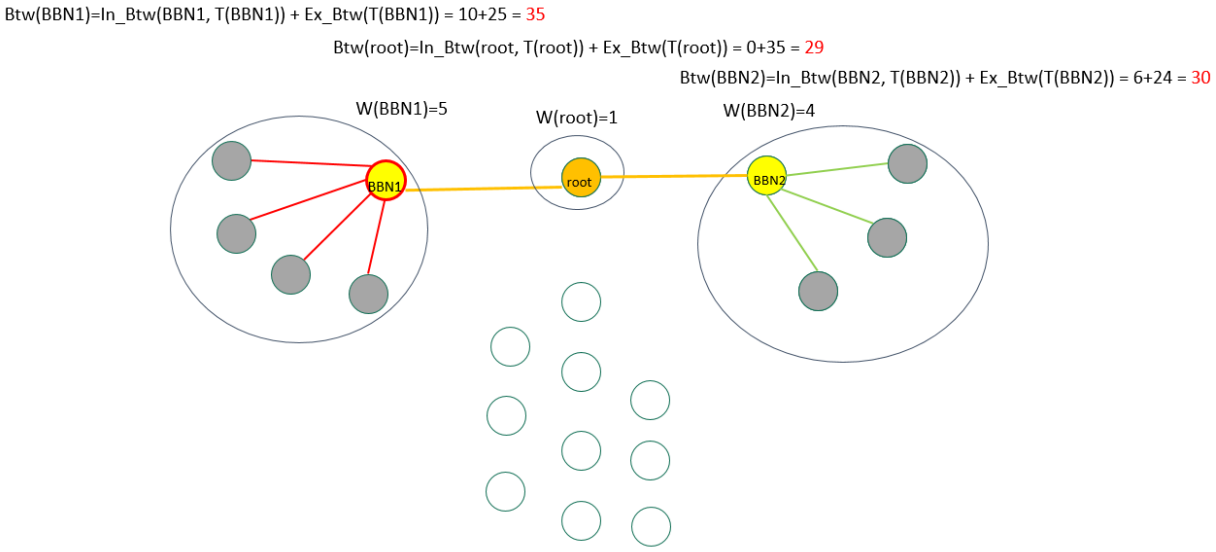


Figure 23(f)

As a result, Figure 23(f) shows that node BBN1 is the BBN for both subgraphs red and green, which is denoted as *BBN12* in the figure. Therefore, in Figure 23(g), we repeat the same procedure to find the global BBN by starting with finding the shortest paths from *BBN12* and *BBN3*. Furthermore, we run DFS from the nodes on the shortest paths, and the DFS stops when the behind of the node is less than that of the local BBN. In Figure 23(h), we include the valid nodes in the candidate set and filter out the invalid nodes. Lastly, we compute the global betweenness of the candidate nodes and find that the root node is the global BBN.

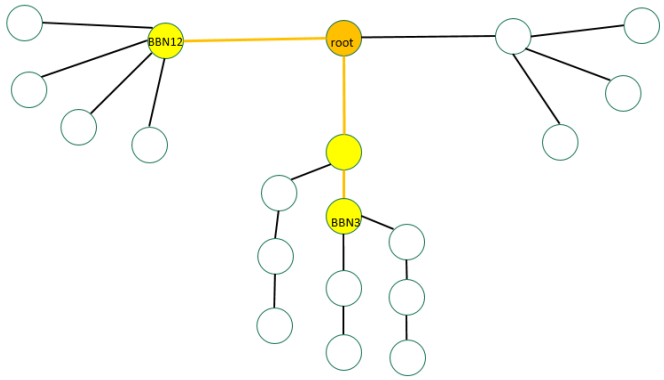


Figure 23(g)

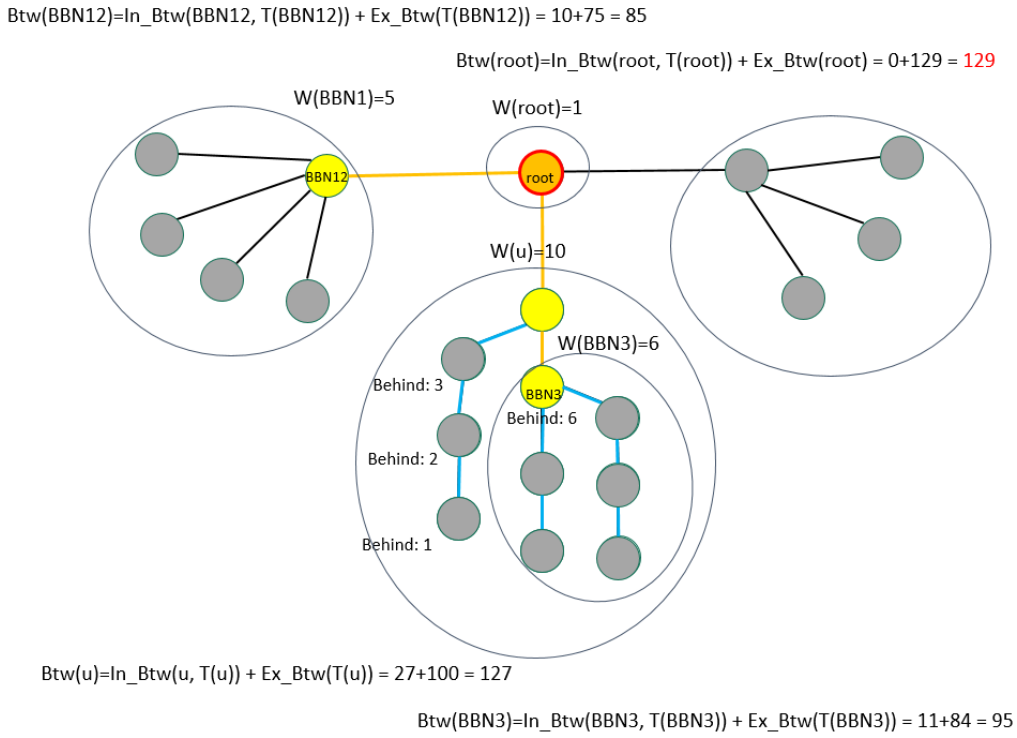
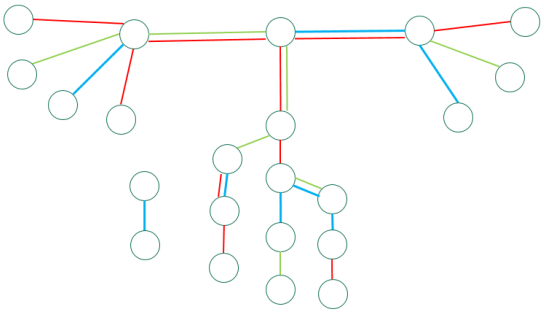
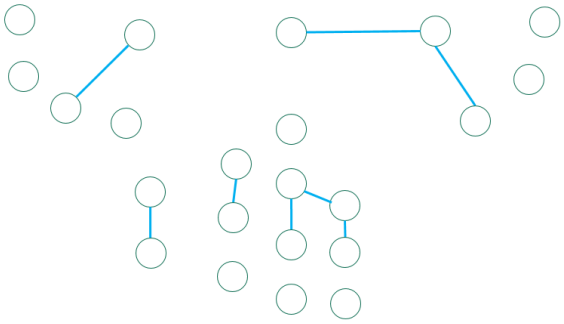


Figure 23(h)

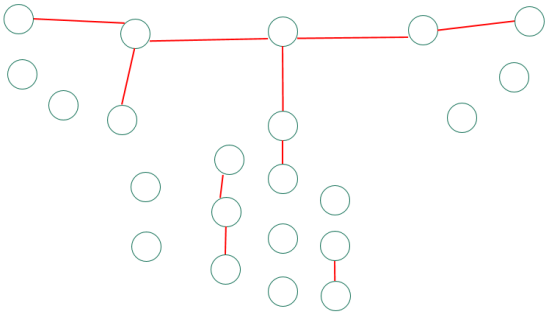
For the same the graph shown in Figure 23, we consider another situation where the colored edges are scattered, such that two adjacent edges rarely share the same color, which is exemplified in Figure 23(i). The three colored (i.e., red, green, and blue) subgraphs are shown in Figure 23(j), (k), and (l). In addition, the local BBNs computed by the traditional betweenness centrality approach are shown in Figure 23(m), (n), and (o) respectively. Since there are more than one cluster in each colored subgraph, the number of local BBNs highlighted in yellow might be more than one. Additionally, there may be two local BBNs sharing the same betweenness centrality values at the same time.



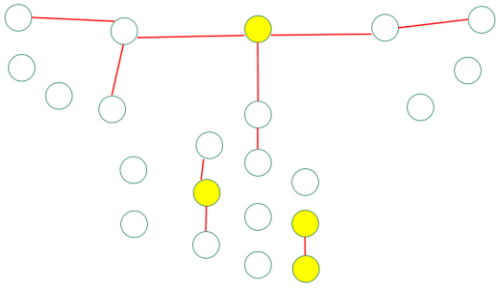
(i)



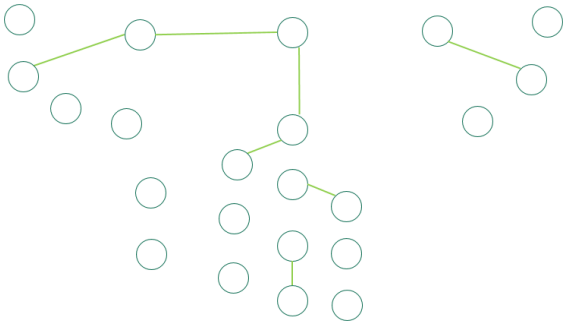
(l)



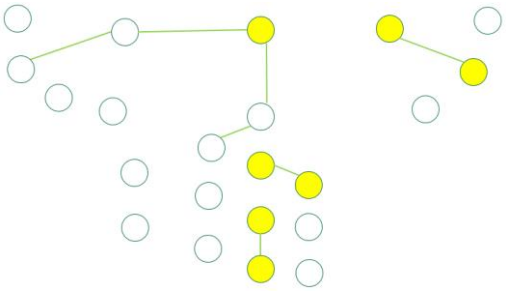
(j)



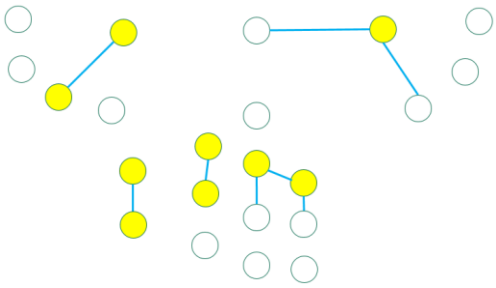
(m)



(k)



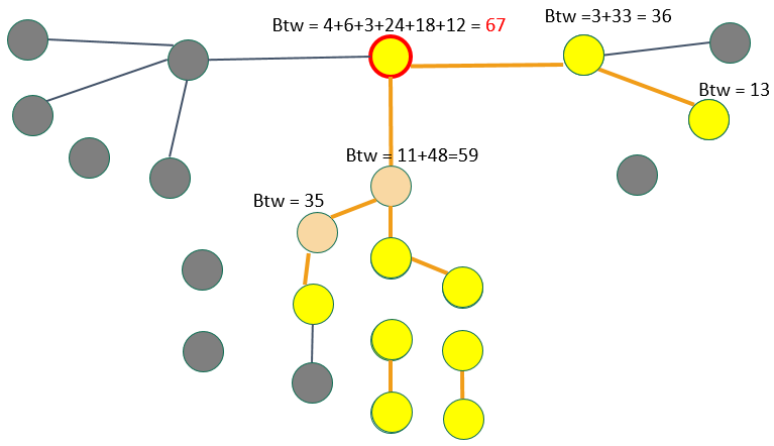
(n)



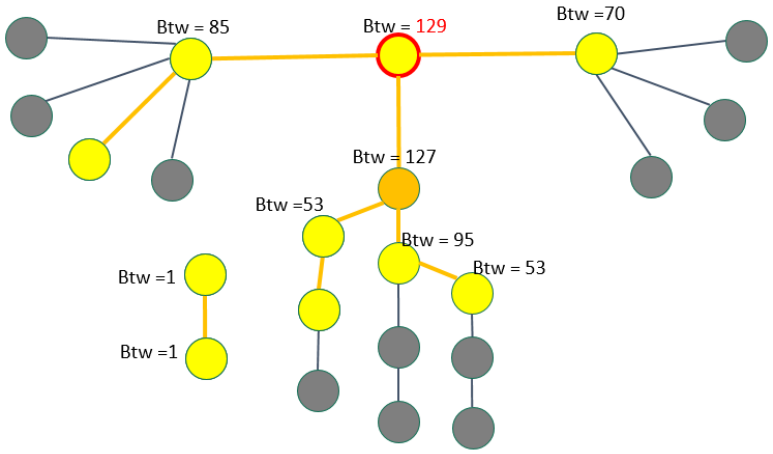
(o)

Next, as shown in Figure 23(p), we find the common BBN in the graph composed of the red subgraph and the green subgraph based on the steps in the SCB algorithm. In this stage, we

compute the common BBN based on the candidate nodes which are the local BBNs shown in yellow and the nodes on the shortest paths between the local BBNs shown in orange. Hence, the yellow node with red outline is the common BBN computed in this graph. In the last step, we merge the graph in Figure 23(p) with the blue subgraph to be the graph in Figure 23(q), and then follow the same procedure as shown in Figure 23(p) to find the global BBN of the composed graph. As a result, the yellow node with red outline is the global BBN in this graph. By observation, we can find that the answer is the same as the graph shown in Figure 23(h) discussed earlier.

