An Approach to Steiner Tree Problem in Graphs by Network Centralities and Chaotic Neurodynamics

(ネットワーク中心性とカオスニューロダイナミクスを用いた
グラフ的シュタイナー木問題の解法)

March 2020

Misa Fujita (藤田 実沙)

Abstract

The aim of this thesis is to develop efficient approximation algorithms for solving the Steiner tree problem in graphs. Given an undirected weighted graph $G = (V, E, w)$ and T, where V is a set of vertices, E is a set of edges, w is a weight function, and $T \subseteq V$, a Steiner tree is defined as a tree that spans T. A vertex in T is called a terminal. The weight of the Steiner tree is the sum of the weights of the edges in the Steiner tree. Then, the Steiner tree problem in graphs is to find a minimum weight Steiner tree of a given graph. The Steiner tree problem in graphs is applied to real-world problems, such as design of communication networks, road networks, and pipelines. However, approximation algorithms that can quickly find good solutions are still required because the Steiner tree problem in graphs is $\mathcal{NP}$-hard.

There are two types of approximation methods for solving combinatorial optimization problems, such as the Steiner tree problem in graphs: construction and improvement. The construction method focuses on finding a solution as quickly as possible, whereas the improvement method focuses on finding the most appropriate solution. In this thesis, we propose an efficient construction method, which uses network centrality, and an efficient improvement method, which uses chaotic neurodynamics, to solve the Steiner tree problem in graphs.

In the construction method for solving the Steiner tree problem in graphs, shortest paths or a minimum spanning tree are typically used. However, previous construction methods sometimes find large-weight solutions because they ignore the overlaps of the shortest paths between terminals. From this viewpoint, we propose to use network centralities to overlap the shortest paths between terminals. The results of numerical experiments revealed that our proposed method improves the average gaps by approximately 3% compared to previous methods, with a slight increase in computation time.

As for the improvement method for solving the Steiner tree problem in graphs, we propose to use the chaotic search. The chaotic search is known as an efficient meta-heuristic for solving the traveling salesman problem, quadratic assignment problem, and vehicle routing problem. The local search for these combinatorial optimization problems always finds feasible neighborhood solutions. On the other hand, the local search for the Steiner tree problem in graphs finds not only feasible solutions but also infeasible solutions. Thus, we temporary ignore neurons that correspond to infeasible solutions. We compare the performance of the chaotic search with that of the tabu search, which is one of the popular methods to solve

the Steiner tree problem in graphs. The results of numerical experiments revealed that the chaotic search finds smaller weight solutions than several conventional methods such as the genetic algorithm, greedy randomized adaptive search procedure, and the tabu search with appropriate parameters.

In this thesis, we firstly propose to use network centralities for distinguish vertices and edges that are important to find a small-weight Steiner tree. The results of numerical experiments indicate that using network centralities, especially vertex betweenness centrality is effective to find a small-weight Steiner tree. This trend is the same for three popular construction methods to solve the Steiner tree problem in graphs. Secondly, we propose to how to apply the chaotic search to solve the Steiner tree problem in graphs. The results of numerical experiments indicate that the chaotic search also shows good performance to solve the Steiner tree problem in graphs. From these results, we found that using network centrality contributes to find good solution and using the chaotic search shows good performance to solve the Steiner tree problem in graphs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The combinatorial optimization problem is one of the recent and most actively researched fields of applied mathematics. It is solved using various techniques such as algorithm theory, combinatorics, complexity theory, data structure, and linear programming. This problem is derived from a wide range of application fields such as designing very large-scale integration [2], multicast tree [3], and pipeline [4]. For example, in the case of designing very large-scale integration, its objective may be minimization of the total length of wires, the total areas of placing electronic components, the production costs, and so on. Various methods have been proposed to solve these combinatorial optimization problems. These combinatorial optimization problems are very difficult to solve exactly in practical time because these are $\mathcal{NP}$-hard. Most of the real-world applications require not only the exact optimum solutions but also approximate solutions that are very close to the exact optimum solutions. Hence, it is very important to develop efficient approximation algorithms for solving these problems.

Many combinatorial optimization problems are defined on graphs. The graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects. The paper written by Leonhard Euler on the Seven Bridges of Königsberg and published in 1736 is regarded as the first paper in the history of the graph theory [5]. The purpose of the graph theory is to determine the properties of graphs. In 1998, Watts and Strogatz investigated the common features of complex graphs in real-world problems, such as the synchronization of the blinking of fireflies, singing of crickets, co-star relationship between actors, power grids, and neural network of C. elegans [6]. This research led to the emergence of a new field of science, which is now called the complex network theory.

In this thesis, we propose to use network centralities to solve the Steiner tree problem in graphs. The Steiner tree problem in graphs is the problem of finding the Steiner tree with the minimum weight. The Steiner tree is a tree that spans special vertices called terminals, and the weight of the Steiner tree is the sum of the edge weights included in it.

The network centralities measure the importance of vertices and edges in a network and it is one of the most important topics in the complex network theory. It is expected that the important vertices and edges are also important to find small-weight Steiner trees. Various network centralities have been proposed from different aspects [7]. For example, the degree centrality [8] evaluates vertices with a large degree as important, the eigenvector centrality [9] evaluates vertices adjacent to important vertices, the closeness centrality [10] measures how close a vertex is to all the other vertices, and the betweenness centrality [8] detects central vertices or edges in the network using the shortest path information.

Meanwhile, an effective meta-heuristic that uses the chaotic neurodynamics [11] was proposed by Hasegawa, Ikeguchi, and Aihara in 1997 [12]. This method is novel and revolutionary in that it drives local search by chaotic neurodynamics. They used this method to solve the traveling salesman problems, and they demonstrate that it shows better performance than the random method. This method can be applied to solve other combinatorial optimization problems by changing local search. Thus, this method is applied to various $\mathcal{NP}$-hard combinatorial optimization problems such as the traveling salesman problem [12, 13, 14, 15], quadratic assignment problem [16, 17, 18], vehicle routing problem [19, 20], packet routing problem [21, 22, 23, 24, 25, 26, 27, 28], and motif extraction problem [29, 30]. This method also shows good performance for these problems. Therefore, we also apply chaotic search to solve the Steiner tree problem in graphs.

In addition, we also investigate the number of optimal solutions found during the search. When solving real-world problems, the optimum solutions are usually unknown. Therefore, the high frequency of good solutions is one of the practical indicator of goodness. The algorithms that can frequently find good solutions during the search are considered to have better performance than the algorithms that do not. From this viewpoint, we also compare the number of optimal solutions found during the search.

The rest of this chapter is organized as follows. We describe the terms of graph theory used in this thesis in Sec. 1.2. Then, we describe the computational complexity that one of the aspects to evaluate the performance of algorithms and the classification of difficulty of problems in Sec 1.3. Then, we refer to the previous research of the combinations of the combinatorial optimization and network centrality in Sec. 1.4, and the one of the combinatorial optimization and chaotic neurodynamics in Sec. 1.5. Finally, this chapter has concluded the outline of this thesis.

1.2 Graph and algorithm

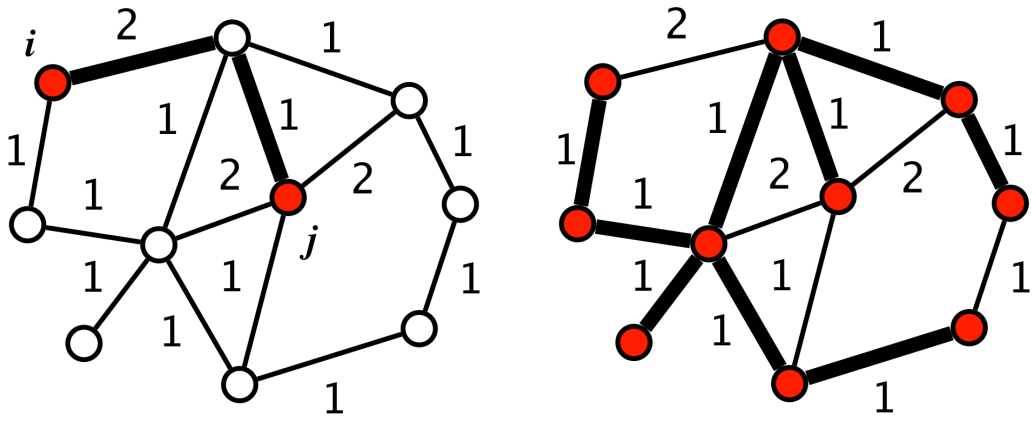
In mathematics, various real-world problems are treated as graphs. A graph consists of vertices and edges, and it represents their relationships. For example, if we assume vertices to be stations and edges to be rails, this graph represents a train network. If we assume vertices to be intersections and edges to be roads, this graph represents a road network. Each edge is incident to two vertices, and these connected vertices are called adjacent. The number of edges that are incident to a vertex is called the degree. We can also consider the direction of edges. A graph whose edges have directions is called a directed graph, whereas a graph whose edges do not have directions is called an undirected graph.

An alternate arrangement of vertices and edges is called a path. Given an undirected graph $G = (V, E)$, the weight function corresponds to the edges $w : E \to \mathbb{R}$, and a vertex pair i and j, the shortest path problem requires a path from i to j with the minimum value of the sum of the edge weights included in the path. Here, V is the set of vertices, E is the set of edges, and $w : E \to \mathbb{R}$ is a map from an edge set to a non-negative whole real number. An example of this problem is shown in Fig. 1.1 (a). In Fig. 1.1 (a), circles indicate vertices, lines indicate edges, and numerals indicate edge weights. In particular, the red circles indicate the vertices that must be connected, and the bold lines indicate the edges included in the shortest path. This problem can be solved easily by using the Dijkstra algorithm [31], which is named after the person who proposed this algorithm. This algorithm produces an optimal solution of this problem.

Moreover, a path starts from a vertex and return to the same vertex is called a loop. In addition, a graph is connected when there is a path between all the vertices. A connected graph without any loops is called a tree. In particular, a tree that connects all the vertices is called a spanning tree. We can consider the following problem by using the definition of a spanning tree: given an undirected graph $G = (V, E)$ and weight function corresponding to the edges $w : E \to \mathbb{R}$, a spanning tree that minimizes the sum of the edge weights included in the spanning tree is required. An example of this problem is shown in Fig. 1.1 (b). This problem is called the minimum spanning tree problem, and it can be solved easily by using the Kruskal algorithm [32], which is named after a person who proposed this algorithm. This algorithm produces an optimal solution of this problem.

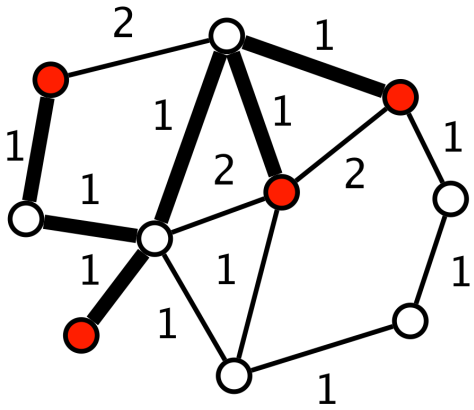
The shortest path problem is the problem of finding a subgraph that connects two selected vertices with the minimum weight. Similarly, the minimum spanning tree problem is the problem of finding a subgraph that connects all the vertices with the minimum weight. On the other hand, what about the problem of finding a subgraph that connects some selected vertices with the minimum weight? An example of this problem is shown in Fig. 1.1 (c). This problem is called the Steiner tree problem, which is named after Jakob Steiner who investigated this problem first. This problem is known to be very hard to solve or $\mathcal{NP}$-hard [33]. However, this problem appears in various engineering fields, such as designing

very large-scale integration, multicast tree, and pipelines. From this viewpoint, we aim to develop an efficient method to solve this problem.



(a)

(b)



(c)

Figure 1.1: Example of (a) the shortest path, (b) a minimum spanning tree, and (c) a Steiner tree. Circles indicate vertices, lines indicate edges, and numerals indicate edge weights. In particular, the red circles indicate the vertices that must be connected, and the bold lines indicate the edges included in (a) the shortest path, (b) the minimum spanning tree, and (c) the Steiner tree.

1.3 Combinatorial optimization problem

Lawler [34] defined the combinatorial optimization problem as follows: combinatorial optimization is the mathematical study of finding an optimal arrangement, grouping, ordering, or selection of discrete objects, which are usually finite in numbers. There are many algorithms for solving combinatorial optimization problems. If these algorithms can find the same solution, the fastest one has the best performance. The computation cost indicates the computational complexity of algorithms. In general, the parameters that represent the size of a problem is called the input size, and the computation cost of an algorithm is expressed as a function of the input size. For example, in the case of the shortest path problem, its input size is the number of vertices $|V|$ and the number of edges $|E|$. Therefore, the computation cost of the Dijkstra algorithm is $\mathcal{O}(|V|^2)$ in the simple implementation. An algorithm whose computation cost is a polynomial function of the input size is called a polynomial time algorithm. The polynomial time algorithm is one of the efficient algorithms. Table 1.1 shows the approximate computation times of algorithms that have different computation costs. The approximate computation time is calculated by assuming a 100 MIPS (million instructions per second) computer. The first three columns in Table 1.1 show the calculation time of various polynomial time ($n,\ n^2,\ n^3$) algorithms. From Table 1.1, it can be seen that polynomial time algorithms up to n^3 are sufficiently practical for large input sizes.

On the other hand, an algorithm whose computation cost is an exponential function of the input size is called an exponential time algorithm. The exponential time algorithm is an inefficient algorithm. The last three columns in Table 1.1 show the calculation times of various exponential time ($2^n,\ 3^n,\ n!$) algorithms. From Table 1.1, $n!$ needs a non-practical time of at most $n = 30$. Additionally, in the case of the slowest one 2^n, it is not practical at $n = 100$.

We can always find the optimum solution by enumerating all the solutions and selecting the best one because the solution space of a combinatorial optimization problem is finite. The objective of the Steiner tree problem in graphs is to find a tree that minimizes the sum of the edge weights included in the tree and spans all terminals. The optimum solution of the Steiner tree problem in graphs is equal to the minimum spanning tree of the induced subgraph of the vertex set of optimum solutions. An induced subgraph is a subgraph of another graph, formed from a subset of the vertices of the graph and all the edges connecting pairs of vertices in that subset. From this viewpoint, the solutions of the Steiner tree problem in graphs can be represented as a set of vertices. Thus, the number of candidate solutions of the Steiner tree problem in graphs is 2^n. Therefore, the enumeration method for the Steiner tree problem in graphs is an exponential time algorithm.

The combinatorial optimization problems that have polynomial-time algorithms are considered easy to solve. For example, the shortest path problem and the minimum spanning tree problem are easy to solve because these problems have polynomial-time algorithms.

Table 1.1: Example of the computation time of polynomial-time (first three columns) and exponential time (last three columns) algorithms. In this table, s represents second, h represents hour, d represents day, y represents year, and hyphen represents more than $2,000$ years [1].

	computation time					
the input size	n	n^2	n^3	2^n	3^n	$n!$
10	1×10^{-7} [s]	1×10^{-6} [s]	1×10^{-5} [s]	2.1×10^{-5} [s]	5.9×10^{-4} [s]	0.036 [s]
20	2×10^{-7} [s]	4×10^{-6} [s]	8×10^{-5} [s]	1.05×10^{-2} [s]	34.9×10^{-4} [s]	771 [y]
30	3×10^{-7} [s]	9×10^{-6} [s]	2.7×10^{-4} [s]	10 [s]	23.8 [d]	-
40	4×10^{-7} [s]	1.6×10^{-5} [s]	6.4×10^{-4} [s]	3.05 [h]	-	-
50	5×10^{-7} [s]	2.5×10^{-5} [s]	1.25×10^{-3} [s]	130 [d]	-	-
100	1×10^{-6} [s]	1×10^{-4} [s]	0.01 [s]	-	-	-
1000	1×10^{-5} [s]	1×10^{-2} [s]	10 [s]	-	-	-
10000	1×10^{-4} [s]	1 [s]	2.78 [h]	-	-	-

These problems are classified as polynomial-time solvable and it is abbreviated $\mathcal{P}$. To prove whether a given problem is in a class $\mathcal{P}$ or not, it is sufficient to find at least one polynomial-time algorithm. However, it is not easy to prove that a given problem does not have any polynomial-time algorithm. For example, a polynomial-time algorithm to solve the Steiner tree problem in graphs has not yet been found despite the efforts of many excellent researchers. However, this does not confirm that there is no polynomial-time algorithm to solve the Steiner tree problem in graphs.

To solve this problem, we introduce a new class of problems. We consider the problem of determining whether the optimal objective function value of the Steiner tree problem in graphs is smaller than a given value C. The answer is either yes or no. A question whose answer is either yes or no is called a decision problem. When the evidence that the answer to a decision problem is yes is constrained by the input size polynomial, the decision problem falls into the class $\mathcal{NP}$. Here, $\mathcal{NP}$ is an acronym for Non-deterministic Polynomial. This means that $\mathcal{NP}$ problems can be solved in polynomial-time by using a nondeterministic computer. From another perspective, problems in which the height of the enumeration tree is at most a polynomial of the input size belong to the class $\mathcal{NP}$. For example, the height of the enumeration tree of the Steiner tree problem in graphs is at most the number of vertices. Thus, the deterministic Steiner tree problem in graphs is an $\mathcal{NP}$ problem. The proof of the answer yes in the solution of the deterministic Steiner tree problem in graphs is a Steiner tree having an objective function value smaller than C.

On the other hand, the proof of the answer no in the solution of the deterministic Steiner tree problem in graphs is not easy to find. Hence, enumerating all the solutions and showing that each solution has an objective function value larger than C become the proof. However,

this process requires an exponential time of the input size. Researchers in computational complexity theory have proven that if a deterministic Steiner tree problem in graphs can be solved efficiently, then all the other problems in class $\mathcal{NP}$ can be solved efficiently. This idea strongly suggests that there is no efficient algorithm for the Steiner tree problem in graphs.

Many researchers believe that $\mathcal{P} \neq \mathcal{NP}$. In this way, the deterministic Steiner tree problem in graphs belongs to the most difficult class in the class $\mathcal{NP}$. This class is called $\mathcal{NP}$-complete. Additionally, if non-deterministic problems are more difficult than $\mathcal{NP}$-complete problems, these problems are classified as $\mathcal{NP}$-hard.

1.4 Approximation algorithm

As a realistic approach to solve $\mathcal{NP}$-hard combinatorial optimization problems, many types of approximation algorithms have been developed. Approximation algorithms that do not guarantee the approximation ratio are called heuristics. Most practical applications do not require exact optimal solutions. Finding approximate solutions within a few percent of the exact optimal value is sufficient. These heuristic approaches find good solutions that are near optimal within a reasonable amount of time. There are two types of methods: construction methods and improvement methods. Construction methods directly find a solution from the input. On the other hand, improvement methods find better solutions than the initial solution obtained using construction methods. Thus, construction methods and improvement methods are closely related. From this viewpoint, both approaches are important.

1.4.1 Construction method

The construction method directly finds a solution from an input. The construction method works very fast; however, the quality of the solution is not very good. The shortest path heuristic [35], distance network heuristic [36], and average distance heuristic [37] are commonly used as the construction method for solving the Steiner tree problem in graphs.

These construction methods use the shortest path between arbitrary vertex pairs and the minimum spanning tree to find a Steiner tree. The shortest path heuristic [35] starts from a tree that consists of an arbitrary terminal. Then, it repeats including the shortest path between two vertices: one is a vertex included in the tree and the other is a terminal not included in the tree. The distance network heuristic [36] starts from a complete graph of the terminals. Then, it makes a minimum spanning tree of the complete graph. After that, it replaces the edges of the minimum spanning tree with the shortest path between both sides of the vertices of the edge in the input graph. The average distance heuristic [37] starts from a forest consisting of the number of terminal trees. Each tree consists of a terminal. Then, it repeats combining trees until it becomes a Steiner tree.

1.4.2 Improvement method

The improvement method finds a better solution than the initial solution. There are two types of methods: local search methods and meta-heuristics. The local search methods find better solutions than the initial solution by using gradient dynamics. On the other hand, meta-heuristics find better solutions by controlling the local search methods. Both local search and meta-heuristics are important.

Local search method

The simplest way to find a better solution is to repeatedly find some neighborhood solutions and select the one that reduces the objective function value the most. The Steiner vertex neighborhood is one of the most popular methods for making neighborhood solutions from a current solution of the Steiner tree problem in graphs. This algorithm finds a neighborhood solution by reversing the state of an arbitrarily selected non-terminal vertex; if the selected vertex is already included in the current solution, the vertex is removed from the neighborhood solution, and vice versa. This algorithm provides not only feasible solutions but also infeasible solutions. Therefore, by repeatedly replacing the current solution with the neighborhood solution when the neighborhood solution is feasible and its objective function value is smaller than the current one, we can find a better solution than the initial solution.

Unfortunately, the above method has a problem: the obtained solution is the local optima. One of the local optima is the global optimum; however, we can find the global optimum only when we can luckily start from an initial solution that converges to the global optimum. The schematic diagram of the local optima and global optimum is shown in Fig. 1.2. As shown in Fig. 1.2, we cannot reach the global optimum when we start from an initial solution that does not converges to the global optimum, because the local search method does not allow uphill movement of the objective function value. Therefore, once the state is trapped in the local minima, it can never escape from it with these dynamics.

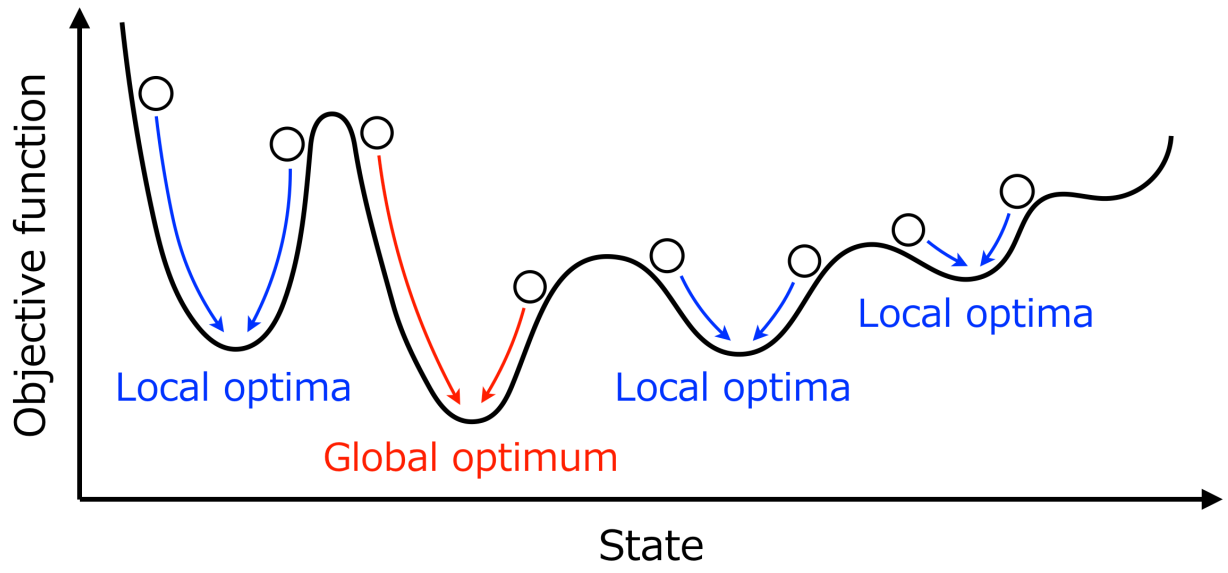


Figure 1.2: Local optima and global optimum.

Meta-heuristics

The local search method has a problem that it cannot escape from the local optima. Therefore, the quality of the obtained solution is determined by the quality of the initial solution, and it is almost impossible to find an initial solution that converges to the global optimum. To overcome this problem, many meta-heuristics that control the application of the local search to escape from the local optima have been proposed [38].

To realize an efficient search, we have to consider two important aspects: intensification and diversification of the search. Intensification means decreasing of an objective function value to find a better solution closer to a local minimum. Diversification means a jump from a searching region to other regions to avoid being trapped in a single local minimum. If the effect of intensification is too strong, the algorithm is trapped at an undesirable local optimum. On the other hand, if the effect of diversification is too strong, the algorithm cannot search solutions with small objective function values. Namely, the balance between them is a very important factor to search good solutions.

There are many meta-heuristics [38] that use different ideas to escape from local optima, such as simulated annealing [39], tabu search [40, 41], and genetic algorithm [42]. Simulated annealing [39] is one of the stochastic algorithms used to solve combinatorial optimization problems. Its name and inspiration come from the annealing process in metallurgy. Annealing is a process in which a metallic material is heated and then gradually cooled to grow crystals in order to reduce defects. The heat causes the atoms to move away from their initial position (the local minimum state where the internal energy is local) and wander to a higher energy state. Slow cooling increases the likelihood that an atom will have a state where the internal energy is even lower than that in the initial state.

In simulated annealing [39], whether the movement to a neighborhood solution could occur or not is decided by the improvement of the objective function value and the parameter temperature; a high temperature indicates strong diversification. The search starts with a high temperature and then it gradually decreases. Thus, the effect of diversification is strong at first and then gradually decreases. Although this algorithm is very simple, it is very useful to be applied to various kinds of optimization problems.

Tabu search [40, 41] is one of the deterministic algorithms used to solve combinatorial optimization problems. It uses the history of the search to escape from a local minimum. The tabu search moves to a neighborhood solution with the smallest objective function value. However, the movement recorded in the tabu list is inhibited. In other words, the tabu search moves to a neighborhood solution with the smallest objective function value that is not registered in the tabu list. The tabu list has a finite length. Thus, when adding a new one, the oldest one is deleted. The length of the tabu list balances the intensification and diversification of the search.

The genetic algorithm [42] is one of the stochastic algorithms used to solve combinatorial

optimization problems. The genetic algorithm is a multi-point search algorithm that has many solutions at the same time. In the genetic algorithm, each solution is expressed as a chromosome, and neighborhood solutions are generated by the crossover and mutation of the chromosome. The procedure of a genetic algorithm is as follows. First, many feasible solutions are prepared. Then, crossover is applied to pairs in those prepared solutions for providing new solutions (they are the children of the initially prepared solutions). Finally, better solutions are used for the next crossover and only better solutions are preserved and updated. This is better for searching a wide area of the search space. However, this approach is not good for searching local optima, namely, this is not good for intensification.

1.5 Combinatorial optimization and network centrality

Many combinatorial optimization problems are defined on graphs. It is expected that knowledge in the graph theory and complex network theory is useful to solve these problems. In particular, the complex network theory, which was born around 2000, has made remarkable progress over the last two decades.

As another $\mathcal{NP}$-hard combinatorial optimization problem, the traveling salesman problem has been well studied. The traveling salesman problem is defined as follows. Find the shortest route that visits all cities only once and returns to the city visited first. The Lin-Kernighan-Helsgaun algorithm [43] is one of the strong algorithms for solving the traveling salesman problem. However, it has a demerit: the computational time is very long. To overcome this issue, the partial optimization meta-heuristic under special intensification conditions method was applied to solve the traveling salesman problem in 2019 [44]. This method limits the candidate edges to obtain solutions, and it successfully reduces the computation time of the Lin-Kernighan-Helsgaun algorithm while retaining its high performances. Thus, it is expected that limiting the candidate vertices and edges also shows good performance for solving the Steiner tree problem in graphs.

One interesting topic in the complex network theory is network centrality. Network centrality measures the importance of vertices and edges in a network. So far, many network centralities have been proposed, such as the degree [8], eigenvector [9], closeness [10], and betweenness centrality [8]. Betweenness centrality is a well-known indicator that is used to detect communities [45]. Community means a dense subgraph in a graph. Community detection is the problem of appropriately dividing groups of vertices. For example, consider a graph that indicates the friendships among people joining two karate clubs. In this case, the objective of community detection is to group people who join the same karate club. However, there is a possibility that some people join both karate clubs. Thus, community detection allows overlap of communities. Girvan and Newman proposed a community detection method that removes edges in the descending order of its betweenness centrality [45].

The Steiner tree problem in graphs is a problem that determines which non-terminal vertices should be used or which edges should be used. The vertices or edges included in the optimum Steiner tree may be important in the graph. Therefore, we proposed to use high network-centrality vertices or edges to find Steiner trees. Network centrality has not been used to solve the Steiner tree problem in graphs; this is a novel approach.

1.6 Combinatorial optimization and chaotic dynamics

The chaotic neuron model was proposed by Aihara, Takabe, and Toyoda in 1990 [11]. This model shows chaotic behavior when parameters are set to suitable values.

Hasegawa, Ikeguchi, and Aihara proposed a new approach to solve combinatorial optimization problems [12]. In this approach, the neurons in chaotic neural networks are related to the candidate solutions produced by the heuristics, which always produce feasible solutions. The most important points for chaotic behavior in a chaotic neuron model are that the output is an analog value and the refractory effect is reproduced; if a neuron fires, this neuron does not fire for a while. These are important behaviors observed in real biological neurons. The chaotic neuron model realizes them by using the sigmoidal function as the activation function and the history of firings.

This method, which uses the chaotic neural network, is called "chaotic search," and it is applied to solve many types of combinatorial optimization problems such as the traveling salesman problem [12, 13, 14, 15], quadratic assignment problem [16, 17, 18], vehicle routing problem [19, 20], packet routing problem [21, 22, 23, 24, 25, 26, 27, 28], and motif extraction problem [29, 30]. The chaotic search shows good performance for these problems with only a few adjustments to the characteristics of the problems.

From this viewpoint, we also apply the chaotic search to solve the Steiner tree problem in graphs. The main difference between the Steiner tree problem in graphs and other above combinatorial optimization problems is that the heuristics for the Steiner tree problem in graphs produces not only feasible solutions but also infeasible solutions. To deal with this problem, we ignore the neurons that correspond to infeasible solutions, and investigate the performance of this chaotic search.

1.7 Outline

This thesis consists of five chapters. Chapter 2 deals with a Steiner tree problem. In Section 2.1, we describe the history of the Steiner tree problem. We introduce five types of Steiner tree problems in Section 2.2. Then, we introduce three applications of Steiner tree problems in Section 2.3. Among the various Steiner tree problems, we concentrate on the third one, namely the Steiner tree problem in graphs. In section 2.4, we show how to represent the solutions of the Steiner tree problem in graphs. Finally, we introduce the benchmark problems used in this thesis in Section 2.5.

Chapter 3 deals with a construction method combined with network centralities. In Section 3.1, we explain why we focus on the network centrality to solve the Steiner tree problem in graphs. Then, we introduce three popular construction methods, namely the shortest path heuristic, the distance network heuristic, and the average distance heuristic, and a postprocessing method for solving the Steiner tree problem in graphs in Section 3.2. Our proposed method is based on these conventional methods. Then, we introduce four popular network centralities, namely the degree centrality, the eigenvector centrality, the closeness centrality, and the betweenness centrality. Section 3.4 describes how to combine construction methods and network centralities . In Section 3.5, we evaluate the performance of our proposed method by using benchmark problems. The results of numerical experiments are compared with the original (not using any network centrality) methods. The performance is evaluated from the gaps in the optimum solution and the computational time by calculating network centralities because the computational time of our proposed method is longer than that of the original methods. In Section 3.6, we discuss the results obtained in the previous section. This chapter is concluded with a summary of the major findings of using network centralities in Section 3.7.

Chapter 4 deals with an improvement method using chaotic neurodynamics. In Section 4.1, we review the history of the solution method using chaotic neurodynamics for combinatorial optimization problems and why we apply this method to solve the Steiner tree problem in graphs. Then, we introduce a popular method to make neighborhood solutions from a Steiner tree in Section 4.2. This method is based on the fact that each Steiner tree can be represented as a set of vertices. Because if there is an optimal Steiner tree, the minimum spanning tree of the induced subgraph of the vertices, which are included in the optimal Steiner tree, is equal to the optimal Steiner tree. From this viewpoint, the Steiner vertex neighborhood provides a neighborhood solution by reversing the state of an arbitrarily selected non-terminal. Here, the state indicates whether the selected non-terminal vertex is included in a current Steiner tree or not. In other words, if the selected non-terminal vertex is included in the current Steiner tree, an induced neighborhood solution is the minimum spanning tree of the vertex set that removes the selected non-terminal vertex from the vertex set of the current Steiner tree, and vice versa. Section 4.3 explains how to apply the solving method using chaotic

neurodynamics or the chaotic search to solve the Steiner tree problem in graphs. In Section 4.4, we evaluate the performance of chaotic search to solve the Steiner tree problem in graphs by using benchmark problems. The results of numerical experiments are compared with the simple tabu search, which is realized by tuning the parameters of the chaotic search. The performance is evaluated from the gaps in the optimum solutions, the computational time, and the number of optimal solutions found during the search. Metaheuristics can find good solutions, but it requires a considerable amount of time. From this viewpoint, it is fair to compare the performance of algorithms taking the same computational time; however, it is very difficult. Therefore, to compare the performance of algorithms as fair as possible, we set a limit for the number of iterations. In other words, the performances of chaotic search and tabu search are compared with the same total number of generated neighborhood solutions. The number of optimal solutions found during the search is first introduced to evaluate the performance of algorithms in [46]. This aspect evaluates the reliability of the algorithms. If an algorithm searches good solutions frequently, the algorithm is capable of searching good solutions stably. Thus, we use this evaluation point to compare the performance of chaotic search and tabu search. Additionally, the results of numerical experiments are compared with other effective conventional metaheuristics such as the sophisticated tabu search, genetic algorithm, and greedy randomized adaptive search procedure. In Section 4.5, we discuss the results obtained in the previous section. This chapter is concluded with a summary of the major findings of using the chaotic search in Section 4.6.

Finally, this thesis is concluded with Chapter 5, which includes a summary of the major findings of this thesis and future researches.

Chapter 2

Steiner tree problem

2.1 History

The original Steiner tree problem was proposed by Fermat (1601–1665). He proposed the following problem: given three points in a plane, find the fourth point such that the sum of its distances to the three given points is minimal. This problem is called the Fermat problem. Torricelli, Cavalieri, and Simpson provided independent solutions to the problem.

Torricelli used equilateral triangles and circumscribing circles to find the fourth point before 1640. The intersection point of three circumscribing circles of three equilateral triangles that have an edge connecting two of the given points is called the Torricelli point. In 1647, Cavalieri showed that line segments from the three given points to the Torricelli point make 120° with each other. Simpson used only equilateral triangles to find the fourth point in 1750. The intersection point of three lines joining the outside vertices of the equilateral triangles to the opposite vertices of the given triangles is equivalent to the Torricelli point. These three lines are called Simpson lines. Heinen proved that the lengths of the three Simpson lines are the same as the sum of lengths from the Torricelli point to the three given points in 1834.

However, if one of the angles in a given triangle is at least 120°, the Torricelli point exists outside the given triangle and is not the minimizing point. The minimizing point in this case is the vertex of the obtuse angle. This fact was observed by Heinen in 1834 and also by Bertrand in 1853.

Jarnik and Kössler asked the following in 1934: find the shortest network that interconnects n points in the plane. In particular, they studied the above problem when the n points are the corners of a regular n-gon. They found the shortest network for $n = 3, 4$, and 5 and proved that any $n - 1$ sides of the regular n-gon consist of the shortest network for $n > 13$. Jarnik and Kössler made no reference to the Fermat problem because their problem was quite different.

Courant and Robbins made the connection that the Fermat problem is the shortest interconnection network with $n = 3$ in their famous book "What is mathematics?" in 1941. They

renamed these problems the Steiner problem after Jacob Steiner, a German mathematician who studied the problem and generalized it to include an arbitrarily large set of points in the plane.

Gilbert and Pollak [47] named Steiner minimal trees as the shortest interconnecting networks (note that whenever edges have positive lengths, the shortest network must be a tree) and Steiner points for vertices in an Steiner minimal trees that are not among the n original points.

The Steiner tree problem has been extended to various metric spaces. Among them, the Euclidean Steiner tree (i.e., Steiner tree in the Euclidean plane), the rectilinear Steiner tree (i.e., Steiner tree in the rectilinear plane), and the Steiner tree problem in graphs (i.e., Steiner tree in an undirected network) have attracted the most attention.

Karp [33] showed that the Steiner tree problem in graphs is $\mathcal{NP}$-hard. Later, in 1978, Garay and Johnson showed that the rectilinear Steiner tree problem is also $\mathcal{NP}$-hard, while Garay, Graham, and Johnson [48] showed that the Euclidean Steiner tree problem is $\mathcal{NP}$-hard. Foulds and Graham [49] showed in 1982 that the Steiner tree problem in an n-dimensional cube is still $\mathcal{NP}$-hard. This means that it is likely that no Steiner tree problems have efficient optimal solutions.

There are two approaches to deal with $\mathcal{NP}$-hard problems: the exact algorithm and the approximate algorithm. The exact algorithm is the approach that obtains exactly optimal solutions even if the computational time becomes long. On the other hand, the approximate algorithm is the approach that obtains near optimal solutions in a reasonable time. Various Steiner tree problems are expected to apply to solve real-world problems. Thus, many researchers tries to develop efficient approximation algorithms to solve Steiner tree problems. The purpose of approximation algorithms to find the best possible solution in the shortest possible time. Approximation algorithms can be roughly divided into two types: a construction method and an improvement method. Construction methods find a feasible solution directly from an inputted instance. On the other hand, improvement methods try to improve a given feasible solution. Thus, both methods are closely related to each other.

The first construction method to solve the Steiner tree problem in graphs was proposed by Moore [47] in 1968. This algorithm uses the shortest path and minimum spanning tree to find a Steiner tree. The worst case approximation ratio of this algorithm is two. The approximation ratio is the ratio between the objective function value of the obtained solution by the algorithm and the objective function value of the optimal solution. Thus, the minimum value of the approximation ratio is unity (it means that the algorithm always can obtain the exact optimal solution). The performance of construction methods which have guarantee of the worst case solution are usually evaluated by the worst case approximation ratio. Zelikovsky [50] proposed the algorithm which uses a 3-Steiner tree to find a Steiner tree in 1990. A k-Steiner tree is a tree with at most k terminals. If k becomes large, making a k-Steiner tree becomes difficult. The worst case approximation ratio of this algorithm is

1.834. It is smaller than the algorithm which is proposed by Moore [47]. Using k-Steiner trees is one trend in developing approximation algorithms to solve the Steiner tree problem in graphs. Berman and Ramaiyer [51] proposed the 1.734 approximation algorithm which uses k-Steiner tree in 1991. Also, Zelikovsky proposed the 1.694 approximation algorithm which used k-Steiner tree in 1995. The next trend was caused by Karpinski and Zelikovsky [52]. They proposed to use the loss, and the worst case approximation ratio of this algorithm is 1.644. Since then, using loss has become a trend in the development of approximation algorithms to solve the Steiner tree problem in graphs. Hourardy and Prömel [53] proposed the 1.598 approximation algorithm in 1999. Also, Robins and Zelikovsky [54] proposed the 1.550 approximation algorithm in 2000. This algorithm is still the state-of-the-art algorithm for solving the Steiner tree problem in graphs.

On the other hand, improvement methods can be roughly divided into two types: local search and meta-heuristics. The local search repeats making neighborhood solutions from a given solution and select the one of them as the next solution. If there are no better solution in the neighborhood, the local search is terminated. The local search can improve a given solution in many case, however, it returns a local (not a global) optimum solution. To overcome this problem, meta-heuristics which controls the application of the local search have been developed.

As local search methods for solving the Steiner tree problem in graphs, Steiner vertex neighborhood and key path neighborhood are frequently used. Steiner vertex neighborhood tries to make a new Steiner tree by adding or removing a Steiner vertex or non-terminal to/from the current Steiner tree. Therefore, Steiner vertex neighborhood has the possibility of finding infeasible solutions. On the other hand, a key path neighborhood tries to make a new Steiner tree based on key paths. The key path is a simple path in a Steiner tree in which both ends are key vertices, i.e., terminals or non-terminals that have at least 3 degrees in the current tree. Thus, a Steiner tree is represented as a set of key paths. If we remove a key path from the current Steiner tree, this Steiner tree is divided into two components. The key path neighborhood reconnects these two components by its shortest path. Therefore, the key path neighborhood always obtains a feasible solution.

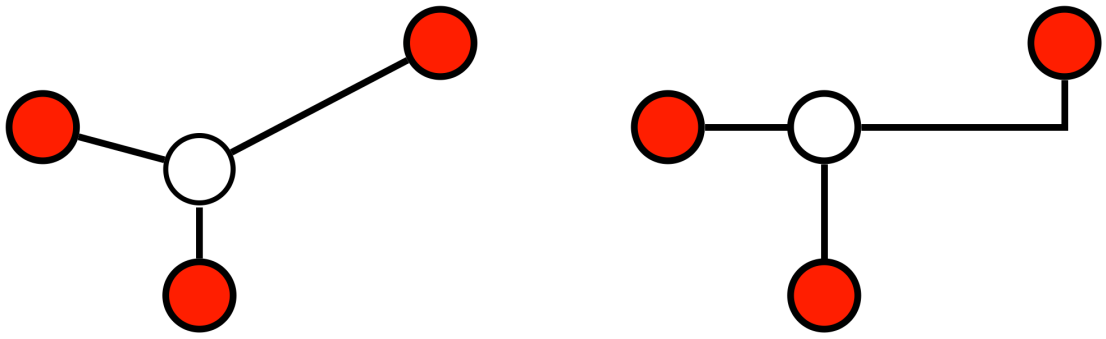
We can improve a Steiner tree by repeatedly applying local search methods. However, this method cannot find good solutions because it is easily trapped at local optima. To escape from such local optima, meta-heuristics are used. Meta-heuristics control the application of the local search. Many meta-heuristics have already proposed and combined with local searches to solve the Steiner tree problem in graphs, including simulated annealing [55], tabu search [56], greedy randomized adaptive search procedure [57] ant colony [58], and genetic algorithm [59].

2.2 Generalizations

The Steiner tree problem is very easy to express but very difficult to solve. Many variants of the Steiner tree problems have been studied by many researchers. In this section, we introduce the popular variants of the Steiner tree problem. The variants of the Steiner tree problem are roughly divided into two types: defined in the plane and defined in the graph.

2.2.1 Geometric Steiner tree problem

The geometric Steiner tree problem is a natural expansion of Fermat's question. In the geometric Steiner tree problem, using different distances is equivalent to using different objective functions. Therefore, the algorithms are tuned for each distance. The Euclidean distance and rectilinear distance are usually used. An example of the geometric Steiner tree problem is shown in Fig. 2.1. In Fig. 2.1, the red circles represent the given points and the white circles represent Steiner points, which are points added to minimize the total length of the network.



(a) Euclidean Steiner tree (b) Rectilinear Steiner tree

Figure 2.1: Example of the geometric Steiner tree problem. Red circles represent the given points and white circles represent the Steiner points: points added to minimize the total length of the network.

Euclidean Steiner tree problem

The Euclidean Steiner tree problem is defined as follows: given n points in a plane, find the shortest network that connects them by line segments under the condition that the addition of new points is permitted. Thus, the Euclidean Steiner tree problem is a generalization of Fermat's question: given three points in the plane, find the fourth point such that the sum of its distances to the three given points is minimum. The Euclidean Steiner tree problem is also known to be $\mathcal{NP}$-hard.

Rectilinear Steiner tree problem

The rectilinear Steiner tree problem is one of the geometric Steiner tree problems in a plane. It uses the rectilinear distance instead of the Euclidean distance. Therefore, the rectilinear Steiner tree problem is defined as follows: given n points in a plane, find the shortest network that connects them only by vertical and horizontal line segments under the condition that the addition of new points is permitted. The rectilinear Steiner tree problem is also known to be $\mathcal{NP}$-hard.

2.2.2 Steiner tree problem in graphs

The Steiner tree problem in graphs is a combinatorial version of the Steiner tree problem. We focus on this problem in this thesis. It should be noted that there is no relationship between algorithms for solving geometric Steiner tree problem and network Steiner tree problem, i.e., the Steiner tree problem in graphs.

Given an undirected graph $G = (V, E)$, where V is a set of vertices and E is a set of edges, a weight function $w : E \to \mathbb{Z}^+$, and a set of terminal vertices $T \subseteq V$, a Steiner tree is a tree $S = (V_S, E_S)$ such that $T \subseteq V_S \subseteq V$ and $E_S \subseteq E$. In other words, a Steiner tree is a tree that spans T. A Steiner tree generally consists of not only terminal vertices but also non-terminal vertices. Non-terminal vertices which included in a Steiner tree are called Steiner vertices.

The objective of the Steiner tree problem in graphs is to find a Steiner tree that minimizes the objective function F in Eq. (2.1).

$$F(S) = \sum_{e \in E} w(e) z(e), \tag{2.1}$$

where S is a Steiner tree, $w(e)$ is the weight of edge e, and $z(e)$ is a decision variable described as follows:

$$z(e) = \begin{cases} 1 & (e \in E_S \subseteq E), \\ 0 & (\text{otherwise}), \end{cases} \tag{2.2}$$

where $E_S \subseteq E$ is a subset of edges included in the Steiner tree S. Namely, the weight of the Steiner tree is the sum of the weight of the edges in the Steiner tree.

The Steiner tree problem in graphs is a generalization of two famous combinatorial optimization problems: the shortest path problem and the minimum spanning tree problem. If $|T| = 2$, where $|\cdot|$ is the number of elements, the Steiner tree problem in graphs is equivalent to finding the shortest path. The shortest path problem belongs to the class $\mathcal{P}$. Thus, it is easy to solve, for example, using the Dijkstra algorithm. On the other hand, if $|T| = |V|$, the Steiner tree problem in graphs is equivalent to finding the minimum spanning tree. The

minimum spanning tree problem also belongs to the class $\mathcal{P}$. Thus, it is easy to solve, for example, using the Kruskal algorithm. Except for these two special cases, the Steiner tree problem in graphs is an $\mathcal{NP}$-hard problem.

The Steiner tree problem in graphs is a basic and important combinatorial optimization problem. The Steiner tree problem in graphs is applied to real-world problems, such as design of very large-scale integration [2], multicast tree [3], and pipelines [60]. Because the Steiner tree problem in graphs is $\mathcal{NP}$-hard [33], many approximation methods have already been proposed to solve this problem.

Constrained Steiner tree problem

The constrained Steiner tree problem considers minimizing the sum of the weight of each directed path from a source vertex to all destinations with the sum of the delay of each directed path smaller than its limit. More precisely, the input to the constrained Steiner tree problem consists of a graph $G = (V, E)$, two weighting functions (weight $w : E \to \mathbb{R}^+$ and delay $d : E \to \mathbb{R}^+$), a source vertex $s \in V$, a set of destination vertices $D \subseteq V \setminus \{s\}$, and a delay constrained δ. The goal is to find a subgraph S of G such that S is a directed tree rooted at the source vertex s that spans all the destination vertices in D and that the sum of the delay from the source vertex s to each destination vertex is at most δ. The total weight of all edges in S is to be minimized.

Prize collecting Steiner tree problem

The prize collecting Steiner tree problem simultaneously considers minimization of the sum of the weights of edges included in a Steiner tree and the sum of the penalties of vertices not included in a Steiner tree. In the prize collecting Steiner tree problem, weights are associated with the edges, and penalties are associated with the vertices of the graph. More precisely, the input to the prize collecting Steiner tree problem consists of a graph $G = (V, E)$, a weight function $w : E \to \mathbb{R}^+$, and a penalty function $p : V \to \mathbb{R}^+$. The goal is to find a subgraph S of G such that S is a tree that minimizes the sum of the weight of edges included in the tree and the sum of the penalties of vertices not included in the tree.

2.3 Applications

The Steiner tree problem in graphs is one of the basic combinatorial optimization problems. Thus, it is applied to many real-world problems, such as the design of electronic circuits, computer networking, and telecommunication. In this section, we explain how these real-world problems correspond to Steiner tree problems.

2.3.1 Very large scale integration

Very large scale integration (VLSI) [2] is the process of creating an integrated circuit by placing thousands or millions of electronic components, such as transistors, diodes, registers, and capacitors on a single chip. It is used in almost all devices and systems, such as computers, information devices, communication devices, home appliances, and automobiles.

Recent years, the number of electronic components on a single chip is rapidly grew. Thus, the efficient algorithm for global routing in VLSI is strongly desired. To deal with it, good positioning of elements is essential, because it can suppress the heating, make chips as small as possible, and decrease the production costs. Many algorithms to minimizing the total wire length, minimizing total area, and minimizing the production cost of the chip have been proposed. Among them, most algorithms focus on minimizing the total wire length.

In the global routing phase of VLSI design, we determine the appropriate locations for macro cells. Macro cells are the logical units that perform the desired functions on the chip. They are characterized by their width, height, contact points (called terminals), and electronic properties. The cells then need to be assigned to a certain rectangular area and connected by wires. These wires usually placed rectilinearly on the chips. Thus, the global routing of the VLSI design is the same as the rectilinear Steiner tree problem.

2.3.2 Multicast routing

The multicast routing [3] is one of the routing methods in computer networks. The multicast routing distributes the data from a source devise to a group of destination devices simultaneously. For example, in teleconferencing, data from a source user is to be sent to a selected number of destination users. From this viewpoint, we can treat the making multicast routing tree problems as the Steiner tree problem in graphs. However, in this case, we have to consider not only the total length of the telecommunication tree but also the delay from a source to each destination. Thus, it is more appropriate to treat this problem as the constrained Steiner tree problems, which minimizes the total length of the tree with the sum of delay of each path between a source and a destination is smaller than its limits.

2.3.3 Pipeline

The pipelines which transport water, oil, and gas require very expensive production cost. Additionally, once these are made, it is difficult to make them from scratch. Therefore, it is important to design the pipeline with as low as possible installation cost and the maintenance cost before construction. The production cost of pipelines depends on the total length. Thus, we can treat the designing layout of pipeline problems as the Euclidean Steiner tree problems.

E.K. Donkoh, S.K. Amponsah, and K.F. Darkwah [4] reported that the total length of the minimum spanning tree and the Steiner tree of the pipeline of some west African countries, Ghana, Benin, Togo, and Nigeria are shorter than the real one. More precisely, the total length of the real pipeline is 788.90 km, however, the one of the minimum spanning tree is 712.30 km, and the one of the Steiner tree is 707.75 km. These results indicate that developing the algorithms to solve Euclidean Steiner tree problem is desired.

2.4 Representation of solutions

The objective function of the Steiner tree problem in graphs (Eq. (2.1)) indicates that a set of edges are required to calculate the objective function value. However, we can represent a solution of the Steiner tree problem in graphs as a set of vertices by using the induced subgraph and the minimum spanning tree. The number of vertices are usually smaller than the number of edges because the input of the Steiner tree problem in graphs must be connected. Thus, we can reduce the memory usage by representing solutions as a vertex set instead of a edge set.

Given a set of vertices V_S, it can be associated with a Steiner tree S corresponding to a minimum spanning tree of the graph induced in G by V_S. Let V_{S^*} be the set of vertices in the optimal solution of the instance. Then, the minimum Steiner tree S^* is also a minimum spanning tree of the graph induced in G by the vertex set V_{S^*}. From this viewpoint, we represent a solution or a Steiner tree as a set of vertices.

If the obtained Steiner tree S has non-terminal leaves or non-terminal vertices whose degree are one, these vertices are removed. This is because these vertices increase the weight of the Steiner tree regardless of whether they contribute to connecting terminal vertices.

2.5 Benchmark problems

The availability of computational test sets for combinatorial optimization is very important for the development algorithms and comparison of their performances. For example, TSPLIB has very much influenced the development of strong algorithms for solving the traveling salesman problem. For the Steiner tree problem in graphs, SteinLib [61] is one of the most popular libraries of instances. Thus, we use instances B, C, D, I080, I160, and I320 in SteinLib.

Tables 2.1–2.6 lists the most important characteristics of instances: instance name, number of vertices $|V|$, number of edges $|E|$, and number of terminals $|T|$. Column "DC" shows the difficulty classification, L, P, and NP. L indicates that the instance may be solved to optimality by applying local reduction techniques. P indicates that the solution to the linear programming relaxation is non-fractional. All other instances are classified as NP, i.e., no polynomial time algorithm is known to date for these instances. A small s, m, h, or d indicates that an instance may be solved to optimality within seconds, minutes, hours, or days with state-of-the-art algorithms, respectively.

The most popular instance generation method was proposed by Aneja in 1980. Each instance corresponds to a randomly generated connected graph with a specified set of $|V|$ vertices and $|E|$ edges. To generate instances that guarantee the existence of a feasible solution, connectivity must be assured. Therefore, first a random spanning tree of the entire set V of vertices is generated. Then, additional edges are added randomly to the graph. Finally, random weights between 1 and 10 are assigned to all edges.

This instance generation method has been applied by various authors. Beasley generated a set of 18 instances according to this method with 50, 75, and 100 vertices. For each number of vertices the number of terminals was chosen to be $|V|/6$, $|V|/4$, and $|V|/2$ and for each graph the number of edges was specified to achieve an average vertex degree of 2.5 and 4. This set of 18 instances became the data set B (Table 2.1). After that, Beasley extended the size of instances when he generated new data sets (Tables 2.2 and 2.3).

Tables 2.4–2.6 show the incidence instances. These problem instances are randomly generated with $|V| = 80, 160$, and 320. For each of them, twenty variants are generated combining four terminal set sizes, $|T| = \log_2 |V|, \sqrt{|V|}, 2\sqrt{|V|}$, and $|V|/4$, with five different densities, $|E| = 3|V|/2, |V| \ln |V|, |V|(|V| - 1)/2, 2|V|$, and $|V|(|V| - 1)/10$, with all values rounded down to the integer value. The instance name shows the information of the number of vertices, terminals, and the density of edges. For example, i080-012 means that the instance has 80 vertices (thus, $|V| = 80$), $\lfloor \log_2 |V| \rfloor = \lfloor \log_2 80 \rfloor = \lfloor 6.321928 \cdots \rfloor = 6$ terminals ($\log_2 |V|$ is the first element of the above sequence of the number of terminals), and $\lfloor |V| \ln |V| \rfloor = \lfloor 80 \ln 80 \rfloor = \lfloor 350.562130 \cdots \rfloor = 350$ edges ($|V| \ln |V|$ is the second element of the above sequence of the number of edges), where $\lfloor n \rfloor$ is the floor function of n. The last numeral 2 indicates that this is the second instance that have the same number of vertices,

terminals, and edges.

In the incidence instances, the weight on each edge (i, j), $w(i, j)$, is defined with a sample r from a normal distribution, rounded to the closest integer value with a minimum value of 1 and maximum value of 500, i.e., $w(i, j) = \min\{500,\ \max\{1, \text{round}(r)\}\}$, where round(n) is a function that rounds down n to the closest integer value. To obtain a graph that is much harder to reduce by preprocessing techniques, three distributions with different mean values are used. Any edge (i, j) is incident to zero, one, or two terminals. The mean of r is 100 for edges (i, j) with $i, j \notin T$ (no incidence with T), 200 on edges (i, j) with one terminal, and 300 on edges (i, j) with both end-vertices $i, j \in T$. The standard deviation for each of the three normal distributions is 5.

Table 2.1: Benchmark problem set B

Name	$\|V\|$	$\|E\|$	$\|T\|$	DC	Opt
b01	50	63	9	Ls	82
b02	50	63	13	Ls	83
b03	50	63	25	Ls	138
b04	50	100	9	Ls	59
b05	50	100	13	Ls	61
b06	50	100	25	Ps	122
b07	75	94	13	Ls	111
b08	75	94	19	Ls	104
b09	75	94	38	Ls	220
b10	75	150	13	Ps	86
b11	75	150	19	Ls	88
b12	75	150	38	Ls	174
b13	100	125	17	Ps	165
b14	100	125	25	Ps	235
b15	100	125	50	Ps	318
b16	100	200	17	Ps	127
b17	100	200	25	Ps	131
b18	100	200	50	Ps	218

Table 2.2: Benchmark problem set C

Name	$\|V\|$	$\|E\|$	$\|T\|$	DC	Opt
c01	500	625	5	Ps	85
c02	500	625	10	Ps	144
c03	500	625	83	Ps	754
c04	500	625	125	Ps	1079
c05	500	625	250	Ls	1579
c06	500	1000	5	Ps	55
c07	500	1000	10	Ps	102
c08	500	1000	83	Ps	509
c09	500	1000	125	Ps	707
c10	500	1000	250	Ps	1093
c11	500	2500	5	Ps	32
c12	500	2500	10	Ps	46
c13	500	2500	83	Ps	258
c14	500	2500	125	Ps	323
c15	500	2500	250	Ls	556
c16	500	12500	5	Ps	11
c17	500	12500	10	Ps	18
c18	500	12500	83	Ps	113
c19	500	12500	125	Ps	146
c20	500	12500	250	Ls	267

Table 2.3: Benchmark problem set D

Name	$\|V\|$	$\|E\|$	$\|T\|$	DC	Opt
d01	1000	1250	5	Ps	106
d02	1000	1250	10	Ps	220
d03	1000	1250	167	Ps	1565
d04	1000	1250	250	Ps	1935
d05	1000	1250	500	Ps	3250
d06	1000	2000	5	Ps	67
d07	1000	2000	10	Ps	103
d08	1000	2000	167	Ps	1072
d09	1000	2000	250	Ps	1448
d10	1000	2000	500	Ps	2110
d11	1000	5000	5	Pm	29
d12	1000	5000	10	Pm	42
d13	1000	5000	167	Ps	500
d14	1000	5000	250	Ps	667
d15	1000	5000	500	Ps	1116
d16	1000	25000	5	Pm	13
d17	1000	25000	10	Pm	23
d18	1000	25000	167	Ps	223
d19	1000	25000	250	Ps	310
d20	1000	25000	500	Ls	537

Table 2.4: Benchmark problem set I080

Name	$\|V\|$	$\|E\|$	$\|T\|$	DC	Opt
i080-001	80	120	6	Ps	1787
i080-002	80	120	6	Ps	1607
i080-003	80	120	6	Ps	1713
i080-004	80	120	6	Ps	1866
i080-005	80	120	6	Ps	1790
i080-011	80	350	6	Ps	1479
i080-012	80	350	6	Ps	1484
i080-013	80	350	6	Ps	1381
i080-014	80	350	6	Ps	1397
i080-015	80	350	6	Ps	1495
i080-021	80	3160	6	Ps	1175
i080-022	80	3160	6	Ps	1178
i080-023	80	3160	6	Ps	1174
i080-024	80	3160	6	Ps	1161
i080-025	80	3160	6	Ps	1162
i080-031	80	160	6	Ps	1570
i080-032	80	160	6	Ps	2088
i080-033	80	160	6	Ps	1794
i080-034	80	160	6	Ps	1688
i080-035	80	160	6	Ps	1862
i080-041	80	632	6	Ps	1276
i080-042	80	632	6	Ps	1287
i080-043	80	632	6	Ps	1295
i080-044	80	632	6	NPs	1366
i080-045	80	632	6	Ps	1310
i080-101	80	120	8	Ps	2608
i080-102	80	120	8	Ps	2403
i080-103	80	120	8	Ps	2603
i080-104	80	120	8	Ps	2486
i080-105	80	120	8	Ps	2203
i080-111	80	350	8	NPs	2051
i080-112	80	350	8	Ps	1885
i080-113	80	350	8	Ps	1884
i080-114	80	350	8	Ps	1895
i080-115	80	350	8	Ps	1868
i080-121	80	3160	8	Ps	1561
i080-122	80	3160	8	Ps	1561
i080-123	80	3160	8	Ps	1569
i080-124	80	3160	8	Ls	1555
i080-125	80	3160	8	Ps	1572
i080-131	80	160	8	Ps	2284
i080-132	80	160	8	Ps	2180
i080-133	80	160	8	Ps	2261
i080-134	80	160	8	Ps	2070
i080-135	80	160	8	Ps	2102
i080-141	80	632	8	Ps	1788
i080-142	80	632	8	Ps	1708
i080-143	80	632	8	NPs	1767
i080-144	80	632	8	Ps	1772
i080-145	80	632	8	Ps	1762

Name	$\|V\|$	$\|E\|$	$\|T\|$	DC	Opt
i080-201	80	120	16	Ps	4760
i080-202	80	120	16	Ps	4650
i080-203	80	120	16	Ps	4599
i080-204	80	120	16	Ps	4492
i080-205	80	120	16	Ps	4564
i080-211	80	350	16	Ps	3631
i080-212	80	350	16	NPs	3677
i080-213	80	350	16	NPs	3678
i080-214	80	350	16	NPs	3734
i080-215	80	350	16	NPs	3681
i080-221	80	3160	16	Ps	3158
i080-222	80	3160	16	Ps	3141
i080-223	80	3160	16	Ps	3156
i080-224	80	3160	16	Ps	3159
i080-225	80	3160	16	Ps	3150
i080-231	80	160	16	Ps	4354
i080-232	80	160	16	Ps	4199
i080-233	80	160	16	Ps	4118
i080-234	80	160	16	Ps	4274
i080-235	80	160	16	NPs	4487
i080-241	80	632	16	NPm	3538
i080-242	80	632	16	Pm	3458
i080-243	80	632	16	NPm	3474
i080-244	80	632	16	NPs	3466
i080-245	80	632	16	NPs	3467
i080-301	80	120	20	Ps	5519
i080-302	80	120	20	Ps	5944
i080-303	80	120	20	Ps	5777
i080-304	80	120	20	Ps	5586
i080-305	80	120	20	NPs	5932
i080-311	80	350	20	Ps	4554
i080-312	80	350	20	NPs	4534
i080-313	80	350	20	Ps	4509
i080-314	80	350	20	NPs	4515
i080-315	80	350	20	NPs	4459
i080-321	80	3160	20	Ps	3932
i080-322	80	3160	20	Ps	3937
i080-323	80	3160	20	Ps	3946
i080-324	80	3160	20	Ps	3932
i080-325	80	3160	20	Ps	3924
i080-331	80	160	20	NPs	5226
i080-332	80	160	20	NPs	5362
i080-333	80	160	20	Ps	5381
i080-334	80	160	20	Ps	5264
i080-335	80	160	20	Ps	4953
i080-341	80	632	20	Ps	4236
i080-342	80	632	20	NPm	4337
i080-343	80	632	20	NPs	4246
i080-344	80	632	20	NPs	4310
i080-345	80	632	20	NPm	4341

Table 2.5: Benchmark problem set I160

Name	$\|V\|$	$\|E\|$	$\|T\|$	DC	Opt
i160-001	160	240	7	Ps	2490
i160-002	160	240	7	Ps	2158
i160-003	160	240	7	Ps	2297
i160-004	160	240	7	Ps	2370
i160-005	160	240	7	Ps	2495
i160-011	160	812	7	Ps	1677
i160-012	160	812	7	Ps	1750
i160-013	160	812	7	Ps	1661
i160-014	160	812	7	Ps	1778
i160-015	160	812	7	Ps	1768
i160-021	160	12720	7	Ps	1352
i160-022	160	12720	7	Ps	1365
i160-023	160	12720	7	Ps	1351
i160-024	160	12720	7	Ps	1371
i160-025	160	12720	7	Ps	1366
i160-031	160	320	7	Ps	2170
i160-032	160	320	7	Ps	2330
i160-033	160	320	7	NPs	2101
i160-034	160	320	7	Ps	2083
i160-035	160	320	7	Ps	2103
i160-041	160	2544	7	Ps	1494
i160-042	160	2544	7	Ps	1486
i160-043	160	2544	7	NPs	1549
i160-044	160	2544	7	Ps	1478
i160-045	160	2544	7	NPs	1554
i160-101	160	240	12	Ps	3859
i160-102	160	240	12	Ps	3747
i160-103	160	240	12	Ps	3837
i160-104	160	240	12	Ps	4063
i160-105	160	240	12	Ps	3563
i160-111	160	812	12	Ps	2869
i160-112	160	812	12	NPs	2924
i160-113	160	812	12	Ps	2866
i160-114	160	812	12	Ps	2989
i160-115	160	812	12	NPm	2937
i160-121	160	12720	12	Pm	2363
i160-122	160	12720	12	Ps	2348
i160-123	160	12720	12	Ps	2355
i160-124	160	12720	12	Ps	2352
i160-125	160	12720	12	Ps	2351
i160-131	160	320	12	Ps	3356
i160-132	160	320	12	Ps	3450
i160-133	160	320	12	Ps	3585
i160-134	160	320	12	Ps	3470
i160-135	160	320	12	Ps	3716
i160-141	160	2544	12	Ps	2549
i160-142	160	2544	12	NPm	2562
i160-143	160	2544	12	Ps	2557
i160-144	160	2544	12	NPm	2607
i160-145	160	2544	12	Ps	2578

Name	$\|V\|$	$\|E\|$	$\|T\|$	DC	Opt
i160-201	160	240	24	NPs	6923
i160-202	160	240	24	Ps	6930
i160-203	160	240	24	Ps	7243
i160-204	160	240	24	Ps	7068
i160-205	160	240	24	Ps	7122
i160-211	160	812	24	NPm	5583
i160-212	160	812	24	NPm	5643
i160-213	160	812	24	NPm	5647
i160-214	160	812	24	NPm	5720
i160-215	160	812	24	NPm	5518
i160-221	160	12720	24	Pm	4729
i160-222	160	12720	24	Pm	4697
i160-223	160	12720	24	Pm	4730
i160-224	160	12720	24	Pm	4721
i160-225	160	12720	24	Pm	4728
i160-231	160	320	24	Ps	6662
i160-232	160	320	24	NPs	6558
i160-233	160	320	24	Ps	6339
i160-234	160	320	24	Ps	6594
i160-235	160	320	24	Ps	6764
i160-241	160	2544	24	NPh	5086
i160-242	160	2544	24	NPh	5106
i160-243	160	2544	24	NPm	5050
i160-244	160	2544	24	NPm	5076
i160-245	160	2544	24	NPm	5084
i160-301	160	240	40	Ps	11816
i160-302	160	240	40	Ps	11497
i160-303	160	240	40	Ps	11445
i160-304	160	240	40	Ps	11448
i160-305	160	240	40	NPs	11423
i160-311	160	812	40	NPm	9135
i160-312	160	812	40	NPm	9052
i160-313	160	812	40	NPh	9159
i160-314	160	812	40	NPm	8941
i160-315	160	812	40	NPm	9086
i160-321	160	12720	40	Pm	7876
i160-322	160	12720	40	NPm	7859
i160-323	160	12720	40	Pm	7876
i160-324	160	12720	40	NPm	7884
i160-325	160	12720	40	NPm	7862
i160-331	160	320	40	Ps	10414
i160-332	160	320	40	NPs	10806
i160-333	160	320	40	Ps	10561
i160-334	160	320	40	Ps	10327
i160-335	160	320	40	Ps	10589
i160-341	160	2544	40	NPh	8331
i160-342	160	2544	40	NPd	8348
i160-343	160	2544	40	NPh	8275
i160-344	160	2544	40	NPh	8307
i160-345	160	2544	40	NPh	8327

Table 2.6: Benchmark problem set I320

Name	$\|V\|$	$\|E\|$	$\|T\|$	DC	Opt
i320-001	320	480	8	Ps	2672
i320-002	320	480	8	Ps	2847
i320-003	320	480	8	Ps	2972
i320-004	320	480	8	Ps	2905
i320-005	320	480	8	Ps	2991
i320-011	320	1845	8	NPs	2053
i320-012	320	1845	8	Ps	1997
i320-013	320	1845	8	Ps	2072
i320-014	320	1845	8	NPs	2061
i320-015	320	1845	8	NPs	2059
i320-021	320	51040	8	Lm	1553
i320-022	320	51040	8	Pm	1565
i320-023	320	51040	8	Pm	1549
i320-024	320	51040	8	Pm	1553
i320-025	320	51040	8	Pm	1550
i320-031	320	640	8	NPs	2673
i320-032	320	640	8	NPs	2770
i320-033	320	640	8	Ps	2769
i320-034	320	640	8	Ps	2521
i320-035	320	640	8	Ps	2385
i320-041	320	10208	8	Ps	1707
i320-042	320	10208	8	Ps	1682
i320-043	320	10208	8	NPm	1723
i320-044	320	10208	8	Ps	1681
i320-045	320	10208	8	Ps	1686
i320-101	320	480	17	Ps	5548
i320-102	320	480	17	Ps	5556
i320-103	320	480	17	Ps	6239
i320-104	320	480	17	Ps	5703
i320-105	320	480	17	Ps	5928
i320-111	320	1845	17	NPm	4273
i320-112	320	1845	17	NPm	4213
i320-113	320	1845	17	NPm	4205
i320-114	320	1845	17	NPm	4104
i320-115	320	1845	17	NPs	4238
i320-121	320	51040	17	Pm	3321
i320-122	320	51040	17	Pm	3314
i320-123	320	51040	17	Pm	3332
i320-124	320	51040	17	Pm	3323
i320-125	320	51040	17	Pm	3340
i320-131	320	640	17	Ps	5255
i320-132	320	640	17	Ps	5052
i320-133	320	640	17	Ps	5125
i320-134	320	640	17	Ps	5272
i320-135	320	640	17	NPs	5342
i320-141	320	10208	17	NPh	3606
i320-142	320	10208	17	Pm	3567
i320-143	320	10208	17	Pm	3561
i320-144	320	10208	17	Ps	3512
i320-145	320	10208	17	NPm	3601

Name	$\|V\|$	$\|E\|$	$\|T\|$	DC	Opt
i320-201	320	480	34	Ps	10044
i320-202	320	480	34	Ps	11223
i320-203	320	480	34	Ps	10148
i320-204	320	480	34	Ps	10275
i320-205	320	480	34	NPs	10573
i320-211	320	1845	34	NPm	8039
i320-212	320	1845	34	NPm	8044
i320-213	320	1845	34	NPm	7984
i320-214	320	1845	34	NPm	8046
i320-215	320	1845	34	NPh	8015
i320-221	320	51040	34	NPh	6679
i320-222	320	51040	34	NPh	6686
i320-223	320	51040	34	NPm	6695
i320-224	320	51040	34	NPm	6694
i320-225	320	51040	34	NPm	6691
i320-231	320	640	34	NPs	9862
i320-232	320	640	34	NPs	9933
i320-233	320	640	34	Ps	9787
i320-234	320	640	34	Ps	9517
i320-235	320	640	34	Ps	9945
i320-241	320	10208	34	NPh	7027
i320-242	320	10208	34	NPh	7072
i320-243	320	10208	34	NPh	7044
i320-244	320	10208	34	NPh	7078
i320-245	320	10208	34	NPh	7046
i320-301	320	480	80	Ps	23279
i320-302	320	480	80	Ps	23387
i320-303	320	480	80	Ps	22693
i320-304	320	480	80	Ps	23451
i320-305	320	480	80	NPs	22547
i320-311	320	1845	80	NPd	17945
i320-312	320	1845	80	NPd	18122
i320-313	320	1845	80	NPd	17991
i320-314	320	1845	80	NPd	18088
i320-315	320	1845	80	NPd	17987
i320-321	320	51040	80	NPh	15648
i320-322	320	51040	80	NPh	15646
i320-323	320	51040	80	NPh	15654
i320-324	320	51040	80	NPh	15667
i320-325	320	51040	80	NPh	15649
i320-331	320	640	80	NPs	21517
i320-332	320	640	80	NPs	21674
i320-333	320	640	80	NPs	21339
i320-334	320	640	80	Ps	21415
i320-335	320	640	80	NPs	21378
i320-341	320	10208	80	NPh	16296
i320-342	320	10208	80	NPh	16228
i320-343	320	10208	80	NPh	16281
i320-344	320	10208	80	NPh	16295
i320-345	320	10208	80	NPh	16289

Chapter 3

Construction method combined with network centralities

3.1 Background of Chapter 3

The construction algorithm is an approximation algorithm that finds an approximate solution from an instance. The construction algorithm plays an important role when we want to obtain approximated solutions in reasonable time. Additionally, it is important to develop efficient construction algorithms because construction algorithms are used to find initial solutions for improvement algorithms.

Many construction algorithms to solve the Steiner tree problem in graphs use the shortest path and the minimum spanning tree. For example, the shortest path heuristic (SPH) [35] proposed by Takahashi and Matsuyama starts from a tree that consists of an arbitrary terminal. Then, it repeats including the shortest path between two vertices: one is a vertex included in a tree and the other is a terminal not included in a tree. The distance network heuristic (DNH) [36] proposed by Kou, Markowsky, and Berman starts from a complete graph of terminals. Then, it makes a minimum spanning tree of the complete graph. After that, it replaces edges of the minimum spanning tree to the shortest path between both sides of vertices of the edge in the input graph. Both SPH and DNH are 2-approximation algorithms; however, SPH can find smaller-weight Steiner trees than DNH. The reason why SPH shows better performance than DNH is that SPH considers the overlap of the shortest paths between terminals. Thus, the overlap of the shortest paths between terminals is important to find good Steiner trees.

However, if multiple shortest paths exist between terminal pairs, SPH sometimes finds large-weight Steiner trees. To overcome this issue, a simple approach is to select a shortest path that is expected to overlap many shortest paths from all shortest paths. Unfortunately, enumerating all shortest paths has high computation costs. Thus, this approach is difficult to realize.

In complex network theory, network centrality that measure the importance of vertices and edges exists. Important vertices and edges in networks are expected to also be important for making small-weight Steiner trees. From this viewpoint, we propose a construction method that uses network centralities.

3.2 Basic algorithms of the construction method

3.2.1 Shortest path heuristic

The shortest path heuristic (SPH) [35] was proposed by Takahashi and Matsuyama. It starts from a tree that consists of an arbitrary terminal. Then, it repeats including the shortest path between two vertices: one is a vertex included in the tree and the other is a terminal not included in the tree.

The procedure of SPH is as follows. First, set $S_{\mathrm{SPH}} = (V_{\mathrm{SPH}}, E_{\mathrm{SPH}})$, where $V_{\mathrm{SPH}} = \{t_0\}$, $E_{\mathrm{SPH}} = \{\emptyset\}$, and t_0 is an arbitrary terminal. We randomly select t_0 from T. Secondly, repeat including the shortest path between $v \in V_{\mathrm{SPH}}$ and $t \in T \notin V_{\mathrm{SPH}}$ to S_{SPH}. If $T \subseteq S_{\mathrm{SPH}}$, the SPH is terminated. The computation cost of the SPH is $\mathcal{O}(|T||E| + |T||V| \log |V|)$, where $|V|$ is the number of vertices, $|E|$ is the number of edges, and $|T|$ is the number of terminals.

Figure 3.1 shows a schematic diagram of the SPH. In Fig. 3.1, circles indicate vertices and lines indicate edges. Red circles are terminals and bold lines are edges included in the Steiner tree. We assume that the weight of each edge is unity for the sake of simplicity. The SPH starts from a tree that consists of an arbitrary terminal. We select vertex a in Fig. 3.1 (a). Then, repeat including the shortest path between a vertex included in the tree and a terminal not included in the tree. We connect vertex a and vertex f by using its shortest path. Then, we connect vertex b and vertex d by using its shortest path (Fig. 3.1 (b)). After that, we connect vertex f and vertex k by using its shortest path (Fig. 3.1 (c)). If all terminals are connected, SPH is terminated (Fig. 3.1 (d)). The objective function value of Fig. 3.1 (d) is six.

3.2.2 Distance network heuristic

The distance network heuristic (DNH) [36] was proposed by Kou, Markowsky and Berman. It starts from a complete graph of terminals, then, makes a minimum spanning tree of the complete graph. After that, replace edges of the minimum spanning tree to the shortest path between both sides of vertices of the edge in the input graph.

The procedure of DNH is as follows. First, construct a complete graph of T, $H = (T, E_H)$, where E_H is the set of edges of the complete graph. Weights of edges in H_E equal the shortest distances between each terminal pair in the input graph G. If there are some replaceable shortest paths, select an arbitrary one. Next, construct a minimum spanning tree of H, MST(H). Finally, replace edges of MST(H) by the shortest path in G. The computation cost of the DNH depends on the computation cost of the shortest path or the Dijkstra algorithm [31], $\mathcal{O}(|T||E| + |T||V| \log |V|)$ with Fibonacci heap [62].

Figure 3.2 shows a schematic diagram of the DNH. In Fig. 3.2, circles indicate vertices and lines indicate edges. Red circles are terminals and bold lines are edges included in the Steiner tree. We assume that the weight of each edge is unity. The DNH considers a

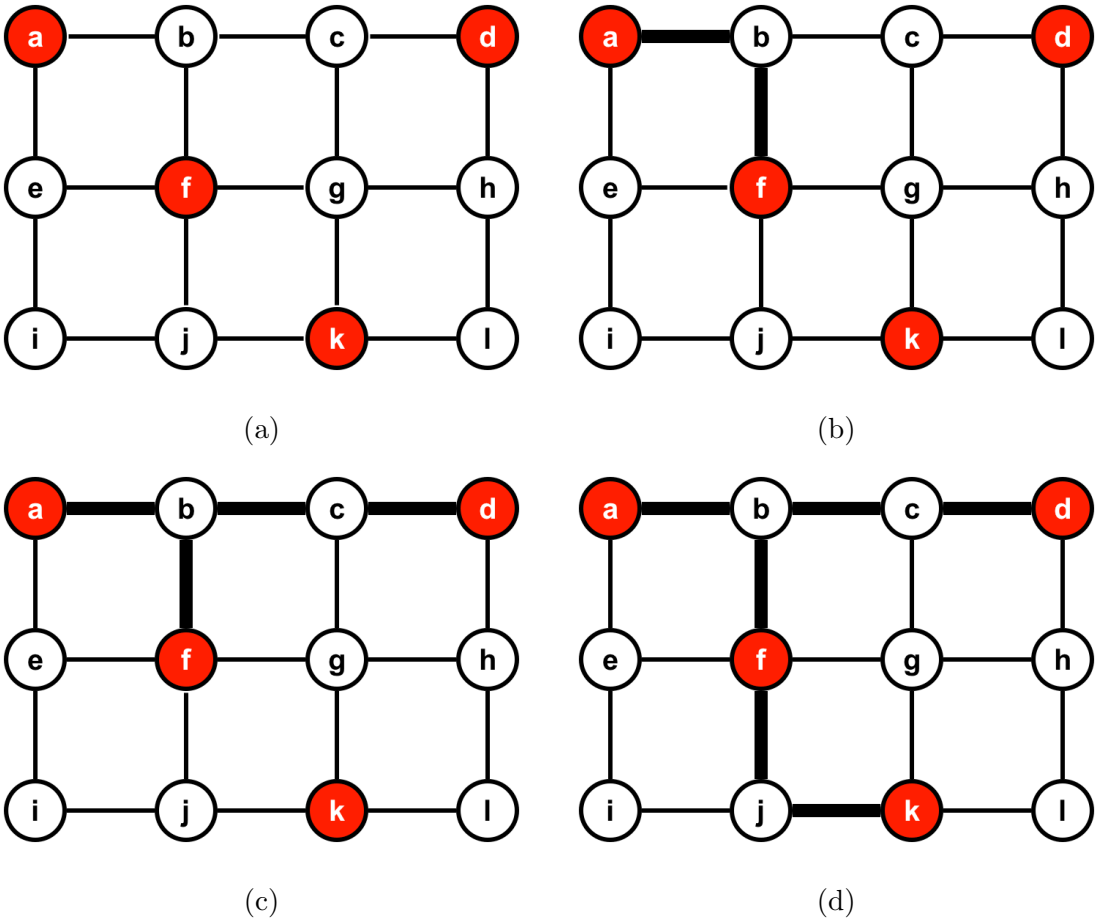


Figure 3.1: Schematic diagram of SPH. Circles indicate vertices and lines indicate edges. Red circles are terminals and bold lines are edges included in the Steiner tree. We assume that the weight of each edge is unity. (a) A given graph. SPH starts from a tree that consists of an arbitrary terminal (vertex a in this example). (b) Connect vertex a and vertex f, which is the nearest terminal to vertex a, by using its shortest path. Two patterns of shortest paths exist between vertices a and f. Here, we select an arbitrary one. (c) Connect vertex b and vertex d by using its shortest path. (d) Connect vertex f and vertex k by using its shortest path. The objective function value of (d) is six (by Eq. (2.1)).

distance network that is a complete graph consisting of terminals (Fig. 3.2 (b)). Then, we consider a minimum spanning tree of the distance network (Fig. 3.2 (c)). After that, edges of the minimum spanning tree are replaced by the shortest path between both end vertices (Fig. 3.2 (d)). The objective function value of Fig. 3.2 (d) is six.

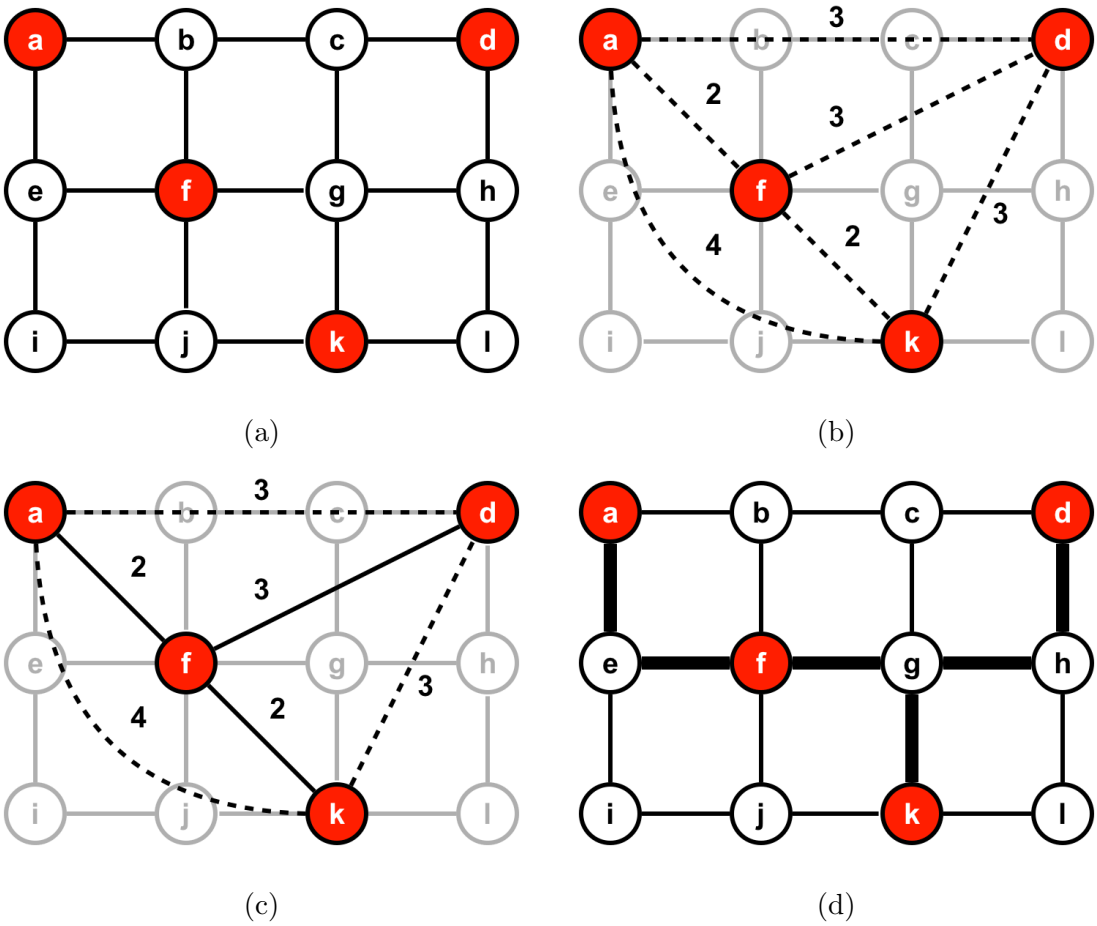


Figure 3.2: Schematic diagram of DNH. (a) A given graph. (b) A distance network of (a). A distance network is a complete graph consisting of all terminals. Broken lines indicate edges included in this distance network. Each edge weight equals the shortest distance between terminal pairs in the input graph. (c) A minimum spanning tree of (b). Solid black lines indicate edges included in this minimum spanning tree. (d) A Steiner tree obtained using DNH. The objective function value of (d) is six (by Eq. (2.1)).

3.2.3 Average distance heuristic

The average distance heuristic (ADH) [37] also uses the shortest path for constructing a Steiner tree. The ADH repeats combining trees until it becomes a Steiner tree. At first, $|T|$

trees consisting of one terminal exist. Then, two trees are combined by using the shortest path between these trees via a vertex v that minimizes an evaluation function L (Eq. (3.1)).

$$L(v) = \min_{1 \leq r \leq k} \sum_{i=0}^{r} \frac{d(v, t_{v.i})}{r}, \tag{3.1}$$

where k is the number of trees, $t_{v.i}$ is the ith nearest tree from v, and $d(v, t_{v.i})$ is the shortest distance between the vertex v and the nearest vertex in $t_{v.i}$ from v. $t_{v.i}$ is ordered by increasing distance from v. Namely, $t_{v.0}, t_{v.1}, ..., t_{v.k}$, where $d(v, t_{v.0}) \leq d(v, t_{v.1}) \leq ... \leq d(v, t_{v.k})$. Then, define $L_1(v), L_2(v), ..., L_k(v)$ by $L_1(v) = d(n, t_{v.0}) + d(n, t_{v.1})$ and $L_r(v) = [(r-1)L_{r-1}(v) + d(v, t_{v.r})]/r, 1 < r \leq k$. Clearly, $L(v) = \min\{L_r(v) : 1 \leq r \leq k\}$.

An example of the process of the ADH is shown in Fig. 3.3. First, four trees (red, blue, orange, and green) exist (Fig. 3.3 (a)). The right table shows the shortest distance between vertices and trees of the graph. From this table, for example, $t_{\mathrm{a}.0} = \mathrm{a}$, $t_{\mathrm{a}.1} = \mathrm{f}$, $t_{\mathrm{a}.2} = \mathrm{d}$, and $t_{\mathrm{a}.3} = \mathrm{k}$. Thus, $L_1(\mathrm{a}) = 0 + 2 = 2$, $L_2(\mathrm{a}) = (2 + 2)/2 = 2$, and $L_3(\mathrm{a}) = (2 \cdot 2 + 4)/3 = 8/3$. Therefore, $L(\mathrm{a}) = 2$. Similarly, $L(\mathrm{b}) = 2$, $L(\mathrm{c}) = 2.333\cdots$, $L(\mathrm{d}) = 3$, $L(\mathrm{e}) = 2$, $L(\mathrm{f}) = 2$, $L(\mathrm{g}) = 2$, $L(\mathrm{h}) = 2$, $L(\mathrm{i}) = 3$, $L(\mathrm{j}) = 2$, $L(\mathrm{k}) = 2$, and $L(\mathrm{l}) = 2$. In this case, one of the vertices that minimizes L is vertex b. Then, two trees nearest vertex b (red and orange) are combined via vertex b (Fig. 3.3 (b)).

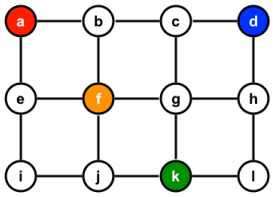
Next, three trees (red, blue, and green) exist (Fig. 3.3 (b)). The right table shows the shortest distance between vertices and trees of the graph. In this case, one of the vertices that minimize L is vertex c. Then, two trees nearest vertex c (red and blue) are combined via vertex c (Fig. 3.3 (c)).

Next, two trees (red and green) exist (Fig. 3.3 (c)). The right table shows the shortest distance between vertices and trees of the graph. In this case, one of the vertices that minimize L is vertex g. Then, two trees nearest vertex g (red and green) are combined via vertex g (Fig. 3.3 (d)). This is the Steiner tree, and the ADH is terminated. The computation cost of the ADH is $\mathcal{O}(|T||V||E| + |T||V|^2 \log |V|)$.

3.2.4 Postprocessing

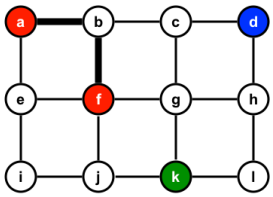
The obtained solutions sometimes have loops. Thus, we applied postprocessing to remove any loops. We use the minimum spanning tree for postprocessing. The procedure of postprocessing is as follows. First, make the minimum spanning tree of the induced subgraph by the obtained solution. Then, remove non-terminal leaves. The induced subgraph is a subgraph of the given graph, which consists of vertices included in the obtained solution and edges belonging to these vertices. A leaf is a vertex with degree one. If a leaf is non-terminal, this vertex does not need to span. Thus, we remove them.

Figure 3.4 shows a schematic diagram of the postprocessing. The postprocessing is applied to obtain a subgraph as in Fig. 3.4 (a). First, make an induced subgraph by the



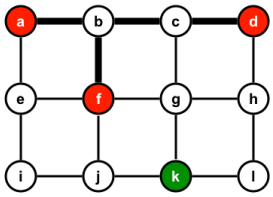
(a)

	a	b	c	d	e	f	g	h	i	j	k	l
a	0	1	2	3	1	2	3	4	2	3	4	5
d	3	2	1	0	4	3	2	1	5	4	3	2
f	2	1	2	3	1	0	1	2	2	1	2	3
k	4	3	2	3	3	2	1	2	2	1	0	1
L	2	2	2.3	3	2	2	2	2	3	2	2	3



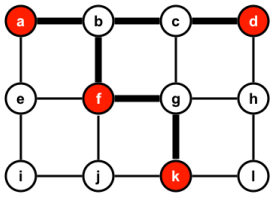
(b)

	a	b	c	d	e	f	g	h	i	j	k	l
{a, b, f}	0	0	1	2	1	0	1	2	2	1	2	3
d	3	2	1	0	4	3	2	1	5	4	3	2
k	4	3	2	3	3	2	1	2	2	1	0	1
L	3	3	2	3	4	2	2	2	4	2	2	3



(c)

	a	b	c	d	e	f	g	h	i	j	k	l
{a, b, c, d, f}	0	0	0	0	1	0	1	1	2	1	2	2
k	4	3	2	3	3	2	1	2	2	1	0	1
L	4	3	2	3	4	2	2	3	4	2	2	3



(d)

Figure 3.3: Schematic diagram of ADH. Colored vertices indicate terminals. Different colors indicate different trees. The right table indicates the shortest distance between vertices and trees. (a) Start with four trees that include a terminal. A vertex that minimizes L is selected to combine two trees. In this case, vertex b is selected to combine the red and orange trees. (b) Red and orange trees in (a) are combined via vertex b. The number of trees is reduced from four to three. Here, vertex c is selected to combine the red and blue trees. (c) Red and blue trees in (b) are combined via vertex c. The number of trees is reduced from three to two. Here, vertex g is selected to combine the red and green trees. (d) Red and green trees in (c) are combined via vertex g. Then, the forest becomes a tree. This tree is a Steiner tree obtained using ADH.

obtained subgraph (Fig. 3.4 (b)). Then, make a minimum spanning tree of the induced subgraph (Fig. 3.4 (c)). Finally, trim any non-terminal leaves of the minimum spanning tree (Fig. 3.4 (d)). After applied the postprocessing, the obtained solution must satisfy the requirements of the Steiner tree, i.e., a tree that spans all terminals.

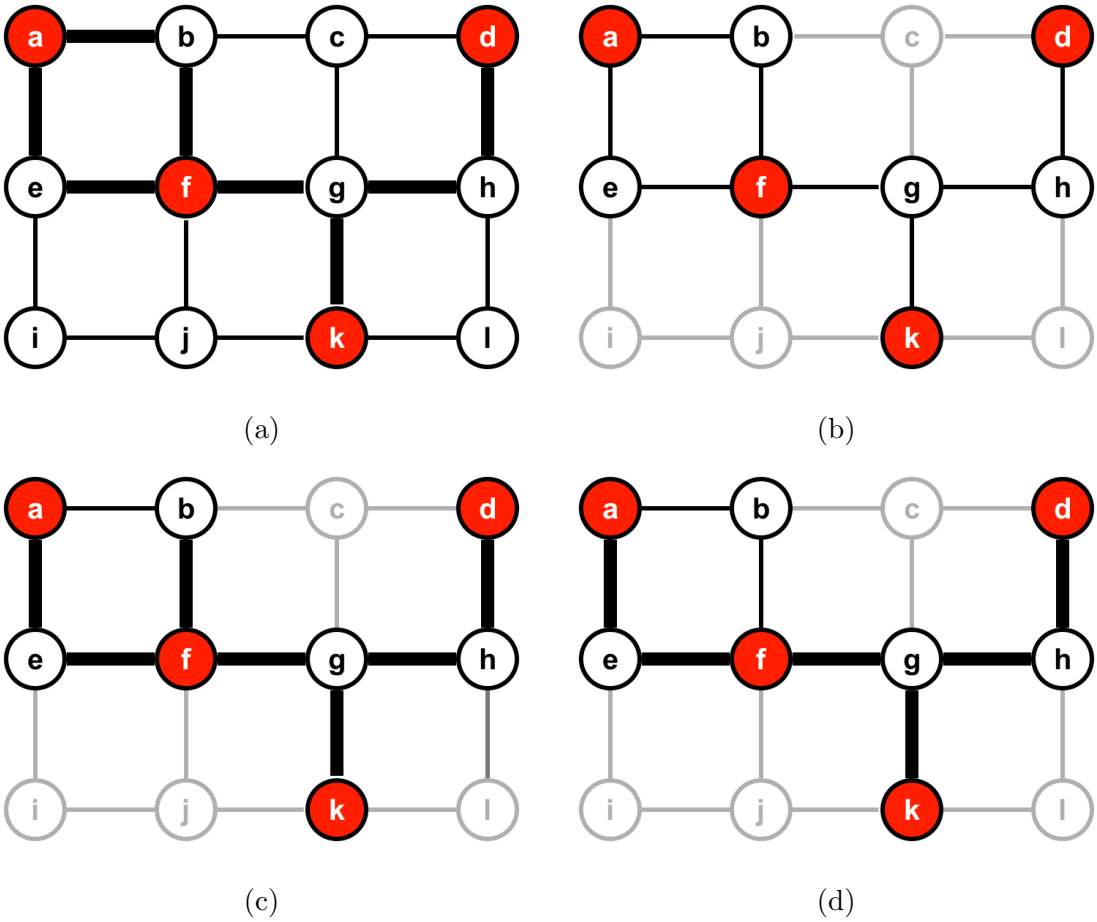


Figure 3.4: Schematic diagram of postprocessing. (a) An example of the obtained solution with a loop (vertex a–b–f–e). (b) An induced subgraph from (a). (c) A minimum spanning tree of (b). In this tree, vertex b is a non-terminal leaf. The vertex b does not need to span because the vertex b is non-terminal. Thus, we remove this leaf from the tree. (d) An obtained Steiner tree.

3.3 Network centralities

3.3.1 Degree centrality

The degree is the number of edges associated with a vertex. Thus, the degree centrality [8] evaluates vertices with large degree as important. The degree centrality of the ith vertex, $d_v(i)$, is defined as follows:

$$d_v(i) = \sum_{j=1}^{|V|} a_{ij}, \tag{3.2}$$

where a_{ij} is the ijth element of the adjacency matrix of the network; if an edge between the ith vertex and jth vertex exists, $a_{ij} = 1$; otherwise, $a_{ij} = 0$.

An example of the degree centrality is shown in Fig. 3.5. For instance, the degree centrality of the vertex 1 is calculated as follows:

$$\begin{aligned} d_v(1) &= a_{11} + a_{12} + a_{13} + a_{14} + a_{15} + a_{16}, && (3.3)\\ &= 0 + 1 + 0 + 1 + 0 + 0, && (3.4)\\ &= 2. && (3.5) \end{aligned}$$

In Fig. 3.5, blue colored numerals show the degree centrality of each vertex. In this network, the vertex 4 has the largest degree centrality, and the vertices 5 and 6 have the smallest degree centrality. The calculation cost of the degree centrality of all vertices is $\mathcal{O}(|V|^2)$ in the dense graph and $\mathcal{O}(|E|)$ in the sparse graph.

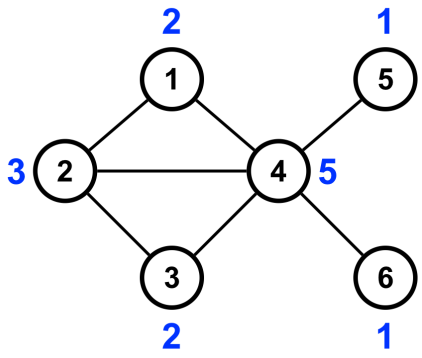


Figure 3.5: Example of the degree centrality

3.3.2 Eigenvector centrality

The degree centrality only considers the number of adjacent vertices to a vertex. However, do all vertices have the same importance? In a real-world network, the importances of vertices

sometimes differ. For example, from the viewpoint of making friends, the connection with a friend who has many friends is stronger than the connection with a friend who has a small number of friends.

To do this, it is necessary to consider the importance of adjacent vertices to a vertex. Here, we assume that the centrality of a vertex is in proportion to the sum of the centralities of adjacent vertices. This situation is expressed as follows:

$$e_v(i) = \frac{1}{\lambda} \sum_{j=1}^{|V|} a_{ij} e_v(j), \tag{3.6}$$

where $e_v(i)$ is the centrality of the ith vertex, $1/\lambda$ is a positive proportion constant, and a_{ij} is the ijth element of the adjacency matrix of the network.

An example of this centrality is shown in Fig. 3.6. For instance, the centrality of the vertex 1 is calculated as follows:

$$\begin{aligned} e_v(1) &= \frac{1}{\lambda}\{a_{11} \cdot e_v(1) + a_{12} \cdot e_v(2) + a_{13} \cdot e_v(3) + a_{14} \cdot e_v(4) + a_{15} \cdot e_v(5) + a_{16} \cdot e_v(6)\} && (3.7) \\ &= \frac{1}{\lambda}\{0 \cdot e_v(1) + 1 \cdot e_v(2) + 0 \cdot e_v(3) + 1 \cdot e_v(4) + 0 \cdot e_v(5) + 0 \cdot e_v(6)\} && (3.8) \\ &= \frac{1}{\lambda}\{e_v(2) + e_v(4)\}. && (3.9) \end{aligned}$$

The adjacency matrix is A and a column vector of the centrality of vertices is $\boldsymbol{c}$. Then, Eq. (3.6) can be expressed as follows:

$$\boldsymbol{c} = \frac{1}{\lambda} A\boldsymbol{c}. \tag{3.10}$$

Then, we can transform it to

$$\lambda \boldsymbol{c} = A\boldsymbol{c}. \tag{3.11}$$

Here, Eq. (3.11) means that λ is an eigenvalue of A and $\boldsymbol{c}$ is its eigenvector. Thus, we can obtain a network centrality that considers the importance of adjacent vertices to a vertex by using the eigenvector of the adjacency matrix. In particular, the first eigenvector that corresponds to the eigenvalue with the maximum value is called the eigenvector centrality [9].

The eigenvector centrality of a vertex reflects the centrality of its adjacent vertices; however, these adjacent vertices also reflect the centrality of its adjacent vertices. Therefore, if a network is connected, the eigenvector centrality reflects the structure of the network.

In an undirected network, we obtained the degree centrality as the sum of each row of the adjacency matrix. This indicates the direct connection or the relationship in one step. We can expand it to the relationship in two or three steps as the sum of each row of A^2 or A^3, respectively. In the case of the m steps, the sum of each row of A^m expresses the centrality of each vertex; however, these values become large when m becomes large.

Here, let the centrality in m steps is A^m/λ_1^m, where λ_1 is the largest eigenvalue of A, the centrality of each vertex, or the sum of each row of A^m/λ_1^m converges to the corresponding element of the first eigenvector of A.

The easiest way to obtain the eigenvector corresponding to the maximum eigenvalue is the power method. If A is a regular matrix,

$$A\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i, \tag{3.12}$$

where λ_i is an eigenvalue and $\boldsymbol{v}_i$ is its eigenvector. Let $\boldsymbol{x}_0 = c_1\boldsymbol{v}_1 + c_2\boldsymbol{v}_2 + \cdots + c_n\boldsymbol{v}_n$,

$$\begin{aligned} A\boldsymbol{x}_0 &= c_1 A\boldsymbol{v}_1 + c_2 A\boldsymbol{v}_2 + \cdots + c_n A\boldsymbol{v}_n, && (3.13)\\ &= c_1\lambda_1\boldsymbol{v}_1 + c_2\lambda_2\boldsymbol{v}_2 + \cdots + c_n\lambda_n\boldsymbol{v}_n. && (3.14) \end{aligned}$$

Thus,

$$\begin{aligned} A^2\boldsymbol{x}_0 &= A(A\boldsymbol{x}_0), && (3.15)\\ &= c_1\lambda_1 A\boldsymbol{v}_1 + c_2\lambda_2 A\boldsymbol{v}_2 + \cdots + c_n\lambda_n A\boldsymbol{v}_n, && (3.16)\\ &= c_1\lambda_1^2\boldsymbol{v}_1 + c_2\lambda_2^2\boldsymbol{v}_2 + \cdots + c_n\lambda_n^2\boldsymbol{v}_n. && (3.17) \end{aligned}$$

Similarly,

$$\begin{aligned} A^k\boldsymbol{x}_0 &= c_1\lambda_1^k\boldsymbol{v}_1 + c_2\lambda_2^k\boldsymbol{v}_2 + \cdots + c_n\lambda_n^k\boldsymbol{v}_n, && (3.18)\\ &= \lambda_1^k\left\{c_1\boldsymbol{v}_1 + c_2\left(\frac{\lambda_2}{\lambda_1}\right)^k\boldsymbol{v}_2 + \cdots + c_n\left(\frac{\lambda_n}{\lambda_1}\right)^k\boldsymbol{v}_n\right\}. && (3.19) \end{aligned}$$

Here, $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n| > 0$. Thus, $|\lambda_2/\lambda_1| < 1$, $|\lambda_3/\lambda_1| < 1$, $\cdots$, $|\lambda_n/\lambda_1| < 1$. Therefore, $|\lambda_s/\lambda_1|^k$ converges to 0, where $k \to \infty$. Consequently, $A^k\boldsymbol{x}_0 \simeq c_1\lambda^k\boldsymbol{v}_1$.

An eigenvector is still an eigenvector if it is multiplied by an arbitrary real value. Thus, the normalized value of them by dividing the maximum value of them are frequently used. In Fig. 3.6, blue colored numerals show the normalized eigenvector centrality of each vertex. In this network, the vertex 4 has the largest centrality, and the vertices 5 and 6 have the smallest centrality.

3.3.3 Closeness centrality

Closeness centrality [10] measures how close a vertex is to all the other vertices in the network. The closeness centrality of the ith vertex, $c_v(i)$, is defined as follows:

$$c_v(i) = \frac{1}{\sum_{j=1}^{|V|} d_{ij}}, \tag{3.20}$$

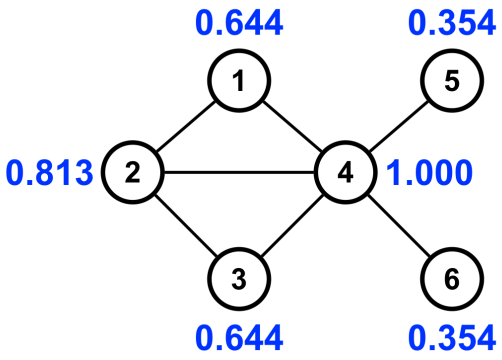


Figure 3.6: Example of eigenvector centrality

where d_{ij} is the ijth element of the distance matrix, i.e., the shortest distance between the ith vertex and the jth vertex. An example of the closeness centrality is shown in Fig. 3.7. For instance, the closeness centrality of the vertex 1 is calculated as follows:

$$c_v(1) = \frac{1}{d_{11} + d_{12} + d_{13} + d_{14} + d_{15} + d_{16}}, \tag{3.21}$$

$$= \frac{1}{0 + 1 + 2 + 1 + 2 + 2}, \tag{3.22}$$

$$= \frac{1}{8}, \tag{3.23}$$

$$= 0.125. \tag{3.24}$$

In Fig. 3.5, blue colored figures show the closeness centrality of each vertex. In this network, the vertex 4 has the largest closeness centrality, and the vertices 5 and 6 have the smallest closeness centrality. The calculation cost of the closeness centrality of all vertices is $\mathcal{O}(|V||E| + |V|^2 \log |V|)$ because the calculation cost of the single-source shortest path is $\mathcal{O}(|E| + |V| \log |V|)$, and this is repeated $|V|$ times.

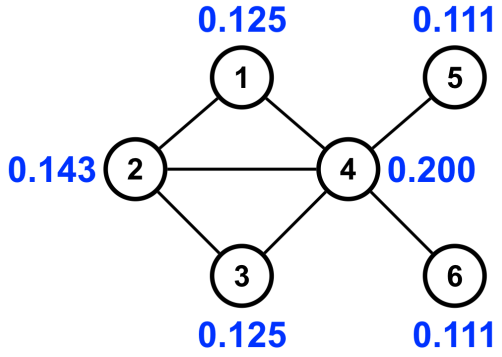


Figure 3.7: Example of closeness centrality

3.3.4 Betweenness centrality

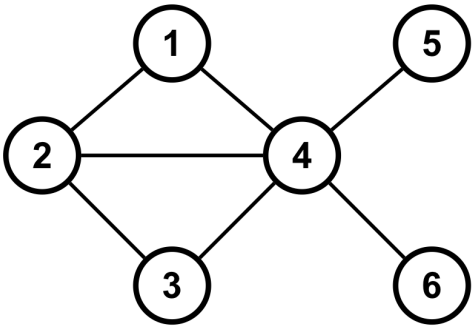
Betweenness centrality [8] detects central vertices or edges in the network using the shortest path information. If a vertex or edge frequently appears on the shortest paths of a network, the vertex or edge has a large betweenness centrality. The betweenness centrality of the ith vertex, $b_v(i)$, is defined as follows:

$$b_v(i) = \sum_{\substack{j=1,\\ j\neq i}}^{|V|} \sum_{\substack{k=j+1,\\ k\neq i}}^{|V|} \frac{P_{jk}(i)}{P_{jk}}, \tag{3.25}$$

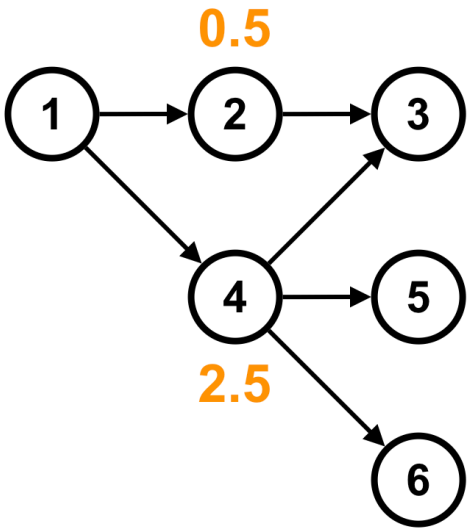
where j and k are start and end vertices of the shortest path, P_{jk} is the total number of shortest paths between j and k, and $P_{jk}(i)$ is the total number of shortest paths between j and k through the vertex i. An example of the calculation process of betweenness centrality is shown in Fig. 3.8. In Fig. 3.8, orange colored figures express the number of shortest paths between j and k through each vertex. For instance, in Fig. 3.8 (b), there is only one shortest path between vertex 1 and vertex 2, vertex 1 and vertex 4, vertex 1 and vertex 5, and vertex 1 and vertex 6. Next, there are two shortest paths between vertex 1 and vertex 3. Thus, vertex 2 appears on only one shortest path and vertex 4 appears on three shortest paths. Because there are two shortest paths between vertex 1 and vertex 3, $1 \to 2 \to 3$ and $1 \to 4 \to 3$, the effects of these paths are divided by two. Therefore, $\frac{P_{1k}(2)}{P_{jk}} = 0.5$ and $\frac{P_{1k}(4)}{P_{jk}} = 2.5$.

By repeating this procedure, the sum of each orange colored value becomes the betweenness centrality of each vertex. From Fig. 3.8, $b_v(1) = 0$, $b_v(2) = 0.5$, $b_v(3) = 0$, $b_v(4) = 7.5$, $b_v(5) = 0$, and $b_v(6) = 0$. An example of the betweenness centrality is shown in Fig. 3.9.

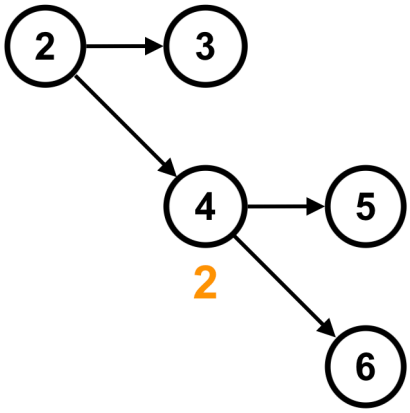
The betweenness centrality can be defined for not only vertices but also edges. Thus, an example of the betweenness centrality of edges is shown in Fig. 3.9. Hereafter, we call the betweenness centrality of vertices the *vertex betweenness centrality* and the betweenness centrality of edges the *edge betweenness centrality*. The calculation cost of both betweenness centralities of all vertices or edges is $\mathcal{O}(|V||E| + |V|^2 \log |V|)$ by [63].



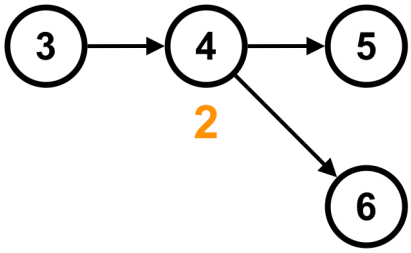
(a) Example network



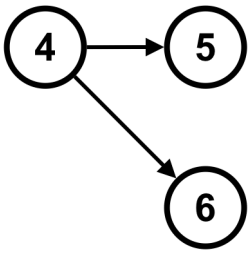
(b) $j = 1$ and $k = 2, 3, 4, 5, 6$



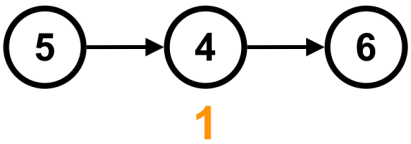
(c) $j = 2$ and $k = 3, 4, 5, 6$



(d) $j = 3$ and $k = 4, 5, 6$



(e) $j = 4$ and $k = 5, 6$



(f) $j = 5$ and $k = 6$

Figure 3.8: Example of the calculation process of betweenness centrality

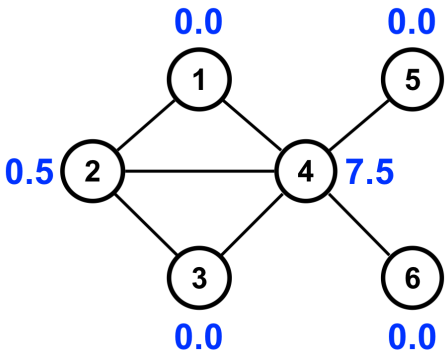


Figure 3.9: Example of vertex betweenness centrality

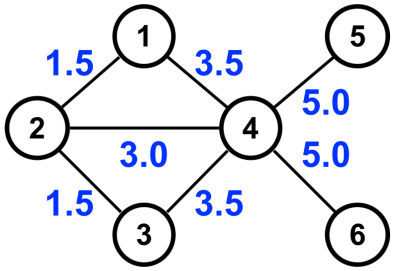


Figure 3.10: Example of edge betweenness centrality

3.4 Construction method combined with network centralities

The SPH [35] and ADH [37] use shortest paths, and the DNH [36] uses not only shortest paths but also a minimum spanning tree to construct the Steiner tree. Both the shortest path and minimum spanning tree use edge weights. From this viewpoint, we use the network centrality information instead of using the edge weights directly. To realize this, new edge weights based on the original edge weights and a network centrality are defined. The new edge weight function w' is defined as follows:

$$w'(i,j) = \alpha w(i,j) + (1-\alpha)\frac{1}{C_e(i,j)}, \tag{3.26}$$

where α is a scaling parameter, $w(i,j)$ is the original edge weight of the edge (i,j), and $C_e(i,j)$ is the one of the network centralities (the degree, eigenvector, closeness, or betweenness centralities) of the edge (i,j).

If vertices or edges have large network centralities, they are important in the network. However, the object of the Steiner tree problem in graphs is to find the minimum weight Steiner tree. Thus, small-weight edges have priority to construct the small-weight Steiner tree. To overcome this contradiction, we use a reciprocal of network centrality. If the network centrality is 0, its reciprocal cannot be calculated. In this case, we set it to 1 because it is the maximum value of the reciprocal. On the other hand, a suitable value of α strongly depends on the range of the original edge weight. To make the tuning of α easy, we use normalized $w(i,j)$ and $1/C_e(i,j)$ by dividing them by their maximum value.

If we use the network centralities of vertices, we transform the network centralities of vertices to those of edges by

$$C_e(i,j) = \frac{C_v(i) + C_v(j)}{2}, \tag{3.27}$$

where (i,j) is the edge between the vertices i and j, and $C_v(i)$ is one of the network centralities (the degree, eigenvector, closeness, or betweenness centralities) of the vertex i.

A flowchart of the proposed method is shown in Figure 3.11. First, calculate a network centrality. If we use network centrality of vertices, we transform it to the network centrality of edges by Eq. (3.27). Then, calculate w' by Eq. (3.26). Next, apply the construction method to an input graph with w'. In this thesis, we use SPH, DNH, and ADH as the construction method. The obtained subgraph sometimes does not satisfy the requirements of the Steiner tree. Thus, apply postprocessing to the obtained subgraph to find the Steiner tree. Finally, replace w' with w and recalculate the weight of the obtained Steiner tree.

The demerit of our proposed method is increased computation cost. The computation costs of the construction methods are $\mathcal{O}(|T||E| + |T||V|\log|V|)$ for SPH and DNH and

Figure 3.11: Flowchat of the construction method using network centrality

$\mathcal{O}(|T||V||E| + |T||V|^2 \log |V|)$ for ADH. On the other hand, the computation costs of the network centralities of all vertices are $\mathcal{O}(|V|^2)$ in the dense graph and $\mathcal{O}(|E|)$ in the sparse graph for the degree centrality, $\mathcal{O}(|V||E| + |V|^2 \log |V|)$ for the closeness centrality, and $\mathcal{O}(|V||E| + |V|^2 \log |V|)$ for the betweenness centrality [63]. Thus, using the betweenness centrality causes an increase in the order of the computation cost.

3.5 Performance evaluation

3.5.1 Experimental conditions

We evaluated the performance of the construction methods combined with network centralities by numerical experiments. We use benchmark problem sets B, C, D, I080, I160, and I320 in SteinLib [61]. Details of these benchmark problem sets are shown in Sec. 2.5. The scaling parameter α of Eq. (3.26) was set to its optimal value in $[0.0, 1.0)$ per instance by pre-numerical experiments. If $\alpha = 0.0$, $w'(i,j) = 1/C_e(i,j)$. On the other hand, if $\alpha = 1.0$, $w'(i,j) = w(i,j)$. Thus, if $0.0 \leq \alpha < 1.0$, $w'(i,j)$ differs from the original method. We construct 50 Steiner trees per instance by each method because these methods have randomness. We use the Dijkstra algorithm [31] to calculate the shortest path and the Kruskal algorithm [32] to calculate the minimum spanning tree. To remove biases of implementation, we shuffled the order of indices of edges randomly before executing the Kruskal algorithm.

3.5.2 Parameter tuning

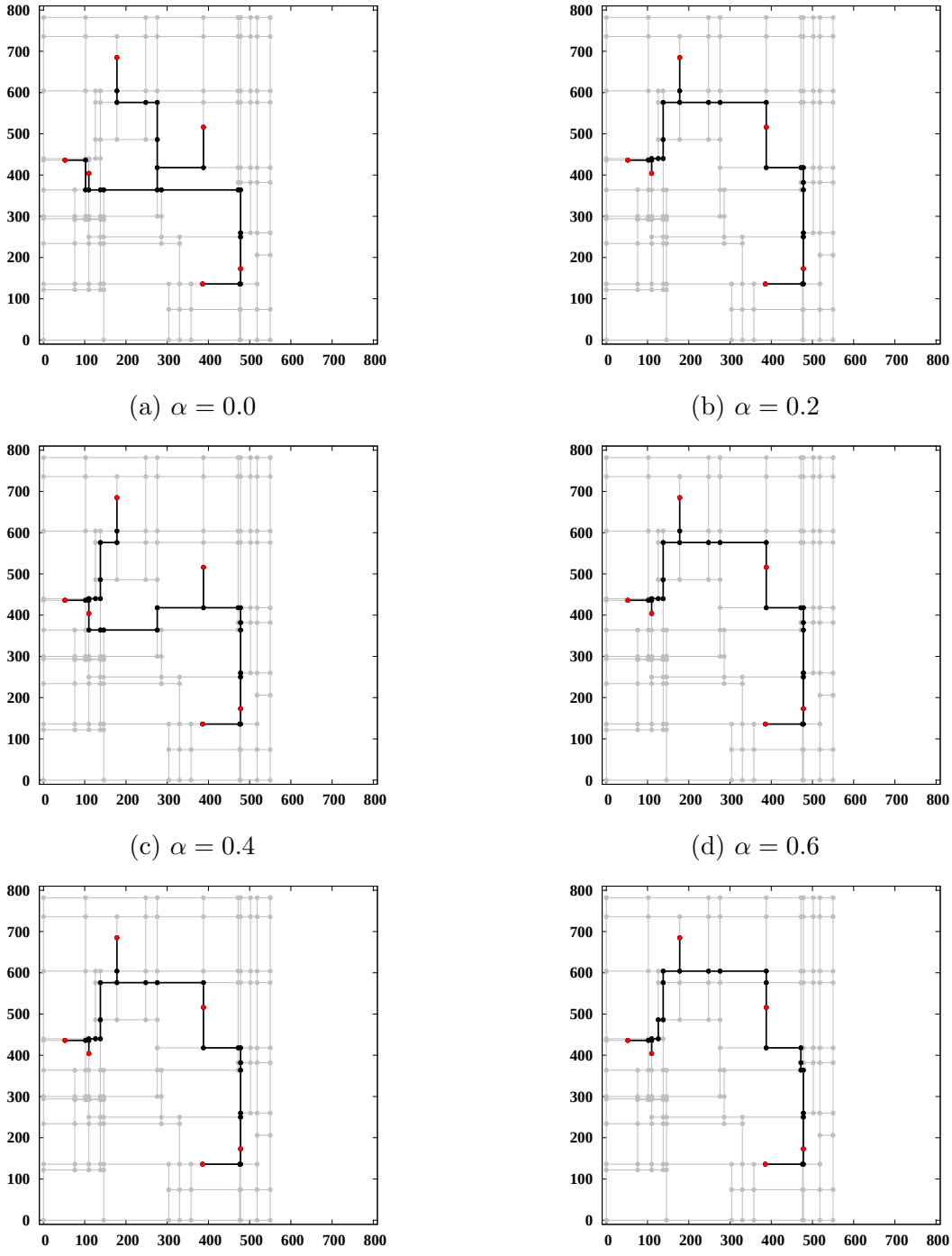
The construction method combined with network centrality has a scaling parameter α. It must be set to an optimal value. If α is too small, the effect of network centrality becomes large, and our proposed method finds Steiner trees by ignoring the given edge weights. It shows poor performance because the objective of the Steiner tree problem in graphs is finding a Steiner tree with minimum weight. On the other hand, if α is too large, the effect of the given edge weights becomes large and our proposed method becomes the conventional methods.

The effect of the value of α for the obtained Steiner tree is shown in Figs. 3.12, 3.13, and 3.14. In Figs. 3.12, 3.13, and 3.14, circles indicate vertices and lines indicate edges. Red circles are terminals, black circles are non-terminals included in a Steiner tree, and gray circles are non-terminals not included in a Steiner tree. Similarly, black lines are edges included in a Steiner tree, and gray lines are edges not included in a Steiner tree. From Figs. 3.12, 3.13, and 3.14, we can obtain small-weight Steiner trees when $0.0 < \alpha < 1.0$ for the instance lin04 in SteinLib [61].

Unfortunately, the optimal value of α changes per construction method, network centrality, and instance. To find the optimal α, we investigate the average gaps by changing α to $[0.0, 1.0)$ in increments of 0.1. The gap p is defined by Eq. (3.28).

$$p = \frac{F(S) - F(S^*)}{F(S^*)} \times 100 \ [\%], \tag{3.28}$$

where $F(S)$ is the objective function value of the obtained Steiner tree S, and $F(S^*)$ is the objective function value of the optimum Steiner tree S^*. The average gaps is the average value of the gaps of each instance.

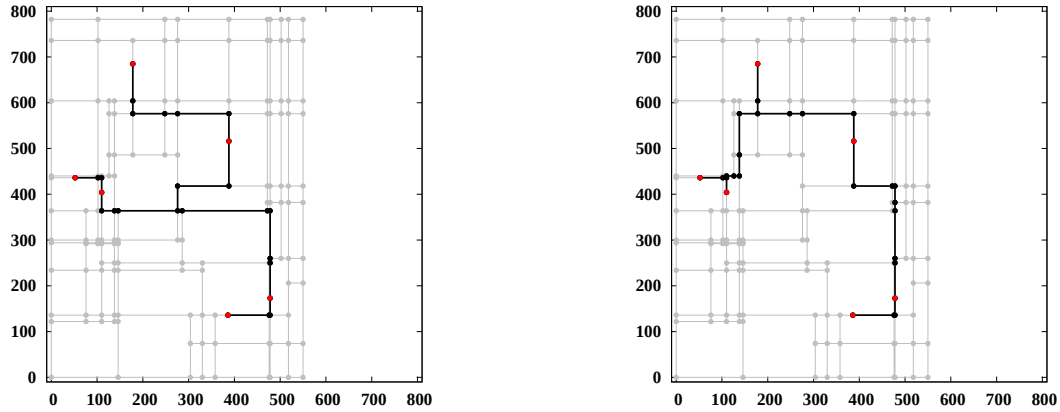


(a) $\alpha = 0.0$ (b) $\alpha = 0.2$

(c) $\alpha = 0.4$ (d) $\alpha = 0.6$

(e) $\alpha = 0.8$ (f) $\alpha = 1.0$ (original)

Figure 3.12: Example of Steiner trees made by SPH combined with edge betweenness centrality (lin04)

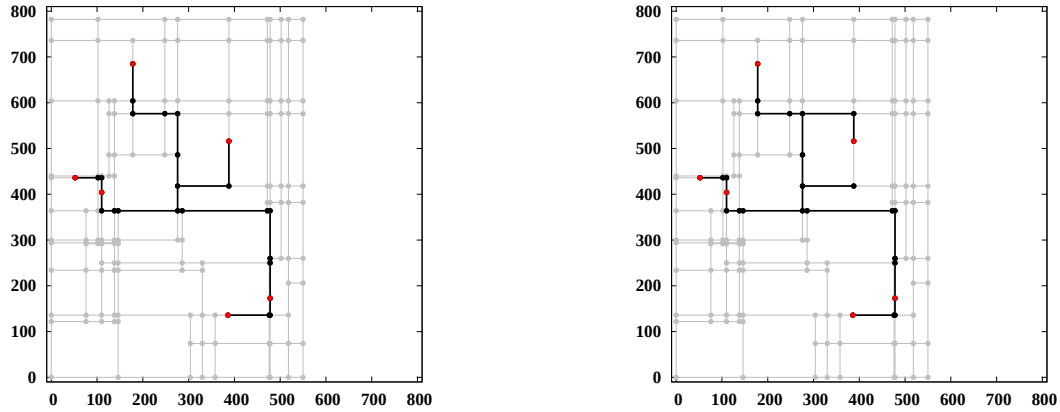


(a) $\alpha = 0.0$ (b) $\alpha = 0.2$

(c) $\alpha = 0.4$ (d) $\alpha = 0.6$

(e) $\alpha = 0.8$ (f) $\alpha = 1.0$ (original)

Figure 3.13: Example of Steiner trees made by DNH combined with edge betweenness centrality (lin04)



(a) $\alpha = 0.0$ (b) $\alpha = 0.2$

(c) $\alpha = 0.4$ (d) $\alpha = 0.6$

(e) $\alpha = 0.8$ (f) $\alpha = 1.0$ (original)

Figure 3.14: Example of Steiner trees made by ADH combined with edge betweenness centrality (lin04)

Table 3.1: Selected α for the benchmark problem set B

	SPH					DNH					ADH				
name	d_v	e_v	c_v	b_v	b_e	d_v	e_v	c_v	b_v	b_e	d_v	e_v	c_v	b_v	b_e
b01	0.9	0.8	0.9	0.2	0.0	0.5	0.3	0.4	0.1	0.0	0.7	0.5	0.8	0.1	0.0
b02	0.9	0.9	0.5	0.0	0.1	0.9	0.7	0.1	0.0	0.1	0.9	0.4	0.7	0.1	0.1
b03	0.8	0.1	0.7	0.3	0.1	0.8	0.1	0.7	0.3	0.1	0.8	0.1	0.7	0.3	0.1
b04	0.9	0.9	0.9	0.3	0.1	0.8	0.9	0.9	0.3	0.1	0.8	0.8	0.8	0.7	0.0
b05	0.8	0.8	0.6	0.6	0.1	0.9	0.8	0.8	0.4	0.1	0.8	0.8	0.2	0.4	0.1
b06	0.7	0.7	0.7	0.7	0.1	0.8	0.7	0.6	0.3	0.3	0.7	0.7	0.6	0.3	0.3
b07	0.9	0.6	0.8	0.1	0.1	0.8	0.4	0.4	0.1	0.1	0.8	0.5	0.6	0.0	0.0
b08	0.8	0.2	0.8	0.1	0.1	0.6	0.1	0.6	0.1	0.1	0.1	0.1	0.6	0.1	0.1
b09	0.9	0.8	0.9	0.5	0.1	0.7	0.4	0.6	0.5	0.1	0.6	0.5	0.6	0.5	0.3
b10	0.9	0.9	0.9	0.3	0.6	0.4	0.4	0.7	0.1	0.0	0.8	0.9	0.8	0.6	0.0
b11	0.9	0.9	0.9	0.4	0.0	0.6	0.7	0.5	0.2	0.0	0.5	0.8	0.8	0.3	0.0
b12	0.8	0.8	0.9	0.6	0.1	0.8	0.8	0.9	0.6	0.1	0.7	0.7	0.7	0.5	0.1
b13	0.2	0.1	0.2	0.0	0.1	0.0	0.9	0.0	0.0	0.2	0.2	0.5	0.1	0.1	0.0
b14	0.5	0.8	0.7	0.7	0.2	0.8	0.4	0.6	0.5	0.5	0.8	0.5	0.1	0.9	0.6
b15	0.6	0.4	0.7	0.7	0.8	0.9	0.5	0.7	0.7	0.8	0.6	0.4	0.5	0.9	0.8
b16	0.8	0.7	0.7	0.6	0.1	0.5	0.7	0.5	0.4	0.1	0.6	0.7	0.5	0.2	0.1
b17	0.8	0.9	0.9	0.3	0.4	0.7	0.8	0.8	0.1	0.1	0.7	0.9	0.8	0.1	0.2
b18	0.8	0.4	0.9	0.6	0.1	0.8	0.3	0.9	0.4	0.1	0.8	0.4	0.9	0.5	0.1

From pre-numerical experiments, we selected α, as shown in Tables 3.1–3.6. In Tables 3.1–3.6, SPH [35] is the shortest path heuristic, DNH [36] is the distance network heuristic, ADH [37] is the average distance heuristic, d_v is the degree centrality [8], e_v is the eigenvector centrality [9], c_v is the closeness centrality [10], b_v is the vertex betweenness centrality, and b_e is the edge betweenness centrality [8]. If the average gap becomes the minimum value for many α, we select the smallest one.

From Tables 3.1–3.6, the degree centrality and closeness centrality tend to select large α, and the vertex betweenness centrality and edge betweenness centrality tend to select small α. On the other hand, α of the eigenvector centrality has a wide range.