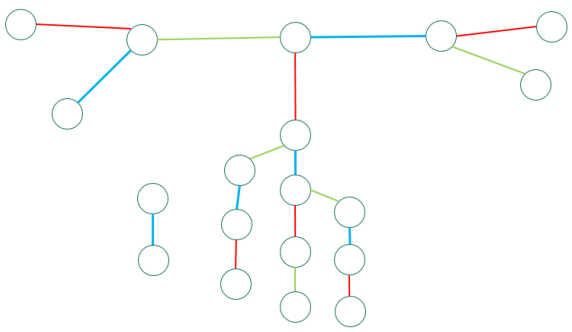


(p)

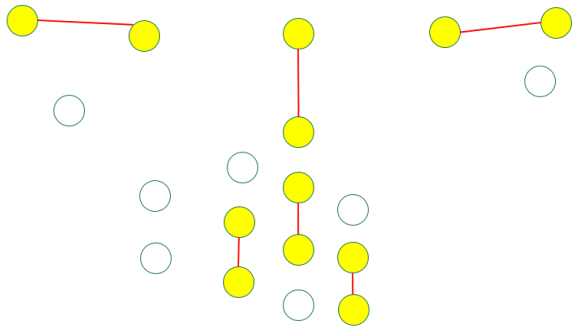


(q)

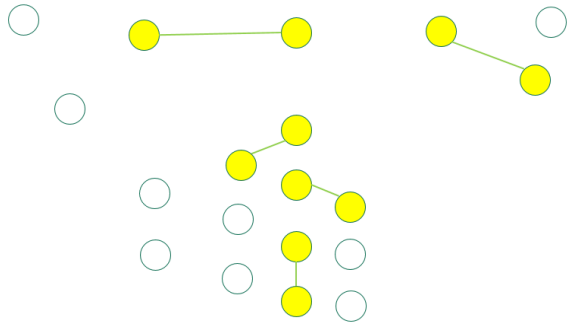
We show a more extreme case where every two adjacent edges do not share the same color. In Figure 23(r) with the same topology but different colored edges, we do the same procedure to find the local BBNs in each colored subgraphs as shown in Figure 23(s), (t), and (u). Basically, we can observe that in such extreme case, each node that is connected with a colored edge in each subgraph is one of the local BBNs. We then combine the red and green subgraphs as shown in Figure 23(v). Since all the local BBNs should be kept as candidates, every node in the aggregated graph composed of red and green will be kept as a candidate. Therefore, it requires more computational time since there is no node can be excluded. The common BBN for both colors is shown in Figure 23(w). The next step is to combine the blue subgraph as shown in Figure 23(x), and the result of the global BBN is shown in Figure 23(y). As a result, we may observe that only few nodes are excluded during the merging step as in Figure 23(v) and (x), so that the computational time of finding the global BBN could be longer than the traditional approach since we have to go through almost every node when merging every two colored subgraphs, and the computational time is proportional to the number of colors *k*.



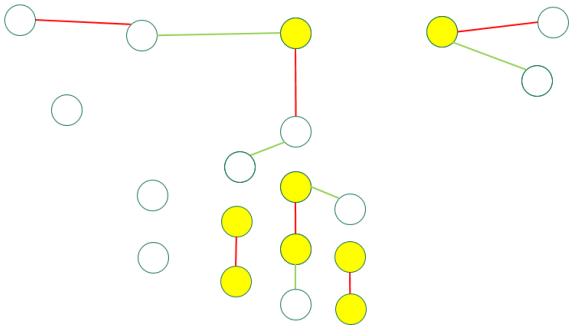
(r)



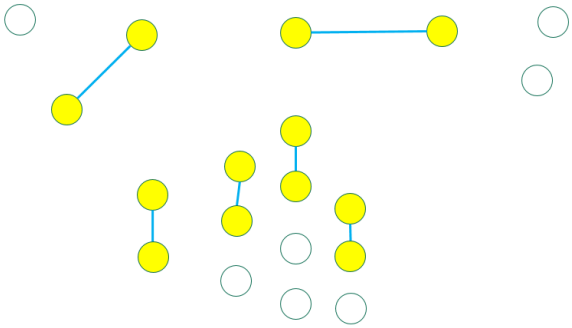
(s)



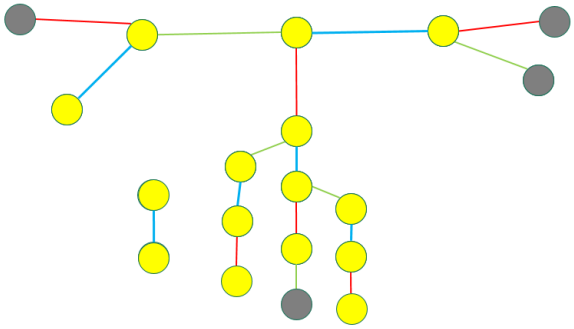
(t)



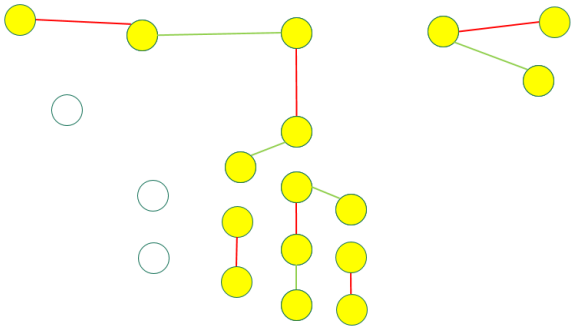
(w)



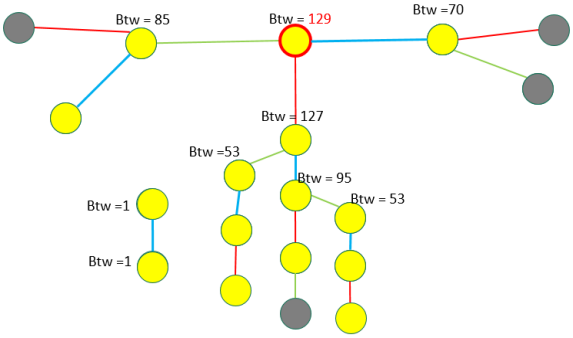
(u)



(x)



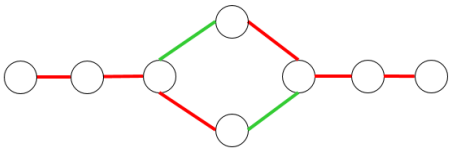
(v)



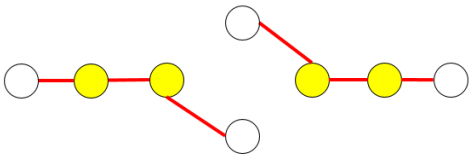
(y)

In the next example, we demonstrate how to deal with cycles containing different color edges, as shown in Figure 24(a). We need to detect cycles and transform them into coarse nodes in each colored subgraph as well as the whole graph. After searching all the nodes in the whole graph, which is considered as a colorless graph, each cycle in the graph is transformed into a coarse node,

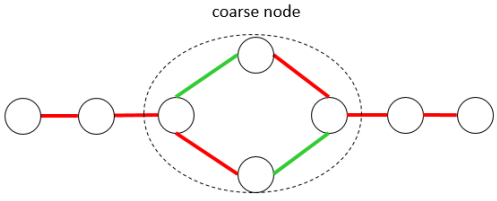
as shown in Figure 24(b). Then, we partition the graph into colored subgraphs with respect to their colors and transform the cycles into coarse nodes, such as the red subgraph in Figure 24(c) and the green subgraph in Figure 24(d). In addition, the local BBNs of both subgraphs are computed based on the same procedure discussed in the previous example, and the local BBNs are highlighted in yellow. Since some of the nodes share the same betweenness centrality values, the number of local BBN might be more than one. Furthermore, since the goal is to find the global BBN, we consider the graph as a colorless graph and find the global BBN of the whole graph in Figure 24(e) based on the candidates highlighted in yellow in Figure 24(c) and (d), including the nodes on the path between the local BBNs and the nodes with more behind nodes. If any of the candidates is inside a coarse node, we use the coarse node instead. In this case, the global BBN is the coarse node highlighted in yellow in Figure 24(e). Since the coarse node is an abstract structure, we need to examine the physical nodes inside the coarse node to figure out the actual global BBN. Therefore, we apply the traditional approach to compute the betweenness centrality of nodes inside the cycle shown in Figure 24(f).



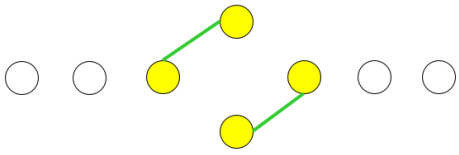
(a)



(c)



(b)



(d)

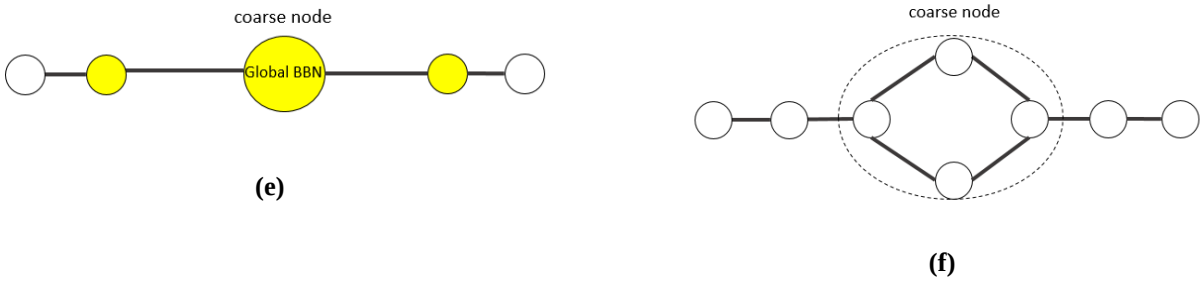


Figure 24. Example 7

5.5. Correctness SCB Algorithm

In the following, we will provide the theoretical foundation to justify the correctness of the SCB algorithm. Especially, we will justify the details of the pruning process.

Before we finish the pruning process and compute the common betweenness of all candidates, we have to assure that no candidate will be left out during the process. As we introduce the idea of coarse node which is an abstract structure, we have to consider the possible nodes inside the coarse nodes.

Consider graph G which contains physical nodes and coarse nodes. The nodes inside any coarse node that is excluded by the candidate set has no chance to be the global BBN. This is because the internal betweenness of the coarse node is computed based on the upper bound $w(c)*(w(c) - 1)/2$ where $w(c)$ is the number of nodes in the coarse node, therefore the internal betweenness of any node inside the coarse node has to be less. On the other hand, any physical node excluded by the candidate set also cannot be the global BBN. Accordingly, the nodes excluded by the candidate set must have less internal betweenness and fewer behind nodes than the local BBN (which is not a coarse node). Thus, their external betweenness must be less than the local BBN. These physical nodes do not have a chance to have a greater global betweenness than the local BBN, and they also cannot be the global BBN.

Accordingly, we compute the common betweenness of each node in the candidate set. After all the irrelevant nodes are excluded, we compute the common betweenness of the candidates. If the common BBN computed is a coarse node, we compute the common betweenness of each node inside the coarse node by summing up their internal betweenness and external betweenness. Subsequently, we can rank all the nodes in the candidate set including all the nodes inside coarse nodes based on their betweenness. If a coarse node is not chosen to be the common BBN, we do not need to examine the nodes inside coarse nodes since these nodes do not have a chance to be the common BBN. We will prove this in the theorem below.

Theorem 2. *Consider graph G which contains physical nodes and coarse nodes. If a coarse node is a candidate but not the common BBN, the nodes inside the coarse node have no chance to be the common BBN, and the common betweenness of the nodes inside the coarse node does not have to be computed.*

Proof of Theorem 2: *Assume that a physical node v is the common BBN that was found by the SCB algorithm and that u is a physical node inside the coarse node c. Node c is a candidate but not the common BBN. According to the assumption, c is a candidate but not the common BBN, so v has higher betweenness centrality than c,*

$$Btw(c, G)' < Btw(v, G) \quad (1)$$

For the left-hand side of (1),

$$Btw(c,G)' = In\_Btw(c,G_j) + w(c)*(w(c)-1)/2 + Ex\_Btw(G_j),$$

because c is a coarse node, and we applied an upper bound for the physical nodes inside c when calculating the local BBNs, which is w(c)(w(c) - 1)/2, where w(c) is the number of nodes in c. Hence,*

Btw(c, G)' = In_Btw(c, Gj)+ w(c)(w(c) - 1)/2 + Ex_Btw(Gj) = In_Btw(c, Gj)' + Ex_Btw(Gj)*

In addition, u is a physical node inside c, and In_Btw(c, Gj)' is the upper bound for the internal betweenness of u in Gj,

Btw(u, G) = In_Btw(u, Gj) + Ex_Btw(u, Gj) <= In_Btw(c, Gj)' + Ex_Btw(Gj) = Btw(c, G)'

Therefore, we have the following equation,

$$Btw(u, G) <= Btw(c, G)'. \quad (2)$$

Based on (1) and (2),

$$Btw(u, G) <= Btw(c, G)' < Btw(v, G),$$

Hence, Btw(u, G) < Btw(v, G) ■

In conclusion, all the nodes (physical and coarse nodes) excluded by the candidate set cannot be the common BBN for the two subgraphs. In addition, the common betweenness of all the candidates can be computed by Formula (4). Therefore, the physical node ranked the highest in the candidate set is indeed the common BBN.

5.6. Complexity and Hierarchical SCB

The SCB algorithm applies the divide-and-conquer methodology, and it merges the result of two subgraphs to find the common BBN. In the conquering stage, we find all the local BBNs for all the subgraphs G_j, j = 1 to k; while in the merging stage, every time we merge two subgraphs by locating the common BBN of both subgraphs, and the common BBN would be either on the path between both local BBNs or with a greater behind set. The concept is also illustrated with an example in Figure 25. As a result, after merging the last two subgraphs, the global BBN can be found.