Table 3.2: Selected α for the benchmark problem set C

	SPH					DNH					ADH				
name	d_v	e_v	c_v	b_v	b_e	d_v	e_v	c_v	b_v	b_e	d_v	e_v	c_v	b_v	b_e
c01	0.7	0.2	0.9	0.1	0.9	0.2	0.1	0.5	0.1	0.1	0.5	0.1	0.3	0.2	0.2
c02	0.9	0.2	0.9	0.1	0.1	0.9	0.1	0.7	0.1	0.1	0.6	0.1	0.4	0.1	0.1
c03	0.9	0.9	0.9	0.7	0.6	0.9	0.9	0.9	0.7	0.5	0.6	0.9	0.9	0.9	0.7
c04	0.8	0.7	0.9	0.3	0.6	0.8	0.3	0.8	0.3	0.7	0.9	0.4	0.9	0.6	0.7
c05	0.9	0.8	0.9	0.7	0.2	0.9	0.1	0.9	0.6	0.1	0.9	0.4	0.9	0.6	0.1
c06	0.6	0.2	0.8	0.5	0.6	0.3	0.2	0.4	0.1	0.1	0.3	0.1	0.4	0.1	0.1
c07	0.9	0.9	0.9	0.5	0.2	0.9	0.6	0.9	0.1	0.0	0.7	0.7	0.8	0.3	0.1
c08	0.9	0.9	0.9	0.9	0.1	0.9	0.9	0.9	0.2	0.1	0.9	0.9	0.9	0.4	0.1
c09	0.9	0.7	0.9	0.6	0.2	0.9	0.6	0.9	0.5	0.1	0.9	0.6	0.9	0.5	0.4
c10	0.9	0.9	0.9	0.8	0.3	0.9	0.7	0.9	0.7	0.1	0.9	0.6	0.9	0.7	0.1
c11	0.9	0.6	0.9	0.2	0.4	0.8	0.7	0.1	0.1	0.1	0.8	0.8	0.9	0.2	0.1
c12	0.9	0.9	0.9	0.7	0.3	0.9	0.8	0.9	0.1	0.0	0.9	0.9	0.8	0.1	0.0
c13	0.9	0.9	0.9	0.4	0.8	0.9	0.9	0.9	0.3	0.1	0.9	0.9	0.9	0.3	0.1
c14	0.9	0.9	0.9	0.5	0.9	0.9	0.9	0.9	0.2	0.1	0.9	0.9	0.9	0.4	0.1
c15	0.9	0.9	0.9	0.8	0.6	0.9	0.9	0.9	0.1	0.1	0.9	0.9	0.9	0.1	0.1
c16	0.9	0.9	0.9	0.5	0.8	0.8	0.8	0.9	0.3	0.0	0.7	0.7	0.8	0.6	0.1
c17	0.9	0.9	0.9	0.2	0.5	0.9	0.9	0.9	0.1	0.1	0.9	0.9	0.9	0.2	0.1
c18	0.9	0.9	0.9	0.7	0.5	0.9	0.9	0.9	0.4	0.0	0.9	0.9	0.9	0.2	0.0
c19	0.9	0.9	0.9	0.5	0.1	0.9	0.9	0.9	0.1	0.1	0.9	0.9	0.8	0.1	0.1
c20	0.9	0.9	0.9	0.2	0.5	0.9	0.9	0.9	0.2	0.1	0.6	0.8	0.5	0.3	0.1

Table 3.3: Selected α for the benchmark problem set D

	SPH					DNH					ADH				
name	d_v	e_v	c_v	b_v	b_e	d_v	e_v	c_v	b_v	b_e	d_v	e_v	c_v	b_v	b_e
d01	0.9	0.4	0.9	0.1	0.8	0.9	0.1	0.2	0.3	0.1	0.5	0.1	0.8	0.1	0.1
d02	0.9	0.5	0.9	0.1	0.1	0.7	0.1	0.7	0.0	0.0	0.8	0.1	0.8	0.5	0.2
d03	0.8	0.5	0.8	0.5	0.6	0.7	0.4	0.9	0.7	0.1	0.8	0.4	0.5	0.5	0.7
d04	0.9	0.3	0.9	0.8	0.4	0.9	0.3	0.9	0.6	0.1	0.9	0.8	0.9	0.7	0.4
d05	0.9	0.1	0.9	0.7	0.6	0.9	0.1	0.9	0.6	0.1	0.9	0.2	0.9	0.3	0.1
d06	0.9	0.2	0.9	0.2	0.8	0.9	0.2	0.8	0.3	0.0	0.7	0.4	0.8	0.3	0.0
d07	0.7	0.2	0.9	0.2	0.4	0.7	0.2	0.8	0.1	0.1	0.7	0.1	0.6	0.6	0.2
d08	0.9	0.9	0.9	0.4	0.7	0.8	0.9	0.9	0.6	0.1	0.9	0.6	0.9	0.2	0.1
d09	0.9	0.9	0.9	0.8	0.5	0.9	0.6	0.9	0.3	0.1	0.9	0.7	0.9	0.6	0.4
d10	0.9	0.9	0.9	0.8	0.8	0.9	0.3	0.9	0.8	0.2	0.9	0.3	0.9	0.5	0.3
d11	0.9	0.8	0.9	0.3	0.2	0.9	0.8	0.9	0.1	0.1	0.9	0.9	0.8	0.1	0.1
d12	0.9	0.9	0.9	0.3	0.1	0.8	0.8	0.9	0.1	0.1	0.7	0.7	0.9	0.1	0.1
d13	0.9	0.9	0.9	0.3	0.5	0.9	0.8	0.9	0.2	0.1	0.9	0.9	0.9	0.3	0.4
d14	0.9	0.9	0.9	0.5	0.2	0.9	0.9	0.9	0.1	0.1	0.9	0.9	0.9	0.1	0.9
d15	0.9	0.9	0.9	0.9	0.7	0.9	0.9	0.9	0.3	0.1	0.9	0.9	0.9	0.2	0.1
d16	0.9	0.9	0.9	0.5	0.7	0.8	0.8	0.9	0.3	0.1	0.8	0.9	0.8	0.2	0.1
d17	0.9	0.9	0.9	0.7	0.3	0.6	0.6	0.7	0.2	0.1	0.9	0.9	0.9	0.3	0.1
d18	0.9	0.9	0.9	0.8	0.2	0.9	0.9	0.9	0.2	0.1	0.9	0.9	0.9	0.2	0.1
d19	0.9	0.9	0.9	0.3	0.8	0.9	0.9	0.9	0.3	0.1	0.9	0.9	0.9	0.4	0.1
d20	0.9	0.9	0.9	0.9	0.5	0.9	0.9	0.9	0.5	0.1	0.8	0.9	0.6	0.6	0.1

Table 3.4: Selected α for the benchmark problem set I080

	SPH					DNH					ADH				
name	d_v	e_v	c_v	b_v	b_e	d_v	e_v	c_v	b_v	b_e	d_v	e_v	c_v	b_v	b_e
i080-001	0.6	0.2	0.6	0.6	0.6	0.8	0.7	0.8	0.0	0.6	0.7	0.8	0.6	0.9	0.7
i080-002	0.0	0.9	0.9	0.0	0.3	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.0	0.0	0.4
i080-003	0.2	0.0	0.8	0.0	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.2	0.0	0.0
i080-004	0.5	0.9	0.7	0.6	0.7	0.0	0.4	0.9	0.1	0.0	0.5	0.8	0.0	0.5	0.3
i080-005	0.8	0.5	0.7	0.5	0.7	0.8	0.1	0.5	0.0	0.0	0.7	0.7	0.5	0.3	0.8
i080-011	0.9	0.9	0.1	0.2	0.7	0.8	0.9	0.1	0.9	0.8	0.8	0.9	0.1	0.0	0.8
i080-012	0.8	0.7	0.5	0.2	0.8	0.6	0.9	0.5	0.6	0.8	0.0	0.8	0.9	0.6	0.1
i080-013	0.7	0.8	0.1	0.0	0.7	0.6	0.8	0.1	0.0	0.1	0.0	0.7	0.1	0.1	0.3
i080-014	0.9	0.9	0.9	0.5	0.2	0.9	0.8	0.8	0.8	0.1	0.9	0.9	0.9	0.8	0.7
i080-015	0.0	0.0	0.0	0.5	0.6	0.0	0.0	0.0	0.8	0.8	0.0	0.9	0.9	0.2	0.2
i080-021	0.1	0.2	0.8	0.1	0.1	0.1	0.2	0.8	0.1	0.1	0.6	0.7	0.4	0.6	0.6
i080-022	0.1	0.2	0.6	0.1	0.1	0.1	0.2	0.6	0.1	0.1	0.6	0.7	0.3	0.6	0.6
i080-023	0.1	0.3	0.4	0.1	0.1	0.1	0.3	0.4	0.1	0.1	0.6	0.9	0.3	0.6	0.6
i080-024	0.1	0.3	0.5	0.1	0.1	0.1	0.3	0.5	0.1	0.1	0.6	0.7	0.6	0.6	0.6
i080-025	0.1	0.2	0.6	0.1	0.1	0.1	0.2	0.6	0.1	0.1	0.5	0.6	0.3	0.5	0.5
i080-031	0.8	0.7	0.8	0.8	0.9	0.8	0.1	0.9	0.8	0.1	0.4	0.1	0.3	0.5	0.8
i080-032	0.5	0.0	0.5	0.8	0.2	0.0	0.0	0.9	0.9	0.1	0.6	0.9	0.9	0.2	0.2
i080-033	0.9	0.9	0.7	0.7	0.9	0.1	0.1	0.8	0.7	0.0	0.6	0.3	0.2	0.7	0.1
i080-034	0.6	0.9	0.9	0.1	0.2	0.1	0.1	0.0	0.1	0.1	0.1	0.2	0.0	0.1	0.3
i080-035	0.3	0.9	0.3	0.7	0.5	0.0	0.0	0.0	0.2	0.1	0.6	0.5	0.0	0.7	0.9
i080-041	0.1	0.2	0.0	0.6	0.7	0.1	0.2	0.0	0.8	0.8	0.5	0.9	0.9	0.2	0.5
i080-042	0.9	0.5	0.9	0.7	0.7	0.8	0.7	0.5	0.7	0.0	0.8	0.6	0.9	0.8	0.5
i080-043	0.9	0.5	0.9	0.9	0.3	0.0	0.2	0.0	0.5	0.8	0.9	0.4	0.7	0.7	0.7
i080-044	0.6	0.7	0.1	0.0	0.5	0.9	0.8	0.7	0.8	0.9	0.9	0.9	0.9	0.7	0.6
i080-045	0.8	0.7	0.0	0.4	0.8	0.1	0.1	0.1	0.8	0.8	0.4	0.6	0.1	0.4	0.9
i080-101	0.2	0.3	0.7	0.6	0.7	0.5	0.1	0.8	0.6	0.4	0.9	0.3	0.6	0.1	0.6
i080-102	0.6	0.9	0.9	0.9	0.8	0.0	0.1	0.9	0.1	0.4	0.5	0.1	0.9	0.8	0.7
i080-103	0.6	0.6	0.3	0.7	0.0	0.6	0.4	0.8	0.2	0.0	0.6	0.9	0.9	0.1	0.2
i080-104	0.7	0.1	0.8	0.8	0.2	0.7	0.1	0.2	0.1	0.1	0.3	0.1	0.2	0.9	0.4
i080-105	0.0	0.0	0.0	0.0	0.4	0.0	0.0	0.0	0.0	0.4	0.0	0.0	0.5	0.2	0.1
i080-111	0.5	0.6	0.9	0.2	0.3	0.7	0.2	0.4	0.2	0.1	0.9	0.8	0.9	0.1	0.9
i080-112	0.7	0.7	0.5	0.7	0.8	0.8	0.5	0.7	0.5	0.9	0.0	0.2	0.2	0.8	0.5
i080-113	0.0	0.9	0.0	0.8	0.9	0.0	0.1	0.0	0.8	0.1	0.9	0.5	0.7	0.9	0.2
i080-114	0.0	0.5	0.0	0.2	0.3	0.2	0.3	0.0	0.0	0.2	0.8	0.9	0.9	0.1	0.6
i080-115	0.4	0.4	0.7	0.2	0.8	0.4	0.0	0.5	0.4	0.3	0.9	0.9	0.7	0.6	0.4
i080-121	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.4	0.6	0.6	0.4	0.4
i080-122	0.1	0.3	0.8	0.1	0.1	0.1	0.3	0.8	0.1	0.1	0.5	0.6	0.3	0.5	0.5
i080-123	0.1	0.4	0.1	0.1	0.1	0.1	0.4	0.1	0.1	0.1	0.4	0.6	0.0	0.4	0.4
i080-124	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.5	0.6	0.1	0.5	0.5
i080-125	0.1	0.2	0.8	0.1	0.1	0.1	0.2	0.8	0.1	0.1	0.4	0.6	0.4	0.4	0.4
i080-131	0.2	0.1	0.5	0.2	0.0	0.8	0.1	0.4	0.5	0.7	0.4	0.1	0.4	0.6	0.5
i080-132	0.5	0.5	0.5	0.6	0.3	0.0	0.8	0.5	0.2	0.1	0.8	0.8	0.1	0.9	0.2
i080-133	0.1	0.7	0.8	0.8	0.2	0.1	0.3	0.3	0.4	0.5	0.1	0.7	0.0	0.6	0.5
i080-134	0.5	0.6	0.9	0.4	0.5	0.0	0.0	0.6	0.0	0.6	0.4	0.7	0.1	0.1	0.3
i080-135	0.7	0.3	0.5	0.9	0.9	0.2	0.1	0.0	0.8	0.7	0.7	0.1	0.8	0.4	0.9
i080-141	0.8	0.6	0.4	0.1	0.5	0.8	0.6	0.4	0.2	0.8	0.8	0.6	0.4	0.5	0.4
i080-142	0.7	0.3	0.1	0.9	0.8	0.2	0.2	0.4	0.7	0.1	0.6	0.5	0.3	0.9	0.5
i080-143	0.1	0.2	0.3	0.6	0.7	0.6	0.2	0.0	0.3	0.8	0.6	0.2	0.4	0.7	0.2
i080-144	0.7	0.6	0.5	0.5	0.7	0.5	0.2	0.4	0.0	0.3	0.9	0.9	0.9	0.7	0.9
i080-145	0.1	0.6	0.0	0.0	0.2	0.0	0.3	0.0	0.0	0.5	0.7	0.9	0.9	0.3	0.9

	SPH					DNH					ADH				
name	d_v	e_v	c_v	b_v	b_e	d_v	e_v	c_v	b_v	b_e	d_v	e_v	c_v	b_v	b_e
i080-145	0.1	0.6	0.0	0.0	0.2	0.0	0.3	0.0	0.0	0.5	0.7	0.9	0.9	0.3	0.9
i080-201	0.6	0.8	0.5	0.6	0.8	0.5	0.3	0.1	0.3	0.2	0.6	0.5	0.6	0.1	0.4
i080-202	0.2	0.1	0.7	0.5	0.6	0.2	0.1	0.2	0.9	0.7	0.2	0.9	0.7	0.8	0.9
i080-203	0.3	0.9	0.1	0.1	0.1	0.0	0.1	0.0	0.0	0.0	0.8	0.2	0.0	0.1	0.1
i080-204	0.8	0.5	0.1	0.1	0.6	0.9	0.8	0.6	0.2	0.5	0.8	0.4	0.0	0.1	0.2
i080-205	0.9	0.1	0.8	0.8	0.3	0.9	0.2	0.8	0.8	0.7	0.9	0.8	0.2	0.9	0.1
i080-211	0.9	0.8	0.8	0.1	0.3	0.8	0.6	0.9	0.8	0.5	0.8	0.9	0.5	0.3	0.9
i080-212	0.9	0.9	0.9	0.5	0.4	0.8	0.9	0.9	0.0	0.0	0.9	0.7	0.8	0.4	0.8
i080-213	0.2	0.2	0.9	0.3	0.5	0.7	0.8	0.9	0.2	0.6	0.8	0.8	0.7	0.2	0.2
i080-214	0.9	0.9	0.3	0.1	0.5	0.9	0.7	0.3	0.1	0.0	0.7	0.7	0.6	0.1	0.2
i080-215	0.2	0.2	0.5	0.4	0.3	0.2	0.2	0.5	0.3	0.8	0.9	0.2	0.8	0.5	0.8
i080-221	0.1	0.4	0.1	0.1	0.1	0.1	0.4	0.1	0.1	0.1	0.3	0.6	0.7	0.3	0.3
i080-222	0.1	0.4	0.1	0.1	0.1	0.1	0.4	0.1	0.1	0.1	0.3	0.7	0.1	0.3	0.3
i080-223	0.1	0.3	0.1	0.1	0.1	0.1	0.3	0.1	0.1	0.1	0.3	0.6	0.7	0.3	0.3
i080-224	0.1	0.5	0.1	0.1	0.1	0.1	0.5	0.1	0.1	0.1	0.3	0.5	0.5	0.3	0.3
i080-225	0.1	0.5	0.1	0.1	0.1	0.1	0.5	0.1	0.1	0.1	0.3	0.5	0.1	0.3	0.3
i080-231	0.8	0.1	0.4	0.7	0.2	0.1	0.1	0.5	0.5	0.0	0.8	0.6	0.9	0.7	0.6
i080-232	0.8	0.8	0.5	0.5	0.9	0.8	0.7	0.8	0.5	0.9	0.5	0.8	0.0	0.9	0.9
i080-233	0.6	0.9	0.2	0.6	0.2	0.5	0.9	0.1	0.0	0.3	0.9	0.3	0.7	0.5	0.4
i080-234	0.8	0.9	0.9	0.9	0.8	0.8	0.9	0.2	0.4	0.0	0.6	0.1	0.7	0.8	0.5
i080-235	0.8	0.8	0.9	0.9	0.9	0.4	0.2	0.8	0.7	0.9	0.4	0.1	0.2	0.3	0.3
i080-241	0.9	0.9	0.9	0.6	0.7	0.9	0.9	0.9	0.6	0.2	0.7	0.9	0.5	0.7	0.7
i080-242	0.5	0.2	0.8	0.6	0.5	0.5	0.2	0.8	0.2	0.1	0.8	0.8	0.5	0.4	0.9
i080-243	0.2	0.2	0.9	0.3	0.6	0.2	0.2	0.9	0.3	0.7	0.9	0.8	0.5	0.7	0.4
i080-244	0.8	0.6	0.4	0.6	0.6	0.7	0.5	0.4	0.3	0.5	0.9	0.5	0.5	0.7	0.4
i080-245	0.8	0.5	0.8	0.7	0.7	0.8	0.5	0.8	0.2	0.4	0.9	0.6	0.6	0.5	0.6
i080-301	0.9	0.3	0.7	0.7	0.8	0.9	0.9	0.9	0.0	0.4	0.8	0.8	0.5	0.8	0.5
i080-302	0.8	0.1	0.9	0.3	0.7	0.8	0.0	0.7	0.1	0.0	0.6	0.9	0.6	0.8	0.9
i080-303	0.6	0.1	0.7	0.0	0.5	0.3	0.1	0.2	0.1	0.3	0.8	0.2	0.7	0.2	0.2
i080-304	0.8	0.7	0.9	0.9	0.5	0.9	0.1	0.8	0.6	0.7	0.9	0.9	0.6	0.8	0.5
i080-305	0.9	0.7	0.2	0.1	0.1	0.7	0.5	0.9	0.0	0.0	0.4	0.3	0.7	0.4	0.2
i080-311	0.9	0.9	0.9	0.1	0.5	0.9	0.9	0.9	0.6	0.1	0.6	0.7	0.6	0.6	0.9
i080-312	0.9	0.4	0.9	0.4	0.5	0.2	0.4	0.9	0.4	0.2	0.9	0.9	0.8	0.9	0.7
i080-313	0.9	0.9	0.3	0.7	0.6	0.5	0.7	0.5	0.6	0.7	0.9	0.9	0.5	0.1	0.5
i080-314	0.6	0.9	0.7	0.6	0.7	0.9	0.9	0.0	0.1	0.0	0.8	0.3	0.8	0.1	0.9
i080-315	0.1	0.7	0.5	0.1	0.4	0.7	0.7	0.7	0.4	0.1	0.9	0.9	0.5	0.9	0.9
i080-321	0.1	0.4	0.1	0.1	0.1	0.1	0.4	0.1	0.1	0.1	0.2	0.5	0.1	0.2	0.2
i080-322	0.1	0.3	0.5	0.1	0.1	0.1	0.3	0.5	0.1	0.1	0.2	0.5	0.9	0.2	0.2
i080-323	0.1	0.5	0.8	0.1	0.1	0.1	0.5	0.8	0.1	0.1	0.2	0.5	0.7	0.2	0.2
i080-324	0.1	0.3	0.1	0.1	0.1	0.1	0.3	0.1	0.1	0.1	0.2	0.5	0.0	0.2	0.2
i080-325	0.1	0.3	0.8	0.1	0.1	0.1	0.3	0.8	0.1	0.1	0.2	0.5	0.5	0.2	0.2
i080-331	0.8	0.9	0.9	0.5	0.6	0.3	0.3	0.0	0.3	0.4	0.7	0.9	0.5	0.9	0.7
i080-332	0.2	0.7	0.6	0.6	0.5	0.1	0.1	0.3	0.7	0.2	0.2	0.5	0.7	0.8	0.3
i080-333	0.5	0.9	0.3	0.5	0.2	0.5	0.8	0.3	0.4	0.1	0.4	0.8	0.5	0.4	0.6
i080-334	0.8	0.9	0.9	0.4	0.9	0.9	0.9	0.0	0.6	0.0	0.9	0.9	0.7	0.8	0.9
i080-335	0.9	0.7	0.6	0.3	0.1	0.7	0.7	0.8	0.1	0.3	0.6	0.7	0.7	0.1	0.2
i080-341	0.9	0.9	0.9	0.6	0.3	0.9	0.9	0.9	0.0	0.1	0.9	0.7	0.9	0.9	0.5
i080-342	0.9	0.9	0.9	0.4	0.4	0.9	0.9	0.9	0.0	0.6	0.8	0.9	0.7	0.8	0.9
i080-343	0.9	0.9	0.9	0.1	0.6	0.9	0.9	0.9	0.0	0.7	0.9	0.7	0.6	0.7	0.5
i080-344	0.7	0.9	0.9	0.1	0.3	0.7	0.9	0.9	0.0	0.2	0.9	0.8	0.8	0.8	0.8
i080-345	0.9	0.9	0.9	0.7	0.7	0.9	0.9	0.9	0.7	0.5	0.9	0.8	0.7	0.2	0.8

Table 3.5: Selected α for the benchmark problem set I160

	SPH					DNH					ADH				
name	d_v	e_v	c_v	b_v	b_e	d_v	e_v	c_v	b_v	b_e	d_v	e_v	c_v	b_v	b_e
i160-001	0.4	0.9	0.6	0.5	0.3	0.0	0.0	0.4	0.3	0.0	0.6	0.1	0.5	0.1	0.7
i160-002	0.6	0.5	0.9	0.9	0.8	0.0	0.5	0.0	0.9	0.4	0.9	0.6	0.0	0.6	0.7
i160-003	0.9	0.8	0.9	0.1	0.7	0.9	0.1	0.9	0.0	0.0	0.9	0.3	0.8	0.8	0.5
i160-004	0.3	0.1	0.1	0.8	0.9	0.4	0.0	0.0	0.6	0.1	0.3	0.0	0.2	0.7	0.1
i160-005	0.9	0.7	0.3	0.5	0.2	0.6	0.8	0.4	0.7	0.0	0.9	0.7	0.6	0.3	0.9
i160-011	0.9	0.8	0.1	0.4	0.3	0.0	0.1	0.1	0.0	0.1	0.9	0.7	0.9	0.1	0.3
i160-012	0.4	0.9	0.5	0.9	0.3	0.7	0.0	0.4	0.2	0.3	0.7	0.7	0.8	0.2	0.3
i160-013	0.9	0.8	0.8	0.1	0.8	0.3	0.1	0.6	0.8	0.8	0.8	0.1	0.0	0.3	0.2
i160-014	0.4	0.8	0.1	0.9	0.9	0.8	0.5	0.1	0.8	0.2	0.0	0.5	0.1	0.6	0.9
i160-015	0.9	0.6	0.7	0.3	0.8	0.1	0.2	0.0	0.0	0.4	0.9	0.8	0.8	0.7	0.7
i160-021	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.5	0.6	0.1	0.5	0.5
i160-022	0.1	0.2	0.1	0.1	0.1	0.1	0.2	0.1	0.1	0.1	0.5	0.6	0.1	0.5	0.5
i160-023	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.5	0.6	0.1	0.5	0.5
i160-024	0.1	0.2	0.1	0.1	0.1	0.1	0.2	0.1	0.1	0.1	0.5	0.6	0.7	0.5	0.5
i160-025	0.1	0.4	0.1	0.1	0.1	0.1	0.4	0.1	0.1	0.1	0.6	0.6	0.1	0.6	0.6
i160-031	0.7	0.9	0.9	0.9	0.8	0.0	0.9	0.9	0.9	0.7	0.3	0.5	0.0	0.5	0.8
i160-032	0.2	0.7	0.0	0.1	0.1	0.8	0.8	0.4	0.6	0.1	0.9	0.3	0.1	0.5	0.4
i160-033	0.8	0.9	0.6	0.5	0.9	0.0	0.0	0.0	0.0	0.1	0.6	0.2	0.6	0.8	0.8
i160-034	0.6	0.5	0.5	0.6	0.8	0.1	0.9	0.4	0.6	0.4	0.6	0.5	0.0	0.6	0.9
i160-035	0.9	0.9	0.4	0.6	0.9	0.8	0.8	0.3	0.4	0.8	0.9	0.3	0.8	0.8	0.5
i160-041	0.9	0.4	0.8	0.4	0.9	0.1	0.3	0.7	0.8	0.8	0.8	0.8	0.6	0.2	0.8
i160-042	0.4	0.5	0.0	0.4	0.3	0.3	0.8	0.6	0.1	0.3	0.7	0.7	0.2	0.1	0.4
i160-043	0.9	0.7	0.1	0.6	0.7	0.9	0.7	0.1	0.4	0.8	0.6	0.7	0.1	0.5	0.3
i160-044	0.9	0.9	0.9	0.4	0.8	0.2	0.9	0.6	0.9	0.8	0.7	0.7	0.3	0.1	0.5
i160-045	0.9	0.7	0.6	0.9	0.7	0.9	0.7	0.6	0.8	0.8	0.9	0.9	0.5	0.5	0.9
i160-101	0.9	0.4	0.5	0.1	0.1	0.1	0.4	0.9	0.1	0.1	0.8	0.5	0.9	0.1	0.2
i160-102	0.9	0.4	0.9	0.1	0.0	0.5	0.1	0.0	0.0	0.0	0.9	0.1	0.4	0.6	0.3
i160-103	0.9	0.8	0.8	0.7	0.9	0.0	0.0	0.0	0.0	0.7	0.0	0.1	0.8	0.1	0.1
i160-104	0.7	0.3	0.4	0.4	0.7	0.0	0.2	0.0	0.3	0.7	0.4	0.0	0.2	0.4	0.1
i160-105	0.9	0.7	0.5	0.5	0.5	0.1	0.3	0.5	0.3	0.6	0.5	0.1	0.9	0.2	0.7
i160-111	0.3	0.9	0.7	0.6	0.8	0.2	0.3	0.6	0.0	0.8	0.4	0.8	0.9	0.2	0.3
i160-112	0.5	0.4	0.6	0.6	0.6	0.4	0.0	0.6	0.0	0.1	0.8	0.7	0.7	0.8	0.6
i160-113	0.9	0.8	0.9	0.1	0.5	0.0	0.3	0.0	0.3	0.0	0.6	0.2	0.9	0.5	0.7
i160-114	0.5	0.5	0.1	0.6	0.3	0.5	0.1	0.1	0.2	0.7	0.8	0.9	0.9	0.6	0.3
i160-115	0.4	0.6	0.5	0.1	0.3	0.8	0.5	0.3	0.2	0.9	0.9	0.7	0.8	0.3	0.3
i160-121	0.1	0.4	0.9	0.1	0.1	0.1	0.4	0.9	0.1	0.1	0.3	0.5	0.1	0.3	0.3
i160-122	0.1	0.3	0.9	0.1	0.1	0.1	0.3	0.9	0.1	0.1	0.3	0.5	0.1	0.3	0.3
i160-123	0.1	0.3	0.8	0.1	0.1	0.1	0.3	0.8	0.1	0.1	0.3	0.5	0.1	0.3	0.3
i160-124	0.1	0.2	0.9	0.1	0.1	0.1	0.2	0.9	0.1	0.1	0.3	0.5	0.1	0.3	0.3
i160-125	0.1	0.3	0.8	0.1	0.1	0.1	0.3	0.8	0.1	0.1	0.3	0.5	0.1	0.3	0.3
i160-131	0.8	0.8	0.9	0.6	0.9	0.9	0.7	0.9	0.9	0.9	0.6	0.7	0.9	0.8	0.6
i160-132	0.9	0.8	0.9	0.9	0.9	0.9	0.6	0.8	0.1	0.5	0.3	0.8	0.7	0.9	0.9
i160-133	0.0	0.9	0.1	0.2	0.3	0.1	0.2	0.1	0.7	0.0	0.5	0.8	0.0	0.3	0.1
i160-134	0.0	0.8	0.0	0.8	0.8	0.0	0.1	0.0	0.6	0.2	0.0	0.9	0.0	0.8	0.3
i160-135	0.9	0.8	0.0	0.3	0.3	0.1	0.2	0.0	0.0	0.0	0.3	0.3	0.9	0.3	0.3
i160-141	0.8	0.8	0.7	0.5	0.5	0.8	0.8	0.7	0.6	0.4	0.7	0.6	0.6	0.5	0.9
i160-142	0.9	0.9	0.1	0.4	0.7	0.9	0.9	0.1	0.8	0.8	0.9	0.8	0.5	0.3	0.3
i160-143	0.7	0.5	0.2	0.1	0.9	0.4	0.3	0.1	0.4	0.1	0.6	0.7	0.6	0.1	0.8
i160-144	0.9	0.9	0.0	0.6	0.9	0.8	0.3	0.5	0.4	0.8	0.9	0.6	0.3	0.6	0.7
i160-145	0.1	0.5	0.0	0.7	0.5	0.1	0.5	0.1	0.8	0.8	0.9	0.7	0.5	0.2	0.6

	SPH					DNH					ADH				
name	d_v	e_v	c_v	b_v	b_e	d_v	e_v	c_v	b_v	b_e	d_v	e_v	c_v	b_v	b_e
i160-145	0.1	0.5	0.0	0.7	0.5	0.1	0.5	0.1	0.8	0.8	0.9	0.7	0.5	0.2	0.6
i160-201	0.3	0.4	0.8	0.5	0.1	0.4	0.1	0.9	0.5	0.0	0.2	0.5	0.4	0.6	0.5
i160-202	0.2	0.9	0.1	0.1	0.9	0.4	0.1	0.0	0.3	0.1	0.9	0.9	0.9	0.9	0.8
i160-203	0.9	0.8	0.9	0.7	0.8	0.0	0.0	0.0	0.1	0.0	0.9	0.1	0.7	0.3	0.6
i160-204	0.5	0.8	0.7	0.0	0.2	0.0	0.1	0.9	0.1	0.9	0.5	0.9	0.9	0.9	0.9
i160-205	0.9	0.1	0.9	0.6	0.3	0.0	0.1	0.0	0.0	0.0	0.5	0.4	0.9	0.8	0.8
i160-211	0.7	0.9	0.8	0.4	0.2	0.7	0.4	0.8	0.2	0.0	0.9	0.8	0.8	0.4	0.5
i160-212	0.9	0.8	0.9	0.1	0.5	0.3	0.4	0.1	0.0	0.6	0.7	0.8	0.8	0.4	0.5
i160-213	0.6	0.7	0.9	0.2	0.5	0.8	0.7	0.9	0.1	0.8	0.8	0.8	0.7	0.4	0.2
i160-214	0.4	0.7	0.1	0.1	0.2	0.2	0.4	0.1	0.0	0.1	0.7	0.7	0.8	0.1	0.7
i160-215	0.9	0.9	0.9	0.2	0.2	0.9	0.7	0.9	0.0	0.1	0.8	0.7	0.9	0.2	0.5
i160-221	0.1	0.4	0.9	0.1	0.1	0.1	0.4	0.9	0.1	0.1	0.2	0.5	0.1	0.2	0.2
i160-222	0.1	0.3	0.8	0.1	0.1	0.1	0.3	0.8	0.1	0.1	0.2	0.5	0.1	0.2	0.2
i160-223	0.1	0.4	0.1	0.1	0.1	0.1	0.4	0.1	0.1	0.1	0.2	0.5	0.1	0.2	0.2
i160-224	0.1	0.2	0.1	0.1	0.1	0.1	0.2	0.1	0.1	0.1	0.2	0.5	0.1	0.2	0.2
i160-225	0.1	0.3	0.9	0.1	0.1	0.1	0.3	0.9	0.1	0.1	0.2	0.5	0.1	0.2	0.2
i160-231	0.4	0.8	0.8	0.1	0.3	0.7	0.5	0.8	0.7	0.8	0.2	0.1	0.9	0.4	0.8
i160-232	0.7	0.2	0.2	0.2	0.2	0.4	0.5	0.9	0.0	0.1	0.8	0.4	0.7	0.2	0.2
i160-233	0.4	0.4	0.7	0.5	0.4	0.1	0.7	0.6	0.0	0.0	0.6	0.6	0.6	0.8	0.6
i160-234	0.9	0.9	0.8	0.4	0.5	0.8	0.5	0.8	0.3	0.3	0.0	0.2	0.5	0.7	0.7
i160-235	0.7	0.7	0.2	0.1	0.1	0.4	0.9	0.7	0.0	0.0	0.9	0.6	0.9	0.3	0.1
i160-241	0.9	0.9	0.1	0.7	0.7	0.9	0.9	0.1	0.3	0.7	0.9	0.7	0.1	0.7	0.7
i160-242	0.9	0.9	0.9	0.3	0.7	0.9	0.9	0.9	0.0	0.7	0.9	0.8	0.5	0.9	0.7
i160-243	0.9	0.9	0.9	0.4	0.7	0.9	0.9	0.9	0.2	0.6	0.9	0.7	0.0	0.5	0.9
i160-244	0.9	0.9	0.9	0.2	0.7	0.9	0.9	0.9	0.2	0.3	0.8	0.9	0.8	0.4	0.9
i160-245	0.9	0.9	0.9	0.5	0.7	0.9	0.9	0.9	0.1	0.1	0.8	0.7	0.9	0.6	0.6
i160-301	0.8	0.6	0.5	0.2	0.7	0.5	0.1	0.7	0.0	0.1	0.9	0.8	0.9	0.3	0.8
i160-302	0.7	0.9	0.9	0.9	0.8	0.0	0.9	0.9	0.0	0.2	0.9	0.7	0.8	0.4	0.4
i160-303	0.3	0.5	0.9	0.1	0.2	0.1	0.2	0.8	0.1	0.0	0.9	0.6	0.8	0.1	0.2
i160-304	0.5	0.2	0.8	0.4	0.5	0.4	0.9	0.9	0.2	0.8	0.9	0.9	0.9	0.6	0.6
i160-305	0.9	0.3	0.5	0.4	0.5	0.9	0.6	0.8	0.1	0.1	0.9	0.8	0.1	0.9	0.9
i160-311	0.8	0.9	0.6	0.4	0.3	0.6	0.5	0.5	0.0	0.1	0.9	0.9	0.9	0.4	0.2
i160-312	0.2	0.2	0.9	0.4	0.4	0.2	0.2	0.9	0.1	0.2	0.6	0.7	0.7	0.6	0.9
i160-313	0.8	0.4	0.9	0.2	0.4	0.8	0.4	0.6	0.1	0.1	0.9	0.8	0.9	0.6	0.9
i160-314	0.6	0.8	0.9	0.2	0.6	0.7	0.5	0.0	0.0	0.0	0.7	0.8	0.7	0.1	0.6
i160-315	0.6	0.9	0.1	0.3	0.2	0.5	0.7	0.1	0.1	0.3	0.9	0.9	0.9	0.4	0.5
i160-321	0.1	0.4	0.1	0.1	0.1	0.1	0.4	0.1	0.1	0.1	0.1	0.5	0.1	0.1	0.1
i160-322	0.1	0.3	0.1	0.1	0.1	0.1	0.3	0.1	0.1	0.1	0.5	0.5	0.4	0.5	0.5
i160-323	0.1	0.4	0.1	0.1	0.1	0.1	0.4	0.1	0.1	0.1	0.1	0.4	0.1	0.1	0.1
i160-324	0.1	0.5	0.1	0.1	0.1	0.1	0.5	0.1	0.1	0.1	0.1	0.4	0.1	0.1	0.1
i160-325	0.1	0.4	0.1	0.1	0.1	0.1	0.4	0.1	0.1	0.1	0.1	0.4	0.1	0.1	0.1
i160-331	0.9	0.1	0.9	0.4	0.9	0.9	0.6	0.9	0.1	0.0	0.7	0.5	0.9	0.9	0.5
i160-332	0.3	0.9	0.3	0.1	0.4	0.3	0.1	0.1	0.0	0.0	0.8	0.5	0.9	0.5	0.2
i160-333	0.3	0.9	0.1	0.1	0.1	0.7	0.2	0.0	0.0	0.0	0.9	0.9	0.9	0.2	0.5
i160-334	0.9	0.9	0.8	0.5	0.9	0.3	0.5	0.8	0.0	0.4	0.8	0.6	0.7	0.5	0.2
i160-335	0.2	0.9	0.1	0.3	0.6	0.5	0.5	0.7	0.0	0.1	0.4	0.3	0.7	0.5	0.2
i160-341	0.9	0.9	0.9	0.6	0.7	0.9	0.9	0.9	0.1	0.7	0.7	0.7	0.5	0.4	0.9
i160-342	0.9	0.9	0.9	0.3	0.7	0.9	0.9	0.9	0.1	0.6	0.8	0.9	0.8	0.6	0.8
i160-343	0.9	0.9	0.9	0.3	0.6	0.9	0.9	0.9	0.2	0.4	0.9	0.7	0.7	0.9	0.8
i160-344	0.9	0.9	0.8	0.6	0.6	0.9	0.9	0.8	0.1	0.2	0.8	0.9	0.9	0.9	0.9
i160-345	0.9	0.9	0.8	0.5	0.7	0.9	0.9	0.8	0.1	0.4	0.7	0.8	0.5	0.8	0.5

Table 3.6: Selected α for the benchmark problem set I320

	SPH					DNH					ADH				
name	d_v	e_v	c_v	b_v	b_e	d_v	e_v	c_v	b_v	b_e	d_v	e_v	c_v	b_v	b_e
i320-001	0.5	0.3	0.9	0.9	0.7	0.0	0.1	0.9	0.1	0.1	0.8	0.7	0.9	0.2	0.7
i320-002	0.6	0.5	0.5	0.2	0.9	0.2	0.3	0.5	0.1	0.4	0.4	0.3	0.6	0.3	0.7
i320-003	0.8	0.2	0.8	0.5	0.4	0.0	0.0	0.6	0.0	0.2	0.9	0.1	0.9	0.4	0.8
i320-004	0.1	0.9	0.5	0.2	0.4	0.7	0.1	0.9	0.2	0.3	0.8	0.2	0.1	0.4	0.5
i320-005	0.7	0.2	0.9	0.5	0.9	0.1	0.1	0.0	0.1	0.3	0.9	0.1	0.1	0.7	0.9
i320-011	0.9	0.9	0.8	0.3	0.2	0.8	0.5	0.1	0.0	0.0	0.9	0.8	0.6	0.4	0.5
i320-012	0.8	0.6	0.2	0.4	0.7	0.0	0.2	0.0	0.2	0.1	0.8	0.9	0.9	0.9	0.7
i320-013	0.6	0.4	0.2	0.9	0.9	0.0	0.0	0.1	0.8	0.8	0.0	0.9	0.4	0.3	0.7
i320-014	0.7	0.9	0.9	0.8	0.8	0.4	0.6	0.7	0.8	0.1	0.9	0.6	0.9	0.4	0.2
i320-015	0.9	0.8	0.9	0.6	0.4	0.1	0.6	0.7	0.0	0.0	0.8	0.9	0.2	0.6	0.7
i320-021	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.5	0.6	0.1	0.5	0.5
i320-022	0.1	0.2	0.1	0.1	0.1	0.1	0.2	0.1	0.1	0.1	0.4	0.6	0.1	0.4	0.4
i320-023	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.4	0.6	0.1	0.4	0.4
i320-024	0.1	0.4	0.8	0.1	0.1	0.1	0.4	0.8	0.1	0.1	0.5	0.6	0.1	0.5	0.5
i320-025	0.1	0.2	0.1	0.1	0.1	0.1	0.2	0.1	0.1	0.1	0.4	0.6	0.1	0.4	0.4
i320-031	0.0	0.1	0.9	0.6	0.9	0.0	0.1	0.9	0.1	0.2	0.7	0.1	0.8	0.4	0.4
i320-032	0.6	0.3	0.7	0.9	0.6	0.9	0.9	0.1	0.2	0.0	0.9	0.8	0.9	0.9	0.6
i320-033	0.5	0.9	0.4	0.7	0.4	0.6	0.4	0.9	0.6	0.4	0.5	0.8	0.6	0.6	0.4
i320-034	0.5	0.3	0.2	0.3	0.2	0.0	0.1	0.6	0.0	0.0	0.8	0.7	0.9	0.6	0.8
i320-035	0.1	0.2	0.2	0.2	0.1	0.8	0.6	0.9	0.0	0.0	0.4	0.3	0.7	0.3	0.4
i320-041	0.4	0.5	0.2	0.1	0.8	0.8	0.6	0.1	0.9	0.9	0.7	0.7	0.5	0.4	0.5
i320-042	0.9	0.9	0.9	0.1	0.5	0.9	0.7	0.0	0.1	0.4	0.5	0.8	0.7	0.5	0.9
i320-043	0.8	0.5	0.9	0.4	0.7	0.8	0.5	0.4	0.8	0.8	0.9	0.8	0.5	0.3	0.5
i320-044	0.9	0.7	0.7	0.8	0.8	0.9	0.7	0.7	0.1	0.8	0.6	0.7	0.2	0.2	0.9
i320-045	0.9	0.4	0.9	0.3	0.8	0.8	0.7	0.4	0.2	0.8	0.9	0.6	0.8	0.6	0.9
i320-101	0.5	0.3	0.5	0.7	0.6	0.2	0.1	0.9	0.1	0.1	0.9	0.2	0.9	0.1	0.3
i320-102	0.7	0.1	0.9	0.1	0.6	0.1	0.1	0.1	0.1	0.5	0.5	0.2	0.3	0.3	0.3
i320-103	0.9	0.2	0.5	0.5	0.5	0.1	0.4	0.8	0.5	0.2	0.8	0.8	0.4	0.9	0.9
i320-104	0.8	0.1	0.1	0.7	0.5	0.7	0.2	0.9	0.0	0.0	0.9	0.3	0.9	0.5	0.6
i320-105	0.6	0.8	0.0	0.8	0.7	0.1	0.2	0.0	0.1	0.1	0.9	0.8	0.8	0.5	0.9
i320-111	0.8	0.6	0.9	0.9	0.7	0.8	0.8	0.1	0.0	0.5	0.7	0.9	0.3	0.1	0.1
i320-112	0.8	0.8	0.8	0.8	0.9	0.8	0.7	0.0	0.1	0.1	0.5	0.9	0.8	0.1	0.3
i320-113	0.8	0.5	0.5	0.1	0.2	0.7	0.1	0.1	0.4	0.8	0.7	0.9	0.8	0.8	0.3
i320-114	0.6	0.9	0.9	0.4	0.5	0.8	0.8	0.6	0.3	0.8	0.9	0.9	0.8	0.3	0.3
i320-115	0.8	0.9	0.7	0.9	0.8	0.9	0.3	0.9	0.8	0.8	0.8	0.9	0.9	0.2	0.9
i320-121	0.1	0.3	0.1	0.1	0.1	0.1	0.3	0.1	0.1	0.1	0.2	0.5	0.1	0.2	0.2
i320-122	0.1	0.2	0.1	0.1	0.1	0.1	0.2	0.1	0.1	0.1	0.2	0.5	0.1	0.2	0.2
i320-123	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.9	0.5	0.9	0.9	0.9
i320-124	0.1	0.3	0.1	0.1	0.1	0.1	0.3	0.1	0.1	0.1	0.2	0.5	0.1	0.2	0.2
i320-125	0.1	0.2	0.1	0.1	0.1	0.1	0.2	0.1	0.1	0.1	0.3	0.5	0.1	0.3	0.3
i320-131	0.9	0.8	0.0	0.8	0.4	0.5	0.7	0.5	0.6	0.4	0.6	0.8	0.6	0.6	0.6
i320-132	0.8	0.9	0.6	0.4	0.2	0.9	0.7	0.6	0.5	0.2	0.6	0.3	0.5	0.2	0.3
i320-133	0.3	0.1	0.4	0.7	0.8	0.0	0.4	0.0	0.9	0.3	0.9	0.2	0.8	0.9	0.8
i320-134	0.6	0.5	0.1	0.3	0.7	0.0	0.2	0.0	0.0	0.0	0.5	0.9	0.5	0.3	0.2
i320-135	0.9	0.8	0.9	0.8	0.6	0.6	0.6	0.7	0.5	0.4	0.5	0.6	0.9	0.9	0.3
i320-141	0.9	0.9	0.9	0.8	0.7	0.9	0.9	0.9	0.2	0.5	0.9	0.8	0.7	0.9	0.8
i320-142	0.9	0.8	0.9	0.5	0.7	0.9	0.8	0.9	0.1	0.8	0.6	0.6	0.7	0.6	0.8
i320-143	0.9	0.7	0.9	0.4	0.7	0.9	0.7	0.4	0.0	0.4	0.9	0.8	0.6	0.4	0.9
i320-144	0.9	0.9	0.9	0.0	0.7	0.9	0.9	0.9	0.1	0.5	0.8	0.9	0.4	0.7	0.8
i320-145	0.9	0.9	0.9	0.0	0.7	0.9	0.9	0.9	0.1	0.5	0.7	0.7	0.1	0.3	0.9

	SPH					DNH					ADH				
name	d_v	e_v	c_v	b_v	b_e	d_v	e_v	c_v	b_v	b_e	d_v	e_v	c_v	b_v	b_e
i320-145	0.9	0.9	0.9	0.0	0.7	0.9	0.9	0.9	0.1	0.5	0.7	0.7	0.1	0.3	0.9
i320-201	0.7	0.4	0.9	0.7	0.5	0.8	0.3	0.8	0.1	0.1	0.9	0.7	0.9	0.8	0.9
i320-202	0.8	0.8	0.1	0.4	0.3	0.5	0.1	0.0	0.0	0.1	0.9	0.9	0.2	0.6	0.7
i320-203	0.2	0.9	0.9	0.8	0.7	0.0	0.1	0.7	0.0	0.0	0.4	0.2	0.7	0.8	0.9
i320-204	0.4	0.1	0.7	0.7	0.2	0.0	0.1	0.1	0.1	0.1	0.7	0.1	0.6	0.5	0.8
i320-205	0.1	0.9	0.7	0.9	0.2	0.1	0.1	0.1	0.0	0.0	0.8	0.5	0.8	0.4	0.4
i320-211	0.8	0.9	0.6	0.9	0.9	0.8	0.4	0.4	0.2	0.2	0.8	0.6	0.8	0.3	0.8
i320-212	0.8	0.7	0.4	0.2	0.2	0.5	0.9	0.0	0.0	0.1	0.9	0.7	0.9	0.3	0.2
i320-213	0.8	0.9	0.9	0.3	0.1	0.8	0.4	0.1	0.0	0.2	0.9	0.8	0.7	0.1	0.8
i320-214	0.9	0.5	0.9	0.3	0.2	0.7	0.1	0.6	0.1	0.4	0.8	0.7	0.6	0.2	0.8
i320-215	0.6	0.7	0.6	0.1	0.4	0.1	0.5	0.0	0.0	0.0	0.8	0.8	0.9	0.4	0.7
i320-221	0.1	0.3	0.1	0.1	0.1	0.1	0.3	0.1	0.1	0.1	0.2	0.6	0.1	0.2	0.2
i320-222	0.1	0.2	0.1	0.1	0.1	0.1	0.2	0.1	0.1	0.1	0.2	0.5	0.1	0.2	0.2
i320-223	0.1	0.3	0.1	0.1	0.1	0.1	0.3	0.1	0.1	0.1	0.2	0.6	0.1	0.2	0.2
i320-224	0.1	0.4	0.1	0.1	0.1	0.1	0.4	0.1	0.1	0.1	0.7	0.5	0.6	0.7	0.7
i320-225	0.1	0.2	0.1	0.1	0.1	0.1	0.2	0.1	0.1	0.1	0.2	0.5	0.1	0.2	0.2
i320-231	0.3	0.4	0.6	0.7	0.9	0.0	0.3	0.0	0.0	0.7	0.4	0.7	0.2	0.7	0.4
i320-232	0.1	0.9	0.2	0.5	0.7	0.2	0.8	0.7	0.2	0.3	0.7	0.8	0.9	0.9	0.8
i320-233	0.9	0.6	0.9	0.9	0.8	0.1	0.1	0.1	0.0	0.1	0.9	0.8	0.3	0.8	0.6
i320-234	0.4	0.5	0.9	0.5	0.5	0.1	0.7	0.0	0.3	0.0	0.2	0.8	0.1	0.5	0.7
i320-235	0.4	0.2	0.2	0.4	0.4	0.9	0.3	0.9	0.2	0.3	0.5	0.6	0.8	0.7	0.8
i320-241	0.9	0.9	0.1	0.4	0.7	0.9	0.9	0.1	0.1	0.7	0.6	0.6	0.9	0.2	0.9
i320-242	0.9	0.9	0.1	0.2	0.7	0.9	0.9	0.1	0.0	0.7	0.8	0.9	0.9	0.7	0.8
i320-243	0.9	0.9	0.1	0.2	0.7	0.9	0.9	0.1	0.0	0.3	0.6	0.9	0.9	0.7	0.8
i320-244	0.9	0.9	0.1	0.3	0.7	0.9	0.9	0.1	0.0	0.7	0.9	0.8	0.7	0.9	0.9
i320-245	0.9	0.9	0.1	0.2	0.7	0.9	0.9	0.1	0.0	0.5	0.7	0.6	0.5	0.9	0.9
i320-301	0.6	0.6	0.9	0.3	0.3	0.9	0.1	0.8	0.1	0.0	0.3	0.7	0.9	0.6	0.2
i320-302	0.9	0.8	0.9	0.9	0.9	0.9	0.6	0.2	0.5	0.5	0.9	0.1	0.6	0.6	0.7
i320-303	0.3	0.8	0.9	0.6	0.5	0.8	0.1	0.9	0.1	0.5	0.6	0.9	0.7	0.3	0.9
i320-304	0.5	0.1	0.2	0.5	0.2	0.4	0.3	0.1	0.1	0.1	0.2	0.3	0.5	0.2	0.2
i320-305	0.9	0.1	0.2	0.2	0.1	0.8	0.1	0.1	0.0	0.0	0.9	0.9	0.5	0.2	0.1
i320-311	0.9	0.4	0.9	0.4	0.3	0.9	0.9	0.8	0.0	0.1	0.8	0.7	0.9	0.2	0.7
i320-312	0.9	0.9	0.9	0.1	0.1	0.9	0.8	0.9	0.0	0.2	0.9	0.7	0.9	0.4	0.4
i320-313	0.7	0.8	0.9	0.1	0.2	0.7	0.8	0.9	0.0	0.1	0.9	0.8	0.9	0.1	0.2
i320-314	0.4	0.6	0.5	0.2	0.5	0.0	0.6	0.6	0.0	0.1	0.9	0.8	0.9	0.3	0.8
i320-315	0.9	0.7	0.1	0.1	0.3	0.6	0.7	0.1	0.0	0.5	0.9	0.8	0.7	0.2	0.7
i320-321	0.1	0.3	0.1	0.1	0.1	0.1	0.3	0.1	0.1	0.1	0.9	0.8	0.8	0.9	0.9
i320-322	0.1	0.3	0.1	0.1	0.1	0.1	0.3	0.1	0.1	0.1	0.9	0.9	0.7	0.9	0.9
i320-323	0.1	0.4	0.1	0.1	0.1	0.1	0.4	0.1	0.1	0.1	0.8	0.8	0.7	0.8	0.8
i320-324	0.1	0.3	0.1	0.1	0.1	0.1	0.3	0.1	0.1	0.1	0.9	0.9	0.8	0.9	0.9
i320-325	0.1	0.4	0.1	0.1	0.1	0.1	0.4	0.1	0.1	0.1	0.9	0.9	0.8	0.9	0.9
i320-331	0.6	0.4	0.9	0.4	0.1	0.7	0.2	0.8	0.1	0.1	0.9	0.4	0.9	0.1	0.5
i320-332	0.9	0.1	0.9	0.5	0.4	0.8	0.8	0.7	0.7	0.5	0.6	0.5	0.6	0.6	0.4
i320-333	0.9	0.5	0.8	0.1	0.2	0.2	0.9	0.7	0.0	0.4	0.6	0.7	0.6	0.2	0.8
i320-334	0.9	0.8	0.9	0.7	0.7	0.5	0.9	0.8	0.1	0.8	0.9	0.9	0.3	0.4	0.9
i320-335	0.4	0.6	0.7	0.7	0.2	0.8	0.3	0.1	0.0	0.0	0.7	0.7	0.7	0.2	0.3
i320-341	0.9	0.9	0.8	0.6	0.7	0.9	0.9	0.8	0.0	0.2	0.9	0.8	0.8	0.8	0.8
i320-342	0.9	0.9	0.8	0.7	0.7	0.9	0.9	0.8	0.0	0.1	0.9	0.8	0.9	0.2	0.9
i320-343	0.9	0.9	0.9	0.4	0.8	0.9	0.9	0.9	0.0	0.4	0.8	0.8	0.7	0.6	0.9
i320-344	0.9	0.9	0.9	0.4	0.7	0.9	0.9	0.9	0.1	0.6	0.9	0.9	0.9	0.7	0.9
i320-345	0.9	0.9	0.9	0.1	0.7	0.9	0.9	0.9	0.1	0.7	0.9	0.8	0.9	0.4	0.9

3.5.3 Experimental results

The results of numerical experiments of the construction methods combined with network centralities are shown in Tables 3.7–3.24 and Figs. 3.15–3.32. In Tables 3.7–3.24, wc is without network centrality (conventional), d_v is the degree centrality, e_v is the eigenvector centrality, c_v is the closeness centrality, b_v is the vertex betweenness centrality, and b_e is the edge betweenness centrality. ave is the average gap, dec is the decrease of average gap by using network centrality, i.e., subtract the average gap of with network centrality from the average gap without network centrality. In other words,

$$\mathrm{dec} = p - p'[\%], \tag{3.29}$$

where p is the gap of the original (without network centrality) method and p' is the gap of the proposed (with network centrality) method. If dec is positive, using network centrality contributes to finding a small-weight Steiner tree. We constructed 50 Steiner trees per instance and averaged the gaps of them. We set the scaling parameter α as a suitable value based on pre-numerical experiments.

Table 3.7 and Fig. 3.15 show the results of SPH with network centralities for the benchmark problem set B. From Table 3.7 and Fig. 3.15, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities improves the average gap for 14 (78%), 12 (67%), 12 (67%), 15 (83%), and 17 (94%) instances, respectively. Thus, using network centralities tends to contribute to finding small-weight Steiner trees for the benchmark problem set B. In these network centralities, the edge betweenness centrality shows the best performance. Unfortunately, using the closeness centrality shows the worst performance, the average gap is larger than in the original method. Using the closeness centrality failed to find small-weight Steiner trees for the instance b02. One of the reasons why the gap becomes large for this instance is the objective function value of the optimum solution is very small (83). Thus, the small mistake of selecting the wrong edges causes large gaps for this instance.

Table 3.8 and Fig. 3.16 show the results of SPH with network centralities for the benchmark problem set C. From Table 3.8 and Fig. 3.16, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities improves the average gap for 5 (25%), 6 (30%), 6 (30%), 11 (55%), and 8 (40%) instances, respectively. In these network centralities, the vertex betweenness centrality shows the best performance. However, using the degree, eigenvector, closeness, and edge betweenness centralities increases the average gaps compared with the original method. These methods show poor performance, especially for the instances c12 and c16. One of their common features is the small number of terminals (10 and 5, respectively). If the number of terminals is small, the objective function value of the optimum solution tends to be small. Thus, the small mistake of selecting the wrong edges causes large gaps for these instances.

Table 3.9 and Fig. 3.17 show the results of SPH with network centralities for the benchmark problem set D. From Table 3.9 and Fig. 3.17, using the degree, eigenvector, closeness,

vertex betweenness, and edge betweenness centralities improves the average gap for 7 (35%), 7 (35%), 2 (10%), 12 (60%), and 15 (75%) instances, respectively. However, using the degree, eigenvector, and closeness centralities increases the average gaps from the original method. These methods show poor performance for instance c19. One of the common features of them is setting $\alpha = 0.9$; thus, the effect of the original edge weight is strong. This result indicates that using these network centralities is not effective to distinguish vertices and edges that contribute to finding the small-weight Steiner tree.

Table 3.10 and Fig. 3.18 show the results of SPH with network centralities for the benchmark problem set I080. From Table 3.10 and Fig. 3.18, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities improves the average gap for 67 (67%), 78 (78%), 61 (61%), 94 (94%), and 94 (94%) instances, respectively. Thus, using network centralities tends to contribute to finding small-weight Steiner trees for the benchmark problem set I080. Unfortunately, we cannot reduce the average gap for the instances i080-021 to i080-025, i080-121 to i080-125, i080-221 to i080-225, and i080-321 to i080-325. This is because these instances are complete graphs. The network centrality of all vertices and edges becomes a similar value in a complete graph. Thus, network centralities cannot distinguish vertices and edges that should be included in the Steiner tree. Therefore, in the case of the complete graph, we should not use network centralities. Additionally, the instances in the benchmark problem sets I080, I160, and I320 have incident edge weights; the weight of edges between terminals is approximately 300, the weight of edges between a terminal and a non-terminal is approximately 200, and the weight of edges between non-terminals is approximately 100. The SPH firstly selects the shortest path between an arbitrary terminal pair to find a Steiner tree. At this step, one of the edges directly connecting terminals must be selected. Thus, the current tree consists of two terminals and an edge between them. At the next step, one of the edges directly connecting terminals must be selected again. Namely, SPH finds a Steiner tree that consists of only terminals and edges between them when the complete graph is an input. However, if the number of terminals is larger than four, we can find a small-weight Steiner tree by including a non-terminal. If we find a Steiner tree without non-terminals, its objective function value is approximately $300 \times 3 = 900$ (we need three edges to connect four terminals in this case). On the other hand, if we find a Steiner tree with one non-terminal, its objective function value is approximately $200 \times 4 = 800$ (we need four edges to connect four terminals in this case). This is the reason why the SPH cannot find a small-weight Steiner tree for complete graph instances. Except for complete graph instances, using network centralities, especially using the vertex betweenness centrality, contributes to reducing the objective function value of the obtained Steiner trees.

Table 3.11 and Fig. 3.19 show the results of SPH with network centralities for the benchmark problem set I160. From Table 3.11 and Fig. 3.19, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities improves the average gap for 60 (60%), 71 (71%), 68 (68%), 92 (92%), and 96 (96%) instances, respectively. Thus, using net-

work centralities tends to contribute to finding small-weight Steiner trees for the benchmark problem set I160. In particular, using the vertex and edge betweenness centralities reduces the gap approximately 20% for the instances i160-312 to i160-315 and 30% for the instances i160-341 to i160-345, respectively. These instances have a high density of terminals, 25%. Therefore, using the vertex and edge betweenness centrality may be effective for instances with high terminal density.

Table 3.12 and Fig. 3.20 show the results of SPH with network centralities for the benchmark problem set I320. From Table 3.12 and Fig. 3.20, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities improves the average gap for 61 (61%), 73 (73%), 66 (66%), 95 (95%), and 92 (92%) instances, respectively. Thus, using network centralities tends to contribute to finding small-weight Steiner trees for the benchmark problem set I320. Similar to the results for the benchmark problem set I160, using the vertex and edge betweenness centralities reduces the gap approximately 20% for the instances i320-311 to i320-315 and 40% for the instances i320-341 to i320-345, respectively. These instances have a high density of terminals, 25%. Therefore, using the vertex and edge betweenness centrality may be effective for instances with high terminal density.

From these results, it is revealed that SPH combined with network centralities tends to find better solutions than the original SPH. In particular, using the vertex or edge betweenness centralities shows the best performance. This is because the betweenness centrality measures the flow in graphs. The minimum Steiner tree is a tree that may connect terminals in the minimum flow. Thus, the betweenness centrality is effective to distinguish vertices and edges that should be included in a Steiner tree. On the other hand, using the closeness centrality shows the worst performance. This is because the closeness centrality ignores the structures of graphs. All of the instances that we used are random graphs. In a random graph, the average distance of all vertices becomes a similar value. Thus, the closeness centrality cannot distinguish vertices and edges that should be included in a Steiner tree.

The combination of SPH and five network centralities shows better performance than the original SPH to solve the benchmark sets B, C, D, I080, I160, and I320. Are these results the same when using other construction methods? To confirm it, we conducted the same numerical experiments by using DNH and ADH.

Table 3.13 and Fig. 3.21 show the results of DNH with network centralities for the benchmark problem set B. From Table 3.13 and Fig. 3.21, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities improves the average gap for 15 (83%), 15 (83%), 14 (78%), 14 (78%), and 16 (89%) instances, respectively. However, using not only the closeness centrality but also the degree and eigenvector centrality increases the average gap from the original method. These methods show poor performance for instance b04. This instance has the smallest number of terminals. These results indicate that using network centralities is not effective to solve the instances with a small number of terminals.

Table 3.14 and Fig. 3.22 show the results of DNH with network centralities for the bench-

mark problem set C. From Table 3.14 and Fig. 3.22, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities improves the average gap for 10 (50%), 8 (40%), 8 (40%), 14 (70%), and 16 (80%) instances, respectively. In contrast to the result of using SPH, using the closeness centrality reduces the average gap. In particular, DNH using the closeness centrality solves the instance c06 optimally, unless the gap of SPH using the closeness centrality is 0.83%. This result suggests that effective network centrality may differ depending on the construction method.

Table 3.15 and Fig. 3.23 show the results of DNH with network centralities for the benchmark problem set D. From Table 3.15 and Fig. 3.23, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities improves the average gap for 9 (45%), 9 (45%), 5 (25%), 15 (75%), and 15 (75%) instances, respectively. In contrast to the result of using SPH, using the degree centrality reduces the average gap. In particular, DNH using the degree centrality reduces the average gap 10.68% for the instance c07. This result contributes to the improvement of the average gap. This result supposes that effective network centrality may differ depending on the construction method.

Table 3.16 and Fig. 3.24 show the results of DNH with network centralities for the benchmark problem set I080. From Table 3.16 and Fig. 3.24, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities improves the average gap for 84 (84%), 95 (95%), 88 (88%), 97 (97%), and 93 (93%) instances, respectively. Similar to the result of using SPH, DNH cannot reduce the gap for the instances with a complete graph. This is because DNH also uses the shortest path between terminals when the first step of finding a Steiner tree or making the complete graph of terminals. Thus, DNH also cannot find a small-weight Steiner tree for the complete graph instances. On the other hand, using the vertex and edge betweenness centralities reduces the gap approximately 20% for the instances i080-341 to i080-345. These instances have a high density of terminals, 25%. Therefore, using the vertex and edge betweenness centrality may be effective for instances with high terminal density in the case of using DNH.

Table 3.17 and Fig. 3.25 show the results of DNH with network centralities for the benchmark problem set I160. From Table 3.17 and Fig. 3.25, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities improves the average gap for 77 (77%), 87 (87%), 92 (92%), 97 (97%), and 97 (97%) instances, respectively. The usage of the vertex and edge betweenness centralities reduces the gap approximately 30% for the instances i160-341 to i160-345. These instances have a high density of terminals, 25%. Therefore, using the vertex and edge betweenness centrality may be effective for instances with high terminal density in the case of using DNH.

Table 3.18 and Fig. 3.26 show the results of DNH with network centralities for the benchmark problem set I320. From Table 3.18 and Fig. 3.26, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities improves the average gap for 78 (78%), 87 (87%), 87 (87%), 99 (99%), and 100 (100%) instances, respectively. The us-

age of the vertex and edge betweenness centralities reduces the gap approximately 35% for the instances i320-341 to i320-345. These instances have a high density of terminals, 25%. Therefore, using the vertex and edge betweenness centrality may be effective for instances with high terminal density in the case of using DNH whatever the number of vertices.

From these results, it is revealed that DNH combined with network centralities tends to find better solutions than the original DNH. This trend is the same as that of using SPH. Similar to the results of using SPH, DNH combined with network centralities cannot find a small-weight Steiner tree when the input instance is a complete graph. This is because DNH uses the shortest path between terminals. Thus, including non-terminal is important to find a small-weight Steiner tree for complete graph instances in the benchmark problem sets I080, I160, and I320. However, DNH cannot include any non-terminals. Therefore, DNH cannot find a small-weight Steiner tree unless combined with network centralities.

Table 3.19 and Fig. 3.27 show the results of ADH with network centralities for the benchmark problem set B. From Table 3.19 and Fig. 3.27, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities improves the average gap for 14 (78%), 15 (83%), 14 (78%), 16 (89%), and 16 (89%) instances, respectively. In contrast to the result of using DNH, ADH using the eigenvector centrality successfully reduces the average gap. The ADH using the eigenvector centrality finds the optimal solution for the instance b13. This result contributes to reducing the average gap. Additionally, this result indicates that using ADH may show different results from SPH and DNH.

Table 3.20 and Fig. 3.28 show the results of ADH with network centralities for the benchmark problem set C. From Table 3.20 and Fig. 3.28, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities improves the average gap for 16 (80%), 14 (70%), 13 (65%), 17 (85%), and 11 (55%) instances, respectively. Among using SPH, DNH, and ADH, only the benchmark problem set C increases the average gap by using the edge betweenness centrality. This result indicates that the instances in the benchmark problem set C have different features from other instances.

Table 3.21 and Fig. 3.29 show the results of ADH with network centralities for the benchmark problem set D. From Table 3.21 and Fig. 3.29, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities improves the average gap for 13 (65%), 14 (70%), 11 (55%), 15 (75%), and 13 (65%) instances, respectively. For the benchmark problem set D, using each network centrality contributes to reducing the average gap. This benchmark problem set is the one with the largest number of vertices that we used. Thus, ADH combined with network centralities may be effective for large size instances.

Table 3.22 and Fig. 3.30 show the results of ADH with network centralities for the benchmark problem set I080. From Table 3.22 and Fig. 3.30, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities improves the average gap for 73 (73%), 79 (79%), 78 (78%), 86 (86%), and 84 (84%) instances, respectively. The ADH finds optimal solutions for complete graph instances. This is because ADH can include

non-terminals because ADH uses the average distance from an arbitrary vertex to all other vertices. Thus, the average gap of ADH is smaller than that of SPH and DNH.