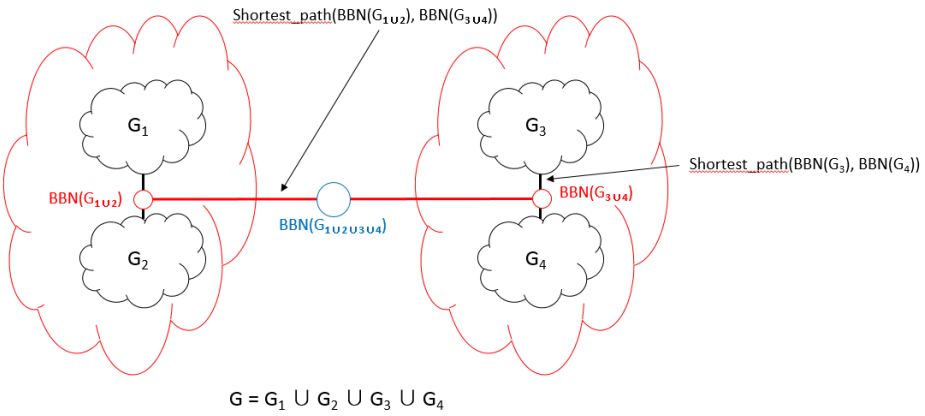


Figure 25. Divide-and-conquer

In the paragraph above, we discussed the situation that given a graph composed of *k* colors, each colored subgraph is a connected component. However, sometimes a colored subgraph might consist of several scattered clusters with the same color, so the number of the scattered clusters might be more than the number of colors *k* in the whole graph. Hence, when we compute the local BBNs, we need to consider all the local BBN for all clusters. Next, we will show that among multiple local BBNs, the common BBN found is indeed the BBN with the highest betweenness centrality.

Consider a graph G which is a tree composed of two colored-subgraphs G_1 and G_2. G_1 has at least m local BBNs since it is composed of m isolated clusters, while G_2 has at least n local BBNs since it is composed of n isolated clusters. According to the SCB algorithm, all the local BBNs in the m+n clusters will be put in the candidate set. Next, we will find all the nodes on the shortest paths between every two local BBNs, and the number of these shortest paths is $O((m+n)^2)$. After these nodes on the paths are added to the candidate set, we will prune the process by acquiring more nodes which has more behind nodes than any of the local BBNs. If a node is not included in the candidate set, it implies that it has less behind nodes than the local BBN of the cluster, and it

will have less common betweenness than the local BBN. Furthermore, we will compute the betweenness centrality of all the candidate nodes by formula (4) in the SCB algorithm. As a result, the common BBN found based on SCB between G_1 and G_2 is indeed the common BBN. If the whole graph is not connected meaning some of the clusters are isolated, according to the SCB algorithm, after all the BBNs of the clusters are found, they will be added to the candidate set. Therefore, the global BBN of graph G will be the one with the highest betweenness centrality.

As discussed earlier in Section 3.2, there are multiple topologies for the colored subgraphs, such as isolated, partially overlapped, complete overlapped, and connected but not overlapped. On one hand, assuming the graph is composed of k colored subgraphs which are connected but not overlapped or isolated, it takes $O((mn+n^2\log n)/k^2)$ as the computational time in each subgraph, so it takes $O((mn+n^2\log n)/k)$ to process k subgraphs. It is observed that every time it requires additional $O(n+m)$ to merge two subgraph and find the common BBN which could be done by using BFS and DFS. In addition, it requires $O(l*(n+m))$ to merge in each layer of the structure in the process of merging, where $l = \log_d k$ is the number of layers, and d is the maximum outgoing degree of each layer which is "two" in this approach. Therefore, the total time required is $O((mn+n^2\log n)/k + l*(n+m))$. The speedup is about $O(k)$ of the traditional approach as the running time of the first part $O((mn+n^2\log n)/k)$ dominates. As a result, the SCB has a better performance than the traditional approach, and the SCB reduce the computational time with respect to the number of colors k in the original graph.

On the other hand, if the original graph is composed of k subgraphs which are partially overlapped, the speedup will be less than $O(k)$ since the size of the original graph is less than the sum of the k subgraphs, and it requires additional computational time for the overlapping parts. If the k subgraphs are completely overlapped where every pair of nodes have exactly k colored edges,

the total computational time for the *k* subgraphs might be *k* times the computational time for the original graph.

5.7. Colored-Node Graphs

As discussed earlier in Section 4, another property of graph is node's color, so that the nodes in the graph can be colored differently. In this type of graph, there are colored nodes instead of colored edges. The definition of the colored-node problem is to find the node with the highest betweenness centrality in a graph where all the nodes belong to specific colors. In other words, we can consider this problem as if we drop the nodes with the other colors, and then we only keep the edges which have valid nodes on both ends, which are defined as the internal edges. The edges which have only one valid node on one side are defined as bridge edges. Therefore, when we take a closer look at an extracted colored subgraph, we can observe that if two nodes are still connected through one edge which is an internal edge, there will be a path between the two nodes, which should be counted towards the betweenness for both nodes. Otherwise, there is no path between them if the two nodes are disconnected.

For colored-node graphs, we can also apply the same SCB algorithm which is applied to colored-edge graphs discussed earlier. According to the definition of colored-node problem stated above, we define all the edges which have its two end nodes sharing the same color as internal edges, while the edges which have different colors on both ends are bridge edges. We can convert the colored-node problem to colored-edge problem by the flowing steps:

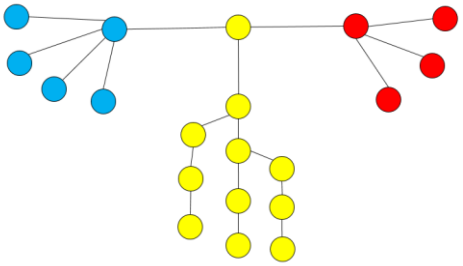
1. Convert the colorless edges into colored internal edges and bridge edges. The internal edges are the edges which have two nodes sharing the same color, while the bridge edges are those have different colors on both ends.

2. Each colored-edge subgraph is only composed of the internal edges associated with the specific color. The bridge edges which do not belong to any colored subgraph form the bridge subgraph.
3. Find the global BBN based on the local BBNs of the colored-edge subgraphs by applying the SCB approach.
4. Remove the colors of the nodes.

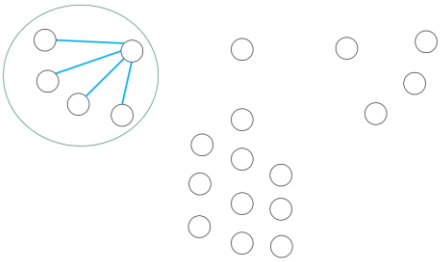
In addition to the transformation from the colored-node problem to the colored-edge problem, the queries of colored-node problem are also transformable into colored-edge queries. For example, for the colored-node problem, we can have the query "find the most important disease in the context of the cardiovascular disease group and the immune system disease group," where each node is a disease, red nodes belong to the cardiovascular disease group, and blue nodes belong to the immune system disease group. This problem can be transformed into another version with respect to colored-edge problem, such as "finding the most important disease node on the subgraph composed of cardiovascular diseases and immune system diseases," where nodes are not colored, and an edge is colored with red if the connected two diseases belong to the cardiovascular disease group. The same can also be applied to the immune system disease group. We provide examples below to demonstrate the transition between colored-node graphs and colored-edge graphs

In the colored-node graph shown in Figure 26 (a), there are three different colors, blue, red, and yellow. First, we convert the color-blind edges into colored edges. For the edges connecting two nodes which share the same color, they are categorized as internal edges and colored with the same color of the two nodes, as shown in Figure 26(b). The edges connecting two nodes with different colors are categorized as bridge edges, and they are assigned with a new color which is not duplicated with any other colors in the graph. In this example, the bridge edges connecting a

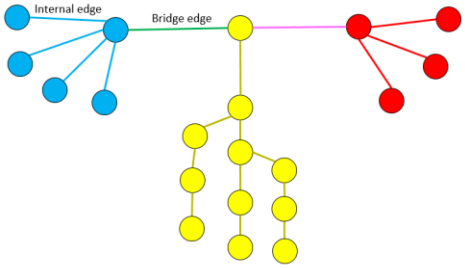
blue node and a yellow node is colored as green; and the bridge edges connecting a yellow node and a red node is colored as pink. Now, we have three colored-edge subgraphs, blue, red, and yellow as shown in Figure 26(c), (d), and (e) correspondingly. When we query betweenness centrality in each of the three colored-edge subgraphs, we can find the local BBN in each subgraph by executing the conventional approach. However, what is missing is the information about the bridge edges, since bridge edges are also essential when finding the global BBN. Hence, we will have one more bridge graph which is composed of all the bridge edges. Therefore, we can transform a colored-node graph into a colored-edge graph, and the total number of subgraphs is the number of colors plus one, which represents the bridge subgraph. The global BBN of the graph will be computed by the SCB algorithm based on the local BBNs in the four colored-edge subgraphs.



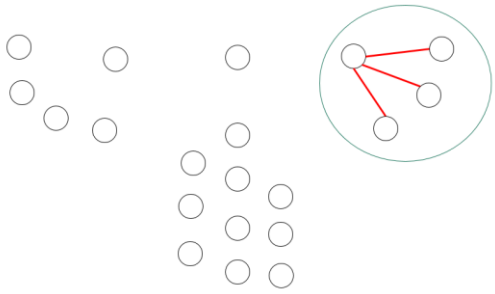
(a)



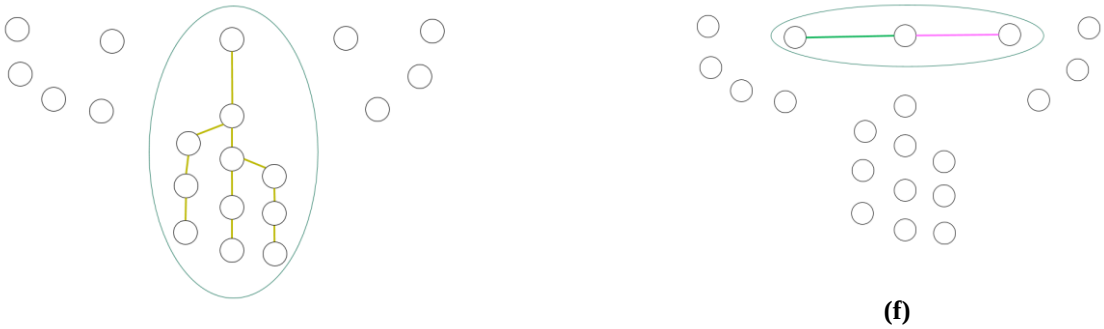
(c)



(b)



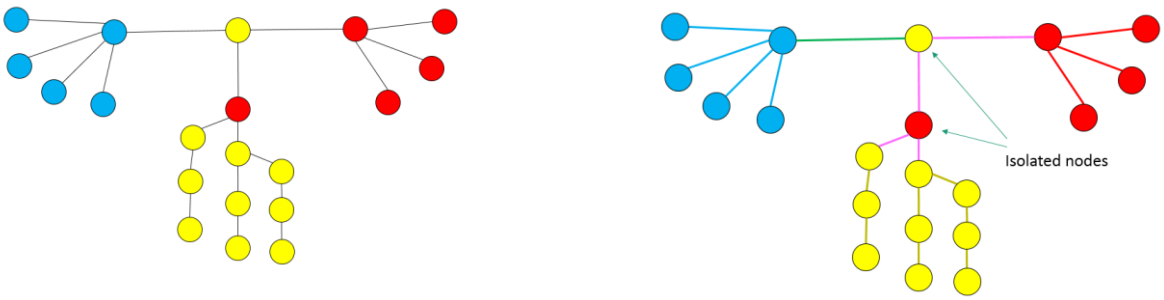
(d)



(e) **(f)**

Figure 26. A sample transformation of colored-node graph

In the next example in Figure 27(a), we show that the color nodes might be scattered as exemplified in Figure 27(b). These nodes are defined as isolated nodes which do not have neighbors having the same color. When transforming the graph into a colored-edge graph as shown in Figure 27(c), for example, the isolated red node will be left out from the red-edge subgraph, since the red node does not have any internal edge connected. This also happens for the isolated yellow node. Although the isolated nodes are left out in the corresponding color subgraphs, they do not affect the computation of local BBN in each subgraph since the isolated nodes have zero betweenness centrality in each colored-edge subgraph. In addition, the bridge edges will be collected in the bridge subgraph. For example, the yellow subgraph shown in Figure 27(c) has two scattered clusters, so that the number of local BBNs can be more than the number of colors.



(a) **(b)**

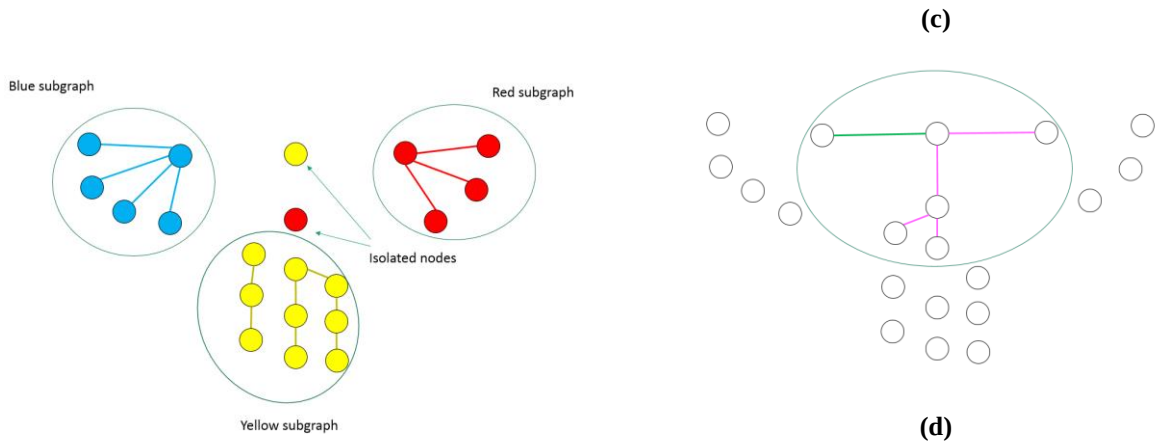


Figure 27. Another sample transformation of colored-node graph

According to the discussion (multiple isolated clusters), the global BBN is computed based on all the local BBNs. As a result, the global BBN of the graph is computed by the SCB algorithm based on the local BBNs from the colored-edge subgraphs.

Since the SCB approach is a divide-and-conquer approach and it merges the information based on two colored-edge subgraphs, we can extend the discussion above to a colored-node graph composed of more colors than two and verify the correctness of applying SCB for the colored-node problems.

Compared to the SCB algorithm on colored-edge graphs, the difference is that the colored-node graphs will have one more subgraph which is the bridge subgraph which covers all the bridge edges. Given k colors in a colored-node graph, the colored-node problem is computed based on k+1 subgraphs. In addition, there might be multiple scattered clusters in each subgraph, and assume the number of additional clusters is d. According to edge-based SCB algorithm, the global BBN of the colored-node problem can be computed based on k+d+1 local BBNs. We will prove in the next theorem that for colored-node graphs, the global BBN found by using the SCB algorithm is indeed the global BBN.

Theorem 3. *Consider a colored-node graph G containing multiple clusters. G contains k colored-node subgraphs G_1, $G_{2, \ldots}G_k$. The global BBN found is indeed the global BBN of the colored-node graph G.*

Proof of Theorem 3: *We prove this by induction.*

Assume x is number of colors in G, M_i is an arbitrary number of clusters in each subgraph G_i, and R_j is an arbitrary number representing the number of clusters in the graph that is composed of j subgraphs.

Step 1: x=1.

G contains only one color and has one subgraph G_1, so that G can be considered as a colorless graph. The number of clusters in G is M_1, and $R_1 = M_1$ because G is composed of only one colored subgraph, so the number of isolate clusters in G is equal to that in G_1. According to the discussion of colored-node problem, the common BBN computed by SCB for G which has one color is indeed the global BBN for colored-node graph G. The theorem holds for x = 1.

Step 2: x=2.

G contains two colors and has two subgraphs G_1 and G_2. Transforming G results in G' that is composed of G_1' and G_2'. There are M_1 cluster in G_1' and M_2 clusters in G_2'. According to the discussions in SCB, the global BBNs on colored-edge graph G' with R_2 clusters are computed. In addition, the discussion of colored-node problem, the answer to the colored-edge version of G' is indeed the answer to the colored-node graph G. Hence, the theorem holds for x = 2.

Step 3: Assume this theorem holds for x = k.

Hence, G has subgraphs G_1, $G_{2, \ldots}G_k$. There are M_1, M_2, … M_k clusters in G_1, $G_{2, \ldots}G_k$ respectively, and assume the results of global BBNs on G with R_k clusters are correct.

Step 4: $x = k+1$.

Introducing another colored subgraph G_{k+1} with M_{k+1} clusters into G. Since there are R_k clusters in the aggregated graph composed of G_1, $G_{2, \ldots}G_k$ in step 3, we can merge the result graph in step 3 with the subgraph G_{k+1} as if we are merging two colored subgraphs with $R_k + M_{k+1}$ isolated clusters. According to step 3, the global BBN can be found because the conditions are the same as in step 3. The theorem holds for $x = k+1$.

AS a result, the global BBN found is indeed the global BBN of the colored-node subgraph G.

5.8. Colored-Node-Edge Graphs

Colored-node-edge graphs integrate colored-edge graphs and colored-node graphs. We can transform the colored-node-edge problem into the colored-edge problem through the flowing steps:

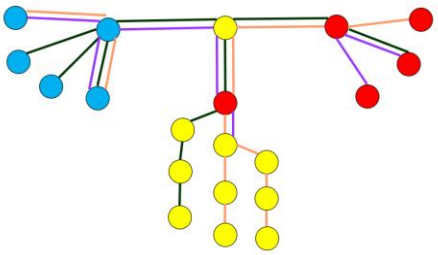
1. Assign a new color to an edge color if the edge color duplicates that of a node color.
2. Assign colored internal edges based on the colors of the nodes. For example, assign blue internal edges to the pair of nodes where both nodes are blue nodes and are connected in the original graph.
3. Assign colored bridge edges with new colors. For example, assign bridge edges with a new color (i.e., green) to an edge that connects a blue node and a yellow node.
4. Convert the colored nodes into colorless nodes, and all the colored nodes are transformed into colored edges.
5. Find the global BBN based on the local BBNs of colored-edge subgraphs by applying the SCB approach.

In addition to the transformation from the colored-node-edge problem to the colored-edge problem, the queries of colored-node-edge problem are also transformable into colored-edge

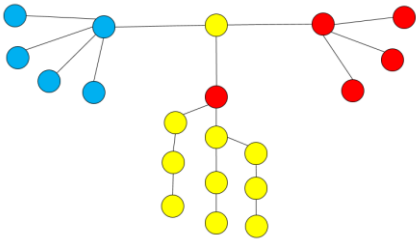
queries. For example, consider a graph in which the nodes are genes and two nodes are connected if they are correlated in a disease. If two genes are correlated in multiple diseases, a set of links are created where each link has a different color and has an assigned edge-weight. Additionally, colors of nodes represent different organs, so that the genes have the same color if they are related to a specific organ. A colored-node-edge query can be "Find the most important gene related to the brain among three types of diseases, such as brain cancer, cardiovascular disease and neurological disorder." This query can be transformed into a colored-edge query, such as "Find the most important gene among four types of human traits subgraphs, including human brain, brain cancer, cardiovascular disease and neurological disorder." We demonstrate the procedure of transforming the problem into colored-edge problem by an example below.

In the colored-node-edge graph shown in Figure 28(a), there are three colors for nodes: red, blue, and yellow, and three colors for edges: black, purple, and orange. In order to transform this problem, we convert the colored nodes in Figure 28(b) into colored edges by assigning new colors which are not duplicated to the colors of edges. In Figure 28(c), we have three colors for internal edges in which both nodes share the same color; and we assign new colors to the bridge edges where both nodes do not have the same color, such as green and pink. Hence, there are three colored-edge subgraphs shown in Figure 28(d) and one more bridge subgraph in Figure 28(e). Next, we add the original colored edges, i.e., black, purple, and orange in the original graph shown in Figure 28(f). As discussed in the colored-edge graph section, we simply partition the graph into three subgraphs based on the edge colors, as shown in Figure 28(g), (h), and (i). As a result, all the colored nodes and colored edges are transformed into colored edges, and the original graph is partitioned into 7 subgraphs with respect to their colors. This problem now becomes the colored-

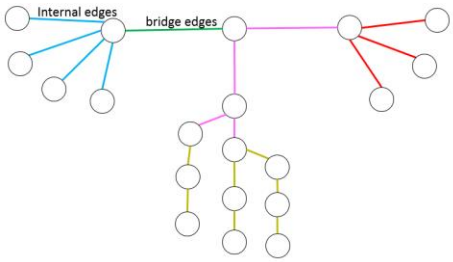
edge problem as shown in Figure 28(j), and we can apply the SCB to the colored-edge graph to compute the global BBN.



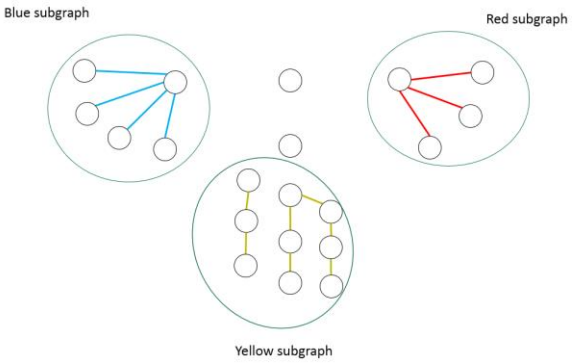
(a)



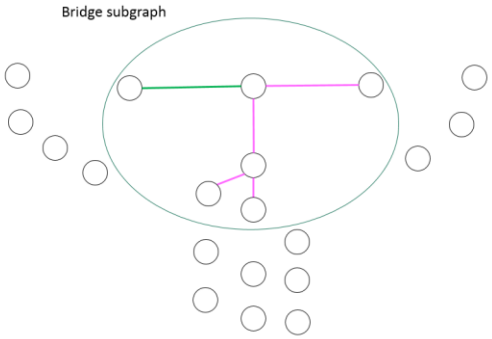
(b)



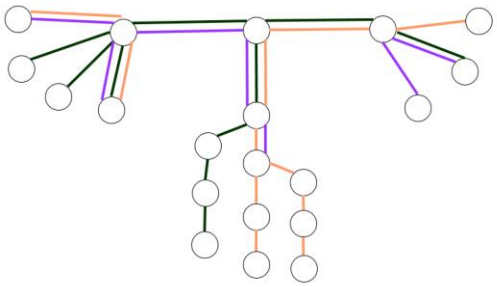
(c)



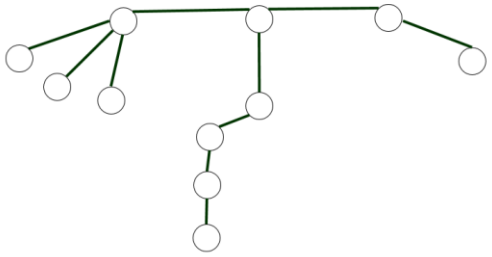
(d)



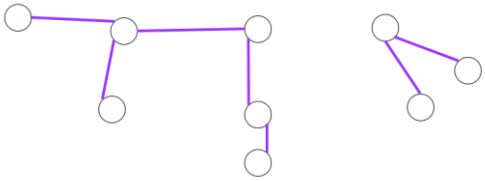
(e)



(f)



(g)



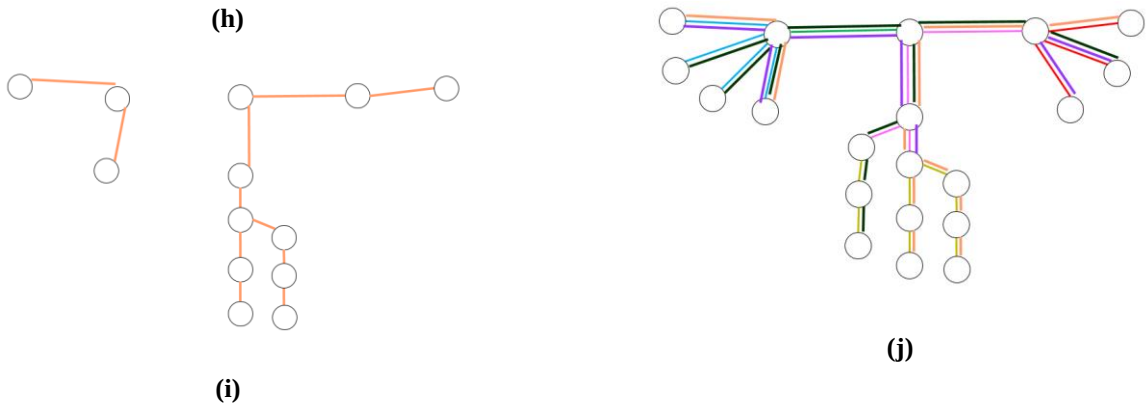


Figure 28. A sample transformation of colored-node-edge graph

CHAPTER 6 The Query-Based Graph Reduction Problem (QGRP)

The query-based graph reduction problem (QGRP) can be defined as follows. Given a weighted graph *G* and a set of queries *Q*, find a reduced graph *G'* such that *G'* is a smallest subgraph of *G* with which for every *q* of *Q*, *q(G) = q(G')*, where *q(G)* is the answer to query *q* in *G*, and *q(G')* is the answer to *q* in *G'*. That is, we want to find a smallest reduced graph *G'* so that it can still answer all the queries for a specific graph problem correctly.

A separate QGRP can be defined for each graph problem. Given a graph *G*, assume an application is only interested in finding the shortest paths between any pair of nodes. Can we reduce *G* to a smaller graph *G'* which preserves all the shortest paths between any pair of nodes? We call this problem the GRSP (Graph Reduction Shortest Path) problem. Our GRSP algorithm based on Dijkstra's shortest algorithm is given in Appendix A.

6.1. The Lossy Graph Data Reduction Problem (LGRP)

The Lossy Graph Reduction Problem (LGRP) can be defined as follows. Given a graph G, a graph problem P, and a set of possible queries Q for the graph problem P, find a smallest reduced graph G' whose size is smaller than G that is able to answer the corresponding queries (in the original form or in a transformed form) in Q' with the error rate bounded by a constant ε.

Formally the goal of an LGRP algorithm can be described as:

1. For any $q \in Q$, the corresponding query $q' \in Q'$ can be computed.
2. $q(G) = q'(G')$, where $q(G)$ is the result of q computed from G (e.g., a shortest distance, a Boolean value, etc.) In other words, ε is zero.

The information lost in lossy graph data reduction can thereby be defined in two types:

1. Some queries cannot be answered by G': There exist some queries $q \in Q$ that cannot be computed from G'.
2. Query results may have some error: There exist some queries that $q(G) \neq q'(G')$.

6.2. Methods for Lossy Graph Data Reduction (LGR)

If an application can tolerate bounded errors for some queries in exchange for graph reduction, the reduction is called $1+\varepsilon$ lossy reduction, where ε $(0 < \varepsilon < 1)$ is the bound for the ratio of errors.

6.2.1. LGRSP

Given a graph G, assume an application is only interested in finding a shortest path between any pair of two nodes. A lossy reduction method not only reduces a graph to the smallest subgraph G' that preserves all the shortest paths between any pair of nodes but also further reduces G' to an even smaller subgraph G'' which preserves most of the shortest paths within a pre-specified tolerance. We call this problem the LGRSP (Lossy Graph Reduction-Shortest Path) problem.