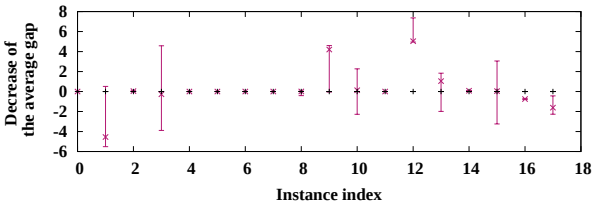
Table 3.23 and Fig. 3.31 show the results of ADH with network centralities for the benchmark problem set I160. From Table 3.23 and Fig. 3.31, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities improves the average gap for 79 (79%), 80 (80%), 78 (78%), 87 (87%), and 89 (89%) instances, respectively. Additionally, the difference in the average gap by using network centralities is small. The computation cost of the degree centrality is smaller than that of ADH. Thus, using the degree centrality may show good performance from the aspects that decreasing the gap and increasing the computation cost.

Table 3.24 and Fig. 3.32 show the results of ADH with network centralities for the benchmark problem set I320. From Table 3.24 and Fig. 3.32, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities improves the average gap for 68 (68%), 89 (89%), 76 (76%), 92 (92%), and 80 (80%) instances, respectively. In particular, using the vertex betweenness centrality shows the best performance. This trend is the same as using SPH and DNH. Thus, using the vertex betweenness centrality is effective combined with whatever construction methods.

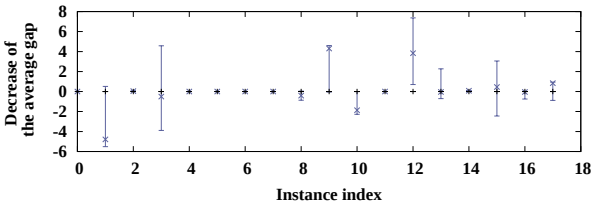
From these results, it is revealed that ADH combined with network centralities tends to find better solutions than the original ADH. In particular, using the vertex and edge betweenness centralities shows the best performance. In contrast to using SPH and DNH, ADH can find a small-weight Steiner tree when the input instance is the complete graph. This is because ADH can include any non-terminals by using the average distance from an arbitrary vertex to all other vertices. Thus, we should better use ADH for complete graph instances. Additionally, the average gap of ADH is smaller than that of SPH and DNH. However, the computation cost of ADH is larger than that of SPH and DNH. Therefore, we have to consider the balance between the decrease of the average gap and the increase of the computation time.

Table 3.7: Gaps [%] (SPH, B)

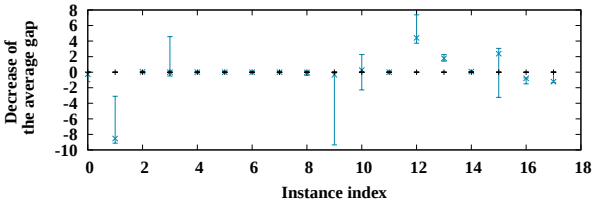
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
b01	0.00	0.00	0.00	0.00	0.00	0.27	-0.27	0.00	0.00	0.00	0.00
b02	0.51	5.06	-4.55	5.30	-4.79	9.04	-8.53	3.61	-3.10	0.00	0.51
b03	0.04	0.00	0.04	0.00	0.04	0.00	0.04	0.00	0.04	0.00	0.04
b04	4.58	4.85	-0.27	5.08	-0.50	4.58	0.00	1.42	3.16	0.00	4.58
b05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
b06	1.64	1.64	0.00	1.64	0.00	1.64	0.00	1.54	0.10	0.26	1.38
b07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
b08	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
b09	0.04	0.03	0.01	0.45	-0.41	0.06	-0.02	0.03	0.01	0.00	0.04
b10	4.60	0.40	4.20	0.30	4.30	4.93	-0.33	0.02	4.58	0.30	4.30
b11	2.27	2.14	0.13	4.14	-1.87	2.00	0.27	0.18	2.09	0.00	2.27
b12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
b13	7.37	2.33	5.04	3.54	3.83	2.96	4.41	3.64	3.73	6.47	0.90
b14	2.27	1.23	1.04	2.31	-0.04	0.48	1.79	2.28	-0.01	2.03	0.24
b15	0.08	0.00	0.08	0.00	0.08	0.00	0.08	0.00	0.08	0.00	0.08
b16	3.06	3.02	0.04	2.61	0.45	0.68	2.38	2.71	0.35	2.77	0.29
b17	0.78	1.53	-0.75	0.82	-0.04	1.57	-0.79	0.78	0.00	0.72	0.06
b18	1.40	3.01	-1.61	0.56	0.84	2.62	-1.22	2.62	-1.22	1.79	-0.39
ave	1.59	1.40	0.19	1.49	0.10	1.71	-0.12	1.05	0.54	0.80	0.79



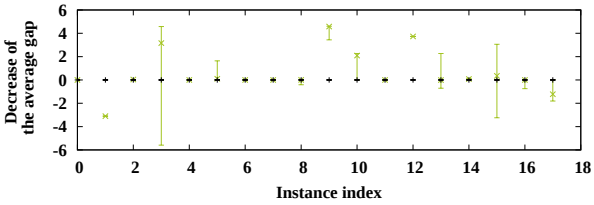
(a) degree centrality



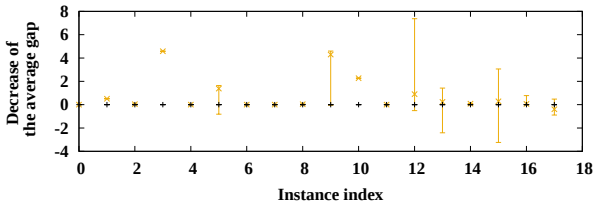
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality

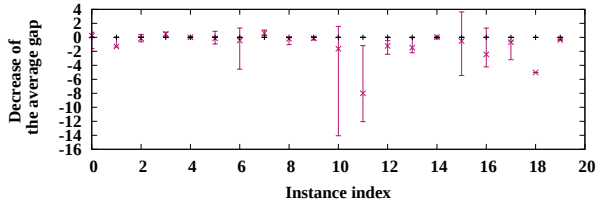


(e) edge betweenness centrality

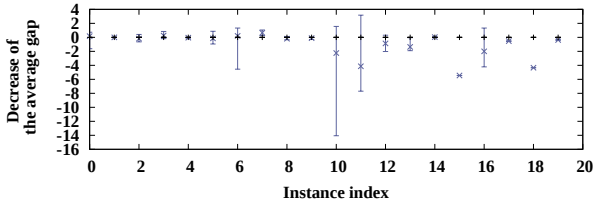
Figure 3.15: Decrease of the average gap (SPH, B)

Table 3.8: Gaps [%] (SPH, C)

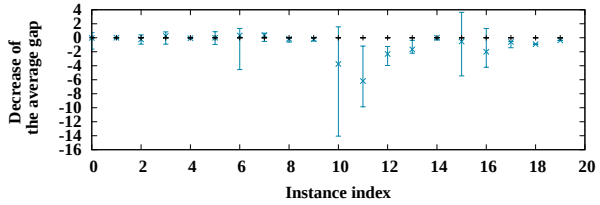
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
c01	0.73	0.47	0.26	0.56	0.17	0.75	-0.02	0.00	0.73	0.78	-0.05
c02	0.00	1.31	-1.31	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
c03	1.07	1.23	-0.16	1.25	-0.18	1.30	-0.23	1.23	-0.16	1.47	-0.40
c04	1.12	0.65	0.47	0.87	0.25	0.87	0.25	0.67	0.45	0.91	0.21
c05	0.13	0.13	0.00	0.18	-0.05	0.19	-0.06	0.18	-0.05	0.18	-0.05
c06	0.87	1.02	-0.15	1.02	-0.15	0.87	0.00	0.80	0.07	0.84	0.03
c07	1.33	1.78	-0.45	1.12	0.21	1.00	0.33	0.90	0.43	0.47	0.86
c08	1.84	1.22	0.62	1.19	0.65	1.48	0.36	1.00	0.84	1.11	0.73
c09	1.22	1.44	-0.22	1.41	-0.19	1.38	-0.16	1.41	-0.19	1.32	-0.10
c10	0.43	0.57	-0.14	0.54	-0.11	0.64	-0.21	0.55	-0.12	0.46	-0.03
c11	1.56	3.19	-1.63	3.81	-2.25	5.31	-3.75	3.56	-2.00	1.94	-0.38
c12	3.17	11.17	-8.00	7.30	-4.13	9.35	-6.18	1.83	1.34	1.83	1.34
c13	2.63	3.85	-1.22	3.49	-0.86	4.95	-2.32	2.29	0.34	2.74	-0.11
c14	2.43	3.91	-1.48	3.80	-1.37	4.09	-1.66	2.65	-0.22	1.54	0.89
c15	0.04	0.01	0.03	0.00	0.04	0.03	0.01	0.17	-0.13	0.17	-0.13
c16	3.64	4.18	-0.54	9.09	-5.45	4.18	-0.54	2.18	1.46	8.18	-4.54
c17	6.89	9.33	-2.44	8.89	-2.00	8.89	-2.00	9.67	-2.78	9.44	-2.55
c18	7.42	8.11	-0.69	7.96	-0.54	8.02	-0.60	8.11	-0.69	8.62	-1.20
c19	6.62	11.64	-5.02	10.96	-4.34	7.53	-0.91	5.48	1.14	6.78	-0.16
c20	0.37	0.75	-0.38	0.75	-0.38	0.75	-0.38	0.37	0.00	0.37	0.00
ave	2.18	3.30	-1.12	3.21	-1.03	3.08	-0.90	2.15	0.03	2.46	-0.28



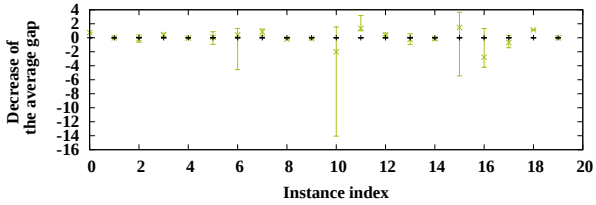
(a) degree centrality



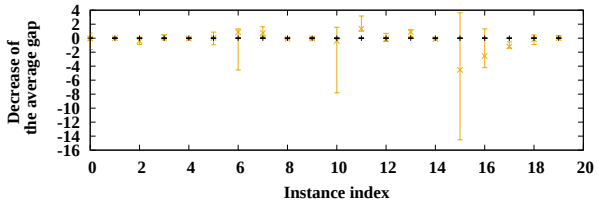
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality

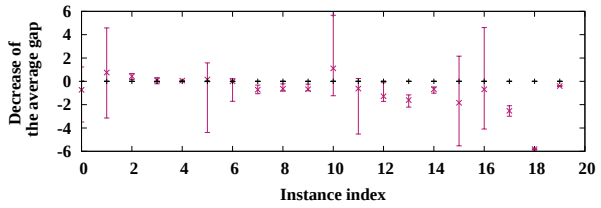


(e) edge betweenness centrality

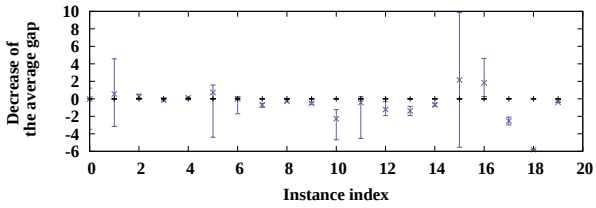
Figure 3.16: Decrease of the average gap (SPH, C)

Table 3.9: Gaps [%] (SPH, D)

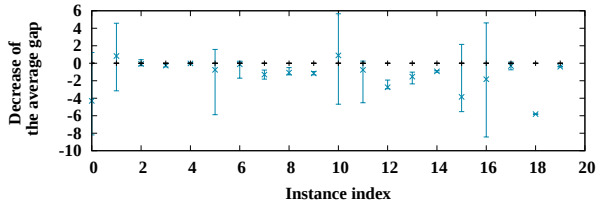
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
d01	1.23	1.96	-0.73	1.25	-0.02	5.53	-4.30	0.79	0.44	1.13	0.10
d02	4.58	3.83	0.75	4.02	0.56	3.75	0.83	3.44	1.14	2.91	1.67
d03	1.88	1.45	0.43	1.65	0.23	1.95	-0.07	1.62	0.26	1.63	0.25
d04	1.03	0.94	0.09	1.16	-0.13	1.30	-0.27	1.33	-0.30	1.33	-0.30
d05	0.34	0.28	0.06	0.21	0.13	0.36	-0.02	0.21	0.13	0.24	0.10
d06	7.55	7.40	0.15	6.81	0.74	8.30	-0.75	6.33	1.22	5.07	2.48
d07	0.23	0.23	0.00	0.23	0.00	0.35	-0.12	0.16	0.07	0.31	-0.08
d08	2.39	3.11	-0.72	3.08	-0.69	3.69	-1.30	1.98	0.41	2.26	0.13
d09	2.47	3.11	-0.64	2.76	-0.29	3.54	-1.07	2.75	-0.28	2.15	0.32
d10	0.71	1.39	-0.68	1.24	-0.53	1.87	-1.16	1.08	-0.37	0.85	-0.14
d11	5.66	4.55	1.11	7.93	-2.27	4.76	0.90	3.45	2.21	4.00	1.66
d12	0.24	0.86	-0.62	0.67	-0.43	1.00	-0.76	0.10	0.14	0.14	0.10
d13	2.28	3.56	-1.28	3.50	-1.22	5.02	-2.74	2.62	-0.34	1.67	0.61
d14	1.39	3.00	-1.61	2.78	-1.39	2.92	-1.53	1.68	-0.29	1.48	-0.09
d15	1.04	1.72	-0.68	1.74	-0.70	1.97	-0.93	1.05	-0.01	0.88	0.16
d16	9.85	11.69	-1.84	7.69	2.16	13.69	-3.84	10.00	-0.15	2.46	7.39
d17	4.61	5.30	-0.69	2.78	1.83	6.43	-1.82	2.35	2.26	1.83	2.78
d18	8.22	10.75	-2.53	10.76	-2.54	8.53	-0.31	8.65	-0.43	7.52	0.70
d19	6.45	12.26	-5.81	12.26	-5.81	12.26	-5.81	6.12	0.33	7.99	-1.54
d20	1.09	1.49	-0.40	1.49	-0.40	1.49	-0.40	0.74	0.35	0.55	0.54
ave	3.16	3.94	-0.78	3.70	-0.54	4.44	-1.28	2.82	0.34	2.32	0.84



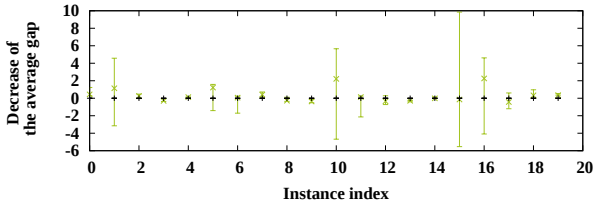
(a) degree centrality



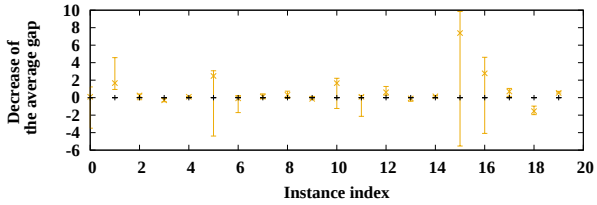
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality



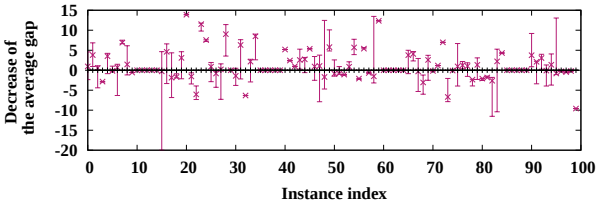
(e) edge betweenness centrality

Figure 3.17: Decrease of the average gap (SPH, D)

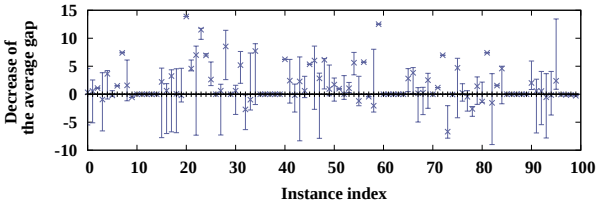
Table 3.10: Gaps [%] (SPH, I080)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
i080-001	4.58	3.63	0.95	4.24	0.34	2.69	1.89	4.51	0.07	4.50	0.08
i080-002	6.84	3.05	3.79	6.28	0.56	6.93	-0.09	0.00	6.84	4.92	1.92
i080-003	1.10	0.55	0.55	0.00	1.10	0.33	0.77	0.00	1.10	0.00	1.10
i080-004	3.84	6.69	-2.85	4.79	-0.95	7.74	-3.90	4.57	-0.73	4.35	-0.51
i080-005	4.45	0.94	3.51	0.80	3.65	4.19	0.26	2.04	2.41	4.27	0.18
i080-011	6.84	7.01	-0.17	7.05	-0.21	2.50	4.34	4.04	2.80	3.62	3.22
i080-012	2.17	1.46	0.71	0.67	1.50	1.30	0.87	0.63	1.54	1.75	0.42
i080-013	7.51	0.61	6.90	0.13	7.38	0.57	6.94	0.00	7.51	1.31	6.20
i080-014	12.49	11.04	1.45	10.90	1.59	13.70	-1.21	8.51	3.98	6.88	5.61
i080-015	6.59	7.16	-0.57	7.16	-0.57	7.16	-0.57	3.33	3.26	5.84	0.75
i080-021	25.19	25.19	0.00	25.19	0.00	25.19	0.00	25.19	0.00	25.19	0.00
i080-022	25.38	25.38	0.00	25.38	0.00	25.38	0.00	25.38	0.00	25.38	0.00
i080-023	25.30	25.30	0.00	25.30	0.00	25.30	0.00	25.30	0.00	25.30	0.00
i080-024	26.87	26.87	0.00	26.87	0.00	26.87	0.00	26.87	0.00	26.87	0.00
i080-025	27.62	27.62	0.00	27.62	0.00	27.62	0.00	27.62	0.00	27.62	0.00
i080-031	4.66	4.95	-0.29	2.47	2.19	1.95	2.71	0.65	4.01	2.89	1.77
i080-032	7.16	2.55	4.61	6.52	0.64	2.28	4.88	3.09	4.07	1.21	5.95
i080-033	5.03	6.88	-1.85	1.78	3.25	6.24	-1.21	3.39	1.64	1.31	3.72
i080-034	4.55	6.24	-1.69	4.49	0.06	6.52	-1.97	5.39	-0.84	5.19	-0.64
i080-035	4.63	1.53	3.10	4.83	-0.20	5.19	-0.56	0.66	3.97	1.59	3.04
i080-041	15.46	1.57	13.89	1.57	13.89	1.57	13.89	12.22	3.24	9.93	5.53
i080-042	12.26	13.82	-1.56	7.69	4.57	13.90	-1.64	7.01	5.25	8.54	3.72
i080-043	8.74	14.78	-6.04	1.75	6.99	5.19	3.55	7.75	0.99	6.91	1.83
i080-044	12.87	1.34	11.53	1.34	11.53	1.34	11.53	1.58	11.29	6.58	6.29
i080-045	12.11	4.55	7.56	5.18	6.93	4.77	7.34	4.85	7.26	6.10	6.01
i080-101	5.76	4.87	0.89	3.14	2.62	1.69	4.07	0.00	5.76	0.00	5.76
i080-102	0.06	0.87	-0.81	0.06	0.00	0.81	-0.75	0.46	-0.40	0.88	-0.82
i080-103	4.24	4.03	0.21	3.62	0.62	3.00	1.24	3.45	0.79	2.96	1.28
i080-104	11.42	2.39	9.03	2.88	8.54	2.45	8.97	2.77	8.65	6.34	5.08
i080-105	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-111	2.04	3.46	-1.42	1.41	0.63	1.64	0.40	0.65	1.39	1.09	0.95
i080-112	7.68	1.37	6.31	2.48	5.20	5.32	2.36	1.12	6.56	5.47	2.21
i080-113	4.82	11.15	-6.33	7.54	-2.72	11.15	-6.33	2.26	2.56	2.44	2.38
i080-114	18.24	16.08	2.16	19.22	-0.98	11.80	6.44	4.51	13.73	5.31	12.93
i080-115	9.50	1.00	8.50	1.79	7.71	1.42	8.08	3.30	6.20	1.21	8.29
i080-121	31.58	31.58	0.00	31.58	0.00	31.58	0.00	31.58	0.00	31.58	0.00
i080-122	31.97	31.97	0.00	31.97	0.00	31.97	0.00	31.97	0.00	31.97	0.00
i080-123	31.36	31.36	0.00	31.36	0.00	31.36	0.00	31.36	0.00	31.36	0.00
i080-124	30.68	30.68	0.00	30.68	0.00	30.68	0.00	30.68	0.00	30.68	0.00
i080-125	31.11	31.11	0.00	31.11	0.00	31.11	0.00	31.11	0.00	31.11	0.00
i080-131	10.04	4.86	5.18	3.81	6.23	6.59	3.45	6.89	3.15	4.90	5.14
i080-132	7.73	5.29	2.44	5.31	2.42	6.34	1.39	5.04	2.69	6.63	1.10
i080-133	1.82	0.84	0.98	1.99	-0.17	3.12	-1.30	1.36	0.46	0.88	0.94
i080-134	6.66	4.11	2.55	4.37	2.29	7.37	-0.71	2.55	4.11	5.30	1.36
i080-135	3.20	0.48	2.72	2.59	0.61	0.34	2.86	3.01	0.19	3.03	0.17
i080-141	10.94	5.59	5.35	5.59	5.35	5.59	5.35	4.23	6.71	6.76	4.18
i080-142	13.63	12.70	0.93	7.61	6.02	14.29	-0.66	12.13	1.50	11.78	1.85
i080-143	4.51	3.44	1.07	1.67	2.84	6.81	-2.30	2.95	1.56	7.60	-3.09
i080-144	13.30	14.99	-1.69	7.16	6.14	7.17	6.13	7.71	5.59	6.73	6.57
i080-145	12.77	7.00	5.77	11.84	0.93	2.67	10.10	2.67	10.10	6.84	5.93

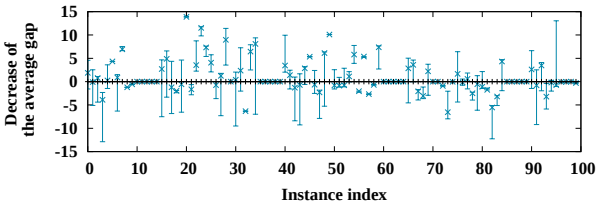
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
i080-201	2.89	3.76	-0.87	1.17	1.72	3.69	-0.80	0.01	2.88	2.36	0.53
i080-202	1.56	2.39	-0.83	0.57	0.99	2.42	-0.86	0.21	1.35	0.84	0.72
i080-203	5.46	6.55	-1.09	5.44	0.02	6.23	-0.77	2.72	2.74	2.92	2.54
i080-204	4.21	3.34	0.87	3.15	1.06	3.10	1.11	3.39	0.82	4.26	-0.05
i080-205	8.02	2.32	5.70	2.39	5.63	2.23	5.79	2.09	5.93	2.99	5.03
i080-211	7.06	9.15	-2.09	8.27	-1.21	9.11	-2.05	6.22	0.84	5.77	1.29
i080-212	18.13	12.74	5.39	12.40	5.73	12.75	5.38	7.38	10.75	5.99	12.14
i080-213	14.02	14.71	-0.69	14.46	-0.44	16.69	-2.67	3.92	10.10	3.41	10.61
i080-214	19.50	21.02	-1.52	21.56	-2.06	20.19	-0.69	2.63	16.87	2.99	16.51
i080-215	20.93	8.58	12.35	8.42	12.51	13.56	7.37	9.23	11.70	4.80	16.13
i080-221	38.89	38.89	0.00	38.89	0.00	38.89	0.00	38.89	0.00	38.89	0.00
i080-222	38.81	38.81	0.00	38.81	0.00	38.81	0.00	38.81	0.00	38.81	0.00
i080-223	39.01	39.01	0.00	39.01	0.00	39.01	0.00	39.01	0.00	39.01	0.00
i080-224	38.46	38.46	0.00	38.46	0.00	38.46	0.00	38.46	0.00	38.46	0.00
i080-225	38.44	38.44	0.00	38.44	0.00	38.44	0.00	38.44	0.00	38.44	0.00
i080-231	7.60	3.83	3.77	4.80	2.80	4.75	2.85	3.67	3.93	3.21	4.39
i080-232	8.99	4.91	4.08	5.17	3.82	5.39	3.60	5.15	3.84	6.70	2.29
i080-233	4.21	4.51	-0.30	4.04	0.17	6.28	-2.07	5.16	-0.95	3.70	0.51
i080-234	3.36	6.41	-3.05	3.10	0.26	6.41	-3.05	3.67	-0.31	4.91	-1.55
i080-235	5.56	2.99	2.57	3.04	2.52	3.34	2.22	3.24	2.32	4.14	1.42
i080-241	26.14	26.31	-0.17	26.14	0.00	26.14	0.00	3.67	22.47	4.76	21.38
i080-242	17.26	16.08	1.18	16.08	1.18	17.26	0.00	4.05	13.21	6.12	11.14
i080-243	21.45	14.48	6.97	14.48	6.97	22.37	-0.92	5.37	16.08	8.49	12.96
i080-244	9.17	15.86	-6.69	15.86	-6.69	15.72	-6.55	4.36	4.81	4.86	4.31
i080-245	28.50	28.58	-0.08	28.55	-0.05	28.53	-0.03	6.32	22.18	8.12	20.38
i080-301	6.95	5.99	0.96	2.24	4.71	5.28	1.67	2.20	4.75	0.47	6.48
i080-302	5.50	4.24	1.26	5.25	0.25	5.67	-0.17	2.57	2.93	2.54	2.96
i080-303	2.28	1.26	1.02	2.74	-0.46	1.70	0.58	1.11	1.17	1.72	0.56
i080-304	1.38	3.72	-2.34	4.00	-2.62	3.86	-2.48	0.28	1.10	0.20	1.18
i080-305	4.47	3.13	1.34	3.03	1.44	5.01	-0.54	1.53	2.94	1.31	3.16
i080-311	17.93	20.02	-2.09	19.24	-1.31	19.03	-1.10	6.43	11.50	5.05	12.88
i080-312	18.93	20.75	-1.82	11.54	7.39	20.69	-1.76	6.57	12.36	6.42	12.51
i080-313	10.18	12.80	-2.62	11.72	-1.54	15.72	-5.54	4.94	5.24	5.58	4.60
i080-314	6.54	4.32	2.22	5.00	1.54	9.72	-3.18	3.71	2.83	4.24	2.30
i080-315	16.50	12.20	4.30	11.91	4.59	12.19	4.31	3.42	13.08	2.73	13.77
i080-321	40.56	40.56	0.00	40.56	0.00	40.56	0.00	40.56	0.00	40.56	0.00
i080-322	39.85	39.85	0.00	39.85	0.00	39.85	0.00	39.85	0.00	39.85	0.00
i080-323	40.02	40.02	0.00	40.02	0.00	40.02	0.00	40.02	0.00	40.02	0.00
i080-324	40.31	40.31	0.00	40.31	0.00	40.31	0.00	40.31	0.00	40.31	0.00
i080-325	41.03	41.03	0.00	41.03	0.00	41.03	0.00	41.03	0.00	41.03	0.00
i080-331	11.12	7.36	3.76	9.11	2.01	8.50	2.62	4.24	6.88	4.49	6.63
i080-332	2.68	0.56	2.12	2.10	0.58	3.47	-0.79	2.93	-0.25	1.34	1.34
i080-333	7.12	4.08	3.04	6.55	0.57	3.67	3.45	2.91	4.21	3.39	3.73
i080-334	3.90	3.98	-0.08	4.44	-0.54	7.12	-3.22	3.07	0.83	2.93	0.97
i080-335	6.21	4.85	1.36	6.35	-0.14	6.39	-0.18	0.85	5.36	1.24	4.97
i080-341	18.29	19.10	-0.81	15.89	2.40	19.03	-0.74	3.49	14.80	3.81	14.48
i080-342	29.49	29.74	-0.25	29.54	-0.05	29.54	-0.05	3.63	25.86	4.52	24.97
i080-343	32.85	33.33	-0.48	32.95	-0.10	32.88	-0.03	6.07	26.78	5.08	27.77
i080-344	31.83	32.02	-0.19	31.95	-0.12	31.83	0.00	3.55	28.28	3.75	28.08
i080-345	15.25	24.83	-9.58	15.53	-0.28	15.55	-0.30	3.96	11.29	4.38	10.87
ave	14.58	13.44	1.14	12.86	1.72	13.61	0.97	9.74	4.84	10.13	4.45



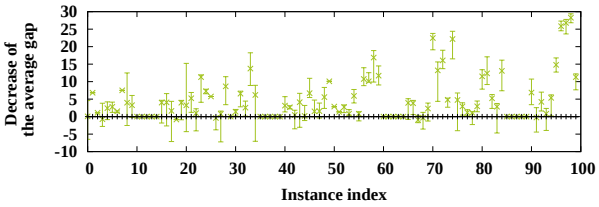
(a) degree centrality



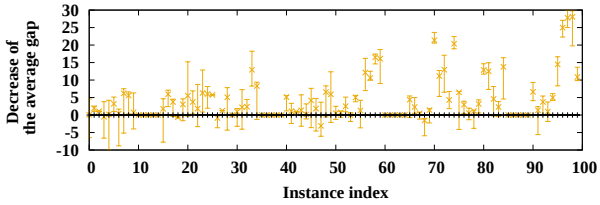
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality



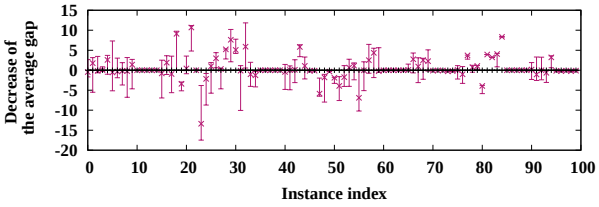
(e) edge betweenness centrality

Figure 3.18: Decrease of the average gap (SPH, I080)

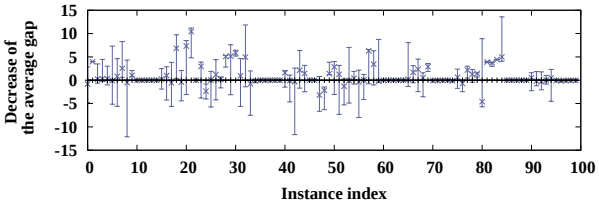
Table 3.11: Gaps [%] (SPH, I160)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
i160-001	3.11	3.57	-0.46	3.92	-0.81	1.40	1.71	3.26	-0.15	1.65	1.46
i160-002	4.08	2.31	1.77	0.09	3.99	2.75	1.33	1.09	2.99	0.54	3.54
i160-003	3.50	3.66	-0.16	3.26	0.24	3.64	-0.14	2.05	1.45	3.17	0.33
i160-004	5.02	4.97	0.05	4.71	0.31	4.42	0.60	5.03	-0.01	5.02	0.00
i160-005	7.45	4.88	2.57	7.13	0.32	7.78	-0.33	1.48	5.97	1.89	5.56
i160-011	7.32	7.88	-0.56	7.39	-0.07	7.55	-0.23	5.14	2.18	5.13	2.19
i160-012	6.06	6.38	-0.32	5.24	0.82	6.79	-0.73	6.00	0.06	5.60	0.46
i160-013	8.28	8.43	-0.15	5.74	2.54	8.04	0.24	6.94	1.34	7.88	0.40
i160-014	4.98	5.19	-0.21	5.49	-0.51	5.41	-0.43	5.46	-0.48	5.72	-0.74
i160-015	8.07	6.67	1.40	7.02	1.05	6.76	1.31	6.02	2.05	5.71	2.36
i160-021	30.47	30.47	0.00	30.47	0.00	30.47	0.00	30.47	0.00	30.47	0.00
i160-022	29.74	29.74	0.00	29.74	0.00	29.74	0.00	29.74	0.00	29.74	0.00
i160-023	29.90	29.90	0.00	29.90	0.00	29.90	0.00	29.90	0.00	29.90	0.00
i160-024	28.96	28.96	0.00	28.96	0.00	28.96	0.00	28.96	0.00	28.96	0.00
i160-025	28.99	28.99	0.00	28.99	0.00	28.99	0.00	28.99	0.00	28.99	0.00
i160-031	2.51	3.31	-0.80	2.27	0.24	2.50	0.01	2.63	-0.12	2.00	0.51
i160-032	5.72	3.83	1.89	4.72	1.00	4.10	1.62	4.29	1.43	3.34	2.38
i160-033	3.85	4.79	-0.94	4.43	-0.58	5.53	-1.68	4.57	-0.72	4.89	-1.04
i160-034	9.75	0.63	9.12	2.90	6.85	0.85	8.90	3.40	6.35	1.72	8.03
i160-035	2.85	6.24	-3.39	3.26	-0.41	6.23	-3.38	2.73	0.12	2.69	0.16
i160-041	8.50	8.05	0.45	1.16	7.34	8.27	0.23	7.54	0.96	4.72	3.78
i160-042	18.26	7.58	10.68	7.83	10.43	7.84	10.42	8.12	10.14	11.37	6.89
i160-043	14.91	14.91	0.00	14.91	0.00	14.91	0.00	3.53	11.38	5.32	9.59
i160-044	9.37	22.76	-13.39	6.41	2.96	12.14	-2.77	8.31	1.06	8.72	0.65
i160-045	4.79	6.94	-2.15	7.14	-2.35	9.94	-5.15	4.50	0.29	6.80	-2.01
i160-101	1.96	0.97	0.99	1.92	0.04	0.47	1.49	1.03	0.93	0.36	1.60
i160-102	6.45	3.47	2.98	5.25	1.20	5.82	0.63	2.54	3.91	2.72	3.73
i160-103	0.72	0.41	0.31	0.54	0.18	0.66	0.06	0.84	-0.12	0.63	0.09
i160-104	5.36	0.13	5.23	0.28	5.08	0.07	5.29	0.17	5.19	1.55	3.81
i160-105	10.19	2.57	7.62	5.03	5.16	9.30	0.89	0.50	9.69	1.75	8.44
i160-111	15.36	10.29	5.07	9.53	5.83	14.07	1.29	10.15	5.21	9.17	6.19
i160-112	5.36	5.47	-0.11	4.41	0.95	5.29	0.07	6.05	-0.69	4.94	0.42
i160-113	11.85	5.96	5.89	6.89	4.96	5.02	6.83	5.38	6.47	6.82	5.03
i160-114	5.57	6.58	-1.01	6.34	-0.77	4.95	0.62	3.75	1.82	3.75	1.82
i160-115	4.45	5.71	-1.26	4.57	-0.12	4.14	0.31	3.06	1.39	4.27	0.18
i160-121	35.84	35.84	0.00	35.84	0.00	35.84	0.00	35.84	0.00	35.84	0.00
i160-122	36.46	36.46	0.00	36.46	0.00	36.46	0.00	36.46	0.00	36.46	0.00
i160-123	36.31	36.31	0.00	36.31	0.00	36.31	0.00	36.31	0.00	36.31	0.00
i160-124	37.76	37.76	0.00	37.76	0.00	37.76	0.00	37.76	0.00	37.76	0.00
i160-125	36.88	36.88	0.00	36.88	0.00	36.88	0.00	36.88	0.00	36.88	0.00
i160-131	2.04	2.46	-0.42	0.44	1.60	0.70	1.34	0.49	1.55	1.01	1.03
i160-132	4.02	3.97	0.05	4.08	-0.06	6.75	-2.73	3.79	0.23	3.58	0.44
i160-133	10.32	10.22	0.10	10.64	-0.32	11.39	-1.07	5.48	4.84	6.79	3.53
i160-134	6.91	1.08	5.83	4.76	2.15	1.20	5.71	2.92	3.99	4.37	2.54
i160-135	6.22	5.14	1.08	4.75	1.47	3.14	3.08	2.98	3.24	1.87	4.35
i160-141	28.09	28.25	-0.16	28.09	0.00	28.09	0.00	7.67	20.42	7.85	20.24
i160-142	27.83	27.95	-0.12	27.83	0.00	27.83	0.00	5.47	22.36	6.27	21.56
i160-143	9.65	15.49	-5.84	12.84	-3.19	11.88	-2.23	8.15	1.50	8.34	1.31
i160-144	7.15	8.95	-1.80	9.33	-2.18	9.23	-2.08	6.19	0.96	6.15	1.00
i160-145	12.90	13.23	-0.33	11.50	1.40	12.55	0.35	7.82	5.08	9.82	3.08

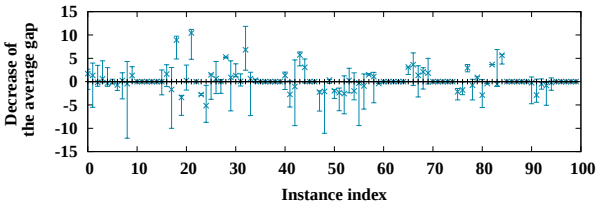
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
i160-201	4.28	6.17	-1.89	1.43	2.85	6.22	-1.94	4.08	0.20	2.50	1.78
i160-202	4.81	8.73	-3.92	3.54	1.27	7.17	-2.36	5.50	-0.69	3.78	1.03
i160-203	2.46	4.20	-1.74	3.75	-1.29	5.02	-2.56	1.86	0.60	2.42	0.04
i160-204	9.98	9.32	0.66	10.03	-0.05	9.73	0.25	5.38	4.60	4.11	5.87
i160-205	4.77	3.65	1.12	4.35	0.42	6.77	-2.00	1.82	2.95	1.73	3.04
i160-211	8.65	15.55	-6.90	9.00	-0.35	8.95	-0.30	5.80	2.85	5.59	3.06
i160-212	10.34	10.36	-0.02	10.29	0.05	11.26	-0.92	7.42	2.92	4.97	5.37
i160-213	18.75	16.24	2.51	12.46	6.29	17.27	1.48	6.20	12.55	6.47	12.28
i160-214	10.36	6.02	4.34	6.92	3.44	9.28	1.08	7.03	3.33	6.75	3.61
i160-215	22.40	22.64	-0.24	22.62	-0.22	22.78	-0.38	8.08	14.32	9.09	13.31
i160-221	41.15	41.15	0.00	41.15	0.00	41.15	0.00	41.15	0.00	41.15	0.00
i160-222	42.09	42.09	0.00	42.09	0.00	42.09	0.00	42.09	0.00	42.09	0.00
i160-223	41.35	41.35	0.00	41.35	0.00	41.35	0.00	41.35	0.00	41.35	0.00
i160-224	41.96	41.96	0.00	41.96	0.00	41.96	0.00	41.96	0.00	41.96	0.00
i160-225	40.67	40.67	0.00	40.67	0.00	40.67	0.00	40.67	0.00	40.67	0.00
i160-231	10.28	10.22	0.06	10.10	0.18	7.16	3.12	6.05	4.23	6.54	3.74
i160-232	10.12	7.36	2.76	8.41	1.71	6.48	3.64	3.50	6.62	2.25	7.87
i160-233	7.73	6.79	0.94	5.39	2.34	6.36	1.37	5.33	2.40	4.63	3.10
i160-234	3.13	0.62	2.51	2.58	0.55	1.05	2.08	2.21	0.92	2.19	0.94
i160-235	10.82	8.56	2.26	7.93	2.89	8.97	1.85	5.40	5.42	4.23	6.59
i160-241	34.17	34.29	-0.12	34.25	-0.08	34.17	0.00	4.15	30.02	6.66	27.51
i160-242	33.45	33.51	-0.06	33.49	-0.04	33.45	0.00	4.45	29.00	5.13	28.32
i160-243	34.91	34.97	-0.06	34.95	-0.04	34.91	0.00	5.39	29.52	7.36	27.55
i160-244	33.59	33.87	-0.28	33.59	0.00	33.59	0.00	4.13	29.46	6.48	27.11
i160-245	33.93	34.15	-0.22	34.05	-0.12	33.93	0.00	5.17	28.76	6.88	27.05
i160-301	4.66	4.60	0.06	4.03	0.63	6.82	-2.16	3.32	1.34	1.15	3.51
i160-302	3.12	4.12	-1.00	3.81	-0.69	4.91	-1.79	2.96	0.16	2.93	0.19
i160-303	5.58	1.86	3.72	3.38	2.20	2.63	2.95	0.29	5.29	1.72	3.86
i160-304	4.20	3.44	0.76	2.92	1.28	4.97	-0.77	1.98	2.22	3.56	0.64
i160-305	4.22	3.16	1.06	2.76	1.46	3.31	0.91	1.87	2.35	1.82	2.40
i160-311	15.96	19.89	-3.93	20.55	-4.59	18.85	-2.89	5.93	10.03	6.04	9.92
i160-312	26.02	22.09	3.93	22.12	3.90	26.43	-0.41	5.88	20.14	5.29	20.73
i160-313	24.38	21.17	3.21	20.62	3.76	20.73	3.65	5.55	18.83	5.84	18.54
i160-314	25.46	21.40	4.06	21.03	4.43	25.99	-0.53	6.89	18.57	6.61	18.85
i160-315	26.06	17.72	8.34	21.07	4.99	20.39	5.67	4.03	22.03	3.46	22.60
i160-321	43.27	43.27	0.00	43.27	0.00	43.27	0.00	43.27	0.00	43.27	0.00
i160-322	43.70	43.70	0.00	43.70	0.00	43.70	0.00	43.70	0.00	43.70	0.00
i160-323	43.38	43.38	0.00	43.38	0.00	43.38	0.00	43.38	0.00	43.38	0.00
i160-324	43.42	43.42	0.00	43.42	0.00	43.42	0.00	43.42	0.00	43.42	0.00
i160-325	43.59	43.59	0.00	43.59	0.00	43.59	0.00	43.59	0.00	43.59	0.00
i160-331	7.06	6.90	0.16	6.52	0.54	7.29	-0.23	4.83	2.23	4.93	2.13
i160-332	6.86	7.93	-1.07	7.07	-0.21	9.73	-2.87	3.65	3.21	2.31	4.55
i160-333	4.66	4.66	0.00	4.97	-0.31	5.16	-0.50	3.25	1.41	2.87	1.79
i160-334	5.29	5.94	-0.65	5.25	0.04	6.11	-0.82	4.88	0.41	5.74	-0.45
i160-335	6.05	2.84	3.21	5.54	0.51	6.23	-0.18	3.73	2.32	3.62	2.43
i160-341	37.13	37.28	-0.15	37.17	-0.04	37.13	0.00	4.90	32.23	5.78	31.35
i160-342	37.28	37.49	-0.21	37.43	-0.15	37.28	0.00	3.14	34.14	3.77	33.51
i160-343	38.82	38.92	-0.10	38.89	-0.07	38.83	-0.01	3.09	35.73	3.92	34.90
i160-344	38.00	38.24	-0.24	38.03	-0.03	38.00	0.00	3.65	34.35	4.55	33.45
i160-345	37.38	37.48	-0.10	37.43	-0.05	37.38	0.00	3.70	33.68	4.08	33.30
ave	17.44	16.92	0.52	16.50	0.94	17.08	0.36	11.09	6.35	11.22	6.22



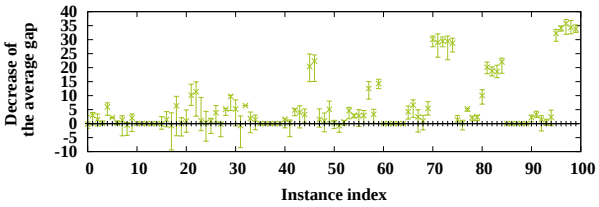
(a) degree centrality



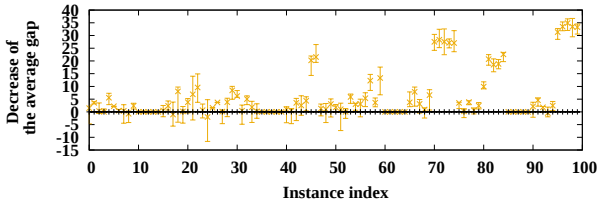
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality



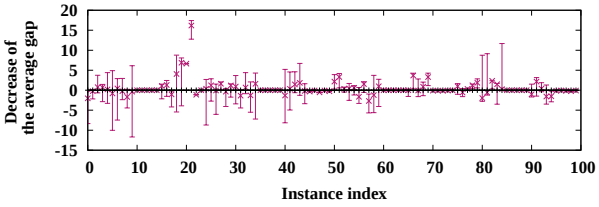
(e) edge betweenness centrality

Figure 3.19: Decrease of the average gap (SPH, I160)

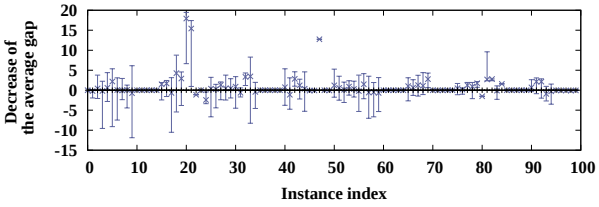
Table 3.12: Gaps [%] (SPH, I320)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
i320-001	0.08	2.10	-2.02	0.07	0.01	0.89	-0.81	0.23	-0.15	2.11	-2.03
i320-002	1.83	2.06	-0.23	2.05	-0.22	2.72	-0.89	1.27	0.56	1.96	-0.13
i320-003	9.75	8.97	0.78	9.29	0.46	8.86	0.89	7.18	2.57	3.94	5.81
i320-004	4.83	4.11	0.72	5.01	-0.18	3.95	0.88	3.60	1.23	4.48	0.35
i320-005	4.50	4.30	0.20	3.89	0.61	3.55	0.95	3.91	0.59	3.43	1.07
i320-011	5.56	6.36	-0.80	3.39	2.17	2.50	3.06	3.98	1.58	5.22	0.34
i320-012	11.56	11.10	0.46	11.51	0.05	6.67	4.89	5.12	6.44	10.27	1.29
i320-013	5.31	5.55	-0.24	5.37	-0.06	5.12	0.19	5.33	-0.02	4.91	0.40
i320-014	2.16	3.84	-1.68	2.08	0.08	1.66	0.50	1.92	0.24	1.61	0.55
i320-015	6.96	7.32	-0.36	7.70	-0.74	7.33	-0.37	1.94	5.02	6.11	0.85
i320-021	31.68	31.68	0.00	31.68	0.00	31.68	0.00	31.68	0.00	31.68	0.00
i320-022	31.57	31.57	0.00	31.57	0.00	31.57	0.00	31.57	0.00	31.57	0.00
i320-023	33.70	33.70	0.00	33.70	0.00	33.70	0.00	33.70	0.00	33.70	0.00
i320-024	31.81	31.81	0.00	31.81	0.00	31.81	0.00	31.81	0.00	31.81	0.00
i320-025	32.00	32.00	0.00	32.00	0.00	32.00	0.00	32.00	0.00	32.00	0.00
i320-031	3.27	2.20	1.07	1.77	1.50	2.51	0.76	1.71	1.56	1.99	1.28
i320-032	3.51	2.20	1.31	1.83	1.68	3.14	0.37	3.86	-0.35	1.74	1.77
i320-033	3.86	4.90	-1.04	4.55	-0.69	4.70	-0.84	3.47	0.39	3.50	0.36
i320-034	9.09	5.03	4.06	4.82	4.27	3.11	5.98	2.16	6.93	2.81	6.28
i320-035	8.67	1.79	6.88	5.75	2.92	3.14	5.53	5.81	2.86	4.96	3.71
i320-041	24.39	17.75	6.64	6.46	17.93	9.75	14.64	10.62	13.77	15.52	8.87
i320-042	23.65	7.48	16.17	8.22	15.43	21.55	2.10	8.05	15.60	16.07	7.58
i320-043	12.69	13.81	-1.12	13.81	-1.12	13.59	-0.90	5.37	7.32	6.59	6.10
i320-044	23.74	23.74	0.00	23.74	0.00	23.74	0.00	6.66	17.08	8.05	15.69
i320-045	10.52	10.15	0.37	12.95	-2.43	10.76	-0.24	10.42	0.10	8.47	2.05
i320-101	4.73	3.52	1.21	4.34	0.39	4.75	-0.02	3.29	1.44	4.84	-0.11
i320-102	3.13	3.20	-0.07	2.87	0.26	3.64	-0.51	2.26	0.87	3.22	-0.09
i320-103	3.83	2.21	1.62	2.51	1.32	1.99	1.84	2.03	1.80	2.11	1.72
i320-104	8.65	9.01	-0.36	8.17	0.48	8.71	-0.06	6.59	2.06	6.42	2.23
i320-105	3.24	2.04	1.20	2.79	0.45	2.11	1.13	2.79	0.45	2.11	1.13
i320-111	5.97	5.05	0.92	5.08	0.89	6.09	-0.12	5.49	0.48	6.18	-0.21
i320-112	5.96	7.33	-1.37	6.60	-0.64	7.34	-1.38	6.37	-0.41	6.43	-0.47
i320-113	6.91	6.23	0.68	3.58	3.33	8.11	-1.20	6.21	0.70	5.51	1.40
i320-114	8.29	9.57	-1.28	4.82	3.47	8.68	-0.39	5.30	2.99	3.71	4.58
i320-115	9.54	7.93	1.61	9.97	-0.43	8.82	0.72	7.05	2.49	6.51	3.03
i320-121	39.51	39.51	0.00	39.51	0.00	39.51	0.00	39.51	0.00	39.51	0.00
i320-122	40.98	40.98	0.00	40.98	0.00	40.98	0.00	40.98	0.00	40.98	0.00
i320-123	39.47	39.47	0.00	39.47	0.00	39.47	0.00	39.47	0.00	39.47	0.00
i320-124	40.02	40.02	0.00	40.02	0.00	40.02	0.00	40.02	0.00	40.02	0.00
i320-125	39.70	39.70	0.00	39.70	0.00	39.70	0.00	39.70	0.00	39.70	0.00
i320-131	7.22	8.55	-1.33	6.47	0.75	6.12	1.10	4.23	2.99	3.11	4.11
i320-132	6.82	6.47	0.35	7.94	-1.12	5.67	1.15	4.41	2.41	5.56	1.26
i320-133	7.47	6.01	1.46	4.62	2.85	6.09	1.38	3.94	3.53	3.25	4.22
i320-134	9.12	7.28	1.84	7.89	1.23	7.76	1.36	4.43	4.69	4.14	4.98
i320-135	4.61	4.77	-0.16	4.24	0.37	4.19	0.42	5.98	-1.37	4.08	0.53
i320-141	32.09	32.42	-0.33	32.20	-0.11	32.20	-0.11	5.96	26.13	9.28	22.81
i320-142	33.19	33.25	-0.06	33.25	-0.06	33.19	0.00	9.55	23.64	9.16	24.03
i320-143	29.46	29.99	-0.53	16.71	12.75	29.90	-0.44	7.87	21.59	9.68	19.78
i320-144	34.82	34.88	-0.06	34.82	0.00	34.82	0.00	7.36	27.46	10.89	23.93
i320-145	31.85	32.10	-0.25	31.91	-0.06	31.88	-0.03	5.62	26.23	9.72	22.13

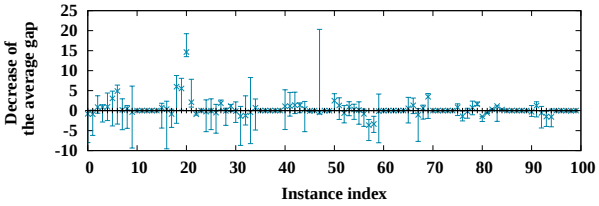
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
i320-201	7.63	5.48	2.15	6.40	1.23	5.15	2.48	3.64	3.99	4.95	2.68
i320-202	5.51	2.20	3.31	4.94	0.57	4.23	1.28	3.11	2.40	2.94	2.57
i320-203	4.81	4.46	0.35	4.71	0.10	5.41	-0.60	4.33	0.48	4.79	0.02
i320-204	5.47	5.04	0.43	4.54	0.93	4.53	0.94	4.50	0.97	3.54	1.93
i320-205	5.11	4.51	0.60	4.81	0.30	4.83	0.28	4.72	0.39	4.04	1.07
i320-211	8.83	10.48	-1.65	9.03	-0.20	8.67	0.16	7.64	1.19	7.41	1.42
i320-212	9.77	8.24	1.53	8.27	1.50	10.58	-0.81	8.32	1.45	7.16	2.61
i320-213	9.60	12.29	-2.69	10.20	-0.60	13.24	-3.64	7.23	2.37	6.87	2.73
i320-214	10.18	11.39	-1.21	10.69	-0.51	13.52	-3.34	6.74	3.44	7.53	2.65
i320-215	9.82	8.83	0.99	10.51	-0.69	9.94	-0.12	7.71	2.11	7.50	2.32
i320-221	42.91	42.91	0.00	42.91	0.00	42.91	0.00	42.91	0.00	42.91	0.00
i320-222	42.88	42.88	0.00	42.88	0.00	42.88	0.00	42.88	0.00	42.88	0.00
i320-223	42.63	42.63	0.00	42.63	0.00	42.63	0.00	42.63	0.00	42.63	0.00
i320-224	42.65	42.65	0.00	42.65	0.00	42.65	0.00	42.65	0.00	42.65	0.00
i320-225	42.94	42.94	0.00	42.94	0.00	42.94	0.00	42.94	0.00	42.94	0.00
i320-231	5.62	5.62	0.00	4.60	1.02	5.03	0.59	4.91	0.71	4.79	0.83
i320-232	7.36	3.69	3.67	6.71	0.65	6.02	1.34	5.75	1.61	4.94	2.42
i320-233	6.86	6.92	-0.06	5.57	1.29	7.91	-1.05	6.82	0.04	7.30	-0.44
i320-234	7.57	6.70	0.87	6.46	1.11	6.94	0.63	5.49	2.08	5.78	1.79
i320-235	6.73	3.44	3.29	3.96	2.77	3.26	3.47	4.98	1.75	4.65	2.08
i320-241	37.77	37.88	-0.11	37.77	0.00	37.77	0.00	5.21	32.56	6.83	30.94
i320-242	37.60	37.71	-0.11	37.64	-0.04	37.60	0.00	4.54	33.06	7.23	30.37
i320-243	37.31	37.44	-0.13	37.31	0.00	37.31	0.00	4.60	32.71	5.89	31.42
i320-244	37.21	37.27	-0.06	37.21	0.00	37.21	0.00	4.83	32.38	5.76	31.45
i320-245	37.87	37.92	-0.05	37.87	0.00	37.87	0.00	5.28	32.59	7.24	30.63
i320-301	3.73	2.75	0.98	3.24	0.49	2.88	0.85	2.07	1.66	3.41	0.32
i320-302	2.97	3.55	-0.58	2.81	0.16	4.38	-1.41	2.44	0.53	3.20	-0.23
i320-303	3.58	3.25	0.33	2.32	1.26	3.77	-0.19	2.05	1.53	2.90	0.68
i320-304	3.96	2.71	1.25	3.04	0.92	3.20	0.76	2.02	1.94	1.63	2.33
i320-305	5.28	3.42	1.86	3.52	1.76	3.67	1.61	2.45	2.83	2.30	2.98
i320-311	22.46	24.41	-1.95	23.97	-1.51	23.90	-1.44	6.82	15.64	6.58	15.88
i320-312	27.12	27.72	-0.60	24.43	2.69	27.72	-0.60	7.78	19.34	6.79	20.33
i320-313	28.70	26.34	2.36	25.90	2.80	28.38	0.32	5.61	23.09	5.82	22.88
i320-314	21.84	20.34	1.50	21.95	-0.11	20.70	1.14	6.98	14.86	6.55	15.29
i320-315	29.68	29.43	0.25	28.09	1.59	29.41	0.27	6.97	22.71	6.81	22.87
i320-321	45.28	45.28	0.00	45.28	0.00	45.28	0.00	45.28	0.00	45.28	0.00
i320-322	45.19	45.19	0.00	45.19	0.00	45.19	0.00	45.19	0.00	45.19	0.00
i320-323	45.41	45.41	0.00	45.41	0.00	45.41	0.00	45.41	0.00	45.41	0.00
i320-324	45.45	45.45	0.00	45.45	0.00	45.45	0.00	45.45	0.00	45.45	0.00
i320-325	45.41	45.41	0.00	45.41	0.00	45.41	0.00	45.41	0.00	45.41	0.00
i320-331	8.37	9.39	-1.02	7.66	0.71	8.65	-0.28	4.95	3.42	4.73	3.64
i320-332	8.02	5.87	2.15	5.88	2.14	6.78	1.24	4.54	3.48	4.64	3.38
i320-333	8.60	8.20	0.40	6.46	2.14	9.17	-0.57	6.59	2.01	5.49	3.11
i320-334	6.58	8.15	-1.57	7.49	-0.91	8.06	-1.48	5.74	0.84	5.58	1.00
i320-335	4.80	6.32	-1.52	5.08	-0.28	6.36	-1.56	4.26	0.54	3.18	1.62
i320-341	41.45	41.59	-0.14	41.46	-0.01	41.46	-0.01	2.34	39.11	3.11	38.34
i320-342	42.11	42.18	-0.07	42.18	-0.07	42.12	-0.01	2.29	39.82	2.84	39.27
i320-343	41.38	41.52	-0.14	41.45	-0.07	41.40	-0.02	2.55	38.83	3.34	38.04
i320-344	41.59	41.83	-0.24	41.69	-0.10	41.64	-0.05	2.15	39.44	3.34	38.25
i320-345	41.98	42.07	-0.09	42.01	-0.03	41.98	0.00	1.88	40.10	3.09	38.89
ave	19.27	18.76	0.51	18.36	0.91	18.85	0.42	11.93	7.34	12.33	6.94



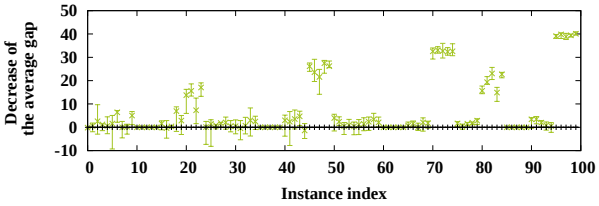
(a) degree centrality



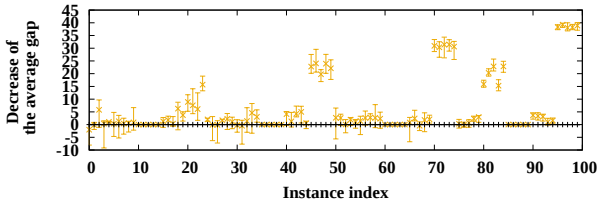
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality

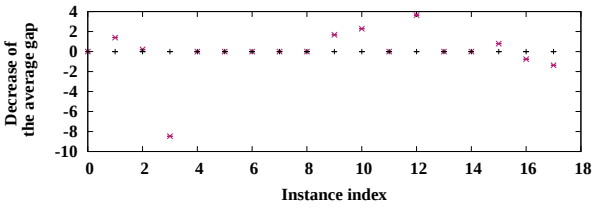


(e) edge betweenness centrality

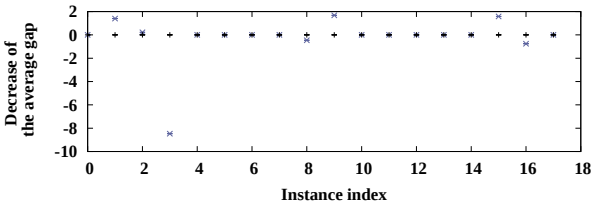
Figure 3.20: Decrease of the average gap (SPH, I320)

Table 3.13: Gaps [%] (DNH, B)

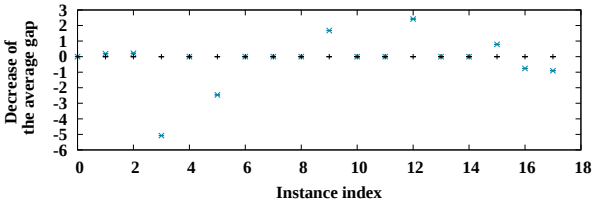
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
b01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
b02	7.42	6.02	1.40	6.02	1.40	7.23	0.19	3.61	3.81	0.00	7.42
b03	0.22	0.00	0.22	0.00	0.22	0.00	0.22	0.00	0.22	0.00	0.22
b04	0.00	8.47	-8.47	8.47	-8.47	5.08	-5.08	5.08	-5.08	0.00	0.00
b05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
b06	1.64	1.64	0.00	1.64	0.00	4.10	-2.46	4.10	-2.46	4.10	-2.46
b07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
b08	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
b09	0.45	0.45	0.00	0.91	-0.46	0.45	0.00	0.45	0.00	0.00	0.45
b10	7.49	5.81	1.68	5.81	1.68	5.81	1.68	1.16	6.33	4.65	2.84
b11	4.55	2.27	2.28	4.55	0.00	4.55	0.00	2.27	2.28	0.00	4.55
b12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
b13	6.06	2.42	3.64	6.06	0.00	3.64	2.42	3.64	2.42	6.06	0.00
b14	0.85	0.85	0.00	0.85	0.00	0.85	0.00	0.85	0.00	0.85	0.00
b15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
b16	7.09	6.30	0.79	5.51	1.58	6.30	0.79	5.51	1.58	5.51	1.58
b17	1.53	2.29	-0.76	2.29	-0.76	2.29	-0.76	2.29	-0.76	1.53	0.00
b18	1.38	2.75	-1.37	1.38	0.00	2.29	-0.91	2.29	-0.91	2.29	-0.91
ave	2.15	2.18	-0.03	2.42	-0.27	2.37	-0.22	1.74	0.41	1.39	0.76



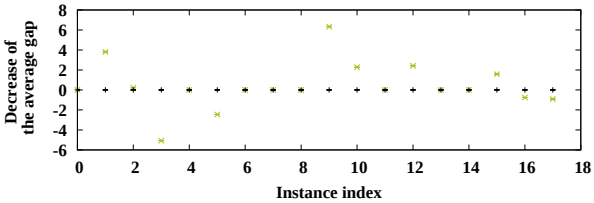
(a) degree centrality



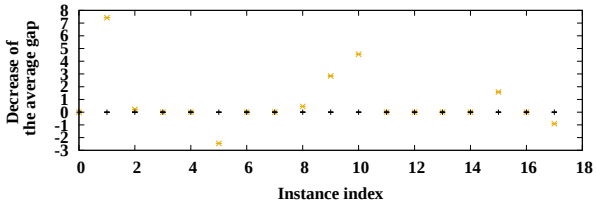
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality

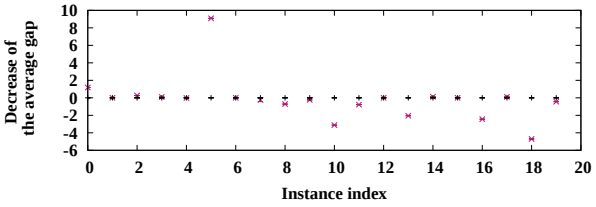


(e) edge betweenness centrality

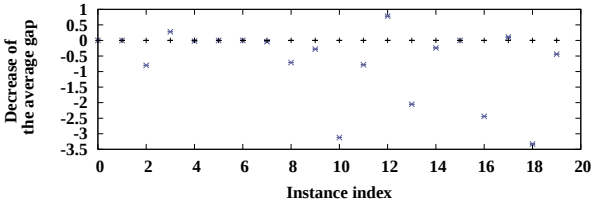
Figure 3.21: Decrease of the average gap (DNH, B)

Table 3.14: Gaps [%] (DNH, C)

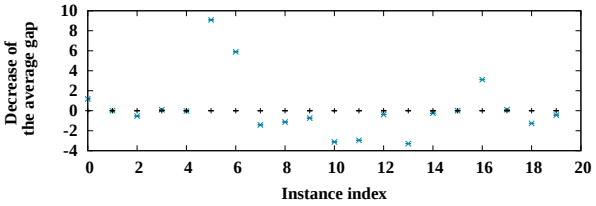
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
c01	3.53	2.35	1.18	3.53	0.00	2.35	1.18	0.00	3.53	1.18	2.35
c02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
c03	1.19	0.93	0.26	1.99	-0.80	1.72	-0.53	1.99	-0.80	1.99	-0.80
c04	2.69	2.59	0.10	2.41	0.28	2.59	0.10	1.76	0.93	1.58	1.11
c05	0.42	0.44	-0.02	0.44	-0.02	0.44	-0.02	0.44	-0.02	0.44	-0.02
c06	9.09	0.00	9.09	9.09	0.00	0.00	9.09	9.09	0.00	9.09	0.00
c07	12.75	12.75	0.00	12.75	0.00	6.86	5.89	11.76	0.99	6.86	5.89
c08	2.91	3.15	-0.24	2.95	-0.04	4.32	-1.41	2.16	0.75	2.55	0.36
c09	1.84	2.55	-0.71	2.55	-0.71	2.97	-1.13	1.84	0.00	1.98	-0.14
c10	1.18	1.41	-0.23	1.46	-0.28	1.92	-0.74	1.01	0.17	0.55	0.63
c11	12.50	15.62	-3.12	15.62	-3.12	15.62	-3.12	15.62	-3.12	9.38	3.12
c12	5.74	6.52	-0.78	6.52	-0.78	8.70	-2.96	2.17	3.57	2.17	3.57
c13	5.43	5.43	0.00	4.65	0.78	5.81	-0.38	2.71	2.72	2.33	3.10
c14	2.28	4.33	-2.05	4.33	-2.05	5.57	-3.29	2.79	-0.51	2.17	0.11
c15	0.66	0.54	0.12	0.90	-0.24	0.90	-0.24	0.18	0.48	0.18	0.48
c16	18.18	18.18	0.00	18.18	0.00	18.18	0.00	18.18	0.00	18.18	0.00
c17	8.67	11.11	-2.44	11.11	-2.44	5.56	3.11	11.11	-2.44	5.56	3.11
c18	8.07	7.96	0.11	7.96	0.11	7.96	0.11	7.08	0.99	7.96	0.11
c19	6.26	10.96	-4.70	9.59	-3.33	7.53	-1.27	4.79	1.47	6.16	0.10
c20	0.31	0.75	-0.44	0.75	-0.44	0.75	-0.44	0.37	-0.06	0.37	-0.06
ave	5.18	5.38	-0.20	5.84	-0.66	4.99	0.19	4.75	0.43	4.03	1.15



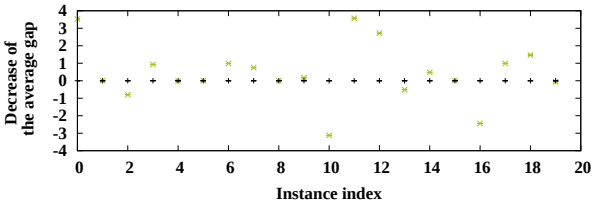
(a) degree centrality



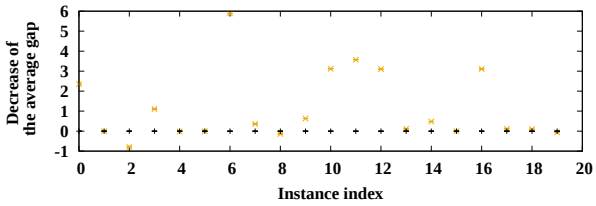
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality

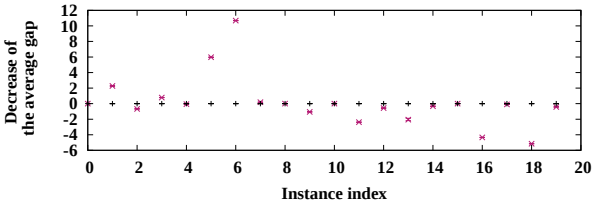


(e) edge betweenness centrality

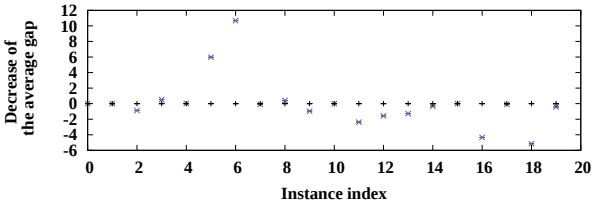
Figure 3.22: Decrease of the average gap (DNH, C)

Table 3.15: Gaps [%] (DNH, D)

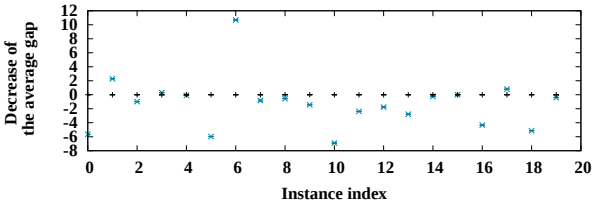
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
d01	0.94	0.94	0.00	0.94	0.00	6.60	-5.66	0.94	0.00	0.94	0.00
d02	7.73	5.45	2.28	7.73	0.00	5.45	2.28	3.64	4.09	3.64	4.09
d03	2.20	2.88	-0.68	3.07	-0.87	3.19	-0.99	3.07	-0.87	3.07	-0.87
d04	2.38	1.60	0.78	1.86	0.52	2.07	0.31	1.81	0.57	2.17	0.21
d05	0.43	0.49	-0.06	0.40	0.03	0.52	-0.09	0.40	0.03	0.40	0.03
d06	11.94	5.97	5.97	5.97	5.97	17.91	-5.97	5.97	5.97	10.45	1.49
d07	12.62	1.94	10.68	1.94	10.68	1.94	10.68	1.94	10.68	1.94	10.68
d08	3.92	3.73	0.19	4.01	-0.09	4.74	-0.82	4.10	-0.18	4.10	-0.18
d09	4.14	4.14	0.00	3.73	0.41	4.70	-0.56	3.66	0.48	2.83	1.31
d10	0.98	2.04	-1.06	1.94	-0.96	2.42	-1.44	1.28	-0.30	0.57	0.41
d11	10.34	10.34	0.00	10.34	0.00	17.24	-6.90	10.34	0.00	6.90	3.44
d12	2.38	4.76	-2.38	4.76	-2.38	4.76	-2.38	0.00	2.38	4.76	-2.38
d13	3.83	4.40	-0.57	5.40	-1.57	5.60	-1.77	4.40	-0.57	3.60	0.23
d14	1.87	3.93	-2.06	3.15	-1.28	4.65	-2.78	3.30	-1.43	2.55	-0.68
d15	1.82	2.15	-0.33	2.15	-0.33	2.06	-0.24	1.34	0.48	1.25	0.57
d16	23.08	23.08	0.00	23.08	0.00	23.08	0.00	15.38	7.70	15.38	7.70
d17	8.70	13.04	-4.34	13.04	-4.34	13.04	-4.34	8.70	0.00	4.35	4.35
d18	9.77	9.87	-0.10	9.87	-0.10	8.97	0.80	8.97	0.80	9.42	0.35
d19	7.11	12.26	-5.15	12.26	-5.15	12.26	-5.15	6.45	0.66	8.71	-1.60
d20	1.07	1.49	-0.42	1.49	-0.42	1.49	-0.42	0.56	0.51	0.56	0.51
ave	5.86	5.72	0.14	5.86	0.00	7.13	-1.27	4.31	1.55	4.38	1.48



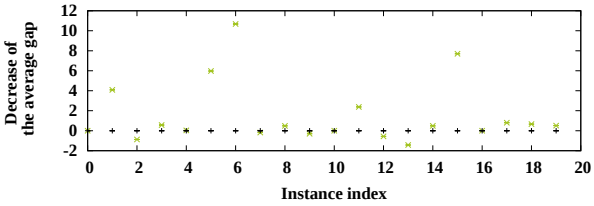
(a) degree centrality



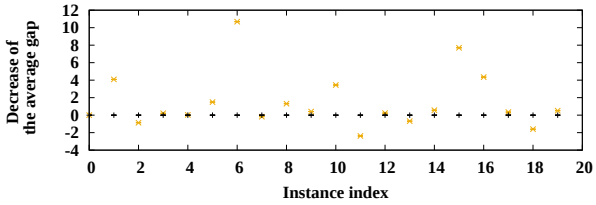
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality



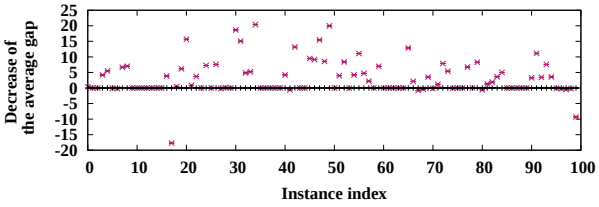
(e) edge betweenness centrality

Figure 3.23: Decrease of the average gap (DNH, D)

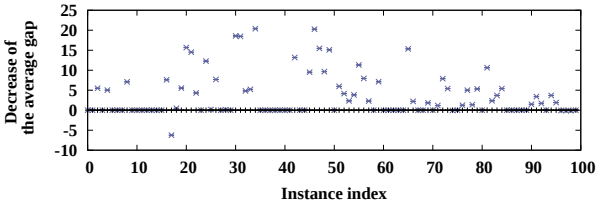
Table 3.16: Gaps [%] (DNH, I080)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
i080-001	11.58	11.19	0.39	11.58	0.00	11.19	0.39	11.08	0.50	12.09	-0.51
i080-002	11.95	11.95	0.00	11.95	0.00	11.95	0.00	0.00	11.95	5.72	6.23
i080-003	5.49	5.49	0.00	0.00	5.49	5.49	0.00	0.00	5.49	0.00	5.49
i080-004	16.45	12.27	4.18	16.45	0.00	16.45	0.00	12.27	4.18	5.68	10.77
i080-005	11.40	5.92	5.48	6.37	5.03	5.92	5.48	4.92	6.48	4.92	6.48
i080-011	7.17	7.23	-0.06	7.17	0.00	2.50	4.67	7.17	0.00	7.23	-0.06
i080-012	0.47	0.67	-0.20	0.47	0.00	0.67	-0.20	0.47	0.00	0.47	0.00
i080-013	6.81	0.14	6.67	6.81	0.00	0.51	6.30	0.00	6.81	0.00	6.81
i080-014	20.83	13.82	7.01	13.74	7.09	13.74	7.09	13.74	7.09	13.17	7.66
i080-015	7.16	7.16	0.00	7.16	0.00	7.16	0.00	7.16	0.00	7.16	0.00
i080-021	25.19	25.19	0.00	25.19	0.00	25.19	0.00	25.19	0.00	25.19	0.00
i080-022	25.38	25.38	0.00	25.38	0.00	25.38	0.00	25.38	0.00	25.38	0.00
i080-023	25.30	25.30	0.00	25.30	0.00	25.30	0.00	25.30	0.00	25.30	0.00
i080-024	26.87	26.87	0.00	26.87	0.00	26.87	0.00	26.87	0.00	26.87	0.00
i080-025	27.62	27.62	0.00	27.62	0.00	27.62	0.00	27.62	0.00	27.62	0.00
i080-031	18.73	18.73	0.00	18.73	0.00	18.73	0.00	18.73	0.00	6.50	12.23
i080-032	12.88	9.00	3.88	5.27	7.61	12.88	0.00	12.88	0.00	4.26	8.62
i080-033	5.35	23.08	-17.73	11.59	-6.24	22.74	-17.39	22.74	-17.39	10.70	-5.35
i080-034	6.93	6.46	0.47	6.46	0.47	6.93	0.00	5.33	1.60	5.09	1.84
i080-035	16.70	10.53	6.17	11.17	5.53	16.70	0.00	5.91	10.79	5.42	11.28
i080-041	17.24	1.57	15.67	1.57	15.67	1.57	15.67	17.32	-0.08	17.32	-0.08
i080-042	22.61	21.76	0.85	8.08	14.53	22.61	0.00	20.20	2.41	18.88	3.73
i080-043	21.31	17.61	3.70	16.99	4.32	17.61	3.70	13.13	8.18	21.31	0.00
i080-044	0.81	0.81	0.00	0.81	0.00	0.81	0.00	0.81	0.00	0.81	0.00
i080-045	18.47	11.22	7.25	6.18	12.29	11.22	7.25	11.22	7.25	11.22	7.25
i080-101	7.94	7.94	0.00	7.78	0.16	7.94	0.00	0.00	7.94	0.00	7.94
i080-102	11.90	4.33	7.57	4.20	7.70	4.20	7.70	3.83	8.07	4.20	7.70
i080-103	6.53	6.72	-0.19	6.53	0.00	6.53	0.00	6.53	0.00	2.96	3.57
i080-104	7.84	7.72	0.12	7.72	0.12	7.72	0.12	7.80	0.04	7.60	0.24
i080-105	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-111	29.64	10.97	18.67	11.07	18.57	0.44	29.20	11.07	18.57	0.15	29.49
i080-112	24.67	9.60	15.07	6.21	18.46	14.59	10.08	6.15	18.52	9.76	14.91
i080-113	15.98	11.15	4.83	11.15	4.83	11.15	4.83	11.15	4.83	9.71	6.27
i080-114	26.28	21.06	5.22	21.06	5.22	10.87	15.41	9.55	16.73	15.09	11.19
i080-115	31.75	11.35	20.40	11.35	20.40	11.35	20.40	16.38	15.37	10.44	21.31
i080-121	31.58	31.58	0.00	31.58	0.00	31.58	0.00	31.58	0.00	31.58	0.00
i080-122	31.97	31.97	0.00	31.97	0.00	31.97	0.00	31.97	0.00	31.97	0.00
i080-123	31.36	31.36	0.00	31.36	0.00	31.36	0.00	31.36	0.00	31.36	0.00
i080-124	30.68	30.68	0.00	30.68	0.00	30.68	0.00	30.68	0.00	30.68	0.00
i080-125	31.11	31.11	0.00	31.11	0.00	31.11	0.00	31.11	0.00	31.11	0.00
i080-131	8.01	3.81	4.20	8.01	0.00	3.81	4.20	4.03	3.98	8.01	0.00
i080-132	8.90	9.50	-0.60	8.90	0.00	9.50	-0.60	5.14	3.76	4.72	4.18
i080-133	14.02	0.84	13.18	0.84	13.18	0.84	13.18	0.00	14.02	0.00	14.02
i080-134	6.62	6.62	0.00	6.62	0.00	6.62	0.00	5.36	1.26	6.62	0.00
i080-135	3.81	3.81	0.00	3.81	0.00	3.81	0.00	3.81	0.00	3.81	0.00
i080-141	15.10	5.59	9.51	5.59	9.51	5.59	9.51	9.68	5.42	15.10	0.00
i080-142	25.31	16.16	9.15	5.04	20.27	16.16	9.15	18.85	6.46	17.68	7.63
i080-143	16.19	0.74	15.45	0.74	15.45	1.47	14.72	5.72	10.47	16.24	-0.05
i080-144	22.57	14.00	8.57	12.92	9.65	12.98	9.59	12.36	10.21	21.33	1.24
i080-145	22.64	2.67	19.97	7.55	15.09	2.67	19.97	2.67	19.97	14.76	7.88

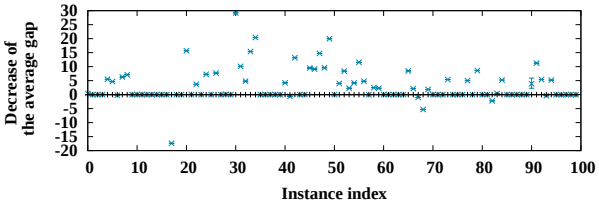
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
i080-201	4.20	4.20	0.00	4.20	0.00	4.20	0.00	4.20	0.00	4.20	0.00
i080-202	6.52	2.54	3.98	0.56	5.96	2.54	3.98	2.65	3.87	0.52	6.00
i080-203	14.98	6.57	8.41	10.83	4.15	6.57	8.41	4.35	10.63	4.41	10.57
i080-204	5.88	5.97	-0.09	3.54	2.34	3.58	2.30	3.54	2.34	3.54	2.34
i080-205	8.37	4.16	4.21	4.54	3.83	4.18	4.19	4.16	4.21	4.23	4.14
i080-211	17.87	6.77	11.10	6.55	11.32	6.33	11.54	6.33	11.54	10.66	7.21
i080-212	20.34	15.61	4.73	12.40	7.94	15.56	4.78	11.78	8.56	10.72	9.62
i080-213	16.67	14.46	2.21	14.38	2.29	16.69	-0.02	4.16	12.51	4.05	12.62
i080-214	22.68	22.68	0.00	22.68	0.00	20.19	2.49	5.38	17.30	4.58	18.10
i080-215	15.54	8.58	6.96	8.42	7.12	13.26	2.28	12.33	3.21	15.54	0.00
i080-221	38.89	38.89	0.00	38.89	0.00	38.89	0.00	38.89	0.00	38.89	0.00
i080-222	38.81	38.81	0.00	38.81	0.00	38.81	0.00	38.81	0.00	38.81	0.00
i080-223	39.01	39.01	0.00	39.01	0.00	39.01	0.00	39.01	0.00	39.01	0.00
i080-224	38.46	38.46	0.00	38.46	0.00	38.46	0.00	38.46	0.00	38.46	0.00
i080-225	38.44	38.44	0.00	38.44	0.00	38.44	0.00	38.44	0.00	38.44	0.00
i080-231	20.28	7.40	12.88	4.94	15.34	11.81	8.47	6.78	13.50	8.61	11.67
i080-232	8.81	6.67	2.14	6.60	2.21	6.69	2.12	6.55	2.26	8.88	-0.07
i080-233	9.98	10.78	-0.80	9.98	0.00	10.98	-1.00	8.40	1.58	8.04	1.94
i080-234	8.73	9.12	-0.39	8.73	0.00	14.04	-5.31	4.87	3.86	6.79	1.94
i080-235	7.93	4.41	3.52	6.08	1.85	6.08	1.85	6.40	1.53	7.93	0.00
i080-241	26.14	26.31	-0.17	26.14	0.00	26.14	0.00	7.55	18.59	16.85	9.29
i080-242	17.26	16.08	1.18	16.08	1.18	17.26	0.00	5.49	11.77	12.90	4.36
i080-243	22.37	14.48	7.89	14.48	7.89	22.37	0.00	5.07	17.30	17.53	4.84
i080-244	16.62	11.22	5.40	11.22	5.40	11.22	5.40	9.43	7.19	15.26	1.36
i080-245	28.50	28.58	-0.08	28.55	-0.05	28.53	-0.03	10.64	17.86	12.43	16.07
i080-301	9.26	9.26	0.00	9.26	0.00	9.26	0.00	4.13	5.13	9.26	0.00
i080-302	6.83	6.83	0.00	5.59	1.24	6.83	0.00	3.50	3.33	3.94	2.89
i080-303	10.30	3.58	6.72	5.30	5.00	5.30	5.00	1.90	8.40	3.74	6.56
i080-304	7.00	7.00	0.00	5.62	1.38	7.00	0.00	7.18	-0.18	7.04	-0.04
i080-305	15.73	7.45	8.28	10.40	5.33	7.13	8.60	4.70	11.03	4.10	11.63
i080-311	19.39	20.00	-0.61	19.39	0.00	19.39	0.00	7.51	11.88	9.82	9.57
i080-312	22.17	20.84	1.33	11.54	10.63	22.17	0.00	10.32	11.85	12.22	9.95
i080-313	17.01	15.10	1.91	14.66	2.35	19.23	-2.22	8.21	8.80	10.20	6.81
i080-314	8.75	5.14	3.61	5.05	3.70	8.36	0.39	5.76	2.99	7.40	1.35
i080-315	16.91	11.91	5.00	11.50	5.41	11.71	5.20	7.51	9.40	10.25	6.66
i080-321	40.56	40.56	0.00	40.56	0.00	40.56	0.00	40.56	0.00	40.56	0.00
i080-322	39.85	39.85	0.00	39.85	0.00	39.85	0.00	39.85	0.00	39.85	0.00
i080-323	40.02	40.02	0.00	40.02	0.00	40.02	0.00	40.02	0.00	40.02	0.00
i080-324	40.31	40.31	0.00	40.31	0.00	40.31	0.00	40.31	0.00	40.31	0.00
i080-325	41.03	41.03	0.00	41.03	0.00	41.03	0.00	41.03	0.00	41.03	0.00
i080-331	13.61	10.33	3.28	12.13	1.48	9.73	3.88	3.65	9.96	5.07	8.54
i080-332	17.23	6.08	11.15	13.82	3.41	5.97	11.26	7.91	9.32	9.46	7.77
i080-333	13.99	10.57	3.42	12.27	1.72	8.59	5.40	7.02	6.97	10.50	3.49
i080-334	11.72	4.14	7.58	11.72	0.00	12.12	-0.40	4.22	7.50	8.43	3.29
i080-335	11.00	7.43	3.57	7.29	3.71	5.83	5.17	5.83	5.17	5.71	5.29
i080-341	19.31	19.38	-0.07	17.40	1.91	19.31	0.00	5.69	13.62	11.95	7.36
i080-342	29.49	29.74	-0.25	29.54	-0.05	29.54	-0.05	5.28	24.21	11.51	17.98
i080-343	32.85	33.33	-0.48	32.95	-0.10	32.88	-0.03	7.85	25.00	9.33	23.52
i080-344	31.83	32.02	-0.19	31.95	-0.12	31.83	0.00	4.59	27.24	4.27	27.56
i080-345	15.53	24.83	-9.30	15.53	0.00	15.55	-0.02	9.49	6.04	11.63	3.90
ave	18.58	15.45	3.13	15.02	3.56	15.64	2.94	12.60	5.98	13.52	5.06



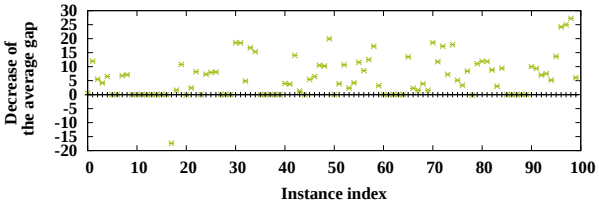
(a) degree centrality



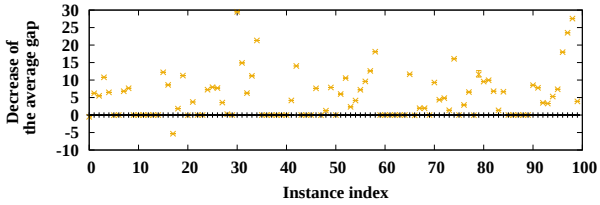
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality



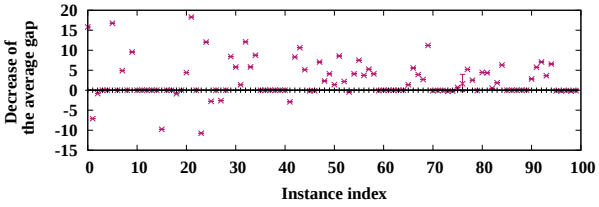
(e) edge betweenness centrality

Figure 3.24: Decrease of the average gap (DNH, I080)

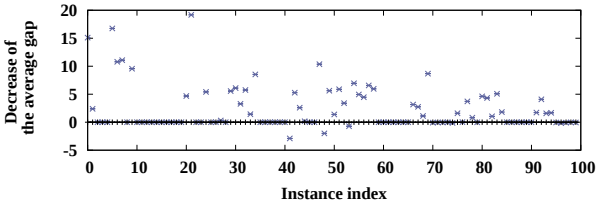
Table 3.17: Gaps [%] (DNH, I160)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
i160-001	24.22	8.43	15.79	9.12	15.10	12.81	11.41	4.50	19.72	4.22	20.00
i160-002	2.49	9.59	-7.10	0.09	2.40	9.59	-7.10	0.09	2.40	0.09	2.40
i160-003	8.45	9.27	-0.82	8.45	0.00	8.45	0.00	5.92	2.53	5.62	2.83
i160-004	5.27	5.27	0.00	5.27	0.00	5.27	0.00	5.27	0.00	5.27	0.00
i160-005	8.26	8.26	0.00	8.26	0.00	8.26	0.00	8.26	0.00	8.18	0.08
i160-011	29.64	12.88	16.76	12.88	16.76	12.88	16.76	11.69	17.95	16.04	13.60
i160-012	24.69	24.69	0.00	13.89	10.80	24.69	0.00	6.29	18.40	17.49	7.20
i160-013	29.14	24.26	4.88	18.06	11.08	11.86	17.28	11.86	17.28	11.86	17.28
i160-014	22.83	22.83	0.00	22.83	0.00	22.83	0.00	22.83	0.00	18.95	3.88
i160-015	28.45	18.89	9.56	18.89	9.56	24.43	4.02	12.44	16.01	17.99	10.46
i160-021	30.47	30.47	0.00	30.47	0.00	30.47	0.00	30.47	0.00	30.47	0.00
i160-022	29.74	29.74	0.00	29.74	0.00	29.74	0.00	29.74	0.00	29.74	0.00
i160-023	29.90	29.90	0.00	29.90	0.00	29.90	0.00	29.90	0.00	29.90	0.00
i160-024	28.96	28.96	0.00	28.96	0.00	28.96	0.00	28.96	0.00	28.96	0.00
i160-025	28.99	28.99	0.00	28.99	0.00	28.99	0.00	28.99	0.00	28.99	0.00
i160-031	9.35	19.12	-9.77	9.35	0.00	9.35	0.00	9.45	-0.10	9.35	0.00
i160-032	6.22	6.22	0.00	6.22	0.00	6.22	0.00	6.22	0.00	1.42	4.80
i160-033	18.94	18.94	0.00	18.94	0.00	18.94	0.00	14.14	4.80	9.71	9.23
i160-034	8.83	9.70	-0.87	8.83	0.00	0.00	8.83	9.70	-0.87	0.00	8.83
i160-035	10.08	10.08	0.00	10.08	0.00	10.08	0.00	10.08	0.00	10.08	0.00
i160-041	17.20	12.78	4.42	12.52	4.68	17.20	0.00	17.20	0.00	10.31	6.89
i160-042	32.23	13.93	18.30	13.06	19.17	13.06	19.17	10.77	21.46	17.03	15.20
i160-043	14.91	14.91	0.00	14.91	0.00	14.91	0.00	12.78	2.13	14.91	0.00
i160-044	12.04	22.80	-10.76	12.04	0.00	12.04	0.00	12.04	0.00	12.04	0.00
i160-045	18.60	6.56	12.04	13.19	5.41	6.69	11.91	6.69	11.91	12.55	6.05
i160-101	8.06	10.83	-2.77	8.06	0.00	5.44	2.62	5.44	2.62	5.44	2.62
i160-102	11.13	11.13	0.00	11.13	0.00	11.13	0.00	5.12	6.01	2.72	8.41
i160-103	5.99	8.60	-2.61	5.68	0.31	11.21	-5.22	6.20	-0.21	5.99	0.00
i160-104	2.54	2.54	0.00	2.54	0.00	2.54	0.00	2.54	0.00	2.54	0.00
i160-105	17.01	8.59	8.42	11.45	5.56	17.01	0.00	8.59	8.42	11.20	5.81
i160-111	17.25	11.47	5.78	11.15	6.10	14.29	2.96	11.01	6.24	17.25	0.00
i160-112	19.66	18.30	1.36	16.38	3.28	17.31	2.35	16.59	3.07	15.53	4.13
i160-113	26.34	14.24	12.10	20.59	5.75	17.38	8.96	9.07	17.27	12.14	14.20
i160-114	18.23	12.41	5.82	16.79	1.44	16.53	1.70	8.80	9.43	17.93	0.30
i160-115	13.07	4.32	8.75	4.53	8.54	4.32	8.75	7.46	5.61	16.79	-3.72
i160-121	35.84	35.84	0.00	35.84	0.00	35.84	0.00	35.84	0.00	35.84	0.00
i160-122	36.46	36.46	0.00	36.46	0.00	36.46	0.00	36.46	0.00	36.46	0.00
i160-123	36.31	36.31	0.00	36.31	0.00	36.31	0.00	36.31	0.00	36.31	0.00
i160-124	37.76	37.76	0.00	37.76	0.00	37.76	0.00	37.76	0.00	37.76	0.00
i160-125	36.88	36.88	0.00	36.88	0.00	36.88	0.00	36.88	0.00	36.88	0.00
i160-131	3.13	3.13	0.00	3.13	0.00	3.13	0.00	3.13	0.00	3.16	-0.03
i160-132	12.09	14.99	-2.90	14.99	-2.90	12.09	0.00	6.14	5.95	11.88	0.21
i160-133	22.12	13.81	8.31	16.85	5.27	16.85	5.27	7.70	14.42	12.86	9.26
i160-134	14.15	3.52	10.63	11.56	2.59	3.52	10.63	8.85	5.30	11.33	2.82
i160-135	13.46	8.34	5.12	13.29	0.17	10.20	3.26	6.11	7.35	5.81	7.65
i160-141	28.09	28.25	-0.16	28.09	0.00	28.09	0.00	17.34	10.75	24.13	3.96
i160-142	27.83	27.95	-0.12	27.83	0.00	27.83	0.00	13.78	14.05	14.95	12.88
i160-143	23.00	15.96	7.04	12.63	10.37	10.83	12.17	13.77	9.23	20.41	2.59
i160-144	11.66	9.36	2.30	13.66	-2.00	9.40	2.26	11.12	0.54	11.66	0.00
i160-145	17.34	13.23	4.11	11.71	5.63	13.23	4.11	17.34	0.00	17.34	0.00

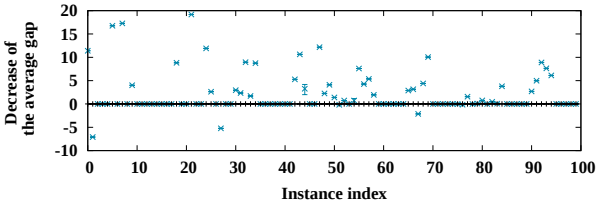
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
i160-201	8.97	7.58	1.39	7.58	1.39	7.58	1.39	7.58	1.39	4.19	4.78
i160-202	22.08	13.51	8.57	16.20	5.88	22.25	-0.17	10.85	11.23	4.49	17.59
i160-203	14.65	12.47	2.18	11.25	3.40	13.92	0.73	6.78	7.87	11.50	3.15
i160-204	9.85	10.30	-0.45	10.57	-0.72	9.85	0.00	8.77	1.08	9.85	0.00
i160-205	14.39	10.28	4.11	7.44	6.95	13.65	0.74	3.13	11.26	4.42	9.97
i160-211	20.10	12.61	7.49	15.15	4.95	12.50	7.60	10.73	9.37	13.72	6.38
i160-212	15.10	11.38	3.72	10.63	4.47	10.83	4.27	12.28	2.82	12.64	2.46
i160-213	18.74	13.48	5.26	12.17	6.57	13.37	5.37	13.53	5.21	13.37	5.37
i160-214	18.90	14.81	4.09	12.94	5.96	16.96	1.94	12.99	5.91	14.90	4.00
i160-215	22.80	22.89	-0.09	22.80	0.00	22.80	0.00	12.18	10.62	16.46	6.34
i160-221	41.15	41.15	0.00	41.15	0.00	41.15	0.00	41.15	0.00	41.15	0.00
i160-222	42.09	42.09	0.00	42.09	0.00	42.09	0.00	42.09	0.00	42.09	0.00
i160-223	41.35	41.35	0.00	41.35	0.00	41.35	0.00	41.35	0.00	41.35	0.00
i160-224	41.96	41.96	0.00	41.96	0.00	41.96	0.00	41.96	0.00	41.96	0.00
i160-225	40.67	40.67	0.00	40.67	0.00	40.67	0.00	40.67	0.00	40.67	0.00
i160-231	11.62	10.25	1.37	11.62	0.00	8.74	2.88	11.62	0.00	11.62	0.00
i160-232	17.93	12.37	5.56	14.78	3.15	14.78	3.15	6.57	11.36	10.25	7.68
i160-233	18.09	14.18	3.91	15.35	2.74	20.22	-2.13	9.42	8.67	10.25	7.84
i160-234	7.26	4.56	2.70	6.14	1.12	2.87	4.39	1.49	5.77	1.62	5.64
i160-235	24.56	13.35	11.21	15.89	8.67	14.52	10.04	13.60	10.96	12.08	12.48
i160-241	34.17	34.29	-0.12	34.25	-0.08	34.17	0.00	7.75	26.42	16.54	17.63
i160-242	33.45	33.51	-0.06	33.49	-0.04	33.45	0.00	13.05	20.40	20.02	13.43
i160-243	34.91	34.97	-0.06	34.95	-0.04	34.91	0.00	4.89	30.02	17.09	17.82
i160-244	33.59	33.87	-0.28	33.59	0.00	33.59	0.00	11.01	22.58	20.06	13.53
i160-245	33.93	34.15	-0.22	34.05	-0.12	33.93	0.00	9.74	24.19	15.76	18.17
i160-301	10.93	10.23	0.70	9.33	1.60	10.94	-0.01	5.76	5.17	5.70	5.23
i160-302	8.87	7.24	1.63	8.85	0.02	9.03	-0.16	5.63	3.24	8.20	0.67
i160-303	8.75	3.55	5.20	5.03	3.72	7.18	1.57	2.66	6.09	2.73	6.02
i160-304	6.27	3.77	2.50	5.46	0.81	6.29	-0.02	3.70	2.57	8.01	-1.74
i160-305	5.70	5.79	-0.09	5.71	-0.01	5.62	0.08	4.39	1.31	3.14	2.56
i160-311	23.05	18.61	4.44	18.43	4.62	22.29	0.76	9.81	13.24	12.62	10.43
i160-312	26.45	22.09	4.36	22.12	4.33	26.45	0.00	10.03	16.42	9.75	16.70
i160-313	21.64	21.17	0.47	20.59	1.05	21.17	0.47	8.46	13.18	8.54	13.10
i160-314	26.40	24.55	1.85	21.33	5.07	26.33	0.07	10.12	16.28	14.07	12.33
i160-315	24.02	17.72	6.30	22.20	1.82	20.23	3.79	7.61	16.41	11.51	12.51
i160-321	43.27	43.27	0.00	43.27	0.00	43.27	0.00	43.27	0.00	43.27	0.00
i160-322	43.70	43.70	0.00	43.70	0.00	43.70	0.00	43.70	0.00	43.70	0.00
i160-323	43.38	43.38	0.00	43.38	0.00	43.38	0.00	43.38	0.00	43.38	0.00
i160-324	43.42	43.42	0.00	43.42	0.00	43.42	0.00	43.42	0.00	43.42	0.00
i160-325	43.59	43.59	0.00	43.59	0.00	43.59	0.00	43.59	0.00	43.59	0.00
i160-331	10.58	7.77	2.81	10.58	0.00	7.89	2.69	7.43	3.15	7.54	3.04
i160-332	15.07	9.32	5.75	13.34	1.73	10.09	4.98	6.20	8.87	7.56	7.51
i160-333	14.75	7.68	7.07	10.67	4.08	5.86	8.89	4.34	10.41	5.44	9.31
i160-334	13.08	9.44	3.64	11.49	1.59	5.45	7.63	8.11	4.97	5.75	7.33
i160-335	12.96	6.40	6.56	11.29	1.67	6.84	6.12	4.89	8.07	7.15	5.81
i160-341	37.13	37.28	-0.15	37.17	-0.04	37.13	0.00	9.51	27.62	18.38	18.75
i160-342	37.28	37.49	-0.21	37.43	-0.15	37.28	0.00	5.21	32.07	17.98	19.30
i160-343	38.82	38.92	-0.10	38.89	-0.07	38.83	-0.01	6.24	32.58	17.00	21.82
i160-344	38.00	38.24	-0.24	38.03	-0.03	38.00	0.00	5.27	32.73	20.22	17.78
i160-345	37.38	37.48	-0.10	37.43	-0.05	37.38	0.00	6.79	30.59	16.15	21.23
ave	22.32	19.96	2.36	20.01	2.31	20.13	2.19	14.60	7.72	16.48	5.84



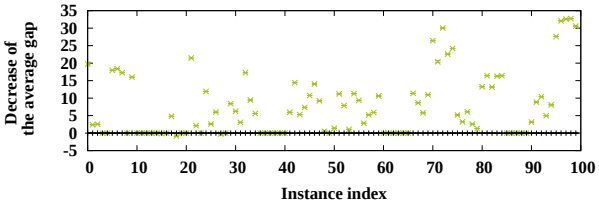
(a) degree centrality



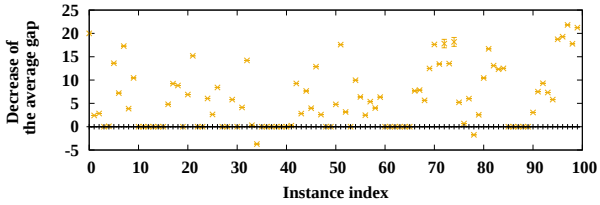
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality



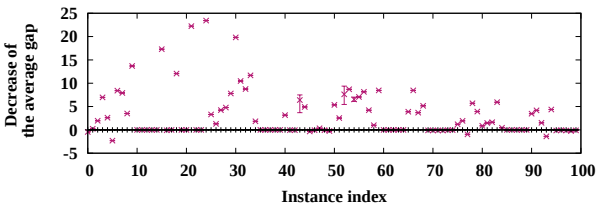
(e) edge betweenness centrality

Figure 3.25: Decrease of the average gap (DNH, I160)

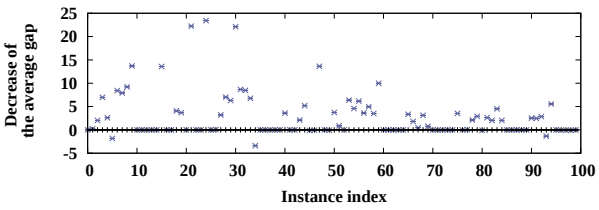
Table 3.18: Gaps [%] (DNH, I320)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
i320-001	11.94	12.39	-0.45	11.94	0.00	11.94	0.00	4.30	7.64	11.38	0.56
i320-002	8.04	7.80	0.24	7.80	0.24	8.04	0.00	4.36	3.68	1.72	6.32
i320-003	22.07	20.12	1.95	20.05	2.02	22.07	0.00	10.03	12.04	12.99	9.08
i320-004	14.22	7.26	6.96	7.26	6.96	7.26	6.96	7.06	7.16	7.06	7.16
i320-005	17.02	14.41	2.61	14.41	2.61	14.41	2.61	14.41	2.61	10.36	6.66
i320-011	17.95	20.26	-2.31	19.78	-1.83	20.26	-2.31	15.78	2.17	8.57	9.38
i320-012	27.54	19.13	8.41	19.13	8.41	9.35	18.19	5.16	22.38	23.54	4.00
i320-013	24.57	16.65	7.92	16.65	7.92	17.23	7.34	24.57	0.00	24.57	0.00
i320-014	15.24	11.74	3.50	6.02	9.22	6.02	9.22	15.24	0.00	15.14	0.10
i320-015	34.48	20.79	13.69	20.79	13.69	34.48	0.00	16.22	18.26	26.81	7.67
i320-021	31.68	31.68	0.00	31.68	0.00	31.68	0.00	31.68	0.00	31.68	0.00
i320-022	31.57	31.57	0.00	31.57	0.00	31.57	0.00	31.57	0.00	31.57	0.00
i320-023	33.70	33.70	0.00	33.70	0.00	33.70	0.00	33.70	0.00	33.70	0.00
i320-024	31.81	31.81	0.00	31.81	0.00	31.81	0.00	31.81	0.00	31.81	0.00
i320-025	32.00	32.00	0.00	32.00	0.00	32.00	0.00	32.00	0.00	32.00	0.00
i320-031	22.71	5.42	17.29	9.13	13.58	15.45	7.26	8.45	14.26	8.45	14.26
i320-032	8.19	8.27	-0.08	8.19	0.00	8.38	-0.19	8.19	0.00	3.43	4.76
i320-033	15.96	15.96	0.00	15.96	0.00	15.96	0.00	15.96	0.00	15.46	0.50
i320-034	19.83	7.77	12.06	15.75	4.08	15.83	4.00	4.09	15.74	11.74	8.09
i320-035	15.72	15.72	0.00	12.03	3.69	15.72	0.00	11.91	3.81	14.47	1.25
i320-041	9.20	9.20	0.00	9.20	0.00	5.16	4.04	9.20	0.00	9.20	0.00
i320-042	34.24	12.01	22.23	12.01	22.23	24.33	9.91	21.94	12.30	31.87	2.37
i320-043	13.81	13.81	0.00	13.81	0.00	13.81	0.00	6.38	7.43	13.81	0.00
i320-044	23.74	23.74	0.00	23.74	0.00	23.74	0.00	19.57	4.17	22.78	0.96
i320-045	32.09	8.66	23.43	8.66	23.43	9.25	22.84	19.81	12.28	20.82	11.27
i320-101	14.01	10.71	3.30	14.01	0.00	14.01	0.00	12.33	1.68	8.98	5.03
i320-102	7.29	5.99	1.30	7.22	0.07	7.29	0.00	5.49	1.80	5.76	1.53
i320-103	13.16	8.93	4.23	9.97	3.19	9.97	3.19	5.24	7.92	3.59	9.57
i320-104	21.83	17.04	4.79	14.82	7.01	14.82	7.01	10.24	11.59	7.01	14.82
i320-105	13.23	5.43	7.80	6.93	6.30	6.92	6.31	5.30	7.93	6.68	6.55
i320-111	31.94	12.12	19.82	9.83	22.11	14.77	17.17	15.02	16.92	27.66	4.28
i320-112	25.92	15.45	10.47	17.26	8.66	16.41	9.51	19.51	6.41	20.77	5.15
i320-113	21.43	12.68	8.75	12.96	8.47	7.82	13.61	17.81	3.62	21.43	0.00
i320-114	19.10	7.41	11.69	12.33	6.77	11.94	7.16	16.74	2.36	14.57	4.53
i320-115	20.58	18.74	1.84	23.97	-3.39	20.86	-0.28	20.58	0.00	20.58	0.00
i320-121	39.51	39.51	0.00	39.51	0.00	39.51	0.00	39.51	0.00	39.51	0.00
i320-122	40.98	40.98	0.00	40.98	0.00	40.98	0.00	40.98	0.00	40.98	0.00
i320-123	39.47	39.47	0.00	39.47	0.00	39.47	0.00	39.47	0.00	39.47	0.00
i320-124	40.02	40.02	0.00	40.02	0.00	40.02	0.00	40.02	0.00	40.02	0.00
i320-125	39.70	39.70	0.00	39.70	0.00	39.70	0.00	39.70	0.00	39.70	0.00
i320-131	18.71	15.55	3.16	15.11	3.60	11.74	6.97	9.67	9.04	9.67	9.04
i320-132	20.92	20.98	-0.06	20.92	0.00	13.84	7.08	20.98	-0.06	9.90	11.02
i320-133	8.18	8.25	-0.07	8.18	0.00	8.25	-0.07	8.18	0.00	2.40	5.78
i320-134	20.31	13.92	6.39	18.21	2.10	16.24	4.07	11.17	9.14	5.65	14.66
i320-135	20.05	15.14	4.91	14.86	5.19	14.92	5.13	13.87	6.18	11.61	8.44
i320-141	32.09	32.42	-0.33	32.20	-0.11	32.20	-0.11	19.02	13.07	21.24	10.85
i320-142	33.19	33.25	-0.06	33.25	-0.06	33.19	0.00	12.78	20.41	33.19	0.00
i320-143	30.33	29.99	0.34	16.71	13.62	30.13	0.20	11.23	19.10	25.72	4.61
i320-144	34.82	34.88	-0.06	34.82	0.00	34.82	0.00	8.17	26.65	19.70	15.12
i320-145	31.85	32.10	-0.25	31.91	-0.06	31.88	-0.03	8.00	23.85	16.58	15.27

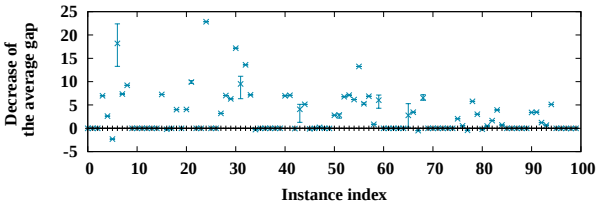
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
i320-201	12.95	7.60	5.35	9.23	3.72	10.15	2.80	6.47	6.48	7.29	5.66
i320-202	12.71	10.18	2.53	11.85	0.86	9.86	2.85	6.71	6.00	8.48	4.23
i320-203	16.57	8.94	7.63	16.57	0.00	9.80	6.77	8.04	8.53	8.19	8.38
i320-204	16.51	7.81	8.70	10.13	6.38	9.40	7.11	7.56	8.95	4.74	11.77
i320-205	16.19	9.66	6.53	11.61	4.58	10.04	6.15	11.19	5.00	8.35	7.84
i320-211	24.36	17.30	7.06	18.20	6.16	11.10	13.26	14.79	9.57	15.24	9.12
i320-212	19.06	10.90	8.16	15.44	3.62	13.80	5.26	14.15	4.91	16.63	2.43
i320-213	21.04	16.82	4.22	16.08	4.96	14.17	6.87	13.17	7.87	18.76	2.28
i320-214	16.26	15.26	1.00	12.76	3.50	15.40	0.86	11.72	4.54	11.56	4.70
i320-215	23.63	15.17	8.46	13.64	9.99	17.60	6.03	16.19	7.44	17.80	5.83
i320-221	42.91	42.91	0.00	42.91	0.00	42.91	0.00	42.91	0.00	42.91	0.00
i320-222	42.88	42.88	0.00	42.88	0.00	42.88	0.00	42.88	0.00	42.88	0.00
i320-223	42.63	42.63	0.00	42.63	0.00	42.63	0.00	42.63	0.00	42.63	0.00
i320-224	42.65	42.65	0.00	42.65	0.00	42.65	0.00	42.65	0.00	42.65	0.00
i320-225	42.94	42.94	0.00	42.94	0.00	42.94	0.00	42.94	0.00	42.94	0.00
i320-231	11.57	7.68	3.89	8.22	3.35	8.80	2.77	9.89	1.68	10.65	0.92
i320-232	15.59	7.13	8.46	13.78	1.81	12.14	3.45	9.99	5.60	11.87	3.72
i320-233	19.47	15.79	3.68	19.00	0.47	19.97	-0.50	10.96	8.51	12.49	6.98
i320-234	18.09	12.93	5.16	14.97	3.12	11.56	6.53	10.51	7.58	9.74	8.35
i320-235	9.79	9.88	-0.09	9.01	0.78	9.79	0.00	9.38	0.41	7.81	1.98
i320-241	37.77	37.88	-0.11	37.77	0.00	37.77	0.00	10.54	27.23	25.43	12.34
i320-242	37.60	37.71	-0.11	37.64	-0.04	37.60	0.00	9.44	28.16	23.80	13.80
i320-243	37.31	37.44	-0.13	37.31	0.00	37.31	0.00	10.69	26.62	23.77	13.54
i320-244	37.21	37.27	-0.06	37.21	0.00	37.21	0.00	13.92	23.29	21.48	15.73
i320-245	37.87	37.92	-0.05	37.87	0.00	37.87	0.00	11.74	26.13	21.84	16.03
i320-301	10.70	9.50	1.20	7.17	3.53	8.66	2.04	7.65	3.05	8.30	2.40
i320-302	8.41	6.47	1.94	8.41	0.00	7.88	0.53	6.48	1.93	5.27	3.14
i320-303	5.15	6.10	-0.95	5.11	0.04	5.57	-0.42	4.79	0.36	4.79	0.36
i320-304	8.99	3.28	5.71	6.87	2.12	3.21	5.78	2.46	6.53	2.06	6.93
i320-305	9.26	5.33	3.93	6.36	2.90	6.24	3.02	3.47	5.79	4.22	5.04
i320-311	23.82	22.94	0.88	23.91	-0.09	24.01	-0.19	11.31	12.51	11.46	12.36
i320-312	29.81	28.34	1.47	27.19	2.62	29.32	0.49	11.05	18.76	13.38	16.43
i320-313	30.02	28.38	1.64	27.98	2.04	28.38	1.64	10.08	19.94	13.53	16.49
i320-314	26.28	20.35	5.93	21.77	4.51	22.36	3.92	9.52	16.76	11.21	15.07
i320-315	30.12	29.67	0.45	28.09	2.03	29.41	0.71	10.96	19.16	12.46	17.66
i320-321	45.28	45.28	0.00	45.28	0.00	45.28	0.00	45.28	0.00	45.28	0.00
i320-322	45.19	45.19	0.00	45.19	0.00	45.19	0.00	45.19	0.00	45.19	0.00
i320-323	45.41	45.41	0.00	45.41	0.00	45.41	0.00	45.41	0.00	45.41	0.00
i320-324	45.45	45.45	0.00	45.45	0.00	45.45	0.00	45.45	0.00	45.45	0.00
i320-325	45.41	45.41	0.00	45.41	0.00	45.41	0.00	45.41	0.00	45.41	0.00
i320-331	14.35	10.88	3.47	11.84	2.51	10.95	3.40	8.20	6.15	7.20	7.15
i320-332	11.60	7.42	4.18	9.14	2.46	8.14	3.46	7.00	4.60	6.89	4.71
i320-333	12.39	10.88	1.51	9.54	2.85	11.17	1.22	10.11	2.28	8.99	3.40
i320-334	9.74	11.14	-1.40	11.09	-1.35	9.07	0.67	8.97	0.77	9.39	0.35
i320-335	15.18	10.81	4.37	9.62	5.56	10.07	5.11	8.18	7.00	7.47	7.71
i320-341	41.45	41.59	-0.14	41.46	-0.01	41.46	-0.01	5.35	36.10	18.67	22.78
i320-342	42.11	42.18	-0.07	42.18	-0.07	42.12	-0.01	4.70	37.41	16.03	26.08
i320-343	41.38	41.52	-0.14	41.45	-0.07	41.40	-0.02	5.82	35.56	18.10	23.28
i320-344	41.59	41.83	-0.24	41.69	-0.10	41.64	-0.05	5.10	36.49	18.13	23.46
i320-345	41.98	42.07	-0.09	42.01	-0.03	41.98	0.00	5.03	36.95	19.95	22.03
ave	25.25	22.05	3.20	22.40	2.85	22.46	2.79	16.58	8.67	18.91	6.34



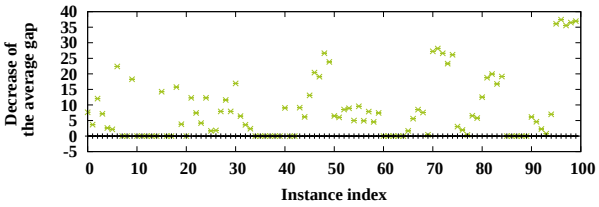
(a) degree centrality



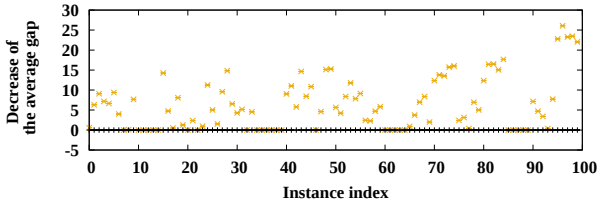
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality

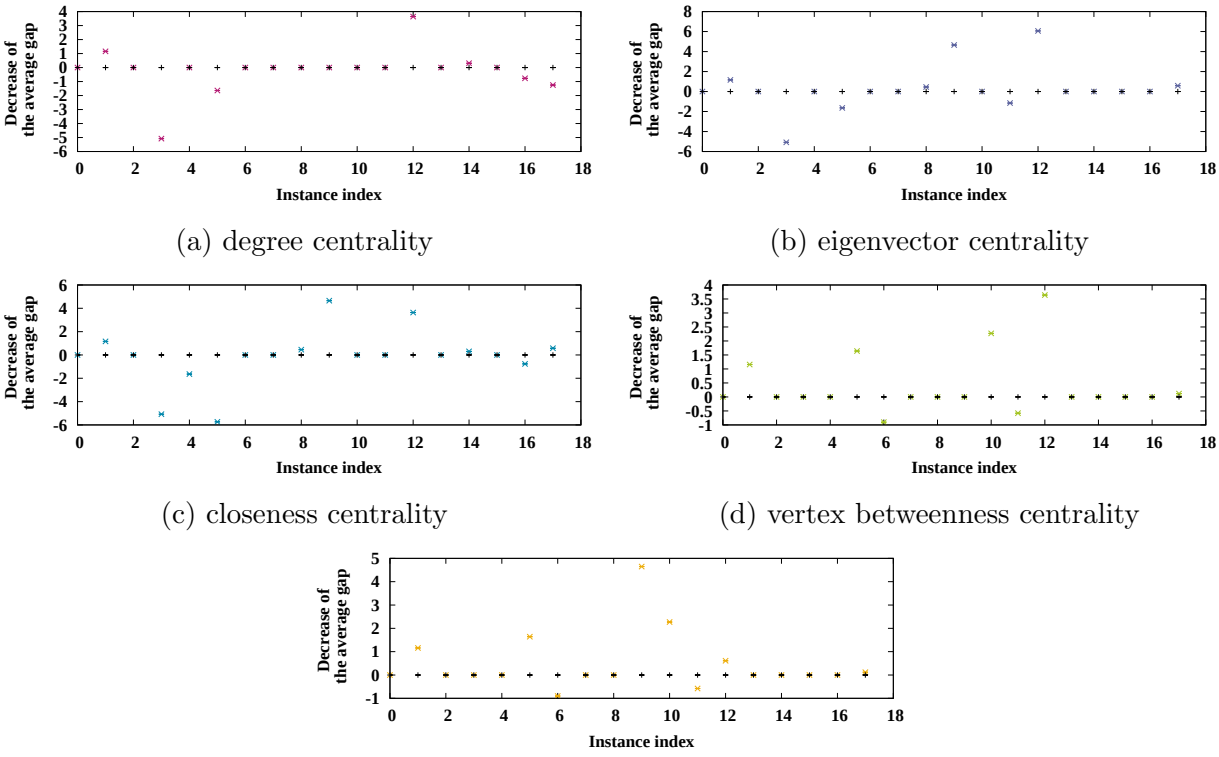


(e) edge betweenness centrality

Figure 3.26: Decrease of the average gap (DNH, I320)

Table 3.19: Gaps [%] (ADH, B)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
b01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
b02	1.16	0.00	1.16	0.00	1.16	0.00	1.16	0.00	1.16	0.00	1.16
b03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
b04	0.00	5.08	-5.08	5.08	-5.08	5.08	-5.08	0.00	0.00	0.00	0.00
b05	0.00	0.00	0.00	0.00	0.00	1.64	-1.64	0.00	0.00	0.00	0.00
b06	1.64	3.28	-1.64	3.28	-1.64	7.38	-5.74	0.00	1.64	0.00	1.64
b07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.90	-0.90	0.90	-0.90
b08	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
b09	0.45	0.45	0.00	0.00	0.45	0.00	0.45	0.45	0.00	0.45	0.00
b10	4.65	4.65	0.00	0.00	4.65	0.00	4.65	4.65	0.00	0.00	4.65
b11	2.27	2.27	0.00	2.27	0.00	2.27	0.00	0.00	2.27	0.00	2.27
b12	0.57	0.57	0.00	1.72	-1.15	0.57	0.00	1.15	-0.58	1.15	-0.58
b13	6.06	2.42	3.64	0.00	6.06	2.42	3.64	2.42	3.64	5.45	0.61
b14	0.43	0.43	0.00	0.43	0.00	0.43	0.00	0.43	0.00	0.43	0.00
b15	0.94	0.63	0.31	0.94	0.00	0.63	0.31	0.94	0.00	0.94	0.00
b16	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
b17	0.76	1.53	-0.77	0.76	0.00	1.53	-0.77	0.76	0.00	0.76	0.00
b18	1.04	2.29	-1.25	0.46	0.58	0.46	0.58	0.92	0.12	0.92	0.12
ave	1.11	1.31	-0.20	0.83	0.28	1.25	-0.14	0.70	0.41	0.61	0.50



(a) degree centrality

(b) eigenvector centrality

(c) closeness centrality

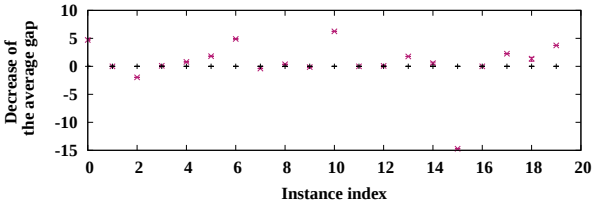
(d) vertex betweenness centrality

(e) edge betweenness centrality

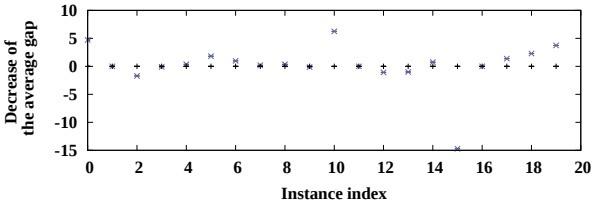
Figure 3.27: Decrease of the average gap (ADH, B)

Table 3.20: Gaps [%] (ADH, C)

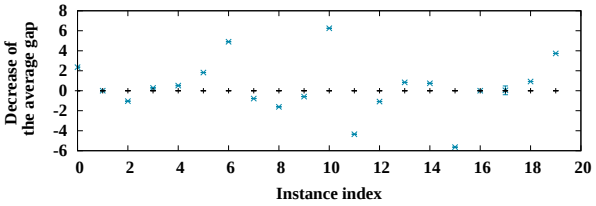
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
c01	4.71	0.00	4.71	0.00	4.71	2.35	2.36	0.00	4.71	4.71	0.00
c02	2.78	2.78	0.00	2.78	0.00	2.78	0.00	2.78	0.00	2.78	0.00
c03	1.88	3.85	-1.97	3.58	-1.70	2.92	-1.04	1.59	0.29	2.12	-0.24
c04	0.74	0.65	0.09	0.83	-0.09	0.46	0.28	0.46	0.28	0.46	0.28
c05	1.18	0.38	0.80	0.82	0.36	0.67	0.51	0.44	0.74	1.27	-0.09
c06	1.82	0.00	1.82	0.00	1.82	0.00	1.82	3.64	-1.82	3.64	-1.82
c07	4.90	0.00	4.90	3.92	0.98	0.00	4.90	3.92	0.98	4.90	0.00
c08	2.95	3.34	-0.39	2.75	0.20	3.73	-0.78	2.75	0.20	3.14	-0.19
c09	2.77	2.40	0.37	2.40	0.37	4.38	-1.61	2.69	0.08	3.54	-0.77
c10	0.87	1.01	-0.14	1.01	-0.14	1.46	-0.59	0.73	0.14	0.82	0.05
c11	6.25	0.00	6.25	0.00	6.25	0.00	6.25	3.12	3.13	3.12	3.13
c12	2.17	2.17	0.00	2.17	0.00	6.52	-4.35	2.17	0.00	2.17	0.00
c13	2.41	2.33	0.08	3.49	-1.08	3.49	-1.08	2.71	-0.30	4.65	-2.24
c14	3.94	2.17	1.77	4.95	-1.01	3.10	0.84	3.72	0.22	3.41	0.53
c15	1.46	0.89	0.57	0.72	0.74	0.72	0.74	0.90	0.56	1.26	0.20
c16	3.45	18.18	-14.73	18.18	-14.73	9.09	-5.64	0.00	3.45	9.09	-5.64
c17	5.56	5.56	0.00	5.56	0.00	5.56	0.00	0.00	5.56	16.67	-11.11
c18	9.33	7.08	2.25	7.96	1.37	9.29	0.04	9.73	-0.40	9.73	-0.40
c19	11.88	10.51	1.37	9.59	2.29	10.96	0.92	8.22	3.66	7.53	4.35
c20	8.22	4.49	3.73	4.49	3.73	4.49	3.73	4.49	3.73	4.49	3.73
ave	3.96	3.39	0.57	3.76	0.20	3.60	0.36	2.70	1.26	4.47	-0.51



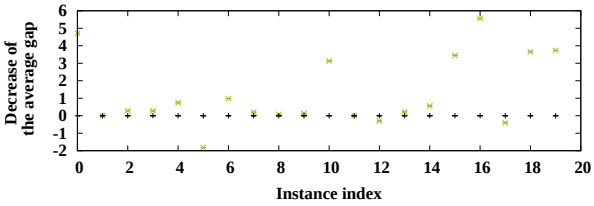
(a) degree centrality



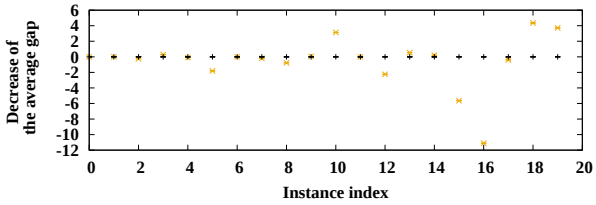
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality

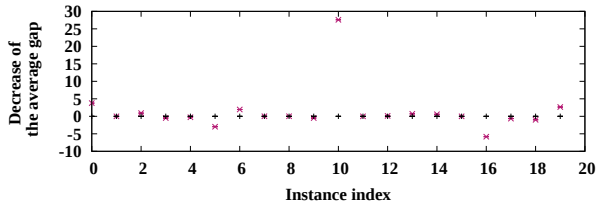


(e) edge betweenness centrality

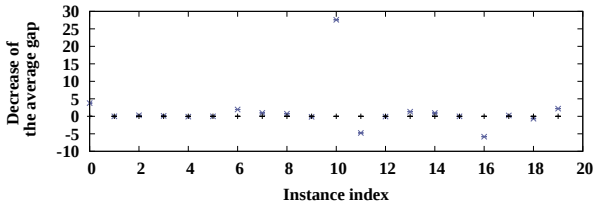
Figure 3.28: Decrease of the average gap (ADH, C)

Table 3.21: Gaps [%] (ADH, D)

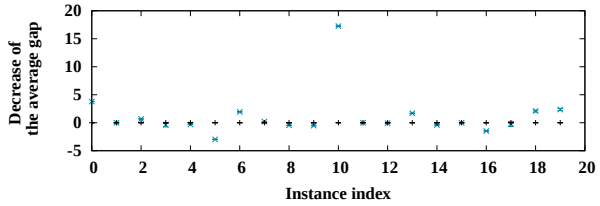
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
d01	7.55	3.77	3.78	3.77	3.78	3.77	3.78	3.77	3.78	3.77	3.78
d02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
d03	2.08	1.15	0.93	1.79	0.29	1.41	0.67	1.92	0.16	2.17	-0.09
d04	0.75	1.24	-0.49	0.62	0.13	1.11	-0.36	0.83	-0.08	0.52	0.23
d05	0.37	0.65	-0.28	0.46	-0.09	0.68	-0.31	0.55	-0.18	0.58	-0.21
d06	4.48	7.46	-2.98	4.48	0.00	7.46	-2.98	4.48	0.00	0.00	4.48
d07	1.94	0.00	1.94	0.00	1.94	0.00	1.94	0.00	1.94	3.88	-1.94
d08	2.74	2.71	0.03	1.77	0.97	2.52	0.22	2.33	0.41	1.96	0.78
d09	2.59	2.56	0.03	1.86	0.73	3.04	-0.45	1.59	1.00	2.07	0.52
d10	0.85	1.33	-0.48	1.00	-0.15	1.37	-0.52	0.85	0.00	0.95	-0.10
d11	27.59	0.00	27.59	0.00	27.59	10.34	17.25	3.45	24.14	3.45	24.14
d12	0.00	0.00	0.00	4.76	-4.76	0.00	0.00	0.00	0.00	0.00	0.00
d13	5.12	5.00	0.12	5.20	-0.08	5.20	-0.08	4.00	1.12	3.60	1.52
d14	3.71	3.00	0.71	2.40	1.31	2.02	1.69	2.40	1.31	2.25	1.46
d15	2.20	1.61	0.59	1.25	0.95	2.59	-0.39	1.52	0.68	1.79	0.41
d16	7.69	7.69	0.00	7.69	0.00	7.69	0.00	0.00	7.69	15.38	-7.69
d17	2.87	8.70	-5.83	8.70	-5.83	4.35	-1.48	4.35	-1.48	0.00	2.87
d18	7.42	8.07	-0.65	7.17	0.25	7.63	-0.21	8.07	-0.65	8.52	-1.10
d19	7.38	8.39	-1.01	8.06	-0.68	5.27	2.11	8.39	-1.01	6.13	1.25
d20	6.66	4.00	2.66	4.47	2.19	4.31	2.35	5.96	0.70	7.82	-1.16
ave	4.70	3.37	1.33	3.27	1.43	3.54	1.16	2.72	1.98	3.24	1.46



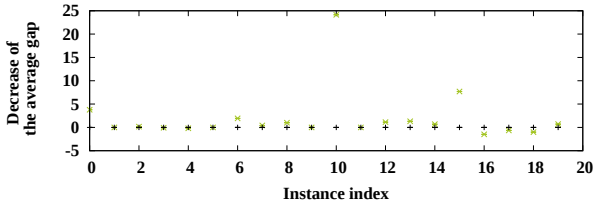
(a) degree centrality



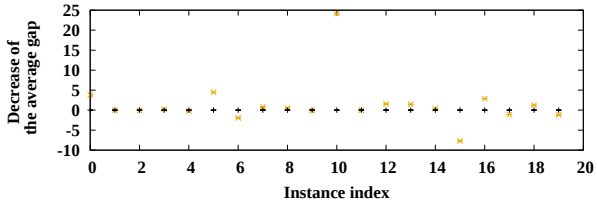
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality



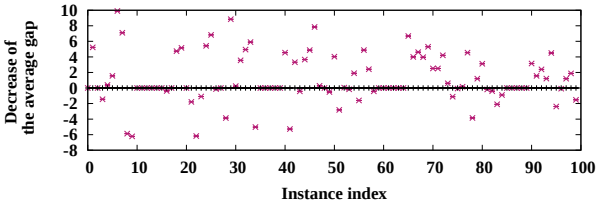
(e) edge betweenness centrality

Figure 3.29: Decrease of the average gap (ADH, D)