6.2.2. The $1+\varepsilon$ LGRSP Algorithm

The basic idea of our 1+ε LGRSP algorithm is to remove the edge between any two nodes if we can find another shortest path connecting the two nodes whose total weight is less than or equal to 1+ε times the weight of the edge. If the path is shorter than the edge, it is safe to remove the edge without causing any error. If the path is longer, the error rate is bounded by 1+ε, where ε can be decided by the user when applying the algorithm. The details of the 1+ε LGRSP algorithm are discussed in Appendix B. This study demonstrates the usefulness of the 1+ε LGRSP algorithm in the context of biomedicine.

6.3. Integrating Graph Decomposition and Graph Reduction

The graph decomposition approach and the graph data reduction approach can be integrated to deal with very large graphs. We showed that GRSP and LGRSP can be used to reduce graph data to compute the betweenness centralities locally. They can be applied to reduce the irrelevant graph data when preserving the same (or nearly the same) result of querying the shortest path problem. After the local betweenness centralities are computed, we can compute the global betweenness centrality from the local values.

CHAPTER 7 Experiments and Discussion

As the advantage of the graph decomposition approach is obvious, to evaluate the efficiency of the graph reduction algorithms and the scalable algorithm, we experimented the SBC, SCB, GRSP, and LGRSP algorithms with several genetic datasets.

7.1. Setup

7.1.1. Experiment Environment

Database	Neo4j Community Edition: 3.0.6
Language	Java (jdk-8u02)
IDE	Eclipse (neon)
OS	Windows server 2012
CPU	Intel Xeon E5-2670 v3, 2.30GHz
RAM	256 GB

7.1.2. Evaluation Function

We define the reduction ratio and speedup as follows:

$$\text{ReductionRatio} = 1 - \frac{Num\ of\ Edges\ after\ Reduction}{Num\ of\ Edges\ before\ Reduction}, \quad (1)$$

$$\text{Speedup} = \frac{Execution\ Time\ of\ shortest\ path\ before\ Reduction}{Execution\ Time\ of\ shortest\ path\ after\ Reduction}, \quad (2)$$

7.1.3. Datasets

Correlation networks have been used increasingly in systems biology applications. For example, gene co-expression network analysis is a systems biology method for describing the correlation patterns among genes across microarray data. Correlation network analysis can be used for finding clusters of highly correlated genes. In addition, correlation networks facilitate network-based gene screening methods that can be used to identify candidate biomarkers or therapeutic targets. These methods have been successfully applied in various biological contexts, for example, Tang [28] use gene co-expression datasets from the GEO database to screen potential biomarkers related to the progression and prognosis of breast cancer. In this study, we use co-expression networks, which represent a major application of the correlation network methodology.

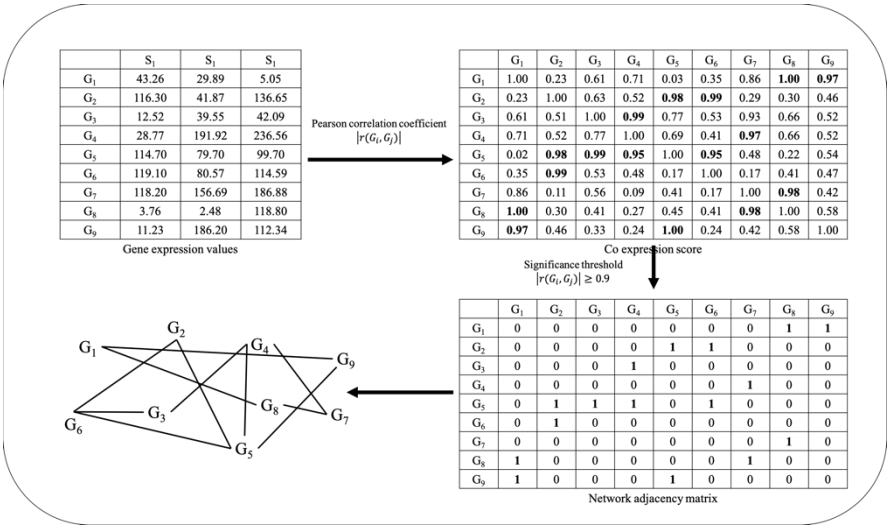
The workflow of the gene expression data process in this study is shown in Figure 29. The gene expression profiles of GSE48216 submitted by Daemen A et al. [29] is downloaded from the Gene Expression Omnibus (GEO) database. The GSE48216 is an expression profiling based on GPL10999 Illumina Genome Analyzer IIx (Homo sapiens) and contains 56 samples. It includes 56 breast cancer cell lines which are profiled to identify patterns of gene expression associated with subtypes and responses to therapeutic compounds. The Robust Multi-Array Average (RMA) algorithm in the affy package within Bioconductor in R is used to preprocess the gene expression profile data. After background correction, quantile normalization, and probe summarization, the dataset with 15,485 genes is further processed, and the top 50% most variant genes by analysis of variance (9,133 genes) are selected for WGCNA analysis [30].

The expression data profile of these 9,133 genes are constructed into a gene co-expression network using the WGCNA package in R. The network is composed of 4,109,069 edges and 17,055 nodes.

The adjacency matrix a_{ij} corresponding to the connection strength between each pair of nodes is calculated as follows:

$$s_{ij} = correlation\,(x_i, x_j)|a_{ij} = S_{ij}^{\beta}\,, \tag{3}$$

where X_i and X_j are vectors of expression value for genes i and j, s_{ij} represents the Pearson correlation coefficient of gene i and gene j, and a_{ij} encodes the network connection strength between gene *i* and gene *j*.



	S1	S1	S1
G1	43.26	29.89	5.05
G2	116.30	41.87	136.65
G3	12.52	39.55	42.09
G4	28.77	191.92	236.56
G5	114.70	79.70	99.70
G6	119.10	80.57	114.59
G7	118.20	156.69	186.88
G8	3.76	2.48	118.80
G9	11.23	186.20	112.34

	G1	G2	G3	G4	G5	G6	G7	G8	G9
G1	1.00	0.23	0.61	0.71	0.03	0.35	0.86	**1.00**	**0.97**
G2	0.23	1.00	0.63	0.52	**0.98**	**0.99**	0.29	0.30	0.46
G3	0.61	0.51	1.00	**0.99**	0.77	0.53	0.93	0.66	0.52
G4	0.71	0.52	0.77	1.00	0.69	0.41	**0.97**	0.66	0.52
G5	0.02	**0.98**	**0.99**	**0.95**	1.00	**0.95**	0.48	0.22	0.54
G6	0.35	**0.99**	0.53	0.48	0.17	1.00	0.17	0.41	0.47
G7	0.86	0.11	0.56	0.09	0.41	0.17	1.00	**0.98**	0.42
G8	**1.00**	0.30	0.41	0.27	0.45	0.41	**0.98**	1.00	0.58
G9	**0.97**	0.46	0.33	0.24	**1.00**	0.24	0.42	0.58	1.00

	G1	G2	G3	G4	G5	G6	G7	G8	G9
G1	0	0	0	0	0	0	0	**1**	**1**
G2	0	0	0	0	**1**	**1**	0	0	0
G3	0	0	0	**1**	0	0	0	0	0
G4	0	0	0	0	0	0	**1**	0	0
G5	0	**1**	**1**	**1**	0	**1**	0	0	0
G6	0	**1**	0	0	0	0	0	0	0
G7	0	0	0	0	0	0	0	**1**	0
G8	**1**	0	0	0	0	0	**1**	0	0
G9	**1**	0	0	0	**1**	0	0	0	0

Figure 29. Data preparation and co-expression network construction.

7.2. SBC on Colorless Graphs

In this section, we show the results of the proposed SBC (scalable betweenness centrality) approach which can enhance the performance of betweenness centrality algorithm. In order to demonstrate the performance of the SBC approach, we also do experiments on a subset of the breast cancer dataset which contains 10,000 edges and 3720 nodes, and we compare the performance of SBC and the conventional betweenness centrality approach proposed by Brandes [21].

7.2.1. Sensitive to *k*

In the SBC algorithm, we randomly select a node in the graph as the root node, and the degree of the root node becomes *k* which is exactly the number of subgraphs. As we can partition the graph into *k* subgraphs, the performance of SBC is sensitive to the number of *k*. However, the number *k* is limited to the degrees of nodes in the graph data, and the maximum *k* is restricted to the maximum degree of the graph. Hence, for this graph data we do experiments based on different *k* values, where $k = \{3,5,7,9\}$. The speedup of SBC can be as large as 2.55 times the Brandes'

approach when *k* increases to nine. The result of execution time is shown in Figure 30, and the result of speedup is shown in Figure 31.

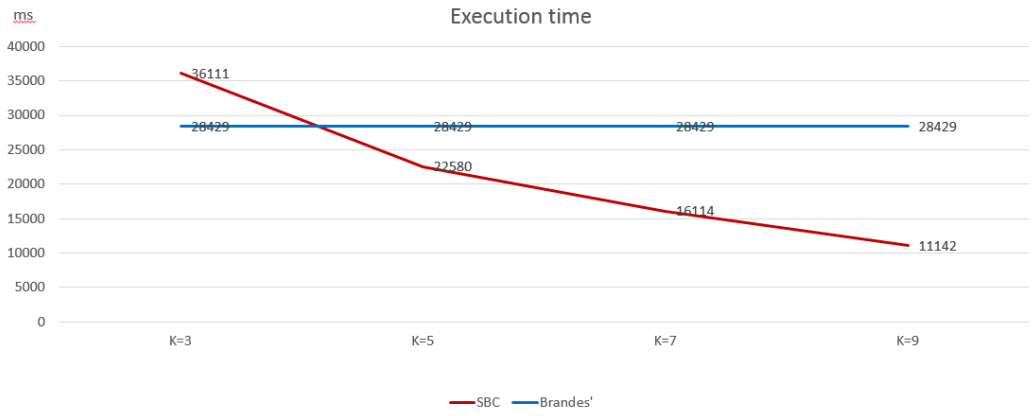


Figure 30. Execution time of SBC and Brandes' approach when *k* increases.

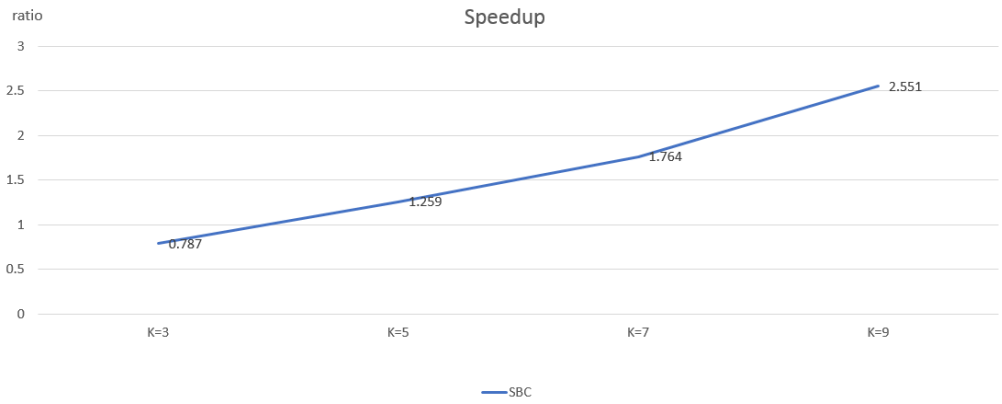


Figure 31. Speedup of SBC compared to Brandes' approach when *k* increases.

7.2.2. Very Large Graphs

From the experiment above, we can observe that the speedup is more eminent when *k* becomes greater. However, as the total size of the breast cancer dataset is moderate, we need a larger graph to demonstrate the advantages of SBC. Hence, we download the road networks dataset from Stanford SNAP network datasets [31]. This dataset is about the road information in California, USA, which contains 2,766,607 edges and 1,965,206 nodes. Since the size of this graph is huge,

and the graph is a dense graph, we first apply the minimum spanning tree algorithm to find a tree structure of the dataset which contains the minimum weights. We run experiments on a subset containing 10,000 edges, and then gradually increase the data size.

First, for the graph with 10,000 edges, we randomly select the node with the highest degree *k* which is 3 in this graph, so that we have a three first-level subgraphs. The speedup of SBC on this graph is 6.21. Second, we run experiment hierarchically on a larger graph containing 50,000 edges. We partition the graph into four second-level subgraphs, and each second-level subgraph has three first-level subgraphs. As a result, the speedup is as high as 13.55.

We also run an experiment to compare the performance between the single-layered structure and the hierarchical structure. For the single-layered structure, we execute the SBC on the road network with 50,000 edges, and the number of subgraphs *k* is equal to 4; for the hierarchical structure, the graph is partitioned into two second-layered subgraphs, and each second-layered subgraph is partitioned further into two first-layered subgraphs. Hence, the total number of subgraphs is 4 for both structures. As a result, the single-layered structure is 6.29 times faster than the conventional approach, while the double-layered structure has a 4.83 speedup than the conventional approach. Therefore, the single-layered structure has a better performance than the hierarchical structure, and the hierarchical structure has a better performance than the conventional approach. The execution times of the experiment are shown in Figure 32, and the speedup results are shown in Figure 33.