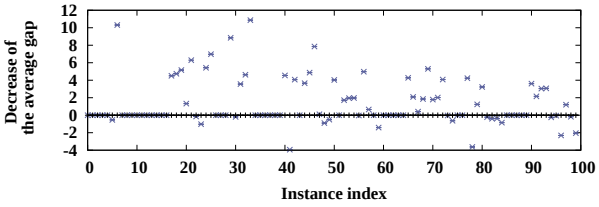
Table 3.22: Gaps [%] (ADH, I080)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
i080-001	0.00	0.00	0.00	0.00	0.00	0.00	0.00	4.70	-4.70	4.70	-4.70
i080-002	5.23	0.00	5.23	5.23	0.00	0.00	5.23	0.00	5.23	0.00	5.23
i080-003	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-004	5.20	6.65	-1.45	5.20	0.00	6.81	-1.61	5.20	0.00	5.20	0.00
i080-005	0.61	0.22	0.39	0.61	0.00	0.84	-0.23	0.22	0.39	0.22	0.39
i080-011	2.37	0.81	1.56	2.91	-0.54	2.50	-0.13	0.00	2.37	0.20	2.17
i080-012	10.78	0.88	9.90	0.47	10.31	0.47	10.31	0.34	10.44	0.34	10.44
i080-013	7.97	0.87	7.10	7.97	0.00	0.51	7.46	0.00	7.97	0.00	7.97
i080-014	0.00	5.87	-5.87	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-015	0.94	7.16	-6.22	0.94	0.00	6.49	-5.55	0.54	0.40	0.54	0.40
i080-021	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-022	0.08	0.08	0.00	0.08	0.00	0.08	0.00	0.08	0.00	0.08	0.00
i080-023	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-024	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-025	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-031	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-032	4.65	5.03	-0.38	4.65	0.00	4.50	0.15	0.05	4.60	0.05	4.60
i080-033	5.18	5.18	0.00	0.67	4.51	0.67	4.51	0.22	4.96	0.00	5.18
i080-034	4.98	0.24	4.74	0.24	4.74	0.24	4.74	5.33	-0.35	5.39	-0.41
i080-035	5.16	0.00	5.16	0.00	5.16	0.00	5.16	0.00	5.16	0.00	5.16
i080-041	1.57	1.57	0.00	0.24	1.33	0.24	1.33	0.24	1.33	0.24	1.33
i080-042	6.29	8.08	-1.79	0.00	6.29	6.29	0.00	5.36	0.93	5.13	1.16
i080-043	0.00	6.18	-6.18	0.15	-0.15	6.25	-6.25	6.25	-6.25	0.15	-0.15
i080-044	0.00	1.10	-1.10	1.02	-1.02	0.00	0.00	0.37	-0.37	0.37	-0.37
i080-045	5.50	0.08	5.42	0.08	5.42	0.08	5.42	0.08	5.42	0.08	5.42
i080-101	10.66	3.83	6.83	3.68	6.98	7.17	3.49	3.91	6.75	3.83	6.83
i080-102	0.08	0.21	-0.13	0.08	0.00	0.08	0.00	0.21	-0.13	4.41	-4.33
i080-103	2.46	2.46	0.00	2.46	0.00	2.46	0.00	2.46	0.00	2.46	0.00
i080-104	0.00	3.86	-3.86	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-105	8.85	0.00	8.85	0.00	8.85	4.09	4.76	4.09	4.76	8.31	0.54
i080-111	0.24	0.00	0.24	0.44	-0.20	5.02	-4.78	0.15	0.09	0.24	0.00
i080-112	4.30	0.74	3.56	0.74	3.56	0.16	4.14	3.66	0.64	5.15	-0.85
i080-113	10.67	5.73	4.94	6.05	4.62	5.73	4.94	5.31	5.36	0.37	10.30
i080-114	10.87	4.96	5.91	0.00	10.87	9.97	0.90	0.00	10.87	0.00	10.87
i080-115	0.11	5.14	-5.03	0.11	0.00	0.11	0.00	0.16	-0.05	0.16	-0.05
i080-121	0.32	0.32	0.00	0.32	0.00	0.32	0.00	0.32	0.00	0.32	0.00
i080-122	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-123	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-124	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-125	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-131	8.36	3.81	4.55	3.81	4.55	3.81	4.55	4.12	4.24	4.33	4.03
i080-132	0.00	5.28	-5.28	3.94	-3.94	5.28	-5.28	0.00	0.00	0.32	-0.32
i080-133	4.16	0.84	3.32	0.09	4.07	0.93	3.23	0.00	4.16	0.88	3.28
i080-134	0.97	1.40	-0.43	0.97	0.00	1.40	-0.43	0.43	0.54	0.43	0.54
i080-135	3.66	0.00	3.66	0.00	3.66	0.00	3.66	0.19	3.47	0.33	3.33
i080-141	4.87	0.00	4.87	0.00	4.87	0.00	4.87	0.00	4.87	0.00	4.87
i080-142	7.85	0.00	7.85	0.00	7.85	0.00	7.85	4.80	3.05	8.43	-0.58
i080-143	0.85	0.57	0.28	0.74	0.11	0.74	0.11	5.43	-4.58	0.06	0.79
i080-144	5.36	5.36	0.00	6.26	-0.90	5.36	0.00	5.36	0.00	5.36	0.00
i080-145	0.00	0.51	-0.51	0.51	-0.51	0.51	-0.51	0.40	-0.40	0.00	0.00

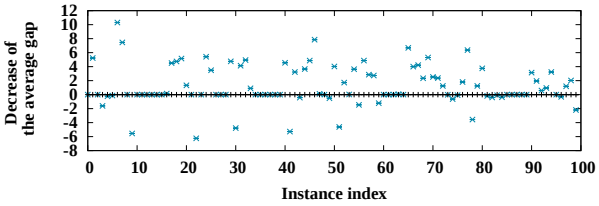
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
i080-201	4.03	0.00	4.03	0.00	4.03	0.00	4.03	0.00	4.03	0.00	4.03
i080-202	0.00	2.82	-2.82	0.00	0.00	4.62	-4.62	0.17	-0.17	0.24	-0.24
i080-203	4.13	4.09	0.04	2.41	1.72	2.41	1.72	3.96	0.17	0.33	3.80
i080-204	3.98	4.14	-0.16	2.03	1.95	3.96	0.02	3.58	0.40	2.03	1.95
i080-205	4.16	2.26	1.90	2.19	1.97	0.53	3.63	2.19	1.97	2.39	1.77
i080-211	2.59	4.19	-1.60	2.62	-0.03	4.05	-1.46	0.44	2.15	2.59	0.00
i080-212	5.28	0.41	4.87	0.30	4.98	0.41	4.87	0.41	4.87	0.41	4.87
i080-213	5.19	2.77	2.42	4.54	0.65	2.34	2.85	2.26	2.93	1.50	3.69
i080-214	3.40	3.83	-0.43	3.40	0.00	0.67	2.73	0.72	2.68	0.51	2.89
i080-215	1.96	1.96	0.00	3.37	-1.41	3.18	-1.22	1.96	0.00	1.96	0.00
i080-221	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-222	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-223	0.06	0.06	0.00	0.06	0.00	0.06	0.00	0.06	0.00	0.06	0.00
i080-224	0.06	0.06	0.00	0.06	0.00	0.00	0.06	0.06	0.00	0.06	0.00
i080-225	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-231	6.68	0.00	6.68	2.41	4.27	0.00	6.68	2.25	4.43	2.41	4.27
i080-232	6.50	2.50	4.00	4.41	2.09	2.50	4.00	6.50	0.00	2.07	4.43
i080-233	5.05	0.44	4.61	4.64	0.41	0.83	4.22	4.91	0.14	0.78	4.27
i080-234	6.46	2.50	3.96	4.61	1.85	4.12	2.34	0.05	6.41	2.18	4.28
i080-235	7.13	1.83	5.30	1.83	5.30	1.83	5.30	2.09	5.04	0.00	7.13
i080-241	3.17	0.65	2.52	1.38	1.79	0.65	2.52	0.00	3.17	0.00	3.17
i080-242	2.83	0.32	2.51	0.81	2.02	0.46	2.37	0.12	2.71	1.68	1.15
i080-243	4.20	0.00	4.20	0.12	4.08	2.96	1.24	1.87	2.33	0.12	4.08
i080-244	3.06	2.45	0.61	3.09	-0.03	3.09	-0.03	3.06	0.00	3.87	-0.81
i080-245	2.77	3.89	-1.12	3.40	-0.63	3.40	-0.63	1.90	0.87	1.90	0.87
i080-301	0.20	0.27	-0.07	0.20	0.00	0.27	-0.07	2.21	-2.01	1.83	-1.63
i080-302	3.47	3.31	0.16	3.47	0.00	1.67	1.80	1.60	1.87	1.60	1.87
i080-303	7.81	3.27	4.54	3.58	4.23	1.45	6.36	1.54	6.27	1.64	6.17
i080-304	1.90	5.75	-3.85	5.53	-3.63	5.46	-3.56	1.90	0.00	1.83	0.07
i080-305	3.41	2.21	1.20	2.17	1.24	2.17	1.24	3.35	0.06	1.25	2.16
i080-311	6.70	3.58	3.12	3.47	3.23	2.94	3.76	3.21	3.49	3.21	3.49
i080-312	0.64	0.82	-0.18	0.86	-0.22	0.82	-0.18	0.55	0.09	0.62	0.02
i080-313	2.55	2.95	-0.40	2.95	-0.40	2.95	-0.40	2.77	-0.22	1.80	0.75
i080-314	0.86	2.95	-2.09	1.24	-0.38	0.95	-0.09	2.72	-1.86	0.86	0.00
i080-315	0.00	0.90	-0.90	0.85	-0.85	0.36	-0.36	0.00	0.00	0.02	-0.02
i080-321	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-322	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-323	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-324	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-325	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i080-331	4.78	1.63	3.15	1.17	3.61	1.63	3.15	1.78	3.00	1.93	2.85
i080-332	2.29	0.73	1.56	0.13	2.16	0.34	1.95	4.03	-1.74	1.75	0.54
i080-333	4.16	1.77	2.39	1.12	3.04	3.57	0.59	3.61	0.55	2.86	1.30
i080-334	3.13	1.92	1.21	0.06	3.07	2.18	0.95	1.67	1.46	1.69	1.44
i080-335	5.09	0.59	4.50	5.33	-0.24	1.86	3.23	0.00	5.09	0.10	4.99
i080-341	0.00	2.38	-2.38	0.05	-0.05	0.00	0.00	0.00	0.00	1.63	-1.63
i080-342	0.16	0.25	-0.09	2.47	-2.31	0.48	-0.32	0.16	0.00	0.21	-0.05
i080-343	3.72	2.52	1.20	2.52	1.20	2.52	1.20	1.46	2.26	1.30	2.42
i080-344	2.02	0.14	1.88	2.20	-0.18	0.00	2.02	2.02	0.00	2.02	0.00
i080-345	0.23	1.75	-1.52	2.28	-2.05	2.42	-2.19	0.39	-0.16	1.75	-1.52
ave	2.87	1.76	1.11	1.50	1.37	1.65	1.22	1.44	1.43	1.25	1.62



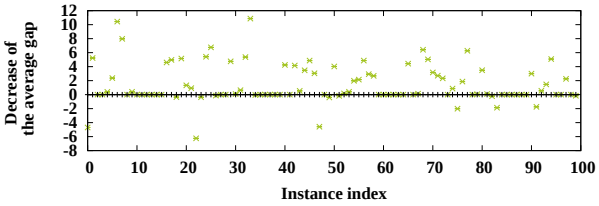
(a) degree centrality



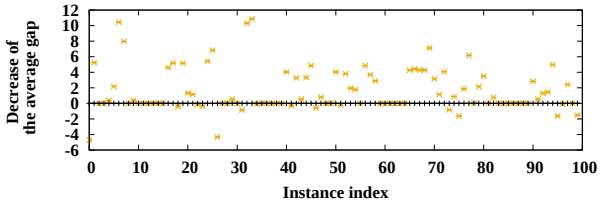
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality



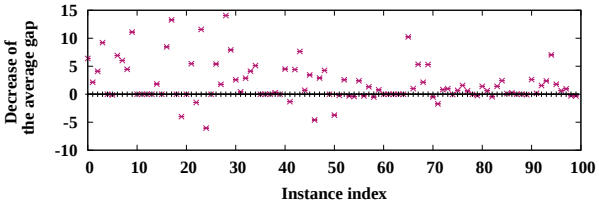
(e) edge betweenness centrality

Figure 3.30: Decrease of the average gap (ADH, I080)

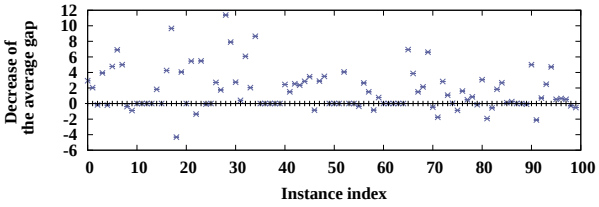
Table 3.23: Gaps [%] (ADH, I160)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
i160-001	10.76	4.38	6.38	7.83	2.93	0.84	9.92	0.44	10.32	0.00	10.76
i160-002	2.14	0.00	2.14	0.09	2.05	0.93	1.21	0.00	2.14	0.09	2.05
i160-003	4.14	0.04	4.10	4.31	-0.17	0.04	4.10	0.00	4.14	0.00	4.14
i160-004	9.20	0.00	9.20	5.27	3.93	5.27	3.93	5.27	3.93	9.20	0.00
i160-005	0.00	0.00	0.00	0.20	-0.20	4.01	-4.01	0.20	-0.20	0.20	-0.20
i160-011	5.31	5.43	-0.12	0.54	4.77	5.07	0.24	5.31	0.00	0.18	5.13
i160-012	6.91	0.00	6.91	0.00	6.91	0.00	6.91	6.23	0.68	0.00	6.91
i160-013	11.50	5.48	6.02	6.50	5.00	7.22	4.28	7.04	4.46	6.14	5.36
i160-014	6.02	1.57	4.45	6.41	-0.39	6.41	-0.39	6.02	0.00	0.51	5.51
i160-015	11.09	0.00	11.09	11.99	-0.90	0.00	11.09	4.69	6.40	4.69	6.40
i160-021	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i160-022	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i160-023	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i160-024	0.36	0.36	0.00	0.36	0.00	0.36	0.00	0.36	0.00	0.36	0.00
i160-025	1.83	0.00	1.83	0.00	1.83	1.83	0.00	0.00	1.83	0.00	1.83
i160-031	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	4.79	-4.79
i160-032	10.69	2.23	8.46	6.44	4.25	6.65	4.04	0.00	10.69	0.00	10.69
i160-033	13.61	0.33	13.28	3.95	9.66	0.00	13.61	0.00	13.61	3.76	9.85
i160-034	0.00	0.00	0.00	4.32	-4.32	0.00	0.00	6.43	-6.43	0.00	0.00
i160-035	5.14	9.13	-3.99	1.09	4.05	9.13	-3.99	7.56	-2.42	0.00	5.14
i160-041	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.55	-3.55
i160-042	6.33	0.87	5.46	0.87	5.46	5.79	0.54	9.76	-3.43	9.35	-3.02
i160-043	0.00	1.48	-1.48	1.36	-1.36	1.03	-1.03	0.00	0.00	0.00	0.00
i160-044	11.57	0.00	11.57	6.09	5.48	0.00	11.57	0.00	11.57	5.41	6.16
i160-045	0.39	6.44	-6.05	0.45	-0.06	0.39	0.00	0.39	0.00	0.39	0.00
i160-101	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i160-102	7.63	2.24	5.39	4.91	2.72	3.15	4.48	0.00	7.63	0.00	7.63
i160-103	2.06	0.29	1.77	0.31	1.75	0.31	1.75	0.42	1.64	0.29	1.77
i160-104	14.10	0.05	14.05	2.71	11.39	0.05	14.05	9.60	4.50	12.06	2.04
i160-105	7.91	0.00	7.91	0.00	7.91	2.58	5.33	0.00	7.91	2.58	5.33
i160-111	7.11	4.53	2.58	4.36	2.75	4.36	2.75	3.31	3.80	3.07	4.04
i160-112	0.48	0.07	0.41	0.07	0.41	0.00	0.48	0.48	0.00	0.03	0.45
i160-113	6.07	3.21	2.86	0.00	6.07	3.11	2.96	6.07	0.00	0.73	5.34
i160-114	4.18	0.07	4.11	2.14	2.04	1.71	2.47	1.71	2.47	1.77	2.41
i160-115	9.19	4.09	5.10	0.54	8.65	4.32	4.87	1.36	7.83	0.99	8.20
i160-121	0.04	0.04	0.00	0.04	0.00	0.04	0.00	0.04	0.00	0.04	0.00
i160-122	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i160-123	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i160-124	0.26	0.00	0.26	0.26	0.00	0.26	0.00	0.00	0.26	0.00	0.26
i160-125	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i160-131	5.10	0.60	4.50	2.65	2.45	0.48	4.62	0.77	4.33	2.53	2.57
i160-132	2.38	3.71	-1.33	0.87	1.51	1.36	1.02	2.61	-0.23	2.32	0.06
i160-133	15.40	10.99	4.41	12.86	2.54	5.66	9.74	2.57	12.83	7.95	7.45
i160-134	8.18	0.55	7.63	5.82	2.36	0.55	7.63	3.26	4.92	3.03	5.15
i160-135	6.14	5.44	0.70	3.28	2.86	3.42	2.72	0.86	5.28	0.81	5.33
i160-141	3.45	0.00	3.45	0.00	3.45	0.00	3.45	0.00	3.45	0.24	3.21
i160-142	1.37	5.97	-4.60	2.22	-0.85	1.91	-0.54	1.37	0.00	1.37	0.00
i160-143	6.61	3.72	2.89	3.72	2.89	3.60	3.01	4.38	2.23	5.94	0.67
i160-144	4.76	0.50	4.26	1.27	3.49	2.95	1.81	0.19	4.57	0.19	4.57
i160-145	1.12	1.12	0.00	1.12	0.00	1.12	0.00	0.16	0.96	2.25	-1.13

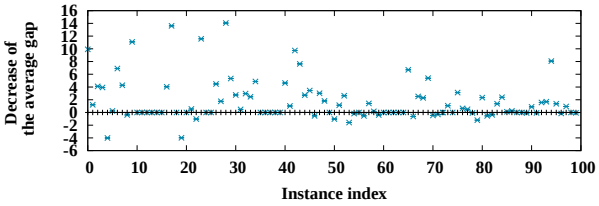
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
i160-201	0.78	4.52	-3.74	0.78	0.00	1.81	-1.03	0.78	0.00	0.78	0.00
i160-202	1.33	1.53	-0.20	1.33	0.00	0.20	1.13	2.84	-1.51	1.33	0.00
i160-203	10.56	7.99	2.57	6.49	4.07	7.94	2.62	4.67	5.89	5.44	5.12
i160-204	2.62	2.94	-0.32	2.62	0.00	4.20	-1.58	1.23	1.39	1.25	1.37
i160-205	2.64	3.06	-0.42	2.64	0.00	2.82	-0.18	2.64	0.00	1.39	1.25
i160-211	4.67	2.27	2.40	5.02	-0.35	4.67	0.00	2.06	2.61	2.06	2.61
i160-212	3.23	3.54	-0.31	0.58	2.65	3.77	-0.54	3.30	-0.07	4.98	-1.75
i160-213	4.04	2.71	1.33	2.53	1.51	2.62	1.42	4.13	-0.09	2.16	1.88
i160-214	3.23	3.74	-0.51	4.07	-0.84	3.06	0.17	2.47	0.76	3.23	0.00
i160-215	6.51	5.73	0.78	5.74	0.77	6.92	-0.41	5.13	1.38	5.09	1.42
i160-221	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i160-222	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i160-223	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i160-224	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i160-225	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i160-231	12.73	2.48	10.25	5.79	6.94	6.02	6.71	8.18	4.55	3.90	8.83
i160-232	6.77	5.76	1.01	2.90	3.87	7.41	-0.64	3.23	3.54	1.77	5.00
i160-233	5.66	0.33	5.33	4.15	1.51	3.14	2.52	2.54	3.12	2.67	2.99
i160-234	2.43	0.29	2.14	0.29	2.14	0.14	2.29	1.06	1.37	1.06	1.37
i160-235	8.55	3.25	5.30	1.94	6.61	3.16	5.39	3.39	5.16	6.09	2.46
i160-241	0.61	1.10	-0.49	1.08	-0.47	1.08	-0.47	0.28	0.33	0.14	0.47
i160-242	1.04	2.76	-1.72	2.80	-1.76	1.35	-0.31	1.19	-0.15	1.00	0.04
i160-243	2.95	2.12	0.83	0.12	2.83	2.97	-0.02	1.23	1.72	1.31	1.64
i160-244	2.34	1.36	0.98	1.26	1.08	1.28	1.06	1.14	1.20	0.57	1.77
i160-245	2.50	2.54	-0.04	2.46	0.04	2.50	0.00	2.18	0.32	1.30	1.20
i160-301	4.80	4.11	0.69	5.67	-0.87	1.68	3.12	1.78	3.02	1.67	3.13
i160-302	3.49	1.92	1.57	1.88	1.61	2.85	0.64	1.01	2.48	3.67	-0.18
i160-303	1.66	1.06	0.60	1.17	0.49	1.17	0.49	0.18	1.48	0.29	1.37
i160-304	0.99	1.01	-0.02	0.13	0.86	1.08	-0.09	0.11	0.88	0.19	0.80
i160-305	1.92	2.17	-0.25	2.07	-0.15	3.13	-1.21	2.12	-0.20	1.99	-0.07
i160-311	4.60	3.19	1.41	1.53	3.07	2.27	2.33	3.82	0.78	2.32	2.28
i160-312	3.00	2.36	0.64	4.93	-1.93	3.54	-0.54	3.00	0.00	3.00	0.00
i160-313	2.13	2.58	-0.45	2.69	-0.56	2.51	-0.38	1.31	0.82	1.31	0.82
i160-314	4.18	2.76	1.42	2.34	1.84	2.81	1.37	3.24	0.94	2.65	1.53
i160-315	3.29	0.87	2.42	0.62	2.67	0.90	2.39	0.53	2.76	0.78	2.51
i160-321	0.44	0.34	0.10	0.34	0.10	0.34	0.10	0.34	0.10	0.34	0.10
i160-322	0.66	0.42	0.24	0.42	0.24	0.42	0.24	0.42	0.24	0.42	0.24
i160-323	0.09	0.09	0.00	0.09	0.00	0.09	0.00	0.09	0.00	0.09	0.00
i160-324	0.36	0.36	0.00	0.36	0.00	0.36	0.00	0.36	0.00	0.36	0.00
i160-325	0.11	0.19	-0.08	0.19	-0.08	0.19	-0.08	0.19	-0.08	0.19	-0.08
i160-331	5.74	3.12	2.62	0.74	5.00	4.86	0.88	2.54	3.20	2.73	3.01
i160-332	2.88	2.71	0.17	4.99	-2.11	2.95	-0.07	2.90	-0.02	1.19	1.69
i160-333	3.59	2.04	1.55	2.88	0.71	2.03	1.56	0.78	2.81	0.75	2.84
i160-334	5.40	3.02	2.38	2.92	2.48	3.71	1.69	3.77	1.63	4.88	0.52
i160-335	8.51	1.49	7.02	3.80	4.71	0.44	8.07	1.28	7.23	2.34	6.17
i160-341	3.16	1.37	1.79	2.62	0.54	1.79	1.37	1.08	2.08	1.08	2.08
i160-342	0.96	0.32	0.64	0.31	0.65	1.11	-0.15	0.81	0.15	0.36	0.60
i160-343	2.14	1.18	0.96	1.60	0.54	1.20	0.94	2.15	-0.01	0.83	1.31
i160-344	0.34	0.63	-0.29	0.63	-0.29	0.34	0.00	0.34	0.00	0.36	-0.02
i160-345	1.14	1.44	-0.30	1.65	-0.51	1.20	-0.06	0.98	0.16	1.35	-0.21
ave	3.81	1.86	1.95	2.18	1.63	2.06	1.75	1.88	1.93	1.79	2.02



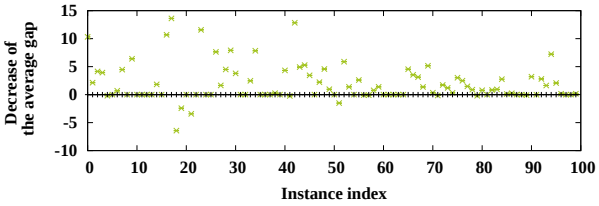
(a) degree centrality



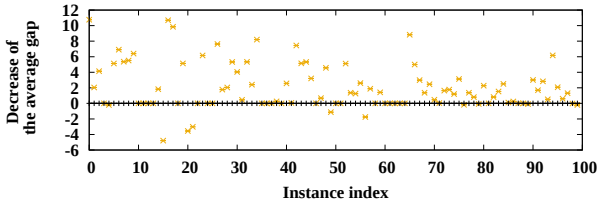
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality



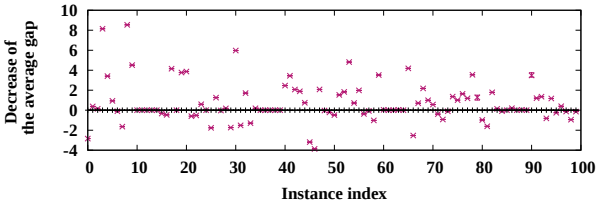
(e) edge betweenness centrality

Figure 3.31: Decrease of the average gap (ADH, I160)

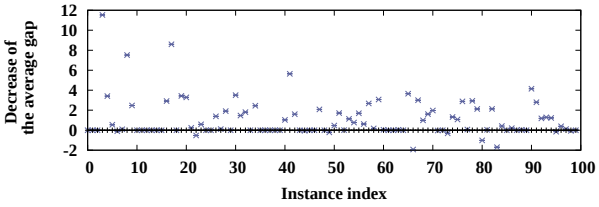
Table 3.24: Gaps [%] (ADH, I320)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
i320-001	0.00	2.84	-2.84	0.00	0.00	2.84	-2.84	0.00	0.00	0.00	0.00
i320-002	0.91	0.53	0.38	0.91	0.00	0.91	0.00	0.00	0.91	0.91	0.00
i320-003	3.47	3.33	0.14	3.47	0.00	3.47	0.00	3.23	0.24	0.07	3.40
i320-004	14.52	6.37	8.15	2.99	11.53	4.17	10.35	3.48	11.04	2.99	11.53
i320-005	3.41	0.00	3.41	0.00	3.41	0.10	3.31	0.00	3.41	0.00	3.41
i320-011	1.61	0.68	0.93	1.07	0.54	1.02	0.59	0.44	1.17	0.44	1.17
i320-012	4.06	4.16	-0.10	4.16	-0.10	12.67	-8.61	4.06	0.00	4.06	0.00
i320-013	6.03	7.67	-1.64	5.94	0.09	2.36	3.67	1.30	4.73	1.30	4.73
i320-014	8.98	0.44	8.54	1.46	7.52	0.87	8.11	4.71	4.27	5.05	3.93
i320-015	10.05	5.54	4.51	7.58	2.47	0.97	9.08	0.00	10.05	0.00	10.05
i320-021	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i320-022	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i320-023	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i320-024	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i320-025	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i320-031	1.35	1.68	-0.33	1.35	0.00	1.68	-0.33	1.35	0.00	0.64	0.71
i320-032	3.61	4.08	-0.47	0.69	2.92	3.36	0.25	3.61	0.00	0.25	3.36
i320-033	12.06	7.91	4.15	3.47	8.59	7.98	4.08	3.14	8.92	3.76	8.30
i320-034	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i320-035	3.77	0.00	3.77	0.34	3.43	0.34	3.43	0.00	3.77	0.00	3.77
i320-041	7.38	3.51	3.87	4.10	3.28	4.10	3.28	7.38	0.00	7.38	0.00
i320-042	0.24	0.83	-0.59	0.00	0.24	0.24	0.00	0.00	0.24	0.00	0.24
i320-043	0.23	0.75	-0.52	0.75	-0.52	0.75	-0.52	0.17	0.06	0.23	0.00
i320-044	0.59	0.00	0.59	0.00	0.59	0.59	0.00	0.59	0.00	0.59	0.00
i320-045	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	10.14	-10.14
i320-101	0.11	1.87	-1.76	0.11	0.00	1.87	-1.76	0.11	0.00	0.11	0.00
i320-102	3.44	2.18	1.26	2.05	1.39	2.18	1.26	1.84	1.60	1.62	1.82
i320-103	2.39	2.42	-0.03	2.28	0.11	0.46	1.93	2.42	-0.03	2.42	-0.03
i320-104	5.35	5.17	0.18	3.44	1.91	5.17	0.18	4.73	0.62	3.26	2.09
i320-105	1.70	3.44	-1.74	1.70	0.00	3.44	-1.74	1.62	0.08	1.70	0.00
i320-111	8.71	2.74	5.97	5.20	3.51	0.70	8.01	5.29	3.42	4.31	4.40
i320-112	2.49	4.01	-1.52	1.02	1.47	1.50	0.99	2.94	-0.45	0.71	1.78
i320-113	2.26	0.55	1.71	0.45	1.81	5.11	-2.85	2.26	0.00	0.21	2.05
i320-114	0.00	1.29	-1.29	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i320-115	7.36	7.17	0.19	4.91	2.45	10.59	-3.23	5.38	1.98	2.60	4.76
i320-121	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i320-122	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i320-123	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i320-124	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i320-125	0.15	0.15	0.00	0.15	0.00	0.15	0.00	0.15	0.00	0.15	0.00
i320-131	8.72	6.26	2.46	7.69	1.03	2.23	6.49	1.62	7.10	2.13	6.59
i320-132	5.64	2.20	3.44	0.00	5.64	0.06	5.58	5.64	0.00	4.08	1.56
i320-133	4.27	2.20	2.07	2.67	1.60	2.20	2.07	2.38	1.89	2.38	1.89
i320-134	5.08	3.19	1.89	5.08	0.00	3.34	1.74	1.99	3.09	3.28	1.80
i320-135	3.43	2.68	0.75	3.48	-0.05	2.34	1.09	3.41	0.02	3.89	-0.46
i320-141	0.00	3.19	-3.19	0.00	0.00	0.00	0.00	0.39	-0.39	1.80	-1.80
i320-142	0.00	3.87	-3.87	0.00	0.00	0.00	0.00	0.00	0.00	1.85	-1.85
i320-143	2.08	0.00	2.08	0.00	2.08	0.00	2.08	0.00	2.08	1.71	0.37
i320-144	0.00	0.03	-0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.03	-0.03
i320-145	0.25	0.47	-0.22	0.47	-0.22	0.47	-0.22	0.00	0.25	0.36	-0.11

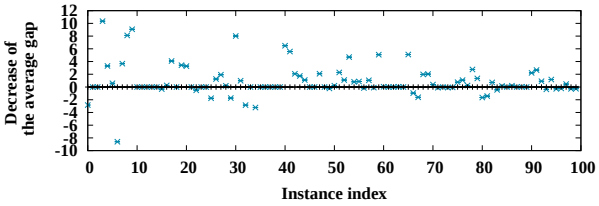
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	dec	ave	dec	ave	dec	ave	dec	ave	dec
i320-201	3.67	4.15	-0.48	3.20	0.47	3.48	0.19	3.59	0.08	3.67	0.00
i320-202	4.50	2.97	1.53	2.79	1.71	2.21	2.29	2.74	1.76	2.96	1.54
i320-203	4.11	2.28	1.83	4.10	0.01	3.01	1.10	5.05	-0.94	3.21	0.90
i320-204	7.86	3.04	4.82	6.73	1.13	3.16	4.70	2.16	5.70	1.70	6.16
i320-205	3.83	3.11	0.72	3.06	0.77	3.02	0.81	3.07	0.76	2.76	1.07
i320-211	5.90	3.92	1.98	4.20	1.70	5.03	0.87	5.90	0.00	5.90	0.00
i320-212	3.89	4.26	-0.37	3.27	0.62	4.04	-0.15	3.64	0.25	2.78	1.11
i320-213	5.17	5.24	-0.07	2.49	2.68	4.11	1.06	4.62	0.55	3.02	2.15
i320-214	5.11	6.13	-1.02	4.93	0.18	5.21	-0.10	2.71	2.40	3.67	1.44
i320-215	6.64	3.12	3.52	3.57	3.07	1.56	5.08	2.63	4.01	1.76	4.88
i320-221	0.31	0.27	0.04	0.27	0.04	0.27	0.04	0.27	0.04	0.27	0.04
i320-222	0.03	0.03	0.00	0.03	0.00	0.03	0.00	0.03	0.00	0.03	0.00
i320-223	0.21	0.21	0.00	0.21	0.00	0.21	0.00	0.21	0.00	0.21	0.00
i320-224	0.04	0.00	0.04	0.00	0.04	0.00	0.04	0.00	0.04	0.00	0.04
i320-225	0.15	0.15	0.00	0.15	0.00	0.15	0.00	0.15	0.00	0.15	0.00
i320-231	8.44	4.25	4.19	4.79	3.65	3.33	5.11	5.54	2.90	4.78	3.66
i320-232	2.74	5.27	-2.53	4.67	-1.93	3.70	-0.96	3.72	-0.98	4.64	-1.90
i320-233	5.83	5.13	0.70	2.83	3.00	7.44	-1.61	3.96	1.87	3.95	1.88
i320-234	5.75	3.57	2.18	4.76	0.99	3.77	1.98	5.99	-0.24	5.88	-0.13
i320-235	4.65	3.66	0.99	3.04	1.61	2.62	2.03	4.54	0.11	5.56	-0.91
i320-241	2.33	1.76	0.57	0.36	1.97	1.95	0.38	0.88	1.45	1.74	0.59
i320-242	1.70	2.09	-0.39	1.73	-0.03	1.82	-0.12	1.70	0.00	2.88	-1.18
i320-243	0.31	1.24	-0.93	0.31	0.00	0.31	0.00	0.31	0.00	1.60	-1.29
i320-244	0.72	0.82	-0.10	1.03	-0.31	0.82	-0.10	0.72	0.00	0.90	-0.18
i320-245	1.50	0.14	1.36	0.18	1.32	1.56	-0.06	0.00	1.50	1.77	-0.27
i320-301	3.76	2.75	1.01	2.69	1.07	3.00	0.76	1.23	2.53	2.03	1.73
i320-302	6.15	4.52	1.63	3.27	2.88	5.07	1.08	1.90	4.25	2.27	3.88
i320-303	2.84	1.65	1.19	2.77	0.07	2.61	0.23	2.35	0.49	3.32	-0.48
i320-304	5.10	1.55	3.55	2.17	2.93	2.34	2.76	2.25	2.85	2.53	2.57
i320-305	3.51	2.23	1.28	1.39	2.12	2.16	1.35	1.54	1.97	1.95	1.56
i320-311	2.76	3.74	-0.98	3.78	-1.02	4.42	-1.66	3.42	-0.66	3.17	-0.41
i320-312	2.64	4.25	-1.61	2.60	0.04	4.07	-1.43	3.01	-0.37	2.09	0.55
i320-313	5.21	3.42	1.79	3.08	2.13	4.49	0.72	2.54	2.67	2.86	2.35
i320-314	1.85	1.72	0.13	3.55	-1.70	2.30	-0.45	1.85	0.00	1.81	0.04
i320-315	2.99	3.08	-0.09	2.57	0.42	2.83	0.16	2.99	0.00	2.87	0.12
i320-321	0.38	0.38	0.00	0.38	0.00	0.38	0.00	0.38	0.00	0.38	0.00
i320-322	0.34	0.14	0.20	0.14	0.20	0.14	0.20	0.14	0.20	0.14	0.20
i320-323	0.20	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00
i320-324	0.31	0.27	0.04	0.27	0.04	0.27	0.04	0.27	0.04	0.27	0.04
i320-325	0.45	0.45	0.00	0.45	0.00	0.45	0.00	0.45	0.00	0.45	0.00
i320-331	8.56	5.08	3.48	4.41	4.15	6.34	2.22	4.25	4.31	3.77	4.79
i320-332	5.85	4.64	1.21	3.05	2.80	3.19	2.66	3.84	2.01	3.17	2.68
i320-333	5.30	3.95	1.35	4.12	1.18	4.40	0.90	3.29	2.01	3.89	1.41
i320-334	5.67	6.49	-0.82	4.38	1.29	6.04	-0.37	3.13	2.54	3.39	2.28
i320-335	4.14	2.97	1.17	2.91	1.23	2.96	1.18	3.30	0.84	3.81	0.33
i320-341	0.62	0.87	-0.25	0.77	-0.15	0.91	-0.29	0.34	0.28	0.64	-0.02
i320-342	0.46	0.06	0.40	0.07	0.39	0.68	-0.22	0.30	0.16	0.62	-0.16
i320-343	1.07	1.20	-0.13	0.96	0.11	0.60	0.47	0.82	0.25	1.04	0.03
i320-344	0.23	1.17	-0.94	0.29	-0.06	0.52	-0.29	0.23	0.00	1.14	-0.91
i320-345	0.63	0.77	-0.14	0.63	0.00	0.91	-0.28	0.09	0.54	0.67	-0.04
ave	3.02	2.30	0.72	1.93	1.09	2.14	0.88	1.84	1.18	1.91	1.11



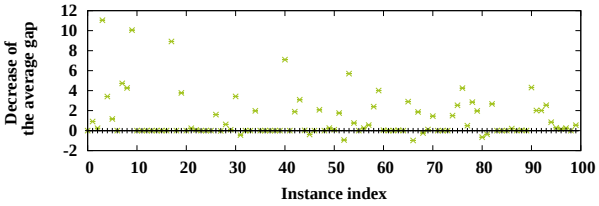
(a) degree centrality



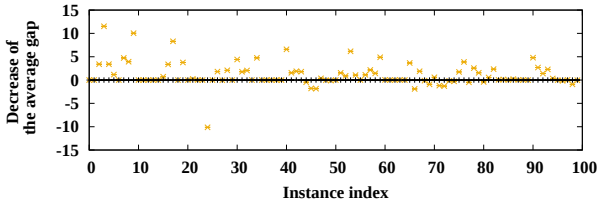
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality



(e) edge betweenness centrality

Figure 3.32: Decrease of the average gap (ADH, I320)

3.5.4 Tradeoff between increase in CPU time and decrease in gaps

The construction methods combined with network centralities tend to construct smaller-weight Steiner trees than the original method. However, our proposed method has two demerits: the calculation time becomes long and parameters must be tuned.

First, we discuss the tradeoff between the increase in computation time and the decrease in the average gap. We evaluate the practical computation time to solve the benchmark problem sets B, C, D, I080, I160, and I320 by eighteen construction methods: SPH without network centrality, SPH with the degree centrality, SPH with the eigenvector centrality, SPH with the closeness centrality, SPH with the vertex betweenness centrality, SPH with the edge betweenness centrality, DNH without network centrality, DNH with the degree centrality, DNH with the eigenvector centrality, DNH with the closeness centrality, DNH with the vertex betweenness centrality, DNH with the edge betweenness centrality, ADH without network centrality, ADH with the degree centrality, ADH with the eigenvector centrality, ADH with the closeness centrality, ADH with the vertex betweenness centrality, and ADH with the edge betweenness centrality. We implemented these methods in C using the gcc compiler (ver. 4.2.1). All numerical experiments were performed on an iMacPro (2017) with 3.2 GHz Intel Xeon W processor and 64 GB 2666 MHz DDR4 random access memory. We measured the CPU time between before inputting an instance and after output of a solution.

The CPU times of these construction methods are listed in Tables 3.25–3.42. In Tables 3.25–3.42, wc is without network centrality (conventional), d_v is the degree centrality, e_v is the eigenvector centrality, c_v is the closeness centrality, b_v is the vertex betweenness centrality, and b_e is the edge betweenness centrality. ave is the average CPU time, inc is the increase in average CPU time using network centrality, i.e., dividing the average CPU time of with network centrality by the average CPU time without network centrality. In other words,

$$inc = t'/t, \tag{3.30}$$

where t' is the average CPU time of the proposed (with network centrality) method and t is the average CPU time of the original (without network centrality) method. For example, inc = 2 means that using network centrality consumes twice the CPU time of the original method. We constructed 50 Steiner trees per instance and averaged their results. We set the scaling parameter α as a suitable value based on pre-numerical experiments.

Tables 3.25–3.30 and Figs. 3.33–3.38 show the results of SPH with network centralities. From Table 3.25 and Fig. 3.33, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities increases the computational time by approximately 1.57, 2.86, 2.88, 3.10, and 3.15 times, respectively. The increase in the computation time was greatest when using eigenvector centrality for instance b02, and the computation time was increased as six times large as the original one. We use the power method to calculate eigenvector centrality, and the computation time for the conversing power method is determined

by the ratio between the largest and the second largest eigenvalue. Thus, the instance b02 may have an adjacency matrix that takes time for converging the power method. However, its computation time is 0.001356 [s], which is enough to short to use.

From Table 3.26 and Fig. 3.34, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities increases the computation time by approximately 1.05, 1.08, 1.78, 1.98, and 2.07 times, respectively. The average computation time of the original SPH is longer than that for the benchmark problem set B because the computation time of the SPH depends on the number of vertices, edges, and terminals. The increase in the computation time was greatest when using edge betweenness centrality for the instance c01, and the computation time was increased 25.29 times of the original one. This value is very large, however, its computation time is 0.043600 [s]. It is enough to short to use. The computation time becomes long for the instances that have a small number of terminals such as c01, c06, c11, and c16. The SPH repeats including the shortest path between arbitrary vertex included in the current tree and arbitrary terminal not included in the current tree. Thus, the number of terminals strongly affects computation time.

From Table 3.27 and Fig. 3.35, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities increases the computation time by approximately 1.02, 1.03, 1.44, 1.58, and 1.65 times, respectively. The increase in the computation time was greatest when using edge betweenness centrality for the instance d01, and the computation time was increased 34.16 times of the original one. This value is very large, however, its computation time is 0.187745 [s]. It is enough to short to use. The average value of the computation time of using the network centrality is at most 1.65 times of the original SPH. This is because the order of the computation cost of the network centrality is smaller than or equals to the computation cost of SPH.

From Table 3.28 and Fig. 3.36, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities increases the computation time by approximately 1.34, 2.08, 3.55, 3.64, and 3.80 times, respectively. The instances in I080 are sorted in ascending order of the number of terminals; the number of terminals of the first 25 instances is 6, the second 25 instances is 8, the third 25 instances is 16, and the last 25 instances is 20. On the other hand, the increase in computation time is high for the first 25 instances and short for the last 25 instances. These results indicate that the computation time depends on the number of terminals.

From Table 3.29 and Fig. 3.37, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities increases the computation time by approximately 1.17, 1.42, 4.02, 4.09, and 4.27 times, respectively. The increase in the computation time was greatest when using edge betweenness centrality for the instance i160-001 to i160-005, and the computation time of them was increased approximately nine times the original one. These instances have the smallest number of edges and terminals. Thus, the computation time of SPH becomes short because the computation cost of SPH depends on the number

of edges and terminals. Therefore, the addition of the computation time for calculating the network centrality affects the total computation time. However, the computation time of them is approximately 0.005 [s], and it is enough to short to use.

From Table 3.28 and Fig. 3.36, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities increases the computation time by approximately 1.11, 1.22, 4.88, 4.87, and 5.12 times, respectively. The computation time of using the vertex and edge betweenness centralities differs unless the computation cost of them is the same. This is because the input size of using edge betweenness centrality is larger than that of using vertex betweenness centrality. We calculate the network centrality by using R and input the instance and the network centrality to C code. The instance of the Steiner tree problem in graphs is a connected graph, thus, the number of edges is larger than the number of vertices. Therefore, the input size of using network centrality of edges becomes larger than that of using network centrality of vertices.

From these results, it is revealed that the increase of the computation time of SPH combined with network centralities becomes large when the number of edges and terminals is small. This is because the computation time of SPH depends on the number of edges and terminals. Therefore, if the number of edges and terminals are large, we should better use the network centrality.

The combination of SPH and five network centralities increase the computation time, however, it is still practical because the order of the computation cost of SPH and network centralities are the same. Are these results the same when using other construction methods? To confirm it, we conducted the same numerical experiments by using DNH and ADH. Tables 3.31–3.36 and Figs. 3.39–3.44 show the results of the DNH with network centralities.

From Table 3.31 and Fig. 3.39, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities increases the computation time by approximately 1.26, 2.07, 2.31, 2.49, and 2.50 times, respectively. The increase in the computation time was greatest when using eigenvector centrality for instance b01, and the computation time was increased 4.11 times of the original one. One of the reasons why the computation time becomes large by using eigenvector centrality is that we use the power method to calculate eigenvector centrality. The computation time for the conversing power method is determined by the ratio between the largest and the second largest eigenvalue. Thus, this result indicates that the instance b01 may have an adjacency matrix that takes time for converging the power method. However, its computation time is 0.000904 [s], which is enough to short to use.

From Table 3.32 and Fig. 3.40, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities increases the computation time by approximately 1.00, 1.03, 1.49, 1.66, and 1.74 times, respectively. In particular, the increase of the computation time is long for the instances c01, c06, c11, and c16. This is because these instances have a small number of terminals (5). The DNH repeats replacing the edges included in the minimum spanning tree of the complete graphs of terminals. Thus, the number of terminals

strongly affects computation time. If the number of terminals is small, the computation time of DNH becomes small. Therefore, the computation time of calculating the network centrality affects the total computation time. From this viewpoint, we should better use the network centrality in the case of the number of terminals are large.

From Table 3.33 and Fig. 3.41, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities increases the computation time by approximately 1.00, 1.02, 1.29, 1.42, and 1.50 times, respectively. The increase in the computation time was greatest when using edge betweenness centrality for the instance d01, and the computation time was increased 25.93 times of the original one. This value can be seen very large, however, its computation time is 0.189605. It is enough to short to use.

From Table 3.34 and Fig. 3.42, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities increases the computation time by approximately 1.26, 1.86, 3.05, 3.13, and 3.28 times, respectively. The increase of the computation time becomes shorter when the index of instances becomes large. The instances in I080 are sorted in ascending order of the number of terminals; the number of terminals of the first 25 instances is 6, the second 25 instances is 8, the third 25 instances is 16, and the last 25 instances is 20. These results indicate that the computation time depends on the number of terminals. Actually, the computation cost of DNH is $\mathcal{O}(|T||E| + |T||V| \log |V|)$. From this viewpoint, we should better use the network centrality in the case of the number of terminals are large.

From Table 3.35 and Fig. 3.43, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities increases the computation time by approximately 1.11, 1.30, 3.30, 3.35, and 3.52 times, respectively. The computation time of using the vertex and edge betweenness centralities differs unless the computation cost of them is the same. This is because the input size of using edge betweenness centrality is larger than that of using vertex betweenness centrality. We calculate the network centrality by using R and input the instance and the network centrality to C code. The instance of the Steiner tree problem in graphs is a connected graph, thus, the number of edges is larger than the number of vertices. Therefore, the input size of using network centrality of edges becomes larger than that of using network centrality of vertices.

From Table 3.36 and Fig. 3.44, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities increases the computation time by approximately 1.06, 1.14, 3.81, 3.79, and 4.05 times, respectively. The increase in the computation time of using the degree centrality is the shortest. This is because the computation cost of degree centrality is very small, $\mathcal{O}(|E|)$. It is smaller than the computation cost of DNH. On the other hand, the computation time of using the closeness and betweenness centralities is long. The computation cost of these network centralities is $\mathcal{O}(|V||E| + |V|^2 \log |V|)$. It is larger than the computation cost of DNH. Thus the computation time of using these network centralities becomes large.

From these results, we found that trends of increase in the computation time when using

DNH are the same as that of using SPH. One of the reasons why the results of using SPH and DNH become the same is that the computation cost of them is the same. Thus, using the network centrality cause the same effect for them.

Tables 3.37–3.42 and Figs. 3.45–3.50 show the results of the ADH with network centralities. From Table 3.37 and Fig. 3.45, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities increases the computation time by approximately 1.04, 1.19, 1.22, 1.26, and 1.27 times, respectively. The increase in the computation time was greatest when using edge betweenness centrality for the instance b01, and the computation time was increased 2.02 times of the original one. In the cases of using SPH and DNH, the increase of the computation time of using edge betweenness centrality is shorter than that of using eigenvector centrality. Thus, this trend is different from the results of using SPH and DNH. However, its computation time is 0.001390. It is enough to short to use.

From Table 3.38 and Fig. 3.46, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities increases the computation time by approximately 1.01, 1.02, 1.06, 1.13, and 1.16 times, respectively. The computation time of using the degree and eigenvector centralities is shorter than that of the original method for the instances c06, c07, c11, c12, c16, c17, c18. One of the reasons why the computation time becomes short unless the procedure of calculating the network centrality is added is the computation cost of these network centralities is shorter than that of ADH. The computation cost of the degree centrality is $\mathcal{O}(|E|)$, however that of ADH is $\mathcal{O}(|T||V||E| + |T||V|^2 \log |V|)$. Therefore, the computation time of ADH with degree and eigenvector centralities is sometimes smaller than that of the original ADH.

From Table 3.39 and Fig. 3.47, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities increases the computation time by approximately 1.03, 1.04, 1.03, 1.08, and 1.11 times, respectively. The computation time of the original ADH is very longer than that of SPH and DNH. This is because the computation cost of ADH is larger than that of SPH and DNH. Also, the increase in the computation time of using the network centrality is small. In particular, the computation time for the instances d05, d10, d15, and d20 is very large. These instances have many terminals (500). Thus, we should better use SPH and DNH in the case of the number of terminals are large.

From Table 3.40 and Fig. 3.48, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities increases the computation time by approximately 1.06, 1.23, 1.57, 1.60, and 1.64 times, respectively. The increase of the computation time is at most twice the original ADH. In contrast to SPH and DNH, ADH can find a small-weight Steiner tree if the instance is a complete graph. Thus, we should better use ADH for I080.

From Table 3.41 and Fig. 3.49, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities increases the computation time by approximately 1.00, 1.04, 1.46, 1.47, and 1.52 times, respectively. The computation time of using the vertex

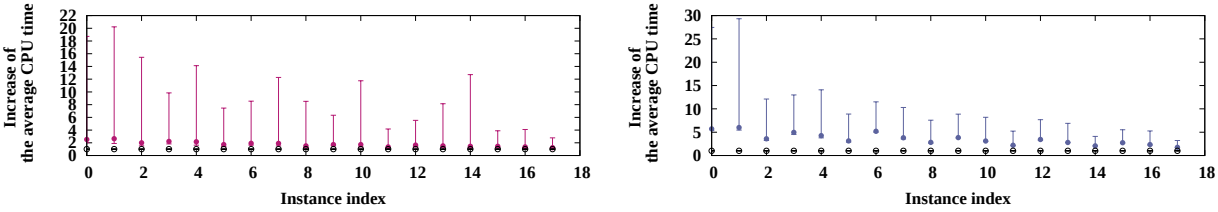
and edge betweenness centralities differs unless the computation cost of them is the same. This is because the input size of using edge betweenness centrality is larger than that of using vertex betweenness centrality. We calculate the network centrality by using R and input the instance and the network centrality to C code. The instance of the Steiner tree problem in graphs is a connected graph, thus, the number of edges is larger than the number of vertices. Therefore, the input size of using network centrality of edges becomes larger than that of using network centrality of vertices.

From Table 3.40 and Fig. 3.48, using the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities increases the computation time by approximately 0.97, 0.99, 1.41, 1.42, and 1.46 times, respectively. Thus, the usage of the degree and eigenvector centrality decreases the average computation time. This is because the computation cost of ADH is larger than that of these network centralities. The usage of network centrality contributes to reducing the average gap. Therefore, we should better to use ADH for I320.

From these results, we found that the increase of the computation time of using ADH is smaller than that of SPH and DNH because the computation cost of ADH is larger than that of them. In particular, using the degree and eigenvector centralities sometimes reduces the computation time from the original ADH. One of the reasons why the computation time becomes short is that the difference in the computation time of ADH is much larger than the computation time of these network centralities. Unfortunately, the computation time of ADH for the benchmark problem set D is very long and it is not enough to use. Thus, we should better use SPH and DNH if the number of vertices is large.

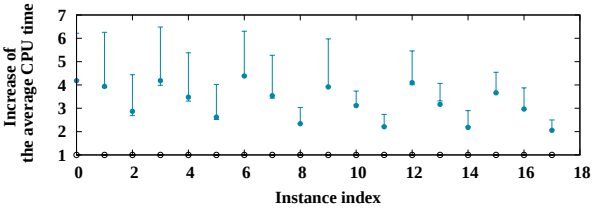
Table 3.25: CPU time [s] (SPH, B)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
b01	0.000197	0.000498	2.53	0.001125	5.71	0.000824	4.18	0.000841	4.27	0.000845	4.29
b02	0.000226	0.000596	2.64	0.001356	6.00	0.000890	3.94	0.000912	4.04	0.000900	3.98
b03	0.000357	0.000715	2.00	0.001278	3.58	0.001025	2.87	0.001033	2.89	0.001040	2.91
b04	0.000231	0.000511	2.21	0.001145	4.96	0.000967	4.19	0.000980	4.24	0.000998	4.32
b05	0.000286	0.000616	2.15	0.001219	4.26	0.000994	3.48	0.001031	3.60	0.001038	3.63
b06	0.000443	0.000758	1.71	0.001389	3.14	0.001161	2.62	0.001206	2.72	0.001236	2.79
b07	0.000319	0.000614	1.92	0.001658	5.20	0.001399	4.39	0.001491	4.67	0.001514	4.75
b08	0.000424	0.000805	1.90	0.001612	3.80	0.001501	3.54	0.001628	3.84	0.001595	3.76
b09	0.000781	0.001182	1.51	0.002196	2.81	0.001827	2.34	0.001931	2.47	0.001954	2.50
b10	0.000419	0.000721	1.72	0.001609	3.84	0.001641	3.92	0.001755	4.19	0.001760	4.20
b11	0.000561	0.000965	1.72	0.001749	3.12	0.001747	3.11	0.001937	3.45	0.001970	3.51
b12	0.001024	0.001382	1.35	0.002241	2.19	0.002262	2.21	0.002389	2.33	0.002485	2.43
b13	0.000528	0.000862	1.63	0.001816	3.44	0.002164	4.10	0.002428	4.60	0.002429	4.60
b14	0.000723	0.001093	1.51	0.002018	2.79	0.002292	3.17	0.002549	3.53	0.002701	3.74
b15	0.001406	0.002043	1.45	0.002911	2.07	0.003062	2.18	0.003294	2.34	0.003315	2.36
b16	0.000724	0.001047	1.45	0.001991	2.75	0.002653	3.66	0.002956	4.08	0.003014	4.16
b17	0.000999	0.001355	1.36	0.002346	2.35	0.002965	2.97	0.003243	3.25	0.003309	3.31
b18	0.001882	0.002323	1.23	0.003312	1.76	0.003862	2.05	0.004176	2.22	0.004260	2.26
ave	0.000641	0.001005	1.57	0.001832	2.86	0.001846	2.88	0.001988	3.10	0.002020	3.15

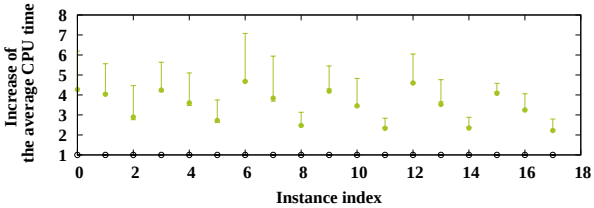


(a) degree centrality

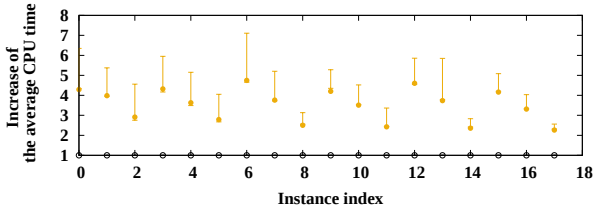
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality

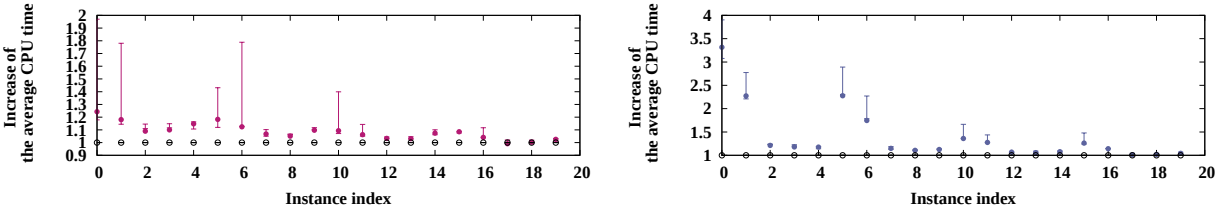


(e) edge betweenness centrality

Figure 3.33: Increase of CPU time from the original method (SPH, B)

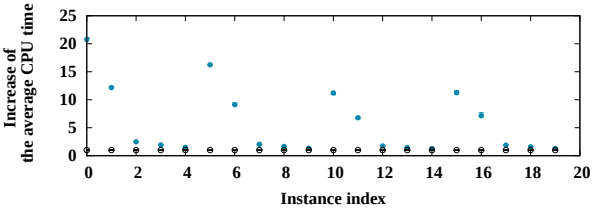
Table 3.26: CPU time [s] (SPH, C)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
c01	0.001724	0.002143	1.24	0.005718	3.32	0.035800	20.77	0.041776	24.23	0.043600	25.29
c02	0.002986	0.003525	1.18	0.006788	2.27	0.036354	12.17	0.043286	14.50	0.044070	14.76
c03	0.024075	0.026234	1.09	0.029183	1.21	0.059755	2.48	0.065574	2.72	0.067546	2.81
c04	0.040309	0.044338	1.10	0.047513	1.18	0.077524	1.92	0.084234	2.09	0.086277	2.14
c05	0.096040	0.110183	1.15	0.112945	1.18	0.143242	1.49	0.150653	1.57	0.152917	1.59
c06	0.002671	0.003158	1.18	0.006081	2.28	0.043421	16.26	0.052032	19.48	0.053860	20.16
c07	0.004962	0.005578	1.12	0.008667	1.75	0.045263	9.12	0.054255	10.93	0.055947	11.28
c08	0.040717	0.043471	1.07	0.046728	1.15	0.083209	2.04	0.093440	2.29	0.094118	2.31
c09	0.065002	0.068296	1.05	0.071842	1.11	0.107232	1.65	0.117236	1.80	0.120317	1.85
c10	0.145680	0.159834	1.10	0.163814	1.12	0.198257	1.36	0.211454	1.45	0.213506	1.47
c11	0.005705	0.006237	1.09	0.007763	1.36	0.064023	11.22	0.077919	13.66	0.080650	14.14
c12	0.010127	0.010760	1.06	0.012921	1.28	0.068710	6.78	0.083744	8.27	0.085846	8.48
c13	0.079907	0.082552	1.03	0.085353	1.07	0.138299	1.73	0.158221	1.98	0.161759	2.02
c14	0.123366	0.127183	1.03	0.130434	1.06	0.180598	1.46	0.204404	1.66	0.210693	1.71
c15	0.261887	0.281328	1.07	0.281835	1.08	0.330113	1.26	0.359596	1.37	0.375263	1.43
c16	0.014435	0.015665	1.09	0.018217	1.26	0.162451	11.25	0.188695	13.07	0.192980	13.37
c17	0.023887	0.024856	1.04	0.027271	1.14	0.170427	7.13	0.197107	8.25	0.202665	8.48
c18	0.162282	0.160874	0.99	0.164114	1.01	0.304258	1.87	0.347002	2.14	0.362206	2.23
c19	0.245312	0.246064	1.00	0.250206	1.02	0.389589	1.59	0.432982	1.77	0.459796	1.87
c20	0.497547	0.510113	1.03	0.518433	1.04	0.647836	1.30	0.704023	1.41	0.763590	1.53
ave	0.088030	0.092019	1.05	0.095039	1.08	0.156493	1.78	0.174649	1.98	0.182267	2.07

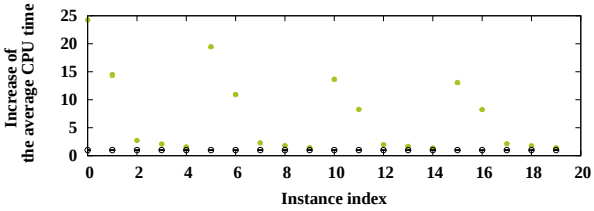


(a) degree centrality

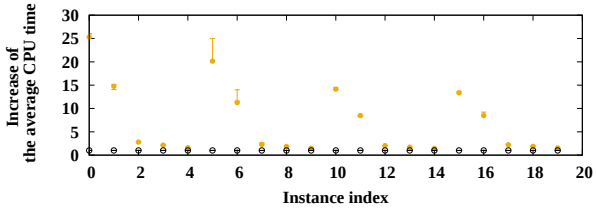
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality

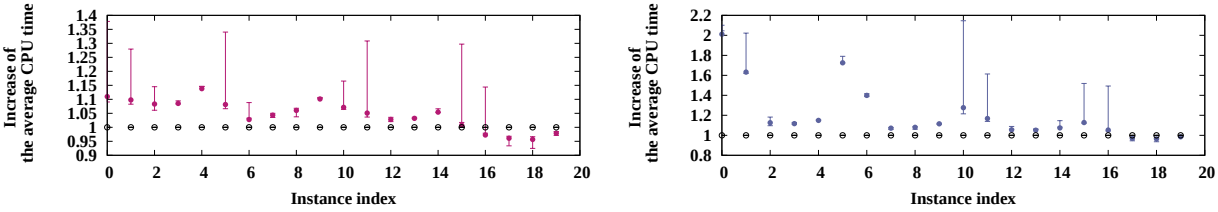


(e) edge betweenness centrality

Figure 3.34: Increase of CPU time from the original method (SPH, C)

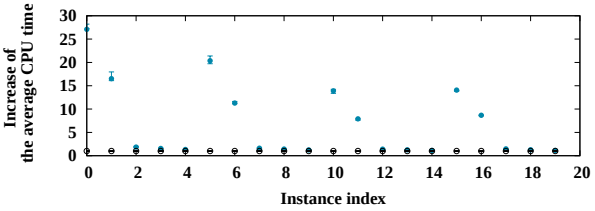
Table 3.27: CPU time [s] (SPH, D)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
d01	0.005496	0.006100	1.11	0.011055	2.01	0.148942	27.10	0.177721	32.34	0.187745	34.16
d02	0.009681	0.010627	1.10	0.015802	1.63	0.159997	16.53	0.185938	19.21	0.195913	20.24
d03	0.188408	0.204103	1.08	0.212114	1.13	0.348888	1.85	0.383365	2.03	0.388255	2.06
d04	0.302470	0.328277	1.09	0.337653	1.12	0.472073	1.56	0.502588	1.66	0.513393	1.70
d05	0.758941	0.863644	1.14	0.872376	1.15	1.009164	1.33	1.045755	1.38	1.056057	1.39
d06	0.009273	0.010028	1.08	0.015996	1.73	0.188859	20.37	0.224303	24.19	0.231282	24.94
d07	0.017186	0.017665	1.03	0.024046	1.40	0.193974	11.29	0.233216	13.57	0.237856	13.84
d08	0.304929	0.318085	1.04	0.326652	1.07	0.489628	1.61	0.534655	1.75	0.543600	1.78
d09	0.476351	0.505092	1.06	0.514509	1.08	0.677825	1.42	0.713972	1.50	0.727861	1.53
d10	1.100780	1.212200	1.10	1.227040	1.11	1.371020	1.25	1.430650	1.30	1.448200	1.32
d11	0.018849	0.020195	1.07	0.024050	1.28	0.261916	13.90	0.326622	17.33	0.343142	18.20
d12	0.035764	0.037588	1.05	0.041783	1.17	0.282827	7.91	0.352127	9.85	0.359766	10.06
d13	0.593965	0.610661	1.03	0.626685	1.06	0.835064	1.41	0.932794	1.57	0.966249	1.63
d14	0.922226	0.951812	1.03	0.969217	1.05	1.166717	1.27	1.285540	1.39	1.343640	1.46
d15	2.011950	2.120990	1.05	2.160330	1.07	2.294460	1.14	2.461860	1.22	2.551910	1.27
d16	0.046889	0.047238	1.01	0.052890	1.13	0.659864	14.07	0.795548	16.97	0.834334	17.79
d17	0.082126	0.079881	0.97	0.086437	1.05	0.711641	8.67	0.852623	10.38	0.879948	10.71
d18	1.214600	1.167110	0.96	1.177180	0.97	1.787350	1.47	2.007500	1.65	2.133640	1.76
d19	1.853380	1.773270	0.96	1.785860	0.96	2.381920	1.29	2.642010	1.43	2.807670	1.51
d20	3.810540	3.730900	0.98	3.754390	0.99	4.326820	1.14	4.685310	1.23	4.983740	1.31
ave	0.655419	0.667403	1.02	0.677908	1.03	0.941379	1.44	1.036860	1.58	1.082580	1.65

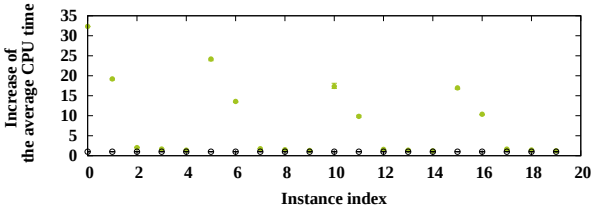


(a) degree centrality

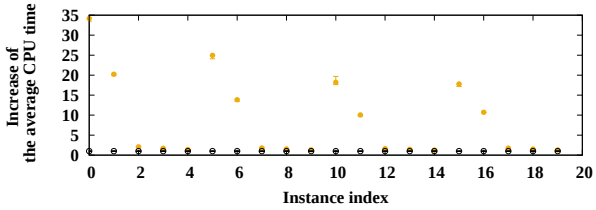
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality



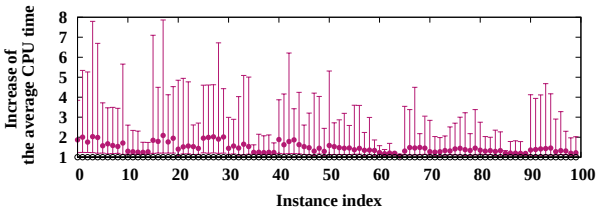
(e) edge betweenness centrality

Figure 3.35: Increase of CPU time from the original method (SPH, D)

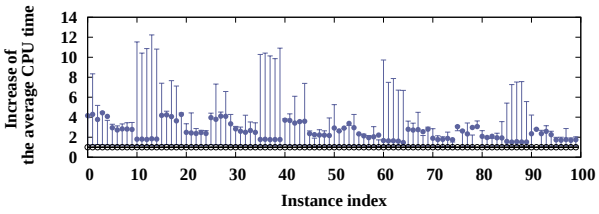
Table 3.28: CPU time [s] (SPH, I080)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
i080-001	0.000245	0.000457	1.87	0.001019	4.16	0.001334	5.44	0.001497	6.11	0.001436	5.86
i080-002	0.000240	0.000481	2.00	0.001028	4.28	0.001430	5.96	0.001486	6.19	0.001550	6.46
i080-003	0.000239	0.000420	1.76	0.000901	3.77	0.001357	5.68	0.001410	5.90	0.001559	6.52
i080-004	0.000243	0.000493	2.03	0.001078	4.44	0.001376	5.66	0.001451	5.97	0.001401	5.77
i080-005	0.000249	0.000495	1.99	0.001015	4.08	0.001360	5.46	0.001518	6.10	0.001527	6.13
i080-011	0.000389	0.000611	1.57	0.001145	2.94	0.001849	4.75	0.001963	5.05	0.001994	5.13
i080-012	0.000396	0.000661	1.67	0.001080	2.73	0.001814	4.58	0.001990	5.03	0.002017	5.09
i080-013	0.000388	0.000615	1.59	0.001100	2.84	0.001787	4.61	0.002019	5.20	0.001997	5.15
i080-014	0.000394	0.000606	1.54	0.001109	2.81	0.001988	5.05	0.002033	5.16	0.002122	5.39
i080-015	0.000389	0.000664	1.71	0.001080	2.78	0.001941	4.99	0.002084	5.36	0.001992	5.12
i080-021	0.001672	0.002170	1.30	0.003007	1.80	0.006622	3.96	0.006788	4.06	0.006938	4.15
i080-022	0.001678	0.002136	1.27	0.003037	1.81	0.006711	4.00	0.006594	3.93	0.006934	4.13
i080-023	0.001671	0.002092	1.25	0.002970	1.78	0.006677	4.00	0.006706	4.01	0.007161	4.29
i080-024	0.001673	0.002079	1.24	0.003089	1.85	0.006703	4.01	0.006612	3.95	0.007039	4.21
i080-025	0.001675	0.002122	1.27	0.003038	1.81	0.006785	4.05	0.006726	4.02	0.007282	4.35
i080-031	0.000274	0.000506	1.85	0.001146	4.18	0.001552	5.66	0.001567	5.72	0.001641	5.99
i080-032	0.000267	0.000478	1.79	0.001130	4.23	0.001502	5.63	0.001564	5.86	0.001640	6.14
i080-033	0.000273	0.000570	2.09	0.001109	4.06	0.001485	5.44	0.001573	5.76	0.001623	5.95
i080-034	0.000273	0.000481	1.76	0.000992	3.63	0.001463	5.36	0.001636	5.99	0.001640	6.01
i080-035	0.000272	0.000530	1.95	0.001163	4.28	0.001498	5.51	0.001668	6.13	0.001624	5.97
i080-041	0.000543	0.000761	1.40	0.001349	2.48	0.002423	4.46	0.002527	4.65	0.002623	4.83
i080-042	0.000544	0.000824	1.51	0.001321	2.43	0.002473	4.55	0.002472	4.54	0.002612	4.80
i080-043	0.000543	0.000846	1.56	0.001287	2.37	0.002455	4.52	0.002634	4.85	0.002719	5.01
i080-044	0.000542	0.000830	1.53	0.001336	2.46	0.002466	4.55	0.002629	4.85	0.002493	4.60
i080-045	0.000547	0.000788	1.44	0.001296	2.37	0.002413	4.41	0.002502	4.57	0.002700	4.94
i080-101	0.000271	0.000529	1.95	0.001073	3.96	0.001409	5.20	0.001456	5.37	0.001496	5.52
i080-102	0.000278	0.000553	1.99	0.001054	3.79	0.001450	5.22	0.001505	5.41	0.001577	5.67
i080-103	0.000271	0.000548	2.02	0.001112	4.10	0.001450	5.35	0.001473	5.44	0.001523	5.62
i080-104	0.000276	0.000526	1.91	0.001128	4.09	0.001407	5.10	0.001547	5.61	0.001573	5.70
i080-105	0.000281	0.000565	2.01	0.000938	3.34	0.001491	5.31	0.001552	5.52	0.001542	5.49
i080-111	0.000454	0.000656	1.44	0.001286	2.83	0.002012	4.43	0.002129	4.69	0.002105	4.64
i080-112	0.000445	0.000694	1.56	0.001154	2.59	0.001837	4.13	0.002117	4.76	0.002166	4.87
i080-113	0.000467	0.000680	1.46	0.001165	2.49	0.002006	4.30	0.002168	4.64	0.002097	4.49
i080-114	0.000454	0.000749	1.65	0.001214	2.67	0.001924	4.24	0.002065	4.55	0.002130	4.69
i080-115	0.000449	0.000689	1.53	0.001117	2.49	0.002028	4.52	0.002121	4.72	0.002130	4.74
i080-121	0.001794	0.002228	1.24	0.003181	1.77	0.006851	3.82	0.006732	3.75	0.007081	3.95
i080-122	0.001789	0.002229	1.25	0.003209	1.79	0.006843	3.83	0.006873	3.84	0.007292	4.08
i080-123	0.001798	0.002209	1.23	0.003162	1.76	0.006817	3.79	0.006850	3.81	0.007340	4.08
i080-124	0.001795	0.002229	1.24	0.003174	1.77	0.006760	3.77	0.006604	3.68	0.007157	3.99
i080-125	0.001786	0.002207	1.24	0.003153	1.77	0.007015	3.93	0.006859	3.84	0.007276	4.07
i080-131	0.000314	0.000591	1.88	0.001164	3.71	0.001605	5.11	0.001675	5.33	0.001728	5.50
i080-132	0.000317	0.000514	1.62	0.001163	3.67	0.001587	5.01	0.001621	5.11	0.001647	5.20
i080-133	0.000320	0.000571	1.78	0.001093	3.42	0.001488	4.65	0.001682	5.26	0.001668	5.21
i080-134	0.000314	0.000586	1.87	0.001117	3.56	0.001491	4.75	0.001658	5.28	0.001604	5.11
i080-135	0.000315	0.000512	1.63	0.001133	3.60	0.001547	4.91	0.001599	5.08	0.001750	5.56
i080-141	0.000611	0.000934	1.53	0.001439	2.36	0.002521	4.13	0.002658	4.35	0.002686	4.40
i080-142	0.000622	0.000917	1.47	0.001403	2.26	0.002573	4.14	0.002518	4.05	0.002824	4.54
i080-143	0.000616	0.000803	1.30	0.001366	2.22	0.002580	4.19	0.002531	4.11	0.002817	4.57
i080-144	0.000611	0.000886	1.45	0.001345	2.20	0.002613	4.28	0.002608	4.27	0.002818	4.61
i080-145	0.000628	0.000811	1.29	0.001364	2.17	0.002374	3.78	0.002593	4.13	0.002750	4.38

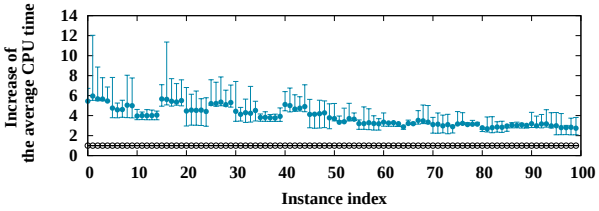
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
i080-201	0.000437	0.000692	1.58	0.001275	2.92	0.001620	3.71	0.001676	3.84	0.001715	3.92
i080-202	0.000456	0.000697	1.53	0.001200	2.63	0.001522	3.34	0.001723	3.78	0.001673	3.67
i080-203	0.000447	0.000661	1.48	0.001303	2.91	0.001513	3.38	0.001736	3.88	0.001699	3.80
i080-204	0.000436	0.000631	1.45	0.001474	3.38	0.001617	3.71	0.001733	3.97	0.001744	4.00
i080-205	0.000441	0.000644	1.46	0.001302	2.95	0.001606	3.64	0.001690	3.83	0.001721	3.90
i080-211	0.000686	0.000942	1.37	0.001597	2.33	0.002198	3.20	0.002353	3.43	0.002434	3.55
i080-212	0.000692	0.000998	1.44	0.001478	2.14	0.002213	3.20	0.002294	3.32	0.002493	3.60
i080-213	0.000704	0.000947	1.35	0.001378	1.96	0.002314	3.29	0.002439	3.46	0.002377	3.38
i080-214	0.000698	0.000946	1.36	0.001425	2.04	0.002238	3.21	0.002295	3.29	0.002383	3.41
i080-215	0.000701	0.000938	1.34	0.001547	2.21	0.002231	3.18	0.002361	3.37	0.002425	3.46
i080-221	0.002245	0.002759	1.23	0.003733	1.66	0.007520	3.35	0.007309	3.26	0.007973	3.55
i080-222	0.002277	0.002675	1.17	0.003710	1.63	0.007434	3.26	0.007335	3.22	0.007795	3.42
i080-223	0.002263	0.002728	1.21	0.003703	1.64	0.007455	3.29	0.007314	3.23	0.007695	3.40
i080-224	0.002291	0.002741	1.20	0.003678	1.61	0.007322	3.20	0.007242	3.16	0.007586	3.31
i080-225	0.002564	0.002720	1.06	0.003763	1.47	0.007384	2.88	0.007219	2.82	0.007878	3.07
i080-231	0.000517	0.000675	1.31	0.001442	2.79	0.001665	3.22	0.001783	3.45	0.001814	3.51
i080-232	0.000509	0.000754	1.48	0.001381	2.71	0.001625	3.19	0.001800	3.54	0.001829	3.59
i080-233	0.000499	0.000728	1.46	0.001376	2.76	0.001759	3.53	0.001877	3.76	0.001928	3.86
i080-234	0.000504	0.000749	1.49	0.001288	2.56	0.001752	3.48	0.001932	3.83	0.001922	3.81
i080-235	0.000497	0.000722	1.45	0.001393	2.80	0.001662	3.34	0.001833	3.69	0.001805	3.63
i080-241	0.000917	0.001171	1.28	0.001739	1.90	0.002846	3.10	0.003070	3.35	0.003076	3.35
i080-242	0.000907	0.001136	1.25	0.001606	1.77	0.002852	3.14	0.002879	3.17	0.002953	3.26
i080-243	0.000906	0.001155	1.27	0.001617	1.78	0.002697	2.98	0.002801	3.09	0.003104	3.43
i080-244	0.000916	0.001222	1.33	0.001722	1.88	0.002839	3.10	0.002977	3.25	0.003037	3.32
i080-245	0.000952	0.001247	1.31	0.001664	1.75	0.002751	2.89	0.002877	3.02	0.002984	3.13
i080-301	0.000517	0.000730	1.41	0.001579	3.05	0.001636	3.16	0.001758	3.40	0.001702	3.29
i080-302	0.000510	0.000736	1.44	0.001338	2.62	0.001646	3.23	0.001691	3.32	0.001736	3.40
i080-303	0.000537	0.000738	1.37	0.001250	2.33	0.001668	3.11	0.001768	3.29	0.001742	3.24
i080-304	0.000519	0.000693	1.34	0.001541	2.97	0.001631	3.14	0.001785	3.44	0.001751	3.37
i080-305	0.000525	0.000762	1.45	0.001607	3.06	0.001653	3.15	0.001744	3.32	0.001761	3.35
i080-311	0.000821	0.001109	1.35	0.001710	2.08	0.002284	2.78	0.002341	2.85	0.002511	3.06
i080-312	0.000853	0.001110	1.30	0.001681	1.97	0.002284	2.68	0.002506	2.94	0.002484	2.91
i080-313	0.000832	0.001105	1.33	0.001742	2.09	0.002330	2.80	0.002389	2.87	0.002620	3.15
i080-314	0.000815	0.001052	1.29	0.001585	1.94	0.002330	2.86	0.002575	3.16	0.002625	3.22
i080-315	0.000821	0.001092	1.33	0.001583	1.93	0.002327	2.83	0.002479	3.02	0.002532	3.08
i080-321	0.002506	0.002999	1.20	0.003959	1.58	0.007424	2.96	0.007337	2.93	0.007893	3.15
i080-322	0.002493	0.003009	1.21	0.003770	1.51	0.007602	3.05	0.007617	3.06	0.008026	3.22
i080-323	0.002477	0.002978	1.20	0.003854	1.56	0.007473	3.02	0.007433	3.00	0.007925	3.20
i080-324	0.002484	0.002959	1.19	0.003777	1.52	0.007664	3.09	0.007494	3.02	0.007812	3.14
i080-325	0.002528	0.003005	1.19	0.003852	1.52	0.007643	3.02	0.007540	2.98	0.008020	3.17
i080-331	0.000597	0.000812	1.36	0.001401	2.35	0.001906	3.19	0.001994	3.34	0.001995	3.34
i080-332	0.000591	0.000817	1.38	0.001649	2.79	0.001762	2.98	0.001855	3.14	0.001962	3.32
i080-333	0.000597	0.000844	1.41	0.001407	2.36	0.001892	3.17	0.001947	3.26	0.001957	3.28
i080-334	0.000583	0.000835	1.43	0.001510	2.59	0.001850	3.17	0.001999	3.43	0.002023	3.47
i080-335	0.000604	0.000881	1.46	0.001360	2.25	0.001793	2.97	0.001913	3.17	0.001986	3.29
i080-341	0.001061	0.001357	1.28	0.001842	1.74	0.003167	2.98	0.003147	2.97	0.003284	3.10
i080-342	0.001058	0.001404	1.33	0.001798	1.70	0.002961	2.80	0.002971	2.81	0.003156	2.98
i080-343	0.001062	0.001394	1.31	0.001878	1.77	0.002978	2.80	0.003148	2.96	0.003285	3.09
i080-344	0.001065	0.001276	1.20	0.001803	1.69	0.003018	2.83	0.003150	2.96	0.003286	3.09
i080-345	0.001058	0.001295	1.22	0.001850	1.75	0.002895	2.74	0.003166	2.99	0.003050	2.88
ave	0.000840	0.001123	1.34	0.001749	2.08	0.002986	3.55	0.003061	3.64	0.003190	3.80



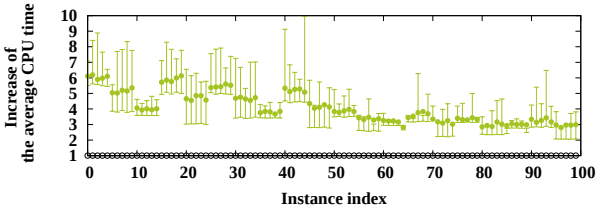
(a) degree centrality



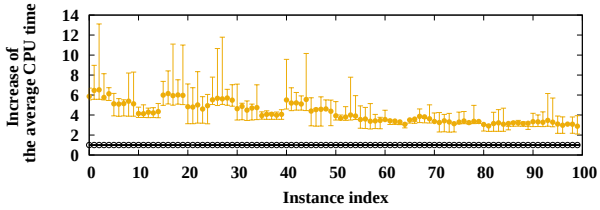
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality



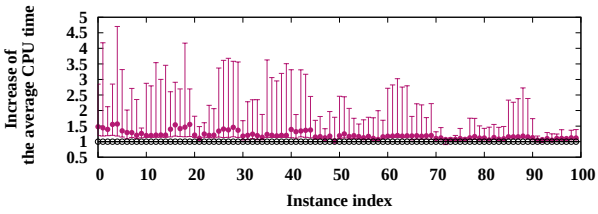
(e) edge betweenness centrality

Figure 3.36: Increase of CPU time from the original method (SPH, I080)

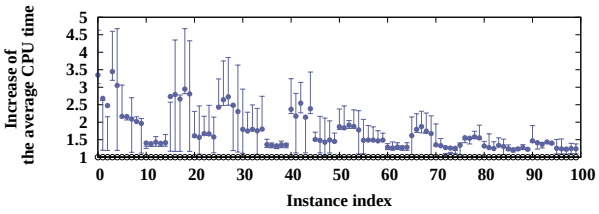
Table 3.29: CPU time [s] (SPH, I160)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
i160-001	0.000535	0.000791	1.48	0.001792	3.35	0.004286	8.01	0.004717	8.82	0.004787	8.95
i160-002	0.000522	0.000754	1.44	0.001399	2.68	0.004027	7.71	0.004465	8.55	0.004697	9.00
i160-003	0.000541	0.000754	1.39	0.001339	2.48	0.004110	7.60	0.004693	8.67	0.004807	8.89
i160-004	0.000519	0.000807	1.55	0.001789	3.45	0.003935	7.58	0.004516	8.70	0.004669	9.00
i160-005	0.000515	0.000807	1.57	0.001571	3.05	0.004148	8.05	0.004654	9.04	0.004790	9.30
i160-011	0.001027	0.001383	1.35	0.002224	2.17	0.006366	6.20	0.006781	6.60	0.007093	6.91
i160-012	0.001061	0.001371	1.29	0.002293	2.16	0.006460	6.09	0.007062	6.66	0.007397	6.97
i160-013	0.001053	0.001358	1.29	0.002202	2.09	0.006306	5.99	0.006756	6.42	0.007084	6.73
i160-014	0.001108	0.001335	1.20	0.002238	2.02	0.006486	5.85	0.007047	6.36	0.007182	6.48
i160-015	0.001092	0.001379	1.26	0.002148	1.97	0.006442	5.90	0.007049	6.46	0.007154	6.55
i160-021	0.006958	0.008329	1.20	0.009768	1.40	0.040522	5.82	0.039504	5.68	0.041660	5.99
i160-022	0.006948	0.008322	1.20	0.009625	1.39	0.040890	5.89	0.039882	5.74	0.042621	6.13
i160-023	0.006822	0.008254	1.21	0.009752	1.43	0.040786	5.98	0.040565	5.95	0.042596	6.24
i160-024	0.006882	0.008358	1.21	0.009596	1.39	0.040687	5.91	0.039511	5.74	0.042342	6.15
i160-025	0.006819	0.008256	1.21	0.009651	1.42	0.040558	5.95	0.039469	5.79	0.041738	6.12
i160-031	0.000624	0.000874	1.40	0.001706	2.73	0.004622	7.41	0.005208	8.35	0.005342	8.56
i160-032	0.000616	0.000948	1.54	0.001719	2.79	0.004452	7.23	0.004991	8.10	0.005294	8.59
i160-033	0.000614	0.000872	1.42	0.001636	2.66	0.004543	7.40	0.005154	8.39	0.005305	8.64
i160-034	0.000625	0.000916	1.47	0.001841	2.95	0.004569	7.31	0.005205	8.33	0.005226	8.36
i160-035	0.000612	0.000951	1.55	0.001718	2.81	0.004539	7.42	0.005089	8.32	0.005312	8.68
i160-041	0.002093	0.002536	1.21	0.003373	1.61	0.012007	5.74	0.012310	5.88	0.012847	6.14
i160-042	0.002102	0.002330	1.11	0.003298	1.57	0.011923	5.67	0.012514	5.95	0.012965	6.17
i160-043	0.002113	0.002635	1.25	0.003534	1.67	0.012249	5.80	0.012677	6.00	0.012954	6.13
i160-044	0.002121	0.002548	1.20	0.003532	1.67	0.012450	5.87	0.012627	5.95	0.012875	6.07
i160-045	0.002124	0.002568	1.21	0.003353	1.58	0.012425	5.85	0.012731	5.99	0.012839	6.04
i160-101	0.000780	0.001045	1.34	0.001895	2.43	0.004421	5.67	0.004920	6.31	0.005071	6.50
i160-102	0.000761	0.001069	1.40	0.002010	2.64	0.004356	5.72	0.004840	6.36	0.005286	6.95
i160-103	0.000756	0.001041	1.38	0.002059	2.72	0.004341	5.74	0.004868	6.44	0.005098	6.74
i160-104	0.000757	0.001106	1.46	0.001881	2.48	0.004374	5.78	0.004939	6.52	0.005070	6.70
i160-105	0.000757	0.001034	1.37	0.001745	2.31	0.004422	5.84	0.004934	6.52	0.005135	6.78
i160-111	0.001509	0.001776	1.18	0.002707	1.79	0.006755	4.48	0.007270	4.82	0.007592	5.03
i160-112	0.001497	0.001802	1.20	0.002617	1.75	0.006811	4.55	0.007259	4.85	0.007551	5.04
i160-113	0.001520	0.001888	1.24	0.002730	1.80	0.007078	4.66	0.007387	4.86	0.007822	5.15
i160-114	0.001487	0.001776	1.19	0.002608	1.75	0.006688	4.50	0.007640	5.14	0.007807	5.25
i160-115	0.001495	0.001706	1.14	0.002694	1.80	0.006561	4.39	0.007218	4.83	0.007541	5.04
i160-121	0.007834	0.009652	1.23	0.010646	1.36	0.041524	5.30	0.040346	5.15	0.042756	5.46
i160-122	0.007866	0.009480	1.21	0.010512	1.34	0.041738	5.31	0.040651	5.17	0.042728	5.43
i160-123	0.007963	0.009437	1.19	0.010546	1.32	0.041714	5.24	0.041119	5.16	0.044208	5.55
i160-124	0.007871	0.009470	1.20	0.010644	1.35	0.041887	5.32	0.041811	5.31	0.044334	5.63
i160-125	0.007837	0.009428	1.20	0.010538	1.34	0.041892	5.35	0.040324	5.15	0.042671	5.44
i160-131	0.000905	0.001259	1.39	0.002144	2.37	0.004882	5.39	0.005377	5.94	0.005524	6.10
i160-132	0.000924	0.001220	1.32	0.002008	2.17	0.004712	5.10	0.005234	5.66	0.005385	5.83
i160-133	0.000902	0.001210	1.34	0.002294	2.54	0.004881	5.41	0.005481	6.08	0.005675	6.29
i160-134	0.000921	0.001256	1.36	0.001975	2.14	0.004976	5.40	0.005481	5.95	0.005686	6.17
i160-135	0.000905	0.001245	1.38	0.002158	2.38	0.004637	5.12	0.005256	5.81	0.005514	6.09
i160-141	0.002829	0.003246	1.15	0.004263	1.51	0.013149	4.65	0.013187	4.66	0.013473	4.76
i160-142	0.002827	0.003284	1.16	0.004188	1.48	0.012805	4.53	0.013169	4.66	0.013500	4.78
i160-143	0.002794	0.003136	1.12	0.003992	1.43	0.012768	4.57	0.013008	4.66	0.013413	4.80
i160-144	0.002804	0.003303	1.18	0.004187	1.49	0.012566	4.48	0.013293	4.74	0.013704	4.89
i160-145	0.002815	0.002890	1.03	0.004086	1.45	0.012740	4.53	0.013403	4.76	0.013661	4.85

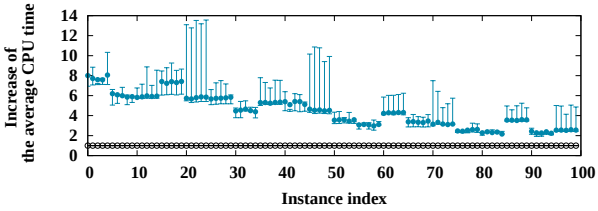
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
i160-201	0.001450	0.001722	1.19	0.002713	1.87	0.005137	3.54	0.005669	3.91	0.005921	4.08
i160-202	0.001430	0.001788	1.25	0.002629	1.84	0.005127	3.59	0.005784	4.04	0.005815	4.07
i160-203	0.001418	0.001653	1.17	0.002714	1.91	0.005090	3.59	0.005629	3.97	0.005750	4.06
i160-204	0.001415	0.001690	1.19	0.002659	1.88	0.004864	3.44	0.005761	4.07	0.005763	4.07
i160-205	0.001442	0.001679	1.16	0.002569	1.78	0.005173	3.59	0.005635	3.91	0.005800	4.02
i160-211	0.002623	0.002951	1.13	0.003887	1.48	0.008056	3.07	0.008588	3.27	0.009202	3.51
i160-212	0.002600	0.003038	1.17	0.003876	1.49	0.008092	3.11	0.008670	3.33	0.008927	3.43
i160-213	0.002620	0.002902	1.11	0.003901	1.49	0.008060	3.08	0.008531	3.26	0.009036	3.45
i160-214	0.002635	0.002819	1.07	0.003865	1.47	0.007827	2.97	0.008579	3.26	0.008802	3.34
i160-215	0.002623	0.003033	1.16	0.003900	1.49	0.008162	3.11	0.008670	3.31	0.008888	3.39
i160-221	0.010252	0.012058	1.18	0.013144	1.28	0.043325	4.23	0.043256	4.22	0.045838	4.47
i160-222	0.010244	0.012091	1.18	0.012950	1.26	0.043896	4.29	0.043548	4.25	0.045883	4.48
i160-223	0.010243	0.012258	1.20	0.013222	1.29	0.043827	4.28	0.043089	4.21	0.045445	4.44
i160-224	0.010224	0.012041	1.18	0.012964	1.27	0.044118	4.32	0.043840	4.29	0.045604	4.46
i160-225	0.010214	0.012130	1.19	0.013121	1.28	0.044071	4.31	0.043671	4.28	0.045931	4.50
i160-231	0.001688	0.001995	1.18	0.002728	1.62	0.005710	3.38	0.006358	3.77	0.006470	3.83
i160-232	0.001675	0.001994	1.19	0.003006	1.79	0.005689	3.40	0.006382	3.81	0.006474	3.87
i160-233	0.001683	0.001977	1.17	0.003157	1.88	0.005621	3.34	0.006364	3.78	0.006525	3.88
i160-234	0.001700	0.002022	1.19	0.002965	1.74	0.005613	3.30	0.006239	3.67	0.006348	3.73
i160-235	0.001661	0.001990	1.20	0.002782	1.67	0.005748	3.46	0.006199	3.73	0.006414	3.86
i160-241	0.004526	0.005001	1.10	0.006129	1.35	0.014225	3.14	0.015321	3.39	0.015614	3.45
i160-242	0.004483	0.005017	1.12	0.005961	1.33	0.014907	3.33	0.015024	3.35	0.015480	3.45
i160-243	0.004699	0.005020	1.07	0.005992	1.28	0.014842	3.16	0.015055	3.20	0.015336	3.26
i160-244	0.004811	0.005151	1.07	0.006067	1.26	0.014938	3.10	0.014664	3.05	0.015307	3.18
i160-245	0.004720	0.005073	1.07	0.005920	1.25	0.014895	3.16	0.015098	3.20	0.015251	3.23
i160-301	0.002440	0.002689	1.10	0.003274	1.34	0.006028	2.47	0.006729	2.76	0.006855	2.81
i160-302	0.002476	0.002655	1.07	0.003854	1.56	0.006037	2.44	0.006613	2.67	0.006782	2.74
i160-303	0.002377	0.002667	1.12	0.003665	1.54	0.006038	2.54	0.006571	2.76	0.006886	2.90
i160-304	0.002325	0.002707	1.16	0.003683	1.58	0.006062	2.61	0.006513	2.80	0.006631	2.85
i160-305	0.002355	0.002627	1.12	0.003657	1.55	0.006177	2.62	0.006727	2.86	0.006925	2.94
i160-311	0.004176	0.004662	1.12	0.005514	1.32	0.009511	2.28	0.010014	2.40	0.010374	2.48
i160-312	0.004226	0.004462	1.06	0.005378	1.27	0.010057	2.38	0.010208	2.42	0.010529	2.49
i160-313	0.004234	0.004796	1.13	0.005282	1.25	0.009747	2.30	0.010041	2.37	0.010296	2.43
i160-314	0.004176	0.004521	1.08	0.005580	1.34	0.009854	2.36	0.010245	2.45	0.010532	2.52
i160-315	0.004244	0.004640	1.09	0.005543	1.31	0.009176	2.16	0.010049	2.37	0.010329	2.43
i160-321	0.013460	0.015559	1.16	0.016683	1.24	0.047832	3.55	0.047136	3.50	0.049508	3.68
i160-322	0.013503	0.015622	1.16	0.016193	1.20	0.047859	3.54	0.046878	3.47	0.049420	3.66
i160-323	0.013445	0.015537	1.16	0.016627	1.24	0.047327	3.52	0.046715	3.47	0.049759	3.70
i160-324	0.013441	0.015843	1.18	0.017148	1.28	0.048333	3.60	0.047086	3.50	0.049333	3.67
i160-325	0.013483	0.015564	1.15	0.016526	1.23	0.048188	3.57	0.046912	3.48	0.049098	3.64
i160-331	0.002829	0.003195	1.13	0.004154	1.47	0.006888	2.43	0.007470	2.64	0.007687	2.72
i160-332	0.002936	0.003088	1.05	0.004155	1.42	0.006682	2.28	0.007522	2.56	0.007537	2.57
i160-333	0.002954	0.003146	1.06	0.004039	1.37	0.006679	2.26	0.007244	2.45	0.007522	2.55
i160-334	0.002893	0.003180	1.10	0.004151	1.43	0.006887	2.38	0.007485	2.59	0.007505	2.59
i160-335	0.002939	0.003083	1.05	0.004120	1.40	0.006581	2.24	0.007493	2.55	0.007630	2.60
i160-341	0.006748	0.007440	1.10	0.008458	1.25	0.017172	2.54	0.017299	2.56	0.017727	2.63
i160-342	0.006744	0.007447	1.10	0.008350	1.24	0.017238	2.56	0.017062	2.53	0.017699	2.62
i160-343	0.006847	0.007512	1.10	0.008437	1.23	0.017278	2.52	0.017228	2.52	0.017857	2.61
i160-344	0.006690	0.007514	1.12	0.008360	1.25	0.017277	2.58	0.017538	2.62	0.017618	2.63
i160-345	0.006806	0.007602	1.12	0.008440	1.24	0.017387	2.55	0.017463	2.57	0.017754	2.61
ave	0.003735	0.004357	1.17	0.005290	1.42	0.015027	4.02	0.015262	4.09	0.015935	4.27



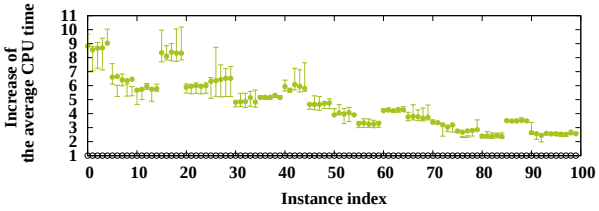
(a) degree centrality



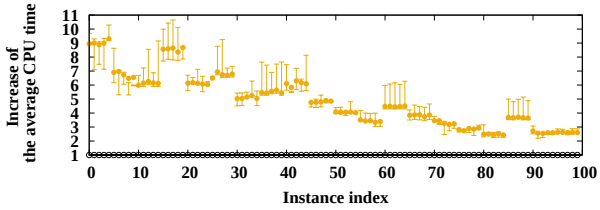
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality



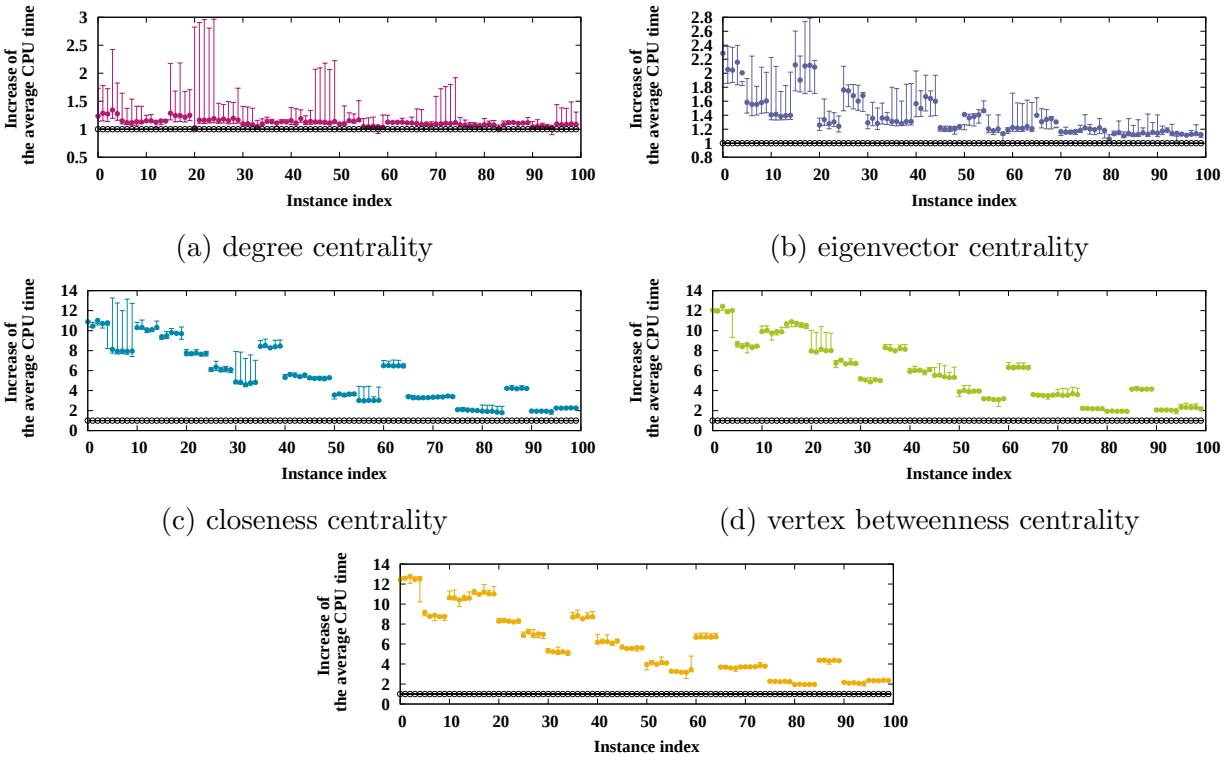
(e) edge betweenness centrality

Figure 3.37: Increase of CPU time from the original method (SPH, I160)

Table 3.30: CPU time [s] (SPH, I320)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
i320-001	0.001447	0.001782	1.23	0.003307	2.29	0.015745	10.88	0.017431	12.05	0.017989	12.43
i320-002	0.001447	0.001860	1.29	0.002971	2.05	0.015119	10.45	0.017320	11.97	0.018202	12.58
i320-003	0.001404	0.001785	1.27	0.002869	2.04	0.015478	11.02	0.017461	12.44	0.017921	12.76
i320-004	0.001461	0.001962	1.34	0.003151	2.16	0.015647	10.71	0.017387	11.90	0.018192	12.45
i320-005	0.001457	0.001861	1.28	0.002923	2.01	0.015704	10.78	0.017494	12.01	0.018232	12.51
i320-011	0.003248	0.003726	1.15	0.005147	1.58	0.026366	8.12	0.028090	8.65	0.029614	9.12
i320-012	0.003239	0.003618	1.12	0.005040	1.56	0.025722	7.94	0.027226	8.41	0.028424	8.78
i320-013	0.003225	0.003603	1.12	0.005013	1.55	0.025655	7.96	0.027785	8.62	0.028673	8.89
i320-014	0.003271	0.003713	1.14	0.005171	1.58	0.025902	7.92	0.027184	8.31	0.028603	8.74
i320-015	0.003235	0.003651	1.13	0.005186	1.60	0.025731	7.95	0.027353	8.46	0.028303	8.75
i320-021	0.029394	0.033778	1.15	0.041297	1.40	0.302985	10.31	0.291395	9.91	0.313199	10.66
i320-022	0.029345	0.033775	1.15	0.041471	1.41	0.302768	10.32	0.294965	10.05	0.311803	10.63
i320-023	0.029902	0.033627	1.12	0.041298	1.38	0.301517	10.08	0.291666	9.75	0.311127	10.40
i320-024	0.029565	0.033922	1.15	0.041202	1.39	0.298670	10.10	0.292924	9.91	0.315293	10.66
i320-025	0.029301	0.033535	1.14	0.040996	1.40	0.302683	10.33	0.289660	9.89	0.310476	10.60
i320-031	0.001754	0.002256	1.29	0.003715	2.12	0.016467	9.39	0.018743	10.69	0.019772	11.27
i320-032	0.001754	0.002182	1.24	0.003337	1.90	0.016626	9.48	0.019108	10.89	0.019245	10.97
i320-033	0.001750	0.002175	1.24	0.003681	2.10	0.017147	9.80	0.018773	10.73	0.019602	11.20
i320-034	0.001767	0.002144	1.21	0.003735	2.11	0.017199	9.73	0.018651	10.56	0.019476	11.02
i320-035	0.001773	0.002212	1.25	0.003702	2.09	0.017196	9.70	0.018669	10.53	0.019532	11.02
i320-041	0.009153	0.009418	1.03	0.011521	1.26	0.071011	7.76	0.073014	7.98	0.075857	8.29
i320-042	0.009313	0.010830	1.16	0.012451	1.34	0.071664	7.70	0.073186	7.86	0.077898	8.36
i320-043	0.009096	0.010549	1.16	0.011611	1.28	0.070575	7.76	0.074003	8.14	0.075347	8.28
i320-044	0.009186	0.010673	1.16	0.011977	1.30	0.070162	7.64	0.073390	7.99	0.075465	8.22
i320-045	0.009211	0.010970	1.19	0.011435	1.24	0.071068	7.72	0.073809	8.01	0.076338	8.29
i320-101	0.002746	0.003173	1.16	0.004837	1.76	0.016783	6.11	0.018500	6.74	0.018847	6.86
i320-102	0.002674	0.003161	1.18	0.004677	1.75	0.016984	6.35	0.018837	7.04	0.019233	7.19
i320-103	0.002755	0.003178	1.15	0.004627	1.68	0.016746	6.08	0.018382	6.67	0.019019	6.90
i320-104	0.002743	0.003277	1.19	0.004392	1.60	0.016831	6.14	0.018565	6.77	0.019217	7.01
i320-105	0.002801	0.003261	1.16	0.004712	1.68	0.017063	6.09	0.018834	6.72	0.019488	6.96
i320-111	0.005892	0.006458	1.10	0.007614	1.29	0.028699	4.87	0.030470	5.17	0.031255	5.30
i320-112	0.005936	0.006527	1.10	0.008040	1.35	0.028665	4.83	0.030026	5.06	0.031011	5.22
i320-113	0.005977	0.006451	1.08	0.007663	1.28	0.027520	4.60	0.029199	4.89	0.031077	5.20
i320-114	0.005869	0.006153	1.05	0.007977	1.36	0.028042	4.78	0.029866	5.09	0.030650	5.22
i320-115	0.005947	0.006509	1.09	0.008038	1.35	0.028731	4.83	0.029751	5.00	0.030616	5.15
i320-121	0.036286	0.041292	1.14	0.047576	1.31	0.305854	8.43	0.301051	8.30	0.315668	8.70
i320-122	0.036274	0.041009	1.13	0.047518	1.31	0.308882	8.52	0.295836	8.16	0.320651	8.84
i320-123	0.037162	0.041277	1.11	0.047604	1.28	0.307600	8.28	0.296963	7.99	0.316444	8.52
i320-124	0.036293	0.041114	1.13	0.047557	1.31	0.304601	8.39	0.298210	8.22	0.316256	8.71
i320-125	0.036338	0.041042	1.13	0.047632	1.31	0.307273	8.46	0.295675	8.14	0.317056	8.73
i320-131	0.003433	0.003959	1.15	0.005373	1.57	0.018303	5.33	0.020326	5.92	0.021164	6.16
i320-132	0.003364	0.003733	1.11	0.005035	1.50	0.018910	5.62	0.020259	6.02	0.020973	6.23
i320-133	0.003386	0.004024	1.19	0.005641	1.67	0.018697	5.52	0.020477	6.05	0.021116	6.24
i320-134	0.003465	0.003891	1.12	0.005679	1.64	0.018723	5.40	0.020344	5.87	0.021093	6.09
i320-135	0.003356	0.003822	1.14	0.005368	1.60	0.018646	5.56	0.020455	6.10	0.021075	6.28
i320-141	0.014327	0.016175	1.13	0.017513	1.22	0.075756	5.29	0.079134	5.52	0.081908	5.72
i320-142	0.014559	0.016441	1.13	0.017411	1.20	0.075954	5.22	0.080939	5.56	0.080801	5.55
i320-143	0.014601	0.016412	1.12	0.017592	1.20	0.076179	5.22	0.078658	5.39	0.081114	5.56
i320-144	0.014463	0.016088	1.11	0.017559	1.21	0.075719	5.24	0.076693	5.30	0.081723	5.65
i320-145	0.014474	0.016474	1.14	0.017924	1.24	0.076655	5.30	0.077139	5.33	0.081190	5.61

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
i320-201	0.005542	0.006064	1.09	0.007810	1.41	0.019793	3.57	0.021527	3.88	0.021906	3.95
i320-202	0.005327	0.005863	1.10	0.007255	1.36	0.019552	3.67	0.021427	4.02	0.022047	4.14
i320-203	0.005567	0.006413	1.15	0.007686	1.38	0.019825	3.56	0.021618	3.88	0.022153	3.98
i320-204	0.005408	0.006172	1.14	0.007600	1.41	0.019775	3.66	0.021300	3.94	0.022468	4.15
i320-205	0.005364	0.006265	1.17	0.007856	1.46	0.019689	3.67	0.021218	3.96	0.021990	4.10
i320-211	0.011172	0.011713	1.05	0.013467	1.21	0.033768	3.02	0.035619	3.19	0.036682	3.28
i320-212	0.011142	0.011631	1.04	0.013122	1.18	0.033172	2.98	0.035587	3.19	0.036318	3.26
i320-213	0.011332	0.011921	1.05	0.013636	1.20	0.034296	3.03	0.035068	3.09	0.036111	3.19
i320-214	0.011401	0.011935	1.05	0.012978	1.14	0.034233	3.00	0.035237	3.09	0.036191	3.17
i320-215	0.011143	0.011527	1.03	0.013236	1.19	0.033807	3.03	0.035550	3.19	0.038309	3.44
i320-221	0.049145	0.055339	1.13	0.060202	1.22	0.318895	6.49	0.310433	6.32	0.327950	6.67
i320-222	0.049284	0.055211	1.12	0.059836	1.21	0.320965	6.51	0.309286	6.28	0.330505	6.71
i320-223	0.049172	0.055241	1.12	0.059631	1.21	0.318599	6.48	0.311966	6.34	0.329748	6.71
i320-224	0.049114	0.055231	1.12	0.060880	1.24	0.318475	6.48	0.310481	6.32	0.328256	6.68
i320-225	0.049241	0.054902	1.11	0.059505	1.21	0.317872	6.46	0.309533	6.29	0.331172	6.73
i320-231	0.006572	0.007300	1.11	0.009184	1.40	0.022277	3.39	0.023700	3.61	0.024252	3.69
i320-232	0.006661	0.007352	1.10	0.008705	1.31	0.021887	3.29	0.023580	3.54	0.024532	3.68
i320-233	0.006738	0.007303	1.08	0.009019	1.34	0.022150	3.29	0.023567	3.50	0.024291	3.61
i320-234	0.006762	0.007407	1.10	0.009128	1.35	0.022277	3.29	0.023598	3.49	0.024343	3.60
i320-235	0.006652	0.007162	1.08	0.008675	1.30	0.021980	3.30	0.023660	3.56	0.024573	3.69
i320-241	0.024886	0.027196	1.09	0.029013	1.17	0.083218	3.34	0.089790	3.61	0.092643	3.72
i320-242	0.024524	0.026885	1.10	0.028449	1.16	0.082402	3.36	0.086520	3.53	0.091175	3.72
i320-243	0.024528	0.027145	1.11	0.028432	1.16	0.082478	3.36	0.086604	3.53	0.091455	3.73
i320-244	0.023982	0.026445	1.10	0.028029	1.17	0.082986	3.46	0.088693	3.70	0.092035	3.84
i320-245	0.024155	0.026992	1.12	0.028583	1.18	0.082111	3.40	0.086521	3.58	0.091356	3.78
i320-301	0.013616	0.014700	1.08	0.016594	1.22	0.028516	2.09	0.030245	2.22	0.031066	2.28
i320-302	0.013601	0.014546	1.07	0.016374	1.20	0.028435	2.09	0.030168	2.22	0.030664	2.25
i320-303	0.014024	0.014895	1.06	0.016739	1.19	0.028699	2.05	0.030611	2.18	0.031377	2.24
i320-304	0.013973	0.014981	1.07	0.017053	1.22	0.028401	2.03	0.030619	2.19	0.031720	2.27
i320-305	0.014058	0.014973	1.07	0.016639	1.18	0.028182	2.00	0.030879	2.20	0.031560	2.24
i320-311	0.026200	0.028287	1.08	0.027780	1.06	0.050949	1.94	0.050536	1.93	0.051091	1.95
i320-312	0.026216	0.028443	1.08	0.029749	1.13	0.050327	1.92	0.051407	1.96	0.051955	1.98
i320-313	0.026156	0.028075	1.07	0.030108	1.15	0.050578	1.93	0.050396	1.93	0.050885	1.95
i320-314	0.026206	0.026200	1.00	0.028945	1.10	0.048499	1.85	0.051005	1.95	0.051453	1.96
i320-315	0.026077	0.028164	1.08	0.029718	1.14	0.046780	1.79	0.050396	1.93	0.050988	1.96
i320-321	0.083580	0.093639	1.12	0.093829	1.12	0.352904	4.22	0.345748	4.14	0.365257	4.37
i320-322	0.083183	0.093167	1.12	0.093512	1.12	0.351905	4.23	0.347869	4.18	0.363159	4.37
i320-323	0.083871	0.092712	1.11	0.096294	1.15	0.351794	4.19	0.346555	4.13	0.362354	4.32
i320-324	0.083315	0.092609	1.11	0.093776	1.13	0.353375	4.24	0.345963	4.15	0.364776	4.38
i320-325	0.083526	0.093155	1.12	0.096459	1.15	0.351347	4.21	0.346535	4.15	0.361738	4.33
i320-331	0.016750	0.017438	1.04	0.019189	1.15	0.032767	1.96	0.034579	2.06	0.036231	2.16
i320-332	0.017114	0.018063	1.06	0.020083	1.17	0.033146	1.94	0.035108	2.05	0.035663	2.08
i320-333	0.016694	0.017610	1.05	0.019721	1.18	0.032448	1.94	0.034530	2.07	0.035440	2.12
i320-334	0.017014	0.017740	1.04	0.019377	1.14	0.032941	1.94	0.034709	2.04	0.035213	2.07
i320-335	0.017372	0.017982	1.04	0.019824	1.14	0.032975	1.90	0.034865	2.01	0.036241	2.09
i320-341	0.050334	0.055075	1.09	0.056823	1.13	0.113333	2.25	0.117365	2.33	0.119066	2.37
i320-342	0.051129	0.055238	1.08	0.056996	1.11	0.114882	2.25	0.118670	2.32	0.119764	2.34
i320-343	0.050688	0.055089	1.09	0.057209	1.13	0.114346	2.26	0.116427	2.30	0.117806	2.32
i320-344	0.050293	0.055082	1.10	0.057081	1.13	0.114119	2.27	0.116235	2.31	0.119253	2.37
i320-345	0.050664	0.054979	1.09	0.056917	1.12	0.114015	2.25	0.109396	2.16	0.119186	2.35
ave	0.019617	0.021722	1.11	0.023945	1.22	0.095651	4.88	0.095586	4.87	0.100491	5.12



(a) degree centrality

(b) eigenvector centrality

(c) closeness centrality

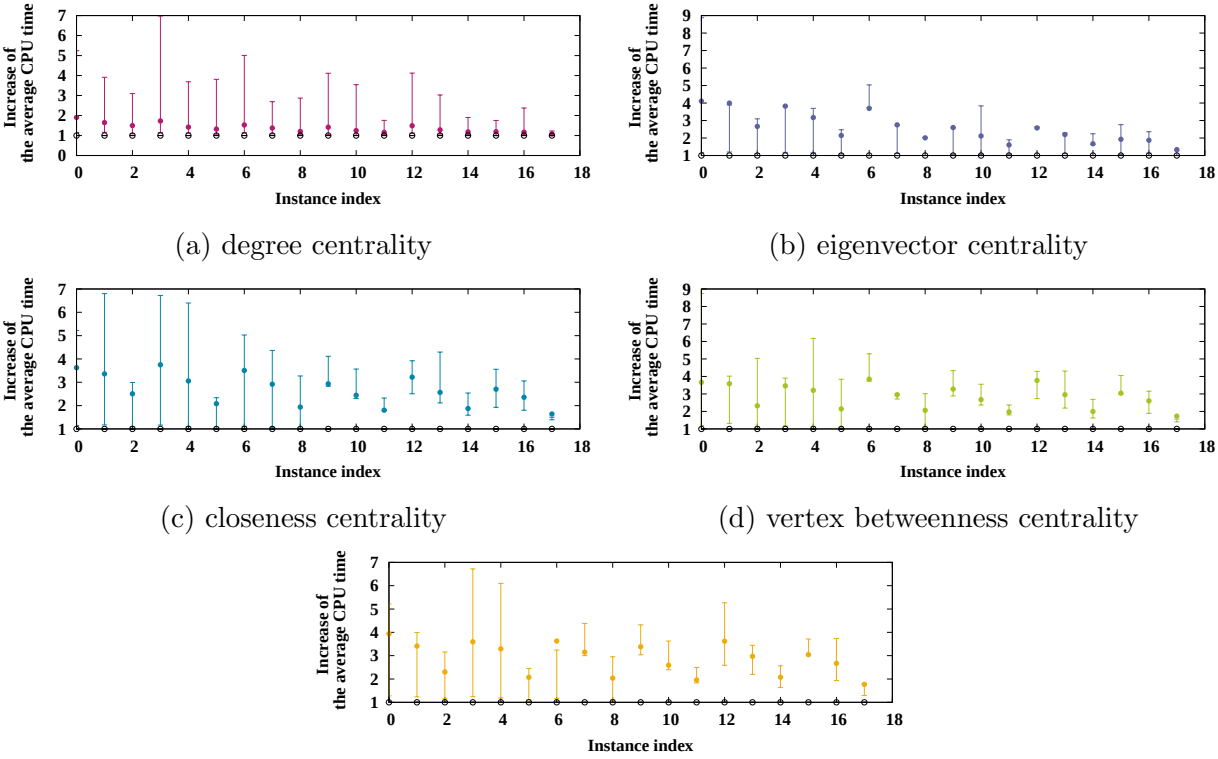
(d) vertex betweenness centrality

(e) edge betweenness centrality

Figure 3.38: Increase of CPU time from the original method (SPH, I320)

Table 3.31: CPU time [s] (DNH, B)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
b01	0.000220	0.000419	1.90	0.000904	4.11	0.000798	3.63	0.000806	3.66	0.000866	3.94
b02	0.000266	0.000438	1.65	0.001063	4.00	0.000894	3.36	0.000954	3.59	0.000907	3.41
b03	0.000453	0.000678	1.50	0.001206	2.66	0.001135	2.51	0.001051	2.32	0.001043	2.30
b04	0.000278	0.000481	1.73	0.001063	3.82	0.001043	3.75	0.000963	3.46	0.000999	3.59
b05	0.000341	0.000485	1.42	0.001081	3.17	0.001042	3.06	0.001093	3.21	0.001123	3.29
b06	0.000621	0.000821	1.32	0.001332	2.14	0.001294	2.08	0.001334	2.15	0.001286	2.07
b07	0.000400	0.000611	1.53	0.001478	3.69	0.001402	3.50	0.001538	3.85	0.001451	3.63
b08	0.000543	0.000746	1.37	0.001494	2.75	0.001583	2.92	0.001606	2.96	0.001711	3.15
b09	0.001055	0.001269	1.20	0.002122	2.01	0.002042	1.94	0.002175	2.06	0.002146	2.03
b10	0.000572	0.000811	1.42	0.001486	2.60	0.001678	2.93	0.001876	3.28	0.001936	3.38
b11	0.000774	0.000965	1.25	0.001640	2.12	0.001890	2.44	0.002073	2.68	0.002005	2.59
b12	0.001398	0.001619	1.16	0.002247	1.61	0.002522	1.80	0.002752	1.97	0.002739	1.96
b13	0.000687	0.001024	1.49	0.001768	2.57	0.002212	3.22	0.002591	3.77	0.002486	3.62
b14	0.000935	0.001201	1.28	0.002059	2.20	0.002399	2.57	0.002766	2.96	0.002779	2.97
b15	0.001847	0.002182	1.18	0.003083	1.67	0.003462	1.87	0.003682	1.99	0.003835	2.08
b16	0.001075	0.001282	1.19	0.002071	1.93	0.002909	2.71	0.003274	3.05	0.003274	3.05
b17	0.001371	0.001605	1.17	0.002568	1.87	0.003231	2.36	0.003569	2.60	0.003662	2.67
b18	0.002877	0.003109	1.08	0.003850	1.34	0.004718	1.64	0.004984	1.73	0.005083	1.77
ave	0.000827	0.001039	1.26	0.001711	2.07	0.001908	2.31	0.002057	2.49	0.002070	2.50



(a) degree centrality

(b) eigenvector centrality

(c) closeness centrality

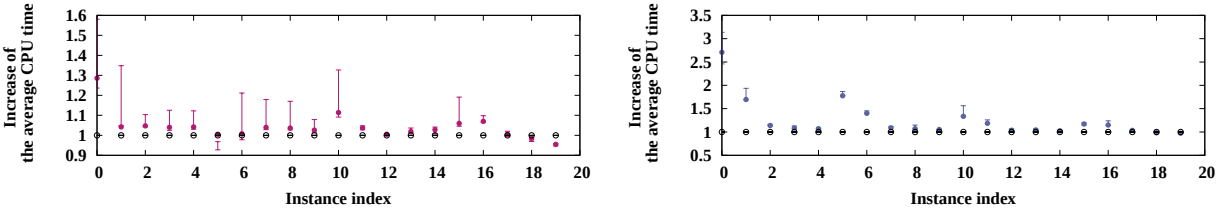
(d) vertex betweenness centrality

(e) edge betweenness centrality

Figure 3.39: Increase of CPU time from the original method (DNH, B)

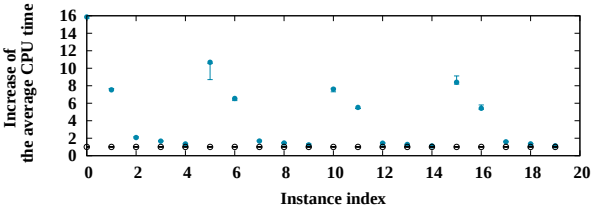
Table 3.32: CPU time [s] (DNH, C)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
c01	0.002299	0.002956	1.29	0.006227	2.71	0.036463	15.86	0.042646	18.55	0.044571	19.39
c02	0.005045	0.005258	1.04	0.008554	1.70	0.037994	7.53	0.045074	8.93	0.045881	9.09
c03	0.032299	0.033828	1.05	0.036747	1.14	0.067179	2.08	0.073217	2.27	0.075463	2.34
c04	0.052295	0.054393	1.04	0.056683	1.08	0.087052	1.66	0.094332	1.80	0.095871	1.83
c05	0.106038	0.110465	1.04	0.113570	1.07	0.143003	1.35	0.152341	1.44	0.152950	1.44
c06	0.004148	0.004169	1.01	0.007367	1.78	0.044296	10.68	0.053332	12.86	0.055213	13.31
c07	0.007241	0.007295	1.01	0.010143	1.40	0.047398	6.55	0.055953	7.73	0.058673	8.10
c08	0.060691	0.063081	1.04	0.066031	1.09	0.102077	1.68	0.112621	1.86	0.113748	1.87
c09	0.089366	0.092580	1.04	0.095723	1.07	0.130320	1.46	0.140920	1.58	0.145030	1.62
c10	0.188142	0.193017	1.03	0.197871	1.05	0.232549	1.24	0.247779	1.32	0.251323	1.34
c11	0.008506	0.009473	1.11	0.011370	1.34	0.064692	7.61	0.081734	9.61	0.083793	9.85
c12	0.012914	0.013391	1.04	0.015315	1.19	0.071277	5.52	0.086617	6.71	0.090082	6.98
c13	0.126825	0.127463	1.01	0.131699	1.04	0.182452	1.44	0.203199	1.60	0.215031	1.70
c14	0.187949	0.191536	1.02	0.196209	1.04	0.241323	1.28	0.270675	1.44	0.289026	1.54
c15	0.376158	0.386770	1.03	0.385872	1.03	0.423652	1.13	0.462969	1.23	0.493242	1.31
c16	0.020198	0.021401	1.06	0.023643	1.17	0.169573	8.40	0.195850	9.70	0.199139	9.86
c17	0.033375	0.035712	1.07	0.038382	1.15	0.181078	5.43	0.206608	6.19	0.213204	6.39
c18	0.250294	0.251615	1.01	0.258973	1.03	0.398306	1.59	0.452078	1.81	0.461294	1.84
c19	0.374196	0.368494	0.98	0.368793	0.99	0.507831	1.36	0.553483	1.48	0.600162	1.60
c20	0.771498	0.736546	0.95	0.751825	0.97	0.869895	1.13	0.953446	1.24	1.025968	1.33
ave	0.129023	0.129021	1.00	0.132428	1.03	0.192305	1.49	0.213565	1.66	0.224270	1.74

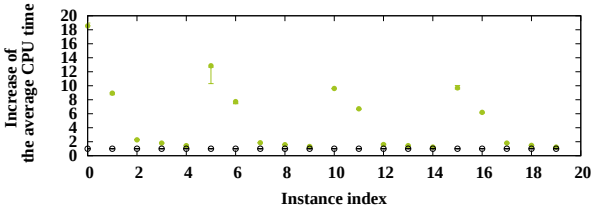


(a) degree centrality

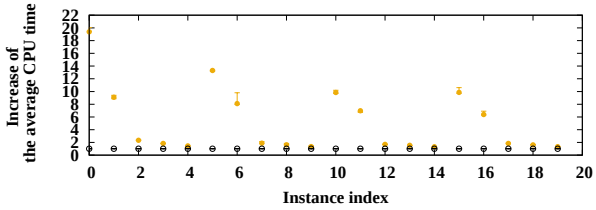
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality

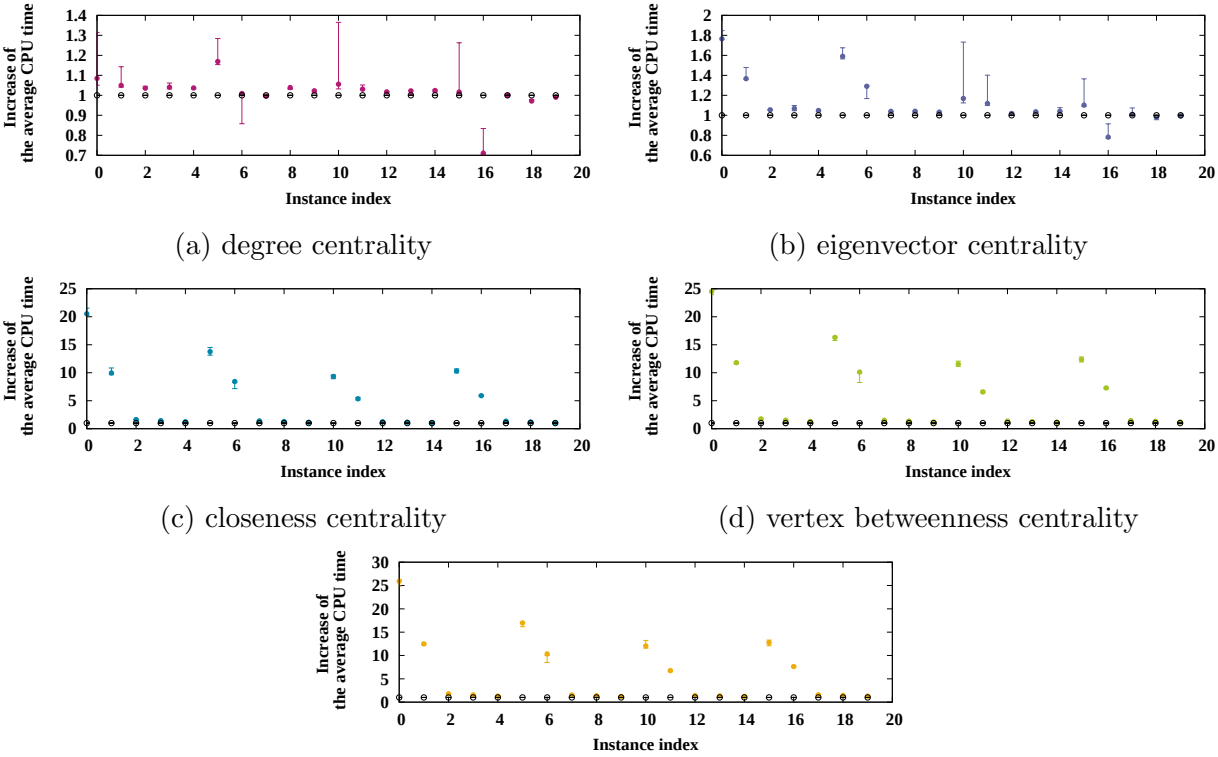


(e) edge betweenness centrality

Figure 3.40: Increase of CPU time from the original method (DNH, C)

Table 3.33: CPU time [s] (DNH, D)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
d01	0.007313	0.007933	1.08	0.012908	1.77	0.150196	20.54	0.179334	24.52	0.189605	25.93
d02	0.016715	0.017543	1.05	0.022840	1.37	0.166443	9.96	0.197181	11.80	0.208697	12.49
d03	0.244831	0.253552	1.04	0.258209	1.05	0.397534	1.62	0.429324	1.75	0.439877	1.80
d04	0.366508	0.381397	1.04	0.391427	1.07	0.522394	1.43	0.561023	1.53	0.566441	1.55
d05	0.735605	0.762708	1.04	0.771232	1.05	0.904250	1.23	0.952599	1.29	0.960025	1.31
d06	0.014156	0.016549	1.17	0.022513	1.59	0.194756	13.76	0.230990	16.32	0.240320	16.98
d07	0.023662	0.023880	1.01	0.030536	1.29	0.199398	8.43	0.239556	10.12	0.244384	10.33
d08	0.453780	0.451656	1.00	0.471604	1.04	0.631653	1.39	0.682695	1.50	0.690858	1.52
d09	0.647165	0.672215	1.04	0.673479	1.04	0.830863	1.28	0.873128	1.35	0.893085	1.38
d10	1.322830	1.353020	1.02	1.365600	1.03	1.504090	1.14	1.571830	1.19	1.596110	1.21
d11	0.029155	0.030772	1.06	0.034068	1.17	0.271822	9.32	0.336441	11.54	0.350774	12.03
d12	0.056612	0.058349	1.03	0.063365	1.12	0.302639	5.35	0.371917	6.57	0.382110	6.75
d13	0.920823	0.936751	1.02	0.937840	1.02	1.136455	1.23	1.247678	1.35	1.305372	1.42
d14	1.390310	1.421520	1.02	1.440000	1.04	1.612860	1.16	1.755200	1.26	1.855030	1.33
d15	2.725090	2.790180	1.02	2.833720	1.04	2.946440	1.08	3.159010	1.16	3.354340	1.23
d16	0.066609	0.067688	1.02	0.073381	1.10	0.685917	10.30	0.823334	12.36	0.850114	12.76
d17	0.122433	0.086934	0.71	0.095680	0.78	0.722413	5.90	0.892647	7.29	0.934493	7.63
d18	1.815680	1.815800	1.00	1.832810	1.01	2.423100	1.33	2.641050	1.45	2.843370	1.57
d19	2.803780	2.723770	0.97	2.752760	0.98	3.357950	1.20	3.706540	1.32	4.023010	1.43
d20	5.496720	5.444730	0.99	5.468100	0.99	5.952150	1.08	6.510130	1.18	7.035080	1.28
ave	0.917132	0.919855	1.00	0.931051	1.02	1.186350	1.29	1.302930	1.42	1.379190	1.50



(a) degree centrality

(b) eigenvector centrality

(c) closeness centrality

(d) vertex betweenness centrality

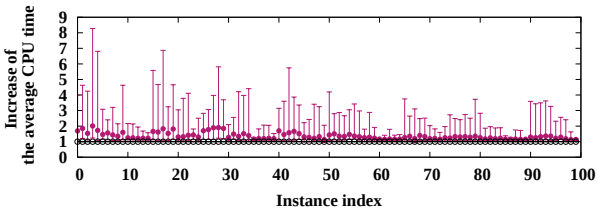
(e) edge betweenness centrality

Figure 3.41: Increase of CPU time from the original method (DNH, D)