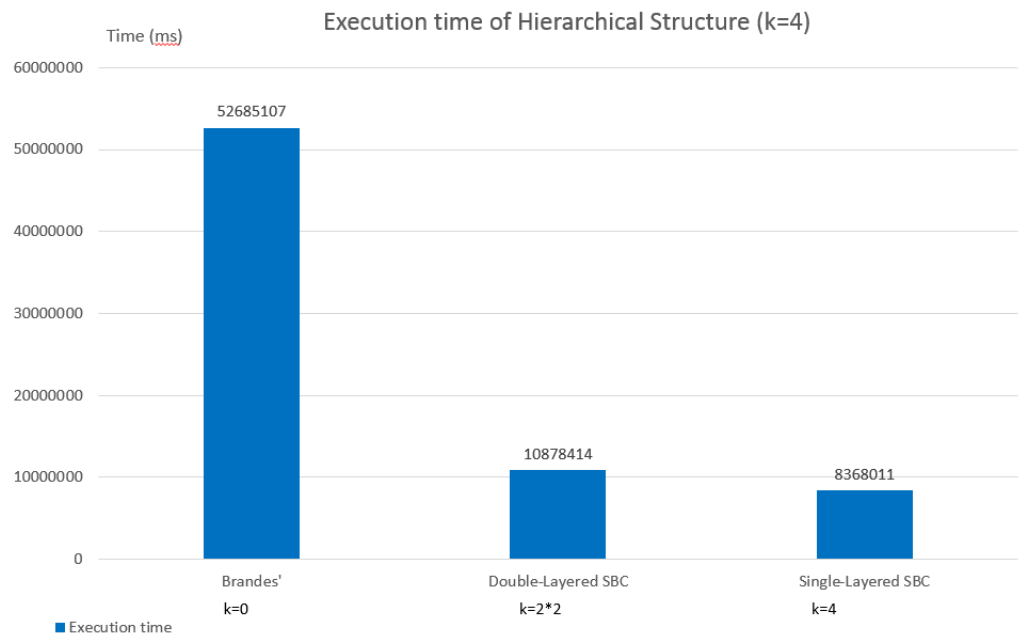


Figure 32. Comparison of the single-layered structure and the hierarchical structure (double-layered).

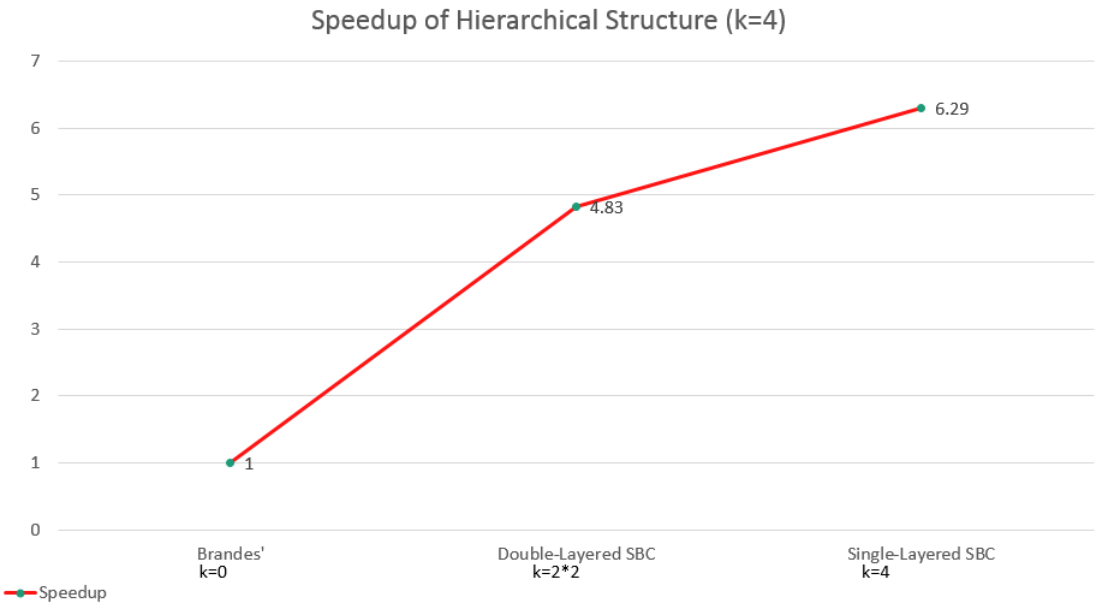


Figure 33. Speedup of the single-layered structure and the hierarchical structure (double-layered).

As we expand the size of the graph to the whole size of the road network containing 2,766,607 edges, we try to run Brandes' approach on this large graph. However, we find that it requires around 24 hours to partially compute all-pair shortest paths for about 5,000 nodes. Thus, the

estimated time to finish the program is beyond 12 months. Since it requires unreasonable amount of time to complete the conventional Brandes' approach on this large graph, we execute the SBC hierarchically on this large graph. We partition the graph into four third-level subgraphs; each third-level subgraph has three second-level subgraphs, and each second-level subgraph has 3 first-level subgraphs. As a result, it spends 116274 seconds (around 32 hours) to complete the SBC on the road network graph.

7.3. GRSP and LGRSP on Large Colorless Graphs

In addition to SBC approach, we discuss the experiment results of graph reduction approaches in this section. We apply both GRSP and LGRSP to the graph corresponding to the dataset in Section 7.1.3 which is composed of 4,109,069 edges and 17,055 nodes.

7.3.1. GRSP

After applying the GRSP algorithm to the graph, 3,184,625 edges are reduced. It takes 2575.4 seconds to finish the betweenness centrality algorithm on the reduced graph, which requires 12252 seconds on the original graph. Thus, the reduction ratio is 77.5%, and the speedup is 4.76.

7.3.2. LGRSP ($\varepsilon = 0.2$)

After applying LGRSP ($\varepsilon = 0.2$) to the graph, the reduction ratio becomes 83.5%, which is higher than that of GRSP, while the speedup can reach 6.1. Since it takes 2009.9 seconds to finish the betweenness centrality algorithm on the reduced graph, which requires 1225.2 seconds on the original graph. Therefore, the ability of LGRSP to speed up is more eminent than that of GRSP.

7.3.3. Evaluation Using Betweenness Centrality

We compute the betweenness centrality of the graph to set a ground truth, and after the reduction algorithm is applied, we compute the betweenness centrality again to compare the difference and compute the accuracy. Since the betweenness centrality algorithm provides a rank

of the most important nodes in the graph, we apply the common rank measure approach that calculates accuracy as the following formula [32], where *tp* stands for the number of true positives, *tn* stands for the number of true negatives, and *n* is the number of items on the list.

$$\text{Accuracy} = (tp + tn) \,/\, n\,, \tag{5}$$

In Table 5, we can see the result for GRSP. We apply the accuracy formula to compare the rank of betweenness for the original graph and the reduced graph after applying the GRSP algorithm. We can see that the accuracy is 100%.

Table 5. Accuracy for betweenness centrality after the GRSP

Two Graphs / Rank of btw.	original graph	after GRSP
1	ENSG00000155249	ENSG00000155249
2	ENSG00000154415	ENSG00000154415
3	ENSG00000155158	ENSG00000155158
4	ENSG00000217598	ENSG00000217598
5	ENSG00000166710	ENSG00000166710
6	ENSG00000154646	ENSG00000154646
7	ENSG00000154645	ENSG00000154645
8	ENSG00000196785	ENSG00000196785
9	ENSG00000154451	ENSG00000154451
10	ENSG00000214088	ENSG00000214088
Accuracy		**100%**

The result for LGRSP is shown in Table 6. In Table 6, we calculate a rank for betweenness centrality for a set of nodes (i.e., nodes with node id 155249, 154415, etc.) which is the ground truth that we will need when verifying the accuracy for the results of LGRSP. Second, for each graph generated by LGRSP with different error rates (ε = 0.1 – 1.0), we also calculate the rank of betweenness. We then calculate the accuracy of the ranks using the accuracy formula, and we

obtain the result that the accuracy for each error rate is in the range between 90% and 100%. We calculate the accuracy based on the comparison of the rank of betweenness (rank of btw.) for the graph before (original) and after applying the LGRSP algorithm with different error rates (ε=0.1 – 1.0). We only keep the last 6 digits of each gene Ensembl in this table.

Table 6. Accuracy for betweenness centrality after the LGRSP

Error rate / Rank	original	ε=0.1	ε=0.2	ε=0.3	ε=0.4	ε=0.5	ε=0.6	ε=0.7	ε=0.8	ε=0.9	ε=1.0
1	155249	155249	155249	155249	155249	155249	155249	155249	155249	155249	155249
2	154415	154415	155158	154415	155158	154415	155158	154415	155158	154415	155158
3	155158	155158	154415	155158	154415	155158	154415	155158	154415	155158	154415
4	217598	217598	217598	217598	217598	217598	217598	217598	217598	217598	217598
5	166710	166710	166710	166710	166710	166710	166710	166710	166710	166710	166710
6	154646	154646	154646	154646	154646	154646	154646	154646	154646	154646	154646
7	154645	154645	154645	154645	154645	154645	154645	154451	154645	154451	154645
8	196785	196785	196785	196785	196785	196785	196785	154645	196785	154645	196785
9	154451	154451	154451	154451	174332	154451	154451	196785	174332	196785	174332
10	214088	154736	214088	214088	154451	214088	214088	214088	154451	134326	154451
Accuracy		**90%**	**100%**	**100%**	**90%**	**100%**	**100%**	**100%**	**90%**	**90%**	**90%**

7.3.4. Sensitivity to ε

As it takes very long to run LGRSP on the original graph for every ε, to save the time of computation we randomly extract a smaller graph dataset which is a subgraph of the original graph for sample values of ε from 0.2 to 1.0 (and $\varepsilon = 0$ for GRSP of course), in order to know that how fast the reduced graph can speed up using GRSP without sacrificing the accuracy. The subgraph contains 10,000 edges and 3720 nodes. We then apply the GRSP algorithm to this subgraph, and it can reduce 1,938 edges. After the reduction, we apply Dijkstra's shortest path algorithm on the

reduced graph and compare the execution time before and after the reduction, based on the formula shown previously in the evaluation section. The reduction ratio is 19.38%, and the speedup is 1.3.

We then apply the 1+ε LGRSP algorithm to the subgraph. When ε is 0.2, the reduction ratio is 21.08% and the speedup is 1.341. We can see that with the error rate limited to 20% (ε =0.2), the reduction ratio could be higher than that of GRSP, and the Dijkstra's shortest path algorithm runs faster on the graph reduced by LGRSP.

We run experiments on the same graph using different values of ε, the result of reduction ratio is shown in Figure 34, and the speedup is shown in Figure 35.

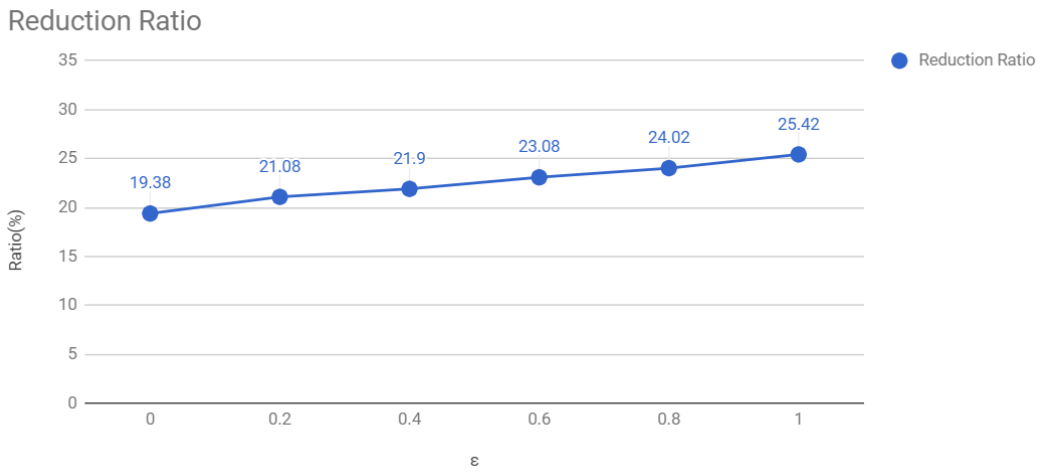


Figure 34. Reduction Ratio of 1+ε LGRSP as ε increases

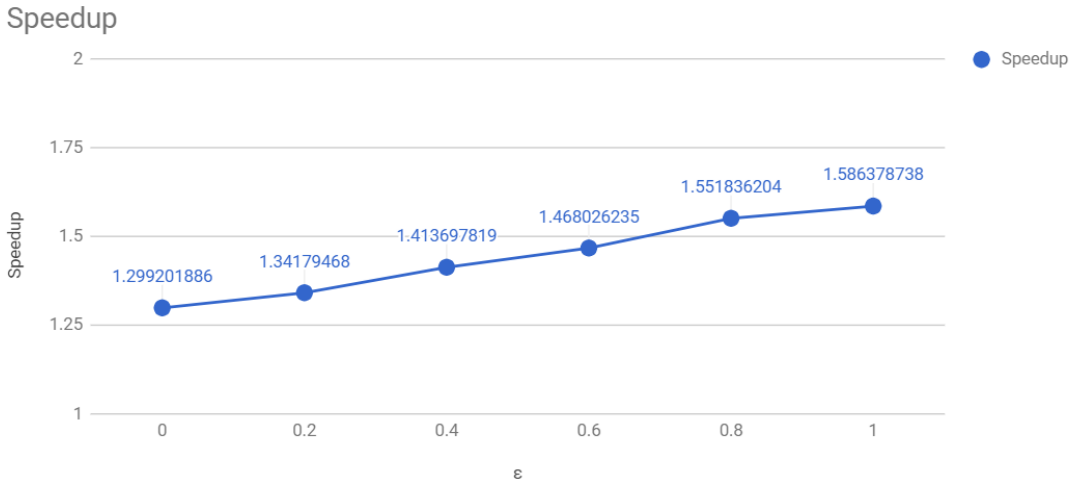


Figure 35. Speedup of 1+ε LGRSP as ε increases

7.4. SCB on Colored Graphs

Betweenness centrality has been significantly applied in biomedicine to discover the most important genes in the context of different diseases. Sun et al. applied the traditional approach to find betweenness centrality of each gene in five main disease categories (cancer, cardiovascular disease, immune system disease, metabolic disease, and nervous system disease) [8]. The goal is to demonstrate that our approach can speed up the computation of betweenness centrality and that with our approach, the computation time of finding the BBN gene can be improved. Hence, we design experiments on the disease-gene networks proposed in Sun's work [8] by running 1) the traditional betweenness centrality approach proposed by Brandes [21], and 2) our approach proposed in this study, and we compare the performance of both approaches.

7.4.1. Dataset

According to the procedure of finding genetic data in Sun's work, we run experiments on a graph composed of human disease genes, which is based on the human PPI network (protein-protein interaction network) [33], and the disease genes in the graph are defined based on the

reported genes in the published GWAS catalog [34]. Frist, we download the gene transcript Ensembl annotations data from UniProtKB [35] and match the annotations with protein identifiers in the human PPI network correspondingly. Second, we download gene Ensembl annotations from Ensembl bioMart online database which is developed by the Ensembl group. We download the gene Ensembl from the latest database which is the Ensembl genes 98 [36], and then we match them with their transcript Ensembl annotations mentioned in the previous step. Moreover, we can replace all the protein labels in human PPI network with the gene Ensembl annotations and categorize the genes based on their associated disease categories published in the GWAS catalog. Therefore, in the context of graph data, we can have a global color-blind graph containing all the disease-edges and 5 disease graphs with different colored disease-edges. Furthermore, in each of the disease graphs, we can further partition the graph into more subgraphs according to the thirteen disease traits reported in EFO (Experimental Factor Ontology) [37]. The association between the 5 categories and 13 detailed traits is listed in Table 7.

7.4.2. Five Major-Disease graphs

As shown in Table 7, we compare our approach with the conventional betweenness centrality approach (i.e., Brandes' approach) by applying betweenness centrality queries on the five disease graphs and obtained the BBNs of each graph. Since our approach is a hierarchical procedure which obtains the BBN by merging the thirteen first-level subgraphs, we run experiments on the thirteen first-level subgraphs and then obtain results for the five second-level graphs. Specifically, the cardiovascular graph is composed of two subgraphs: cardiovascular disease and cardiovascular measurement; the immune system is composed of two subgraphs: immune system disorder and inflammatory; the metabolic graph is composed of seven subgraphs: biological process, body measurement, hematological, digestive system disorder, lipid

measurement, liver enzyme measurement, and metabolic disorder. The cancer and the nervous system graphs are not composed of more than one subgraph.

Brandes' approach is not a scalable algorithm, so when querying betweenness centrality on different graph data using Brandes' method, we need to answer the queries by repetitively running a complete conventional algorithm on different graphs. To be more specific, we have to run the experiments on the thirteen first-level subgraphs, the five second-level graphs, and the global graph as well. We compare the performance of finding the BBNs on the five major-disease graphs using the conventional approach and or approach. Figure 36 shows the execution times of our approach on different datasets, while the speedup compared to the conventional approach is shown in Figure 37. Therefore, the approach is 2.63 times faster than the conventional one on the metabolic disease graph.

Table 7. BBNs in the five major diseases.

5 Major Category	Disease Categories	BBN of subgraphs	BBN of Major Categories
Cancer	3. Cancer	ENSG00000141510	ENSG00000141510
cardiovascular disease	4. Cardiovascular disease	ENSG00000091831	ENSG00000166949
	5. Cardiovascular Measurement	ENSG00000166949	
Immune system disease	8. Immune system disorder	ENSG00000177885	ENSG00000177885
	9. Inflammatory	ENSG00000113916	
Metabolic disease	1. Biological Process	ENSG00000036257	ENSG00000150991
	2. Body Measurement	ENSG00000150991	
	7. Hematological	ENSG00000036257	
	6. Digestive System Disorder	ENSG00000166949	
	10. Lipid or lipoprotein measurement	ENSG00000166949	
	11. Liver Enzyme Measurement	ENSG00000188612	
	12. Metabolic Disorder	ENSG00000154229	
nervous system disease	13. Neurological Disorder	ENSG00000010810	ENSG00000010810

Since cancer and nerve system disease graphs contain only one first-level subgraph, the execution time cannot be improved and remains the same as that of Brandes' approach. However, the speedup could be less significant or even smaller than 1.00, and this happens to the immune system graph. It is because when the size of the first-level subgraph is close to the second-level graph, the computational time of the first-level subgraph would be similar to the second-level one, and there is no chance for our approach to speed up. Therefore, the performance depends on the structure of the graph or how we partition the graph.

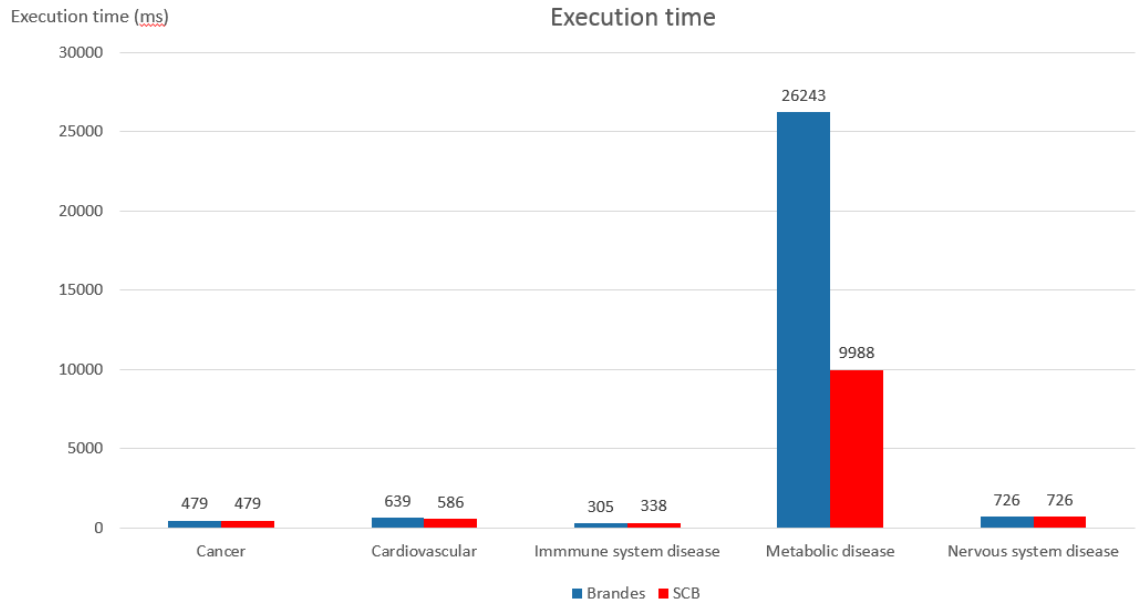


Figure 36. Computational time SCB compared to Brandes' on different dataset.

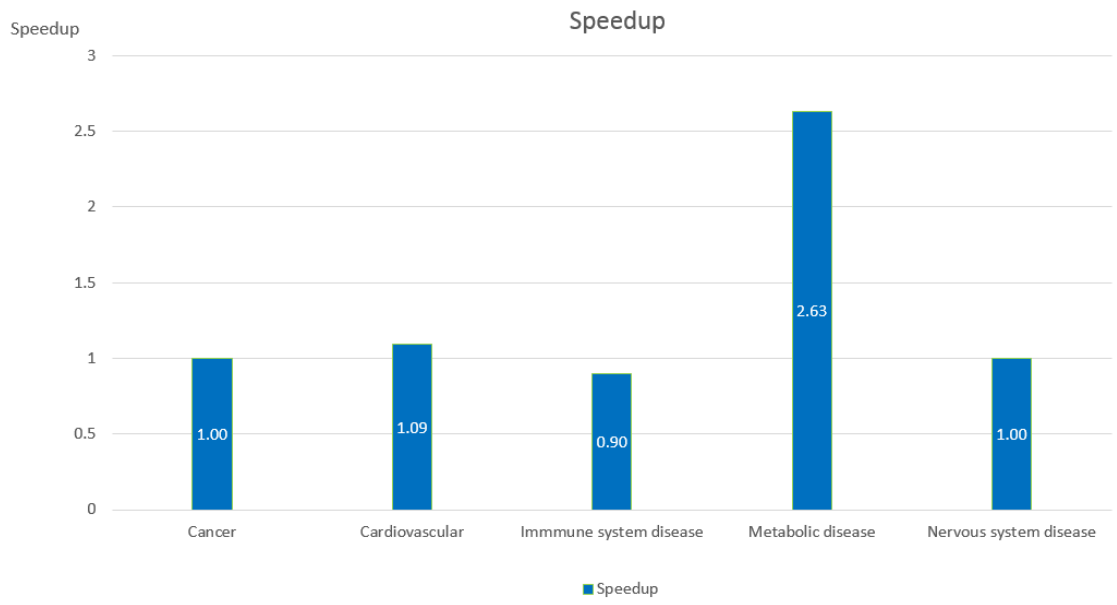


Figure 37. Speedup of the SCB approach compared to Brandes'

7.4.3. Human Disease graph

In addition to finding the BBNs of five major-disease graphs, we apply the SCB to find the global BBN of the human disease graph hierarchically based on the results of the five second-level graphs. The human disease graph can be seen as a graph composed of five colored subgraphs. In Table 8, we demonstrate the local BBNs of the five graphs and the global BBN found based on the five local BBNs. As a result, the execution time of finding the global BBN is less than that of the conventional approach, and the speedup is 2.15, which is shown in Figure 38.

Table 8. BBNs of the five major-disease categories and the human disease graph.

5 Major Category	BBN of Major Categories	BBN of Human Disease Graph
Cancer	ENSG00000141510	ENSG00000150991
cardiovascular disease	ENSG00000166949	
Immune system disease	ENSG00000177885	
Metabolic disease	ENSG00000150991	
nervous system disease	ENSG00000010810	

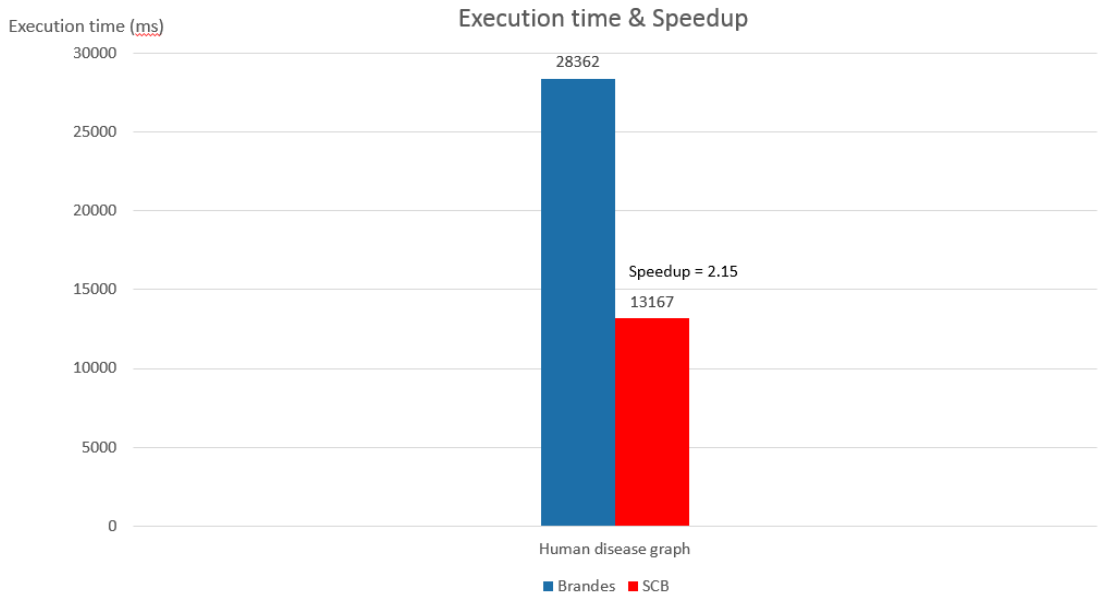


Figure 38. Speedup of the SCB approach compared to Brandes' based on the human disease graph

7.5. Discussion

For colorless graphs, the results show that with a higher reduction ratio, LGRSP can speed up more, and it can preserve a pretty good accuracy (90%~100%). Compared to [18], because their best case varies from 10%~46%, the ε=0.1 in our approach means it is the error rate for the shortest path, but the corresponding accuracy for betweenness centrality that applies LGRSP (ε=0.1) first is 90% which is better than their approach. For other values of ε, when it goes higher the accuracy

can still be kept at least 90%. For the SBC, the results show that when the number of subgraphs *k* becomes larger, the speedup can be more eminent. However, the maximum degree of every node in the graph can be limited generally, so we cannot further partition the graph into more subgraphs. Hence, we apply a hierarchical approach to partition the graph and run the SBC level by level. As a result, the speedup can be as high as 13.55 on the subset of a road network graph containing 50,000 edges, and even a large graph with two million edges which cannot be computed by the traditional approach can now be computed using the hierarchical approach. Regarding the performance of the SBC approach, the bottle neck of the speedup of SBC is the structure of the graph. If the graph data is a sparse graph which is not a complete graph or containing complexed cycles, the speedup can be evidently than the conventional approach, which is considered as the average case. If the graph is a tree without any cycles which is the best case, the SBC algorithm is even faster because there would be no extra computation inside the cycles (coarse nodes). The worst case happens when the graph is a complete graph, or the whole graph is a cycle.

On the other hand, for colored graphs, when biomedical scientists are looking for the most important genes with respect to certain types of diseases by querying betweenness centrality on genetic disease networks, it is always time consuming to process the betweenness centrality algorithm on such genetic networks. In addition, genes are not only correlated in single type of diseases, but every two genes are usually correlated in multiple diseases. Given that we apply color as one of the graph properties to represent different diseases in a genetic network, color edges indicate a specific common disease between the two genes connected by the color edge, and the weight on the edge means the correlation coefficient of both genes. Every two genes in the overall genetic network can have multiple edges with different colors representing multiple common diseases.

Conventionally, scientists would be interested in finding the backbone nodes in a specific disease network, in an integrated network composed of several types of diseases, or in an overall network including all the diseases. This has to be done by executing the traditional betweenness centrality algorithm over and over on different combinations of diseases subgraphs, while the total number of combined subgraphs is 2^k, where k is the number of colors (i.e., diseases). A major benefit of our approach is that once we have all the base cases which are the local information about the backbone node in each disease subgraph, we can answer all the questions such as looking for the backbone node in any combination of multiple-disease networks. Therefore, these kinds of queries could be done faster without re-calculating betweenness centrality values in the new diseases network, since we only need to find the shortest path between the two local backbone nodes and then find the backbone node which is the answer by calculating the external betweenness centrality only for the nodes in the candidate set.

Compared to Sun's work [8], which runs all-pair shortest paths before computing the betweenness centrality, our approach can speed up the process of finding the common BBN for each combination of subgraphs sine it takes as less as $O(m+n)$ to find the common BBN based on the base cases, where the bases cases are the local BBNs computed for each of the colored subgraphs.

Another benefit of our approach is that it can be applied to improve the performance of computing betweenness centrality of the original graph. This approach can speed up the executional time of betweenness centrality by cutting the original graph into several subgraphs based on edge colors and computing betweenness measures of each subgraph. According to the SCB algorithm, considering a graph G with multiple colors on the edges, and k is the number of colors of the edges, G can be separated into k sub-graphs (G_1, G_2...G_k). We can find the

betweenness centrality for the backbone node. The time complexity of conventional approach discussed in Section 2.1 is $O(mn+n^2\log n)$ based on Dijkstra's version of all-pair shortest path algorithm using Fibonacci heap. For the subgraph-level betweenness approach we proposed, the time complexity can be reduced. It is noted that in the worst case, if the graph *G* cannot be equally partitioned and some of the graphs have the same number of edges as the original graph *G*, the execution time would not be improved. However, in the best case, if the graph *G* can be partitioned into *k* subgraphs equally, the process for finding the global backbone node can be expedited.

Assuming the best case, we compute the betweenness measure of each subgraph, and then find the shortest path between every two backbone nodes in the two subgraphs. Therefore, for the best case in which the number of edges in each group is equal, the time complexity is $O((mn+n^2\log n)/k + l*(n+m))$, where the first part of the execution time is caused by computing the betweenness centrality of each subgraph, and the second part is caused by merging the results. Our approach detects cycles which are transformed into coarse nodes, and it requires additional computation on the coarse nodes using the traditional algorithm of betweenness centrality when the coarse nodes are the BBNs. For the best case where the whole graph is a sparse graph, the whole graph can be considered as a tree since all the cycles are transformed into coarse nodes. In this case, the weights on the edges are not necessary when finding the BBNs, and these weights only affect the computation when computing the betweenness centrality using the traditional approach for the nodes inside the coarse nodes which are the BBNs.

For the worst case in which the whole graph is a complete graph or a cycle, it requires additional computation based on the edge weights inside the coarse node, which is exactly the whole graph itself, so the time complexity is bounded by $O(mn+n^2\log n)$. Additionally, if the

number of edges in each group is extremely unbalanced or the number of colors is exactly one, the time complexity remains the same as the conventional approach.

In brief, in this approach, the number of colors *k* plays an important role such that when *k* is greater, the computational time speeds up more evidently.

CHAPTER 8 Conclusions

Betweenness centrality has been one of the most important structure analytics in graph networks. Especially, biomedical scientists have applied betweenness centrality to discover the importance of gene nodes and find the most important genes in genetic networks. In this research, we propose a scalable approach which is more efficient to compute betweenness centrality on graphs that is composed of multiple color subgraphs. Conventionally, when querying betweenness centrality on different subgraphs with different colors, we need to repetitively running the same algorithm on different subgraphs. However, this study propose a method to find the common BBN of multiple subgraphs by merging the results of each subgraph's BBN. In addition, the method is further extended to a hierarchical approach to compute BBN for multiple colored subgraphs efficiently. With such, we can answer various combinations of subgraphs by computing the subgraphs' BBNs and merging them to find the common BBN. This approach is more efficient than running the conventional approach on each of the combination over and over.

We conduct experiments on the human disease graph data based on the human PPI network using the GWAS catalog. We compare the proposed approach with the conventional approach, and we discover that the performance is influenced by the number of subgraphs and the degree of how the subgraphs are overlapped. As a result, the greater the number of colored subgraphs, the better performance the SCB can have.

Although our approach deals with a graph including cycles, the performance of SCB is restricted by the topology of the input graphs; if the whole graph is a cycle, the performance will be bounded by the computational time of Brandes's traditional approach since the graph will be considered as a coarse node. On top of our SCB approach which is based on a tree structure, we hope to extend it to another scalable version which deals with cycle nodes. As we discussed previously in Section 5, the information of shortest paths on a cycle may be affected once we decompose the cycle into two subgraphs. We have to deal with the error rate of betweenness centrality on cycle nodes in the extended version. In addition to betweenness centrality, there are other centrality analytics that have been applied to biomedicine, such as closeness centrality and degree centrality. We can apply our methodology to these analytics and propose different scalable algorithms which have better performance.

Appendix A

The GRSP algorithm can be designed based on the Dijkstra algorithm or the Floyd algorithm, i.e., when deciding an edge is to be removed or not, the shortest distance between the incident vertices can be calculated by the Dijkstra or Floyd algorithm. Due to space limitation, only the Dijkstra algorithm is considered.

ALGORITHM 1: The GRSP Algorithm

Input: Graph *G*
Output: Reduction graph *G'*

```
1. G' ← G
2. N ← NumberOfNodes(G)
3. For i ← 1 to N do
4.     Dist[i] ← Dijkstra(G,i) // use Dijkstra algorithm to compute the shortest path from i to other nodes
5.     For j ← 1 to N do
6.         If edge E(i,j) exists and E(i,j) > Dist[i][j] then
7.             Remove E(i,j) from G'
8.         End
9.     End
```

10. **End**
11. **Return** *G'*

Appendix B

For the 1+ε LGRSP algorithm, simply going through every edge and removing any edge that has another path less than or equal to 1+ε times of the edge weight may cause a problem. Assume an edge e with weight $w(e)$ is removed because we find another path p whose weight is (1+ε) $w(e)$. Later we remove another edge e' as we find another p' whose weight is (1+ε) $w(e')$. It is possible that e' belongs to p, and removing e' would affect p so that the weight of the path connecting the incident nodes of e becomes (1+ε) $w(e)$ + ε $w(e')$, i.e., the error accumulates and becomes unbounded. Therefore, instead of removing edges, we build a reduced graph by inserting edges in increasing weight order. The algorithm identifies the shortest, un-processed edge in each loop, and only if it cannot find an alternative path to connect the incident nodes of the edge or if the total weight of the shortest path is greater than (1+ε) times the weight of the edge, the edge can be inserted into the graph. The 1+ε LGRSP algorithm is described below.

ALGORITHM 2: The 1+ε LGRSP algorithm (Find a reduced graph that preserves the shortest paths between any pair of nodes in a graph with the error bound ε)

Input: Graph *G*
Output:1+ε Lossy reduction graph *G'*

1. Let *V* be the nodes set of *G*, *E* be the edges set of *G*
2. *G'* ← (*V*, ∅) //*G'* contains only nodes in *V* without any edge
3. Sort *E* in ascending order of weight
4. **For each** edge *e* in *E* **do**

5. Let nodes *x*, *y* be the incident nodes of *e*
6. *Dist*=Dijkstra(*G'*, *x*, *y*); //find the shortest distance of the corresponding nodes in *G'*
7. **If** *Dist* does not exist or $w(e) * (1+\varepsilon) < Dist$
8. Add *e* to *G'*
9. **End**
10. **End**
11. **Return** *G'*

The time complexity of the $1+\varepsilon$ LGRSP algorithm (based on Dijkstra) is $n*T$(Dijkstra).

Bibliography

[1] Li, Bi-Qing, et al., Identification of lung-cancer-related genes with the shortest path approach in a protein-protein interaction network, BioMed research international 2013.

[2] Koschützki, Dirk, and Falk Schreiber, Centrality analysis methods for biological networks and their application to gene regulatory networks, Gene Regulation and Systems Biology, 2008; Volume 2, pp. 193-201.

[3] Shao-Ting Wang, Jennifer Jin, Pete Rivett, and Atsushi Kitazawa, Graph databases and applications, International Journal of Semantic Computing, 2015; 9(4), pp. 523-545.

[4] Jovelin R, Phillips PC, Evolutionary rates and centrality in the yeast gene regulatory network, Genome Biol. 2009; 10(4): R35.

[5] Mistry, Divya, Roger P. Wise, and Julie A. Dickerson, DiffSLC: A graph centrality method to detect essential proteins of a protein-protein interaction network, PloS One 2017, 12(11): e0187091

[6] Joyce, Karen E., et al. A new measure of centrality for brain networks, PLoS One 2010, 5(8): e12200

[7] Chou, Chung-Hsien, et al. "GOLAP: Graph-Based Online Analytical Processing." International Journal of Semantic Computing 12.04 (2018): 595-608.

[8] Sun, J., Zhu, K., Zheng, W. J., & Xu, H. (2015, December). A comparative study of disease genes and drug targets in the human protein interactome. In BMC bioinformatics (Vol. 16, No. 5, p. S1). BioMed Central.

[9] Brandes, Ulrik. "On variants of shortest-path betweenness centrality and their generic computation." Social Networks 30.2 (2008): 136-145.

[10] Everett, Martin G., and Stephen P. Borgatti. "The centrality of groups and classes." The Journal of mathematical sociology 23.3 (1999): 181-201.

[11] Kolaczyk, Eric D., David B. Chua, and Marc Barthélemy. "Group betweenness and co-betweenness: Inter-related notions of coalition centrality." Social Networks 31.3 (2009): 190-203.

[12] Y. Choi and W. Szpankowski, Compression of graphical structures: fundamental limits, algorithms, and experiments, IEEE Transactions on Information Theory, February 2012, 58(2):620–638.

[13] Ahlswede, Rudolf, E-H. Yang, and Zhen Zhang, Identification via compressed data, IEEE Transactions on Information Theory, 1997, 43(1): pp. 48-70.

[14] U. N. Raghavan, R. Albert, and S. Kumara, Near linear time algorithm to detect community structures in large-scale networks, Physical Review E, 2007, 76(3): 036106.

[15] S. Chen and J. Reif, Efficient lossless compression of trees and graphs, in Proceedings of the IEEE Data Compression Conference (DCC '96), Mar 1996, pp. 428–437.

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

[17] W. Fan, X. Wang, and Y. Wu, Querying big graphs within bounded resources, in SIGMOD, 2014, pp. 301-312.

[18] Bader, David A., et al., Approximating betweenness centrality, International Workshop on Algorithms and Models for the Web-Graph. Springer, Berlin, Heidelberg, 2007. p. 124-137.

[19] Y. Tang, Y. Zhang, H. Chen, "A Parallel Shortest Path Algorithm Based on Graph Partitioning and Iterative Correcting", in Proc. of IEEE HPCC'08, pp. 155-161, 2008.

[20] Pavlopoulos, Georgios A., et al., Using graph theory to analyze biological networks, BioData Mining, 2011, 4(1): 10.

[21] Brandes, U., A faster algorithm for betweenness centrality, J. Mathematical Sociology, 2001, 25(2), pp. 163–177.

[22] Bader, David A., and Kamesh Madduri. Parallel algorithms for evaluating centrality indices in real-world networks, Parallel Processing International Conference on. IEEE, 2006, pp. 539-550.

[23] Brandes, Ulrik, and Christian Pich. Centrality estimation in large networks, International Journal of Bifurcation and Chaos, 2007, 17(7), pp. 2303-2318

[24] Brandes, Ulrik. "On variants of shortest-path betweenness centrality and their generic computation." Social Networks 30.2 (2008): 136-145.

[25] Kruskal, Joseph B. "On the shortest spanning subtree of a graph and the traveling salesman problem." Proceedings of the American Mathematical society 7.1 (1956): 48-50.

[26] Thomas H. Cormen; Charles E. Leiserson; Ronald L. Rivest; Clifford Stein (31 July 2009). Introduction to Algorithms. MIT Press. ISBN 978-0-262-53305-8.

[27] Computing Betweenness Centrality of Large Graphs based on Data Reduction and Decomposition with a Case Study on Breast Cancer Analytics

[28] Tang, Jianing, et al, Prognostic genes of breast cancer identified by gene co-expression network analysis, Frontiers in oncology, 2018, 8: 374.

[29] Daemen, Anneleen, et al., Modeling precision treatment of breast cancer, Genome Biology, 2013, 14(10): R110.

[30] Langfelder, P., & Horvath, S., WGCNA: An R package for weighted correlation network analysis, BMC Bioinformatics, 2008, 9(1), 559.

[31] Jure Leskovec and Andrej Krevl, "SNAP Datasets: Stanford Large Network Dataset Collection", http://snap.stanford.edu/data, June 2014.

[32] Huang, Jin, and Charles X. Ling, Rank measures for ordering, European Conference on Principles of Data Mining and Knowledge Discovery, Springer, Berlin, Heidelberg, 2005. p. 503-510.

[33] Cowley MJ, Pinese M, Kassahn KS, Waddell N, Pearson JV, Grimmond SM, Biankin AV, Hautaniemi S, Wu J. PINA v2.0: mining interactome modules. Nucleic acids research. 2012. pp. D862–865.

[34] Welter D, MacArthur J, Morales J, Burdett T, Hall P, Junkins H, Klemm A, Flicek P, Manolio T, Hindorff L, Parkinson H, Nucleic Acids Res. 2014 Jan; 42(Database issue):D1001-6.

[35] Bairoch, A., Apweiler, R., Wu, C. H., Barker, W. C., Boeckmann, B., Ferro, S., ... & Martin, M. J. (2005). The universal protein resource (UniProt). Nucleic acids research, 33(suppl_1), D154-D159.

[36] Zerbino, Daniel R., et al. "Ensembl 2018." Nucleic acids research 46.D1 (2017): D754-D761.

[37] Malone, J., Holloway, E., Adamusiak, T., Kapushesky, M., Zheng, J., Kolesnikov, N., ... & Parkinson, H. (2010). Modeling sample variables with an Experimental Factor Ontology. Bioinformatics, 26(8), 1112-1118.