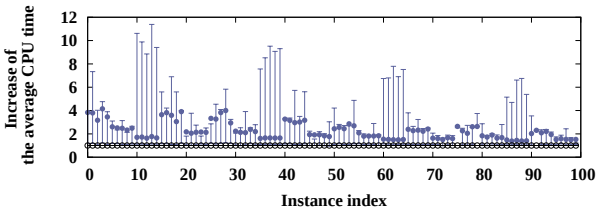
Table 3.34: CPU time [s] (DNH, I080)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
i080-001	0.000274	0.000463	1.69	0.001050	3.83	0.001368	4.99	0.001572	5.74	0.001485	5.42
i080-002	0.000285	0.000527	1.85	0.001082	3.80	0.001460	5.12	0.001528	5.36	0.001619	5.68
i080-003	0.000287	0.000439	1.53	0.000909	3.17	0.001369	4.77	0.001411	4.92	0.001612	5.62
i080-004	0.000270	0.000542	2.01	0.001120	4.15	0.001406	5.21	0.001503	5.57	0.001478	5.47
i080-005	0.000304	0.000524	1.72	0.001051	3.46	0.001387	4.56	0.001576	5.18	0.001575	5.18
i080-011	0.000477	0.000697	1.46	0.001241	2.60	0.001927	4.04	0.002050	4.30	0.002078	4.36
i080-012	0.000461	0.000718	1.56	0.001156	2.51	0.001876	4.07	0.002047	4.44	0.002076	4.50
i080-013	0.000471	0.000675	1.43	0.001169	2.48	0.001846	3.92	0.002092	4.44	0.002103	4.46
i080-014	0.000519	0.000698	1.34	0.001207	2.33	0.002075	4.00	0.002113	4.07	0.002201	4.24
i080-015	0.000457	0.000729	1.60	0.001136	2.49	0.001996	4.37	0.002155	4.72	0.002057	4.50
i080-021	0.001825	0.002272	1.24	0.003109	1.70	0.006753	3.70	0.006938	3.80	0.007068	3.87
i080-022	0.001830	0.002277	1.24	0.003156	1.72	0.006824	3.73	0.006701	3.66	0.007002	3.83
i080-023	0.001891	0.002293	1.21	0.003080	1.63	0.006849	3.62	0.006895	3.65	0.007346	3.88
i080-024	0.001816	0.002215	1.22	0.003214	1.77	0.006838	3.77	0.006703	3.69	0.007150	3.94
i080-025	0.001943	0.002369	1.22	0.003215	1.65	0.006984	3.59	0.006932	3.57	0.007393	3.80
i080-031	0.000319	0.000529	1.66	0.001162	3.64	0.001589	4.98	0.001608	5.04	0.001690	5.30
i080-032	0.000302	0.000489	1.62	0.001153	3.82	0.001538	5.09	0.001600	5.30	0.001676	5.55
i080-033	0.000314	0.000573	1.82	0.001127	3.59	0.001511	4.81	0.001592	5.07	0.001703	5.42
i080-034	0.000331	0.000506	1.53	0.001010	3.05	0.001494	4.51	0.001674	5.06	0.001682	5.08
i080-035	0.000311	0.000564	1.81	0.001216	3.91	0.001540	4.95	0.001746	5.61	0.001702	5.47
i080-041	0.000643	0.000837	1.30	0.001395	2.17	0.002476	3.85	0.002657	4.13	0.002720	4.23
i080-042	0.000671	0.000884	1.32	0.001382	2.06	0.002574	3.84	0.002561	3.82	0.002893	4.31
i080-043	0.000631	0.000889	1.41	0.001347	2.13	0.002508	3.97	0.002699	4.28	0.002795	4.43
i080-044	0.000673	0.000965	1.43	0.001452	2.16	0.002605	3.87	0.002754	4.09	0.002597	3.86
i080-045	0.000640	0.000834	1.30	0.001364	2.13	0.002491	3.89	0.002597	4.06	0.002776	4.34
i080-101	0.000342	0.000582	1.70	0.001139	3.33	0.001459	4.27	0.001507	4.41	0.001555	4.55
i080-102	0.000332	0.000589	1.77	0.001087	3.27	0.001491	4.49	0.001537	4.63	0.001611	4.85
i080-103	0.000300	0.000568	1.89	0.001147	3.82	0.001479	4.93	0.001500	5.00	0.001528	5.09
i080-104	0.000290	0.000553	1.91	0.001160	4.00	0.001405	4.84	0.001602	5.52	0.001614	5.57
i080-105	0.000333	0.000616	1.85	0.000979	2.94	0.001538	4.62	0.001593	4.78	0.001587	4.77
i080-111	0.000627	0.000798	1.27	0.001399	2.23	0.002128	3.39	0.002238	3.57	0.002219	3.54
i080-112	0.000544	0.000808	1.49	0.001157	2.13	0.001932	3.55	0.002174	4.00	0.002225	4.09
i080-113	0.000588	0.000786	1.34	0.001241	2.11	0.002094	3.56	0.002279	3.88	0.002239	3.81
i080-114	0.000548	0.000829	1.51	0.001307	2.39	0.001993	3.64	0.002166	3.95	0.002251	4.11
i080-115	0.000537	0.000746	1.39	0.001179	2.20	0.002076	3.87	0.002208	4.11	0.002278	4.24
i080-121	0.002151	0.002517	1.17	0.003461	1.61	0.007119	3.31	0.007003	3.26	0.007362	3.42
i080-122	0.002064	0.002476	1.20	0.003407	1.65	0.007053	3.42	0.007082	3.43	0.007499	3.63
i080-123	0.002051	0.002458	1.20	0.003396	1.66	0.007023	3.42	0.007051	3.44	0.007568	3.69
i080-124	0.002040	0.002470	1.21	0.003362	1.65	0.006963	3.41	0.006823	3.34	0.007385	3.62
i080-125	0.002049	0.002449	1.20	0.003367	1.64	0.007189	3.51	0.007071	3.45	0.007502	3.66
i080-131	0.000368	0.000624	1.70	0.001201	3.26	0.001640	4.46	0.001706	4.64	0.001769	4.81
i080-132	0.000392	0.000570	1.45	0.001235	3.15	0.001637	4.18	0.001675	4.27	0.001717	4.38
i080-133	0.000386	0.000609	1.58	0.001144	2.96	0.001539	3.99	0.001733	4.49	0.001731	4.48
i080-134	0.000388	0.000646	1.66	0.001171	3.02	0.001564	4.03	0.001749	4.51	0.001675	4.32
i080-135	0.000376	0.000569	1.51	0.001191	3.17	0.001594	4.24	0.001656	4.40	0.001800	4.79
i080-141	0.000796	0.001038	1.30	0.001544	1.94	0.002622	3.29	0.002804	3.52	0.002836	3.56
i080-142	0.000781	0.000996	1.28	0.001499	1.92	0.002709	3.47	0.002677	3.43	0.003066	3.93
i080-143	0.000772	0.000934	1.21	0.001486	1.92	0.002675	3.47	0.002631	3.41	0.002936	3.80
i080-144	0.000757	0.001003	1.32	0.001387	1.83	0.002709	3.58	0.002744	3.62	0.003015	3.98
i080-145	0.000814	0.000924	1.14	0.001442	1.77	0.002497	3.07	0.002724	3.35	0.002820	3.46

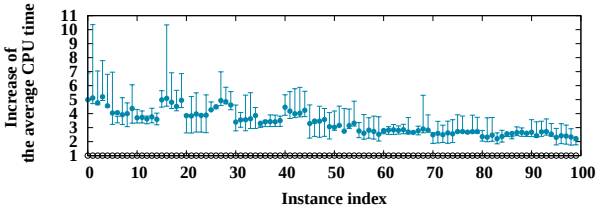
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
i080-201	0.000580	0.000828	1.43	0.001413	2.44	0.001755	3.03	0.001815	3.13	0.001870	3.22
i080-202	0.000499	0.000750	1.50	0.001277	2.56	0.001577	3.16	0.001785	3.58	0.001757	3.52
i080-203	0.000596	0.000806	1.35	0.001455	2.44	0.001636	2.74	0.001911	3.21	0.001870	3.14
i080-204	0.000559	0.000746	1.33	0.001600	2.86	0.001760	3.15	0.001870	3.35	0.001865	3.34
i080-205	0.000518	0.000758	1.46	0.001396	2.69	0.001718	3.32	0.001814	3.50	0.001837	3.55
i080-211	0.000880	0.001198	1.36	0.001838	2.09	0.002430	2.76	0.002569	2.92	0.002647	3.01
i080-212	0.000954	0.001255	1.32	0.001717	1.80	0.002480	2.60	0.002625	2.75	0.002835	2.97
i080-213	0.000891	0.001120	1.26	0.001620	1.82	0.002512	2.82	0.002594	2.91	0.002615	2.93
i080-214	0.000899	0.001153	1.28	0.001630	1.81	0.002448	2.72	0.002553	2.84	0.002651	2.95
i080-215	0.000987	0.001208	1.22	0.001815	1.84	0.002493	2.53	0.002625	2.66	0.002694	2.73
i080-221	0.002774	0.003240	1.17	0.004284	1.54	0.007752	2.79	0.007747	2.79	0.008387	3.02
i080-222	0.002814	0.003226	1.15	0.004280	1.52	0.007943	2.82	0.007853	2.79	0.008294	2.95
i080-223	0.002792	0.003263	1.17	0.004196	1.50	0.007933	2.84	0.007813	2.80	0.008215	2.94
i080-224	0.002775	0.003215	1.16	0.004143	1.49	0.007783	2.80	0.007710	2.78	0.008033	2.89
i080-225	0.002747	0.003201	1.17	0.004170	1.52	0.007840	2.85	0.007649	2.78	0.008327	3.03
i080-231	0.000681	0.000858	1.26	0.001629	2.39	0.001830	2.69	0.001983	2.91	0.002050	3.01
i080-232	0.000671	0.000904	1.35	0.001528	2.28	0.001784	2.66	0.001959	2.92	0.002006	2.99
i080-233	0.000661	0.000792	1.20	0.001534	2.32	0.001833	2.77	0.002037	3.08	0.002067	3.13
i080-234	0.000646	0.000903	1.40	0.001437	2.22	0.001875	2.90	0.002084	3.23	0.002101	3.25
i080-235	0.000637	0.000843	1.32	0.001539	2.42	0.001793	2.81	0.001972	3.10	0.001934	3.04
i080-241	0.001292	0.001538	1.19	0.002109	1.63	0.003216	2.49	0.003229	2.50	0.003474	2.69
i080-242	0.001238	0.001493	1.21	0.001988	1.61	0.003198	2.58	0.003122	2.52	0.003377	2.73
i080-243	0.001198	0.001382	1.15	0.001816	1.52	0.002985	2.49	0.003027	2.53	0.003309	2.76
i080-244	0.001173	0.001469	1.25	0.001958	1.67	0.003072	2.62	0.003163	2.70	0.003334	2.84
i080-245	0.001165	0.001434	1.23	0.001860	1.60	0.002974	2.55	0.003130	2.69	0.003351	2.88
i080-301	0.000646	0.000859	1.33	0.001708	2.64	0.001756	2.72	0.001928	2.98	0.001817	2.81
i080-302	0.000631	0.000820	1.30	0.001425	2.26	0.001716	2.72	0.001822	2.89	0.001989	3.15
i080-303	0.000665	0.000880	1.32	0.001361	2.05	0.001779	2.68	0.001877	2.82	0.001906	2.87
i080-304	0.000655	0.000841	1.28	0.001712	2.61	0.001777	2.71	0.001943	2.97	0.001889	2.88
i080-305	0.000660	0.000892	1.35	0.001737	2.63	0.001794	2.72	0.001998	3.03	0.002007	3.04
i080-311	0.001089	0.001367	1.26	0.002002	1.84	0.002563	2.35	0.002659	2.44	0.002868	2.63
i080-312	0.001100	0.001290	1.17	0.001902	1.73	0.002547	2.32	0.002726	2.48	0.002778	2.53
i080-313	0.001070	0.001333	1.25	0.002039	1.91	0.002632	2.46	0.002653	2.48	0.002808	2.62
i080-314	0.001147	0.001367	1.19	0.001882	1.64	0.002534	2.21	0.002783	2.43	0.003068	2.67
i080-315	0.001109	0.001350	1.22	0.001865	1.68	0.002622	2.36	0.002778	2.50	0.003002	2.71
i080-321	0.003110	0.003597	1.16	0.004607	1.48	0.007956	2.56	0.007900	2.54	0.008434	2.71
i080-322	0.003204	0.003716	1.16	0.004431	1.38	0.008271	2.58	0.008244	2.57	0.008671	2.71
i080-323	0.003015	0.003516	1.17	0.004347	1.44	0.007979	2.65	0.007965	2.64	0.008423	2.79
i080-324	0.003067	0.003539	1.15	0.004273	1.39	0.008177	2.67	0.008014	2.61	0.008351	2.72
i080-325	0.003159	0.003653	1.16	0.004442	1.41	0.008209	2.60	0.008143	2.58	0.008589	2.72
i080-331	0.000762	0.000986	1.29	0.001546	2.03	0.002036	2.67	0.002150	2.82	0.002148	2.82
i080-332	0.000784	0.000984	1.26	0.001807	2.30	0.001905	2.43	0.002047	2.61	0.002168	2.77
i080-333	0.000751	0.000993	1.32	0.001571	2.09	0.002025	2.70	0.002077	2.77	0.002084	2.77
i080-334	0.000745	0.001013	1.36	0.001646	2.21	0.002025	2.72	0.002160	2.90	0.002344	3.15
i080-335	0.000780	0.001056	1.35	0.001520	1.95	0.001978	2.54	0.002160	2.77	0.002148	2.75
i080-341	0.001489	0.001813	1.22	0.002256	1.52	0.003428	2.30	0.003488	2.34	0.003807	2.56
i080-342	0.001370	0.001769	1.29	0.002172	1.59	0.003316	2.42	0.003401	2.48	0.003504	2.56
i080-343	0.001390	0.001647	1.18	0.002129	1.53	0.003327	2.39	0.003418	2.46	0.003664	2.64
i080-344	0.001442	0.001633	1.13	0.002154	1.49	0.003359	2.33	0.003627	2.52	0.003646	2.53
i080-345	0.001499	0.001715	1.14	0.002277	1.52	0.003321	2.22	0.003443	2.30	0.003438	2.29
ave	0.001035	0.001306	1.26	0.001927	1.86	0.003159	3.05	0.003244	3.13	0.003393	3.28



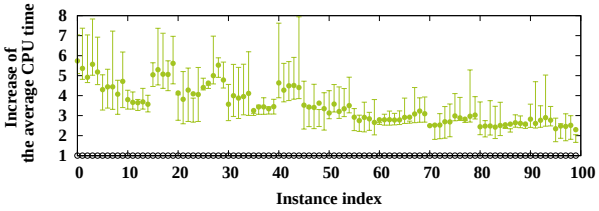
(a) degree centrality



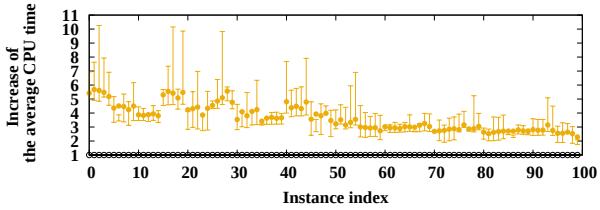
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality



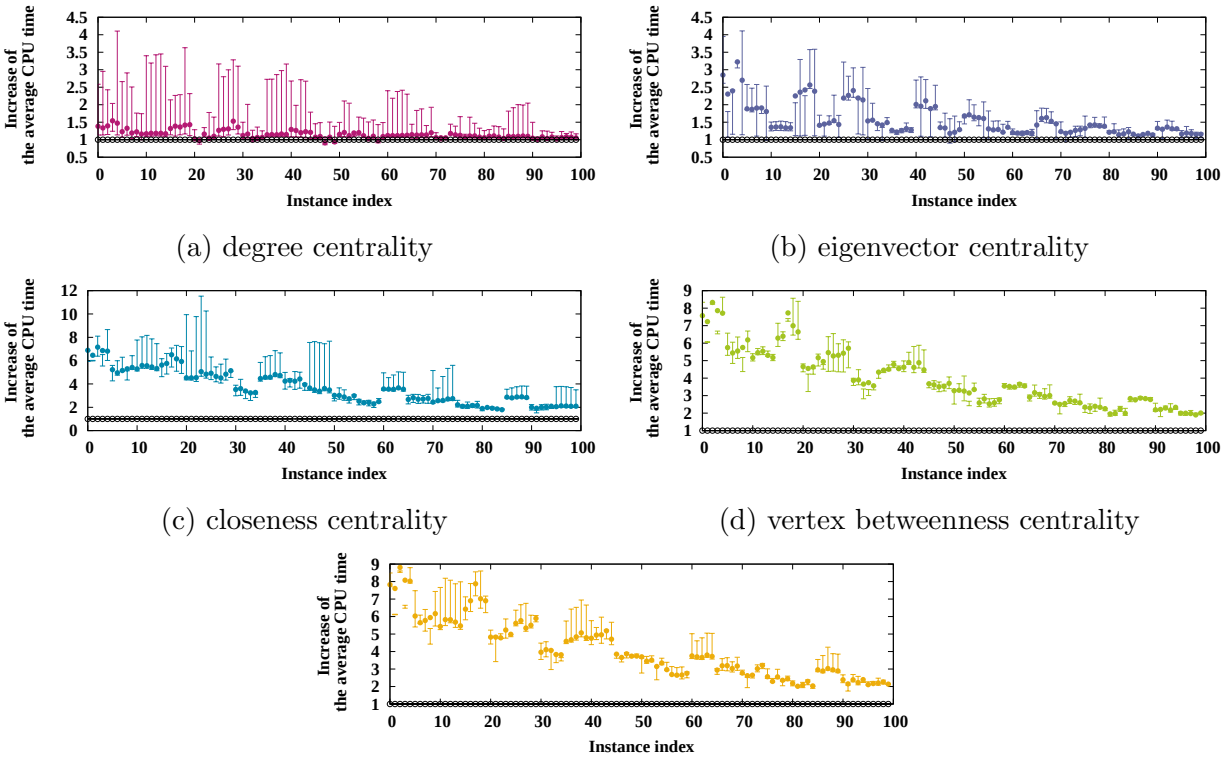
(e) edge betweenness centrality

Figure 3.42: Increase of CPU time from the original method (DNH, I080)

Table 3.35: CPU time [s] (DNH, I160)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
i160-001	0.000638	0.000882	1.38	0.001817	2.85	0.004395	6.89	0.004829	7.57	0.004993	7.83
i160-002	0.000631	0.000844	1.34	0.001454	2.30	0.004078	6.46	0.004563	7.23	0.004798	7.60
i160-003	0.000583	0.000814	1.40	0.001397	2.40	0.004172	7.16	0.004854	8.33	0.005135	8.81
i160-004	0.000586	0.000903	1.54	0.001890	3.23	0.004022	6.86	0.004606	7.86	0.004734	8.08
i160-005	0.000616	0.000908	1.47	0.001663	2.70	0.004198	6.81	0.004750	7.71	0.004942	8.02
i160-011	0.001255	0.001544	1.23	0.002363	1.88	0.006554	5.22	0.007212	5.75	0.007566	6.03
i160-012	0.001364	0.001810	1.33	0.002549	1.87	0.006753	4.95	0.007419	5.44	0.007702	5.65
i160-013	0.001277	0.001519	1.19	0.002431	1.90	0.006582	5.15	0.007086	5.55	0.007374	5.77
i160-014	0.001262	0.001546	1.23	0.002406	1.91	0.006684	5.30	0.007257	5.75	0.007498	5.94
i160-015	0.001196	0.001388	1.16	0.002159	1.81	0.006479	5.42	0.007397	6.18	0.007377	6.17
i160-021	0.007795	0.009133	1.17	0.010546	1.35	0.041244	5.29	0.040270	5.17	0.042364	5.43
i160-022	0.007386	0.008738	1.18	0.010062	1.36	0.041259	5.59	0.040297	5.46	0.043036	5.83
i160-023	0.007393	0.008753	1.18	0.010114	1.37	0.041102	5.56	0.040929	5.54	0.043016	5.82
i160-024	0.007555	0.008942	1.18	0.010119	1.34	0.041222	5.46	0.040074	5.30	0.042932	5.68
i160-025	0.007763	0.009092	1.17	0.010410	1.34	0.041314	5.32	0.040163	5.17	0.042434	5.47
i160-031	0.000867	0.001143	1.32	0.001952	2.25	0.004871	5.62	0.005450	6.29	0.005572	6.43
i160-032	0.000804	0.001116	1.39	0.001897	2.36	0.004628	5.76	0.005142	6.40	0.005550	6.90
i160-033	0.000706	0.000963	1.36	0.001710	2.42	0.004590	6.50	0.005455	7.73	0.005557	7.87
i160-034	0.000764	0.001082	1.42	0.001959	2.56	0.004699	6.15	0.005343	6.99	0.005363	7.02
i160-035	0.000795	0.001136	1.43	0.001896	2.38	0.004712	5.93	0.005283	6.65	0.005487	6.90
i160-041	0.002787	0.002856	1.02	0.003936	1.41	0.012630	4.53	0.012978	4.66	0.013443	4.82
i160-042	0.002830	0.002788	0.99	0.004090	1.45	0.012795	4.52	0.012882	4.55	0.013658	4.83
i160-043	0.002810	0.003262	1.16	0.004149	1.48	0.012763	4.54	0.013041	4.64	0.013448	4.79
i160-044	0.002530	0.002651	1.05	0.003905	1.54	0.012784	5.05	0.013070	5.17	0.013225	5.23
i160-045	0.002665	0.002925	1.10	0.003820	1.43	0.012888	4.84	0.013179	4.95	0.013290	4.99
i160-101	0.000942	0.001195	1.27	0.002064	2.19	0.004634	4.92	0.005141	5.46	0.005287	5.61
i160-102	0.000974	0.001263	1.30	0.002208	2.27	0.004527	4.65	0.005132	5.27	0.005614	5.76
i160-103	0.001000	0.001309	1.31	0.002404	2.40	0.004512	4.51	0.005308	5.31	0.005340	5.34
i160-104	0.000958	0.001466	1.53	0.002098	2.19	0.004633	4.84	0.005196	5.42	0.005271	5.50
i160-105	0.000884	0.001201	1.36	0.001888	2.14	0.004539	5.13	0.005044	5.71	0.005209	5.89
i160-111	0.002050	0.002294	1.12	0.003151	1.54	0.007223	3.52	0.007929	3.87	0.008122	3.96
i160-112	0.002027	0.002355	1.16	0.003161	1.56	0.007293	3.60	0.007900	3.90	0.008326	4.11
i160-113	0.002171	0.002254	1.04	0.003159	1.46	0.007368	3.39	0.007941	3.66	0.008823	4.06
i160-114	0.002216	0.002358	1.06	0.003144	1.42	0.007160	3.23	0.008294	3.74	0.008490	3.83
i160-115	0.002139	0.002274	1.06	0.003187	1.49	0.006989	3.27	0.007614	3.56	0.008157	3.81
i160-121	0.009628	0.010980	1.14	0.012201	1.27	0.042959	4.46	0.041783	4.34	0.044147	4.59
i160-122	0.009401	0.010773	1.15	0.011475	1.22	0.042995	4.57	0.041870	4.45	0.043954	4.68
i160-123	0.009400	0.010741	1.14	0.011870	1.26	0.043032	4.58	0.042397	4.51	0.045491	4.84
i160-124	0.008939	0.010397	1.16	0.011774	1.32	0.042943	4.80	0.042722	4.78	0.045311	5.07
i160-125	0.009126	0.010480	1.15	0.011611	1.27	0.043131	4.73	0.041428	4.54	0.043726	4.79
i160-131	0.001224	0.001581	1.29	0.002457	2.01	0.005192	4.24	0.005663	4.63	0.005830	4.76
i160-132	0.001142	0.001442	1.26	0.002235	1.96	0.004916	4.30	0.005592	4.90	0.005641	4.94
i160-133	0.001221	0.001466	1.20	0.002579	2.11	0.005174	4.24	0.005633	4.61	0.006051	4.96
i160-134	0.001156	0.001421	1.23	0.002183	1.89	0.005116	4.43	0.005646	4.88	0.005995	5.19
i160-135	0.001280	0.001549	1.21	0.002493	1.95	0.005044	3.94	0.005727	4.47	0.006026	4.71
i160-141	0.003819	0.004085	1.07	0.005139	1.35	0.013943	3.65	0.014049	3.68	0.014707	3.85
i160-142	0.003899	0.004261	1.09	0.005204	1.33	0.013621	3.49	0.014102	3.62	0.014287	3.66
i160-143	0.003981	0.003634	0.91	0.004702	1.18	0.013471	3.38	0.013959	3.51	0.015409	3.87
i160-144	0.003963	0.004256	1.07	0.004805	1.21	0.014332	3.62	0.013992	3.53	0.014813	3.74
i160-145	0.003883	0.003637	0.94	0.005005	1.29	0.013519	3.48	0.014374	3.70	0.014525	3.74

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
i160-201	0.001812	0.002089	1.15	0.003045	1.68	0.005482	3.03	0.005951	3.28	0.006697	3.70
i160-202	0.001878	0.002267	1.21	0.003217	1.71	0.005657	3.01	0.006213	3.31	0.006478	3.45
i160-203	0.001895	0.002118	1.12	0.003116	1.64	0.005459	2.88	0.006179	3.26	0.006645	3.51
i160-204	0.001962	0.002338	1.19	0.003202	1.63	0.005356	2.73	0.006180	3.15	0.006175	3.15
i160-205	0.001851	0.002225	1.20	0.002969	1.60	0.005525	2.98	0.006207	3.35	0.006191	3.34
i160-211	0.003658	0.004075	1.11	0.004787	1.31	0.009223	2.52	0.009454	2.58	0.010851	2.97
i160-212	0.003633	0.003775	1.04	0.004653	1.28	0.008657	2.38	0.010232	2.82	0.009765	2.69
i160-213	0.003854	0.004235	1.10	0.004984	1.29	0.009316	2.42	0.009745	2.53	0.010208	2.65
i160-214	0.003854	0.003729	0.97	0.004682	1.21	0.008791	2.28	0.009993	2.59	0.010241	2.66
i160-215	0.003745	0.004085	1.09	0.005075	1.36	0.009310	2.49	0.010317	2.75	0.010326	2.76
i160-221	0.012825	0.014349	1.12	0.015415	1.20	0.045801	3.57	0.045505	3.55	0.048080	3.75
i160-222	0.013043	0.014580	1.12	0.015522	1.19	0.046355	3.55	0.045986	3.53	0.048256	3.70
i160-223	0.013047	0.014559	1.12	0.015483	1.19	0.046242	3.54	0.045584	3.49	0.047802	3.66
i160-224	0.012616	0.014159	1.12	0.015204	1.21	0.046230	3.66	0.046080	3.65	0.047940	3.80
i160-225	0.012988	0.014645	1.13	0.015479	1.19	0.046135	3.55	0.046125	3.55	0.048344	3.72
i160-231	0.002438	0.002786	1.14	0.003459	1.42	0.006511	2.67	0.007087	2.91	0.007174	2.94
i160-232	0.002258	0.002556	1.13	0.003627	1.61	0.006381	2.83	0.007165	3.17	0.007193	3.19
i160-233	0.002312	0.002631	1.14	0.003770	1.63	0.006220	2.69	0.007099	3.07	0.007385	3.19
i160-234	0.002289	0.002603	1.14	0.003497	1.53	0.006190	2.70	0.006772	2.96	0.006936	3.03
i160-235	0.002341	0.002809	1.20	0.003403	1.45	0.006457	2.76	0.007034	3.00	0.007424	3.17
i160-241	0.006427	0.006824	1.06	0.007921	1.23	0.015689	2.44	0.016553	2.58	0.017810	2.77
i160-242	0.006583	0.006948	1.06	0.007784	1.18	0.016899	2.57	0.016455	2.50	0.017254	2.62
i160-243	0.006506	0.006856	1.05	0.007882	1.21	0.016792	2.58	0.016552	2.54	0.017237	2.65
i160-244	0.006041	0.007119	1.18	0.007644	1.27	0.016393	2.71	0.016454	2.72	0.018239	3.02
i160-245	0.005933	0.006755	1.14	0.007620	1.28	0.016276	2.74	0.015960	2.69	0.019002	3.20
i160-301	0.003077	0.003461	1.12	0.004075	1.32	0.006832	2.22	0.007976	2.59	0.007878	2.56
i160-302	0.003307	0.003524	1.07	0.004697	1.42	0.006906	2.09	0.007744	2.34	0.007575	2.29
i160-303	0.003236	0.003587	1.11	0.004598	1.42	0.006760	2.09	0.007436	2.30	0.008253	2.55
i160-304	0.003140	0.003488	1.11	0.004365	1.39	0.006748	2.15	0.007499	2.39	0.007405	2.36
i160-305	0.003231	0.003473	1.07	0.004473	1.38	0.006978	2.16	0.007541	2.33	0.007892	2.44
i160-311	0.005752	0.006084	1.06	0.006974	1.21	0.010906	1.90	0.012966	2.25	0.012538	2.18
i160-312	0.006065	0.006668	1.10	0.007455	1.23	0.012041	1.99	0.011876	1.96	0.012226	2.02
i160-313	0.006003	0.006658	1.11	0.006856	1.14	0.011524	1.92	0.011902	1.98	0.012499	2.08
i160-314	0.005733	0.006170	1.08	0.006692	1.17	0.010637	1.86	0.012905	2.25	0.013137	2.29
i160-315	0.006025	0.006248	1.04	0.007389	1.23	0.010796	1.79	0.011993	1.99	0.012270	2.04
i160-321	0.018165	0.019951	1.10	0.021039	1.16	0.052116	2.87	0.051441	2.83	0.053681	2.96
i160-322	0.018773	0.020464	1.09	0.020769	1.11	0.052713	2.81	0.051770	2.76	0.054215	2.89
i160-323	0.017750	0.019473	1.10	0.020402	1.15	0.051130	2.88	0.050708	2.86	0.053751	3.03
i160-324	0.018159	0.020051	1.10	0.021495	1.18	0.052558	2.89	0.051373	2.83	0.053679	2.96
i160-325	0.018679	0.020479	1.10	0.021267	1.14	0.053088	2.84	0.051890	2.78	0.053992	2.89
i160-331	0.003891	0.004066	1.04	0.005171	1.33	0.007779	2.00	0.008523	2.19	0.009273	2.38
i160-332	0.003943	0.003982	1.01	0.005126	1.30	0.007563	1.92	0.008709	2.21	0.008475	2.15
i160-333	0.003900	0.004224	1.08	0.005306	1.36	0.007750	1.99	0.008932	2.29	0.009295	2.38
i160-334	0.003910	0.004040	1.03	0.005134	1.31	0.007867	2.01	0.008503	2.17	0.008707	2.23
i160-335	0.003683	0.003958	1.07	0.004833	1.31	0.007698	2.09	0.008579	2.33	0.008828	2.40
i160-341	0.009765	0.009970	1.02	0.011360	1.16	0.020098	2.06	0.019465	1.99	0.020565	2.11
i160-342	0.009597	0.010171	1.06	0.011414	1.19	0.020342	2.12	0.019143	1.99	0.020814	2.17
i160-343	0.009713	0.010553	1.09	0.011454	1.18	0.020495	2.11	0.019195	1.98	0.021416	2.20
i160-344	0.009605	0.010293	1.07	0.011007	1.15	0.020028	2.09	0.018247	1.90	0.021681	2.26
i160-345	0.009619	0.010117	1.05	0.011127	1.16	0.020144	2.09	0.019268	2.00	0.020536	2.13
ave	0.004879	0.005423	1.11	0.006345	1.30	0.016084	3.30	0.016341	3.35	0.017181	3.52



(a) degree centrality

(b) eigenvector centrality

(c) closeness centrality

(d) vertex betweenness centrality

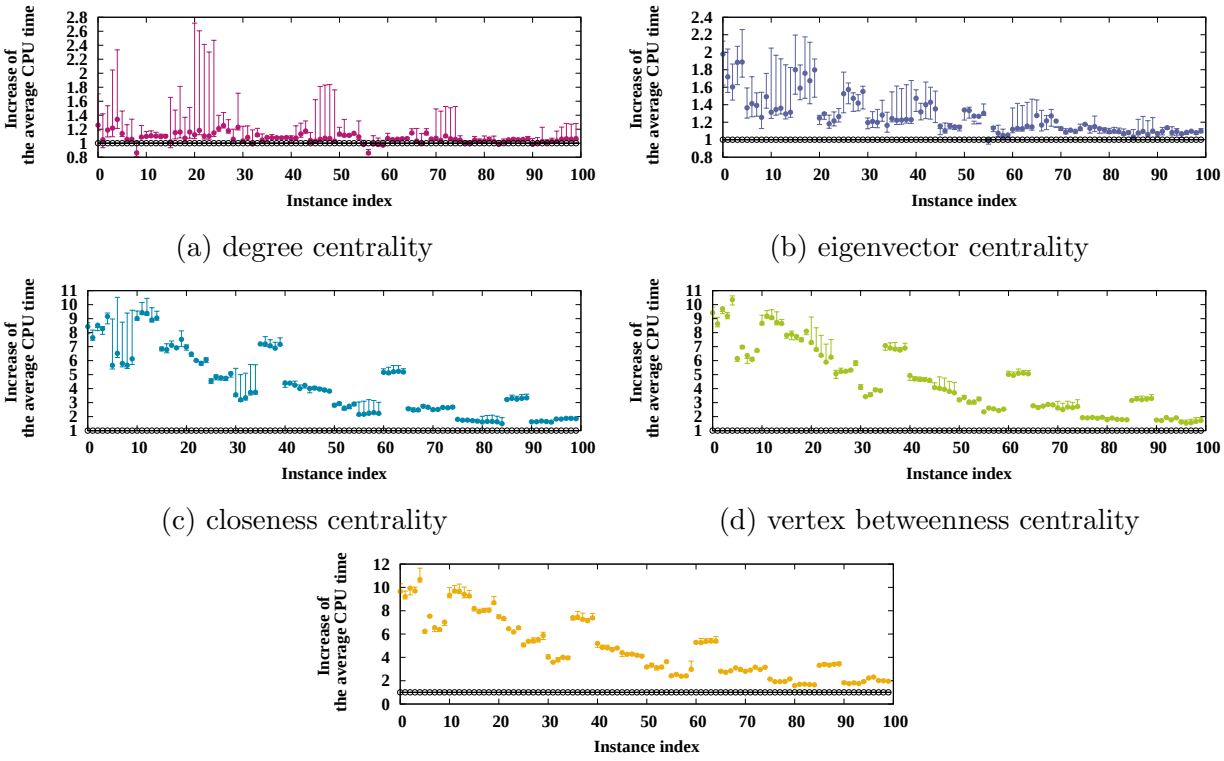
(e) edge betweenness centrality

Figure 3.43: Increase of CPU time from the original method (DNH, I160)

Table 3.36: CPU time [s] (DNH, I320)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
i320-001	0.001920	0.002415	1.26	0.003799	1.98	0.016195	8.43	0.018085	9.42	0.018529	9.65
i320-002	0.002040	0.002139	1.05	0.003506	1.72	0.015582	7.64	0.017618	8.64	0.018793	9.21
i320-003	0.001867	0.002218	1.19	0.002992	1.60	0.015931	8.53	0.018123	9.71	0.018540	9.93
i320-004	0.001947	0.002367	1.22	0.003669	1.88	0.016122	8.28	0.017894	9.19	0.018870	9.69
i320-005	0.001754	0.002350	1.34	0.003312	1.89	0.016061	9.16	0.018159	10.35	0.018723	10.67
i320-011	0.004869	0.005542	1.14	0.006645	1.36	0.027576	5.66	0.029943	6.15	0.030423	6.25
i320-012	0.004038	0.004256	1.05	0.005706	1.41	0.026320	6.52	0.028146	6.97	0.030400	7.53
i320-013	0.004582	0.004838	1.06	0.006376	1.39	0.026574	5.80	0.029160	6.36	0.030032	6.55
i320-014	0.004698	0.004052	0.86	0.005899	1.26	0.026631	5.67	0.028586	6.08	0.030011	6.39
i320-015	0.004359	0.004741	1.09	0.006501	1.49	0.026722	6.13	0.029349	6.73	0.030612	7.02
i320-021	0.034010	0.037316	1.10	0.044708	1.31	0.306640	9.02	0.294940	8.67	0.316933	9.32
i320-022	0.032334	0.035759	1.11	0.043486	1.34	0.304810	9.43	0.296870	9.18	0.313416	9.69
i320-023	0.032388	0.035672	1.10	0.044040	1.36	0.303695	9.38	0.293815	9.07	0.313107	9.67
i320-024	0.033928	0.037127	1.09	0.043937	1.30	0.302273	8.91	0.296341	8.73	0.318651	9.39
i320-025	0.033843	0.037188	1.10	0.044452	1.31	0.306306	9.05	0.293023	8.66	0.313872	9.27
i320-031	0.002519	0.002564	1.02	0.004535	1.80	0.017232	6.84	0.019687	7.82	0.020655	8.20
i320-032	0.002533	0.002915	1.15	0.004025	1.59	0.017194	6.79	0.019991	7.89	0.020057	7.92
i320-033	0.002533	0.002937	1.16	0.004459	1.76	0.017974	7.10	0.019571	7.73	0.020318	8.02
i320-034	0.002579	0.002766	1.07	0.004318	1.67	0.017835	6.92	0.019330	7.50	0.020725	8.04
i320-035	0.002372	0.002751	1.16	0.004264	1.80	0.017831	7.52	0.019227	8.11	0.020595	8.68
i320-041	0.010280	0.011515	1.12	0.012831	1.25	0.071693	6.97	0.075057	7.30	0.076840	7.47
i320-042	0.011076	0.013106	1.18	0.014347	1.30	0.071750	6.48	0.075414	6.81	0.081290	7.34
i320-043	0.012060	0.013294	1.10	0.014245	1.18	0.072539	6.01	0.077013	6.39	0.077954	6.46
i320-044	0.012650	0.013936	1.10	0.015379	1.22	0.073490	5.81	0.074588	5.90	0.078140	6.18
i320-045	0.012077	0.013843	1.15	0.015273	1.26	0.073192	6.06	0.075508	6.25	0.078960	6.54
i320-101	0.003953	0.004784	1.21	0.006033	1.53	0.017962	4.54	0.020021	5.06	0.020002	5.06
i320-102	0.003741	0.004684	1.25	0.005888	1.57	0.018130	4.85	0.019547	5.23	0.020117	5.38
i320-103	0.003697	0.004353	1.18	0.005442	1.47	0.017651	4.77	0.019395	5.25	0.020061	5.43
i320-104	0.003740	0.003951	1.06	0.005306	1.42	0.017710	4.74	0.019916	5.33	0.020727	5.54
i320-105	0.003491	0.004288	1.23	0.005416	1.55	0.017698	5.07	0.020349	5.83	0.020500	5.87
i320-111	0.008497	0.008781	1.03	0.010165	1.20	0.030242	3.56	0.034759	4.09	0.034150	4.02
i320-112	0.009584	0.010381	1.08	0.011589	1.21	0.030702	3.20	0.032820	3.42	0.034358	3.58
i320-113	0.009026	0.008961	0.99	0.010810	1.20	0.030230	3.35	0.032119	3.56	0.034229	3.79
i320-114	0.008243	0.009233	1.12	0.010566	1.28	0.030726	3.73	0.032242	3.91	0.033066	4.01
i320-115	0.008328	0.008707	1.05	0.009693	1.16	0.031198	3.75	0.032106	3.86	0.032981	3.96
i320-121	0.043305	0.046880	1.08	0.053791	1.24	0.311590	7.20	0.306275	7.07	0.321102	7.41
i320-122	0.043811	0.047183	1.08	0.053541	1.22	0.314825	7.19	0.302099	6.90	0.326801	7.46
i320-123	0.044537	0.048009	1.08	0.054509	1.22	0.314608	7.06	0.304258	6.83	0.323239	7.26
i320-124	0.045285	0.048702	1.08	0.055761	1.23	0.312140	6.89	0.305985	6.76	0.323586	7.15
i320-125	0.043771	0.047304	1.08	0.053785	1.23	0.313682	7.17	0.301816	6.90	0.323131	7.38
i320-131	0.004325	0.004616	1.07	0.006370	1.47	0.019056	4.41	0.021364	4.94	0.022458	5.19
i320-132	0.004562	0.004897	1.07	0.006013	1.32	0.020039	4.39	0.021397	4.69	0.022124	4.85
i320-133	0.004597	0.005204	1.13	0.006437	1.40	0.019507	4.24	0.021474	4.67	0.022140	4.82
i320-134	0.005005	0.005875	1.17	0.007151	1.43	0.020036	4.00	0.023263	4.65	0.023300	4.66
i320-135	0.004674	0.004862	1.04	0.006322	1.35	0.019784	4.23	0.021489	4.60	0.022433	4.80
i320-141	0.020323	0.020918	1.03	0.023511	1.16	0.081504	4.01	0.082889	4.08	0.089555	4.41
i320-142	0.020184	0.021300	1.06	0.022244	1.10	0.081442	4.03	0.081323	4.03	0.085956	4.26
i320-143	0.020373	0.021841	1.07	0.023599	1.16	0.080840	3.97	0.080510	3.95	0.087157	4.28
i320-144	0.021080	0.022482	1.07	0.024184	1.15	0.082171	3.90	0.080186	3.80	0.088236	4.19
i320-145	0.021903	0.022531	1.03	0.025039	1.14	0.083749	3.82	0.081226	3.71	0.089911	4.10

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
i320-201	0.008027	0.009088	1.13	0.010751	1.34	0.022605	2.82	0.025640	3.19	0.025560	3.18
i320-202	0.007522	0.008371	1.11	0.010032	1.33	0.022052	2.93	0.025335	3.37	0.024963	3.32
i320-203	0.008657	0.009625	1.11	0.010999	1.27	0.022585	2.61	0.026379	3.05	0.027023	3.12
i320-204	0.007956	0.009085	1.14	0.010083	1.27	0.021840	2.75	0.024265	3.05	0.025239	3.17
i320-205	0.007431	0.008101	1.09	0.009682	1.30	0.021568	2.90	0.024182	3.25	0.026955	3.63
i320-211	0.017654	0.017336	0.98	0.017502	0.99	0.038121	2.16	0.041384	2.34	0.042812	2.43
i320-212	0.017027	0.014638	0.86	0.019298	1.13	0.036973	2.17	0.044474	2.61	0.043291	2.54
i320-213	0.016646	0.016500	0.99	0.017883	1.07	0.037256	2.24	0.042511	2.55	0.039748	2.39
i320-214	0.017046	0.016736	0.98	0.017884	1.05	0.039010	2.29	0.041451	2.43	0.041163	2.41
i320-215	0.016691	0.016219	0.97	0.017580	1.05	0.037254	2.23	0.042367	2.54	0.049876	2.99
i320-221	0.064420	0.068933	1.07	0.072101	1.12	0.332595	5.16	0.323443	5.02	0.340481	5.29
i320-222	0.065233	0.068536	1.05	0.073718	1.13	0.334363	5.13	0.322616	4.95	0.343847	5.27
i320-223	0.063696	0.067367	1.06	0.071723	1.13	0.331768	5.21	0.324758	5.10	0.342134	5.37
i320-224	0.063190	0.067199	1.06	0.073037	1.16	0.331117	5.24	0.322732	5.11	0.340747	5.39
i320-225	0.063793	0.067970	1.07	0.073101	1.15	0.330851	5.19	0.322724	5.06	0.344424	5.40
i320-231	0.009796	0.011234	1.15	0.012501	1.28	0.025081	2.56	0.027192	2.78	0.027493	2.81
i320-232	0.010169	0.010483	1.03	0.011880	1.17	0.025098	2.47	0.026928	2.65	0.027574	2.71
i320-233	0.009886	0.009859	1.00	0.012022	1.22	0.024533	2.48	0.027200	2.75	0.028114	2.84
i320-234	0.009096	0.010431	1.15	0.011564	1.27	0.024924	2.74	0.026145	2.87	0.028234	3.10
i320-235	0.009344	0.009903	1.06	0.011329	1.21	0.024956	2.67	0.026541	2.84	0.027765	2.97
i320-241	0.037417	0.040045	1.07	0.042170	1.13	0.093439	2.50	0.098088	2.62	0.104888	2.80
i320-242	0.036135	0.037584	1.04	0.039255	1.09	0.090685	2.51	0.090471	2.50	0.104958	2.90
i320-243	0.033913	0.037430	1.10	0.037604	1.11	0.090089	2.66	0.091267	2.69	0.107048	3.16
i320-244	0.034618	0.036818	1.06	0.037951	1.10	0.091236	2.64	0.091476	2.64	0.102542	2.96
i320-245	0.033361	0.035235	1.06	0.037606	1.13	0.089392	2.68	0.090941	2.73	0.104923	3.15
i320-301	0.017946	0.018744	1.04	0.021093	1.18	0.032391	1.80	0.034627	1.93	0.038324	2.14
i320-302	0.017899	0.017969	1.00	0.020219	1.13	0.031227	1.74	0.034114	1.91	0.034310	1.92
i320-303	0.018722	0.018630	1.00	0.021538	1.15	0.032730	1.75	0.036459	1.95	0.036095	1.93
i320-304	0.019158	0.020019	1.04	0.021571	1.13	0.032797	1.71	0.036094	1.88	0.037066	1.93
i320-305	0.018940	0.019444	1.03	0.021058	1.11	0.031788	1.68	0.036923	1.95	0.040733	2.15
i320-311	0.039413	0.040592	1.03	0.042975	1.09	0.063756	1.62	0.070193	1.78	0.062348	1.58
i320-312	0.036892	0.038965	1.06	0.040460	1.10	0.061223	1.66	0.069878	1.89	0.062343	1.69
i320-313	0.037914	0.038838	1.02	0.041425	1.09	0.062733	1.65	0.068442	1.81	0.064593	1.70
i320-314	0.039049	0.038433	0.98	0.041818	1.07	0.063648	1.63	0.070002	1.79	0.065065	1.67
i320-315	0.037841	0.038445	1.02	0.040919	1.08	0.056712	1.50	0.067084	1.77	0.062211	1.64
i320-321	0.119622	0.125340	1.05	0.123443	1.03	0.385489	3.22	0.377822	3.16	0.396690	3.32
i320-322	0.115473	0.122015	1.06	0.124326	1.08	0.381282	3.30	0.376981	3.26	0.392027	3.39
i320-323	0.117841	0.123771	1.05	0.129060	1.10	0.382355	3.24	0.377888	3.21	0.393161	3.34
i320-324	0.115551	0.121786	1.05	0.122381	1.06	0.382224	3.31	0.375236	3.25	0.393879	3.41
i320-325	0.113313	0.120380	1.06	0.124508	1.10	0.378520	3.34	0.373495	3.30	0.388449	3.43
i320-331	0.023469	0.023131	0.99	0.025111	1.07	0.038236	1.63	0.041157	1.75	0.043011	1.83
i320-332	0.024129	0.024233	1.00	0.026406	1.09	0.039433	1.63	0.041474	1.72	0.042177	1.75
i320-333	0.022483	0.023035	1.02	0.025584	1.14	0.037809	1.68	0.043498	1.93	0.040786	1.81
i320-334	0.023672	0.023956	1.01	0.025620	1.08	0.038965	1.65	0.041919	1.77	0.041530	1.75
i320-335	0.024418	0.025079	1.03	0.026697	1.09	0.039244	1.61	0.046775	1.92	0.046916	1.92
i320-341	0.074003	0.076261	1.03	0.079023	1.07	0.135332	1.83	0.119077	1.61	0.165237	2.23
i320-342	0.077376	0.081951	1.06	0.083899	1.08	0.141239	1.83	0.119127	1.54	0.179017	2.31
i320-343	0.075232	0.080026	1.06	0.082530	1.10	0.140672	1.87	0.117861	1.57	0.151255	2.01
i320-344	0.073387	0.077550	1.06	0.079278	1.08	0.137419	1.87	0.121138	1.65	0.146009	1.99
i320-345	0.074431	0.079314	1.07	0.081777	1.10	0.137996	1.85	0.127997	1.72	0.145462	1.95
ave	0.026794	0.028326	1.06	0.030672	1.14	0.102079	3.81	0.101585	3.79	0.108399	4.05



(a) degree centrality

(b) eigenvector centrality

(c) closeness centrality

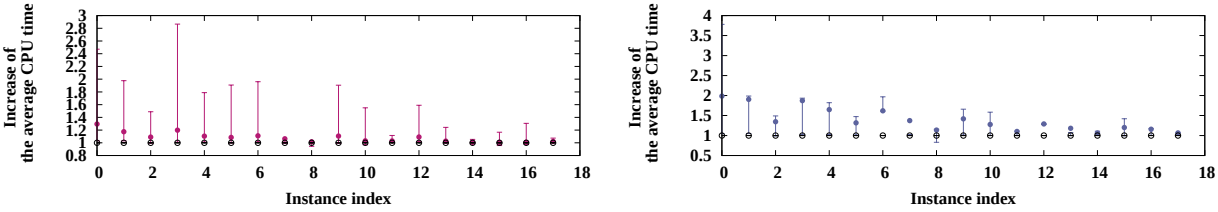
(d) vertex betweenness centrality

(e) edge betweenness centrality

Figure 3.44: Increase of CPU time from the original method (DNH, I320)

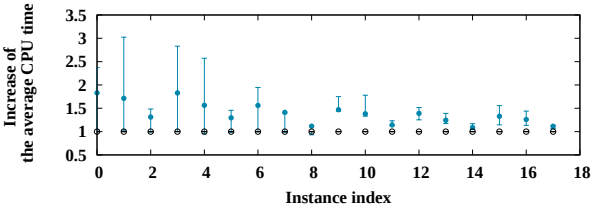
Table 3.37: CPU time [s] (ADH, B)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
b01	0.000689	0.000892	1.29	0.001368	1.99	0.001261	1.83	0.001296	1.88	0.001390	2.02
b02	0.000862	0.001012	1.17	0.001642	1.90	0.001476	1.71	0.001503	1.74	0.001475	1.71
b03	0.002033	0.002215	1.09	0.002736	1.35	0.002667	1.31	0.002568	1.26	0.002577	1.27
b04	0.000883	0.001058	1.20	0.001654	1.87	0.001615	1.83	0.001570	1.78	0.001626	1.84
b05	0.001094	0.001208	1.10	0.001802	1.65	0.001712	1.56	0.001825	1.67	0.001871	1.71
b06	0.002243	0.002431	1.08	0.002955	1.32	0.002908	1.30	0.002961	1.32	0.002917	1.30
b07	0.001735	0.001926	1.11	0.002807	1.62	0.002707	1.56	0.002972	1.71	0.002936	1.69
b08	0.002444	0.002598	1.06	0.003355	1.37	0.003452	1.41	0.003477	1.42	0.003559	1.46
b09	0.006936	0.007050	1.02	0.007877	1.14	0.007748	1.12	0.007773	1.12	0.007907	1.14
b10	0.002257	0.002496	1.11	0.003201	1.42	0.003318	1.47	0.003547	1.57	0.003639	1.61
b11	0.003004	0.003096	1.03	0.003835	1.28	0.004153	1.38	0.004297	1.43	0.004298	1.43
b12	0.007605	0.007826	1.03	0.008374	1.10	0.008682	1.14	0.008932	1.17	0.008878	1.17
b13	0.003543	0.003868	1.09	0.004566	1.29	0.004930	1.39	0.005364	1.51	0.005722	1.62
b14	0.005268	0.005412	1.03	0.006217	1.18	0.006545	1.24	0.006874	1.30	0.006900	1.31
b15	0.016692	0.016835	1.01	0.017734	1.06	0.018138	1.09	0.018386	1.10	0.018598	1.11
b16	0.004700	0.004730	1.01	0.005642	1.20	0.006245	1.33	0.006808	1.45	0.006937	1.48
b17	0.006607	0.006672	1.01	0.007640	1.16	0.008324	1.26	0.008661	1.31	0.008697	1.32
b18	0.017748	0.018128	1.02	0.018927	1.07	0.019779	1.11	0.019985	1.13	0.020147	1.14
ave	0.004544	0.004708	1.04	0.005386	1.19	0.005561	1.22	0.005726	1.26	0.005793	1.27

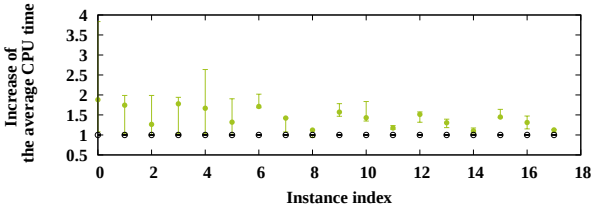


(a) degree centrality

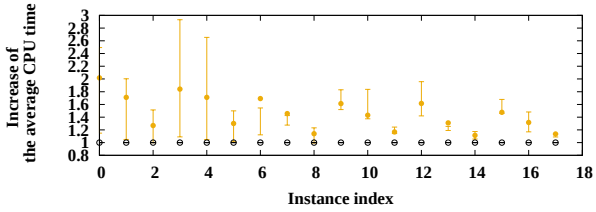
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality

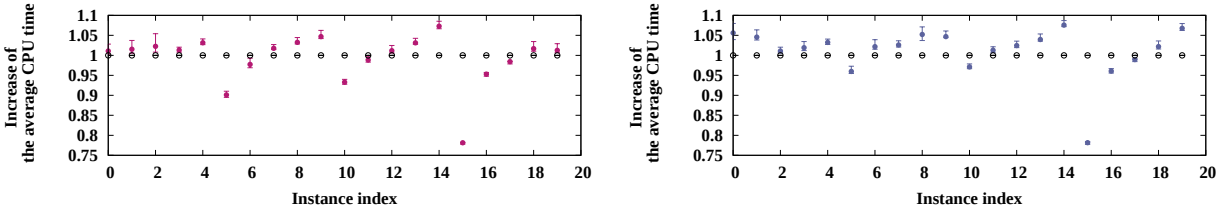


(e) edge betweenness centrality

Figure 3.45: Increase of CPU time from the original method (ADH, B)

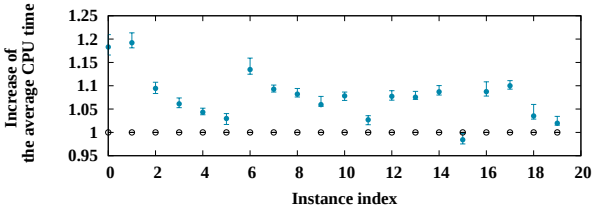
Table 3.38: CPU time [s] (ADH, C)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
c01	0.135164	0.136630	1.01	0.142728	1.06	0.159905	1.18	0.185836	1.37	0.185352	1.37
c02	0.137760	0.139868	1.02	0.144047	1.05	0.164233	1.19	0.180867	1.31	0.182181	1.32
c03	0.367741	0.375989	1.02	0.371690	1.01	0.402407	1.09	0.409461	1.11	0.411880	1.12
c04	0.691080	0.700115	1.01	0.704293	1.02	0.733406	1.06	0.739423	1.07	0.743572	1.08
c05	2.438500	2.515390	1.03	2.517290	1.03	2.544290	1.04	2.576470	1.06	2.556160	1.05
c06	0.228076	0.205538	0.90	0.218926	0.96	0.234845	1.03	0.277928	1.22	0.284732	1.25
c07	0.228374	0.223192	0.98	0.233342	1.02	0.259147	1.13	0.280712	1.23	0.284868	1.25
c08	0.470448	0.478786	1.02	0.482610	1.03	0.514042	1.09	0.528885	1.12	0.532246	1.13
c09	0.776203	0.801461	1.03	0.816711	1.05	0.839860	1.08	0.851752	1.10	0.853980	1.10
c10	2.563080	2.683160	1.05	2.683250	1.05	2.715990	1.06	2.729350	1.06	2.735980	1.07
c11	0.465899	0.434664	0.93	0.452482	0.97	0.502317	1.08	0.545483	1.17	0.588997	1.26
c12	0.459424	0.454184	0.99	0.465223	1.01	0.471795	1.03	0.537374	1.17	0.627677	1.37
c13	0.691839	0.699922	1.01	0.708715	1.02	0.745307	1.08	0.785456	1.14	0.822537	1.19
c14	0.989714	1.020770	1.03	1.028940	1.04	1.064350	1.08	1.126680	1.14	1.143620	1.16
c15	2.709790	2.907190	1.07	2.914580	1.08	2.945730	1.09	2.967150	1.09	2.995040	1.11
c16	0.937587	0.732649	0.78	0.732867	0.78	0.922856	0.98	1.117175	1.19	1.204280	1.28
c17	0.944734	0.900293	0.95	0.907470	0.96	1.027264	1.09	1.104107	1.17	1.213370	1.28
c18	1.157650	1.139630	0.98	1.145540	0.99	1.273480	1.10	1.372390	1.19	1.455730	1.26
c19	1.449900	1.474140	1.02	1.481530	1.02	1.500910	1.04	1.684650	1.16	1.786190	1.23
c20	3.072990	3.111320	1.01	3.280290	1.07	3.133590	1.02	3.553970	1.16	3.642950	1.19
ave	0.995998	1.006420	1.01	1.020600	1.02	1.055030	1.06	1.121670	1.13	1.154830	1.16

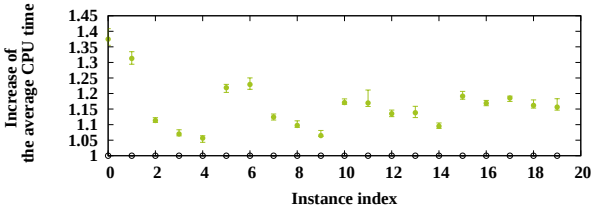


(a) degree centrality

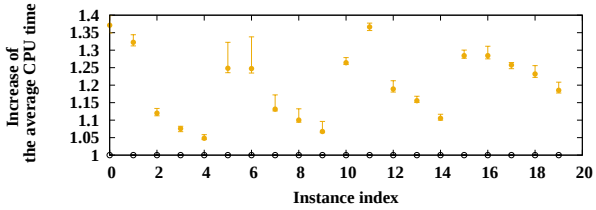
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality

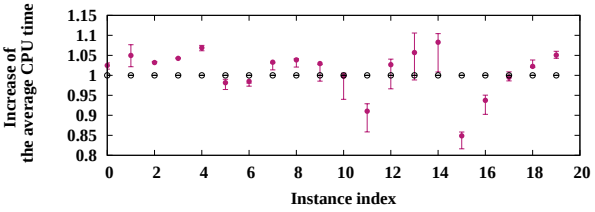


(e) edge betweenness centrality

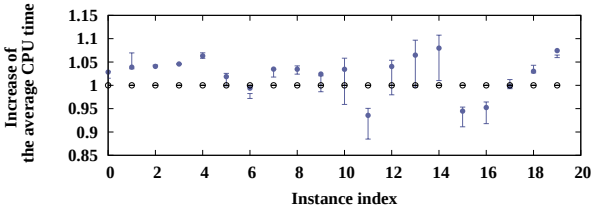
Figure 3.46: Increase of CPU time from the original method (ADH, C)

Table 3.39: CPU time [s] (ADH, D)

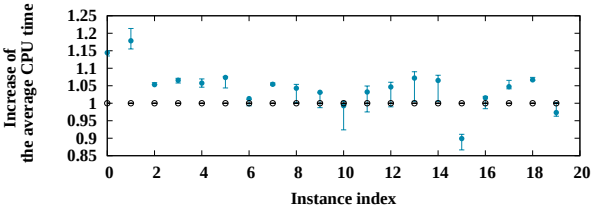
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
d01	0.939428	0.962629	1.02	0.966013	1.03	1.075111	1.14	1.285280	1.37	1.156546	1.23
d02	0.906859	0.951994	1.05	0.942231	1.04	1.068794	1.18	1.133208	1.25	1.244190	1.37
d03	3.167550	3.269060	1.03	3.295970	1.04	3.335920	1.05	3.461460	1.09	3.469470	1.10
d04	5.883150	6.133010	1.04	6.152040	1.05	6.268680	1.07	6.326290	1.08	6.331840	1.08
d05	22.261000	23.796800	1.07	23.664340	1.06	23.543560	1.06	23.402520	1.05	23.434960	1.05
d06	1.639920	1.609790	0.98	1.670600	1.02	1.760650	1.07	1.898320	1.16	2.168050	1.32
d07	1.624090	1.599740	0.99	1.614360	0.99	1.645560	1.01	1.874680	1.15	1.891670	1.16
d08	3.846880	3.973370	1.03	3.980320	1.03	4.054830	1.05	4.130320	1.07	4.230700	1.10
d09	6.544010	6.797050	1.04	6.770450	1.03	6.825570	1.04	6.875410	1.05	6.908000	1.06
d10	22.700700	23.366760	1.03	23.250900	1.02	23.409800	1.03	23.710520	1.04	23.875520	1.05
d11	3.387480	3.379900	1.00	3.504100	1.03	3.363700	0.99	3.836240	1.13	4.048720	1.20
d12	3.400800	3.095990	0.91	3.181860	0.94	3.510440	1.03	3.690730	1.09	4.029250	1.18
d13	5.540600	5.688090	1.03	5.764550	1.04	5.799090	1.05	6.094440	1.10	6.271040	1.13
d14	8.246220	8.715330	1.06	8.779560	1.06	8.839560	1.07	9.133550	1.11	9.238770	1.12
d15	23.881500	25.860720	1.08	25.786580	1.08	25.436180	1.07	25.721400	1.08	26.325120	1.10
d16	6.923630	5.875820	0.85	6.540740	0.94	6.224760	0.90	7.450420	1.08	8.368310	1.21
d17	6.932840	6.498900	0.94	6.603000	0.95	7.040150	1.02	7.692410	1.11	8.480910	1.22
d18	8.990840	8.946840	1.00	8.971770	1.00	9.415510	1.05	10.052630	1.12	10.721170	1.19
d19	11.564200	11.822120	1.02	11.913700	1.03	12.336640	1.07	13.017700	1.13	13.594280	1.18
d20	26.291400	27.620060	1.05	28.250900	1.07	25.594500	0.97	28.060540	1.07	28.567720	1.09
ave	8.317770	8.569710	1.03	8.647810	1.04	8.597570	1.03	8.992770	1.08	9.255060	1.11



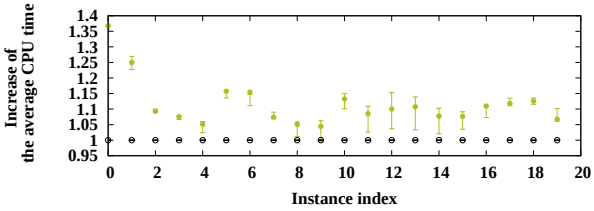
(a) degree centrality



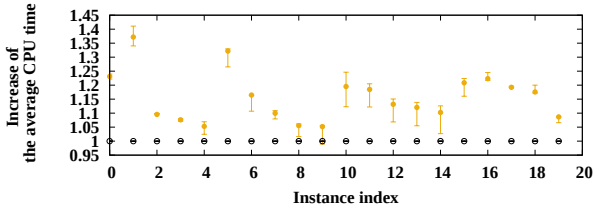
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality



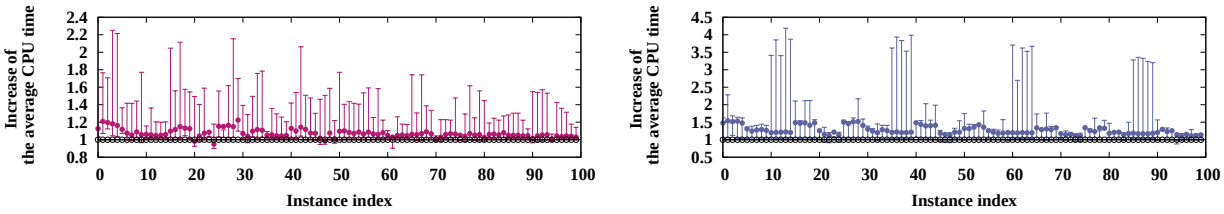
(e) edge betweenness centrality

Figure 3.47: Increase of CPU time from the original method (ADH, D)

Table 3.40: CPU time [s] (ADH, I080)

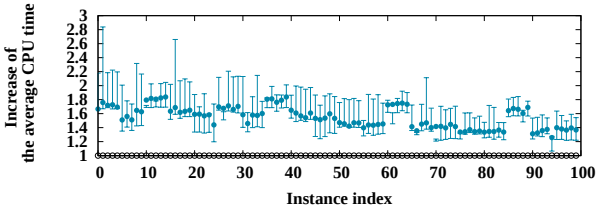
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
i080-001	0.001600	0.001803	1.13	0.002366	1.48	0.002665	1.67	0.002848	1.78	0.002795	1.75
i080-002	0.001543	0.001865	1.21	0.002378	1.54	0.002713	1.76	0.003005	1.95	0.002905	1.88
i080-003	0.001531	0.001831	1.20	0.002324	1.52	0.002630	1.72	0.002896	1.89	0.003051	1.99
i080-004	0.001572	0.001857	1.18	0.002407	1.53	0.002715	1.73	0.002817	1.79	0.002754	1.75
i080-005	0.001594	0.001853	1.16	0.002354	1.48	0.002699	1.69	0.002899	1.82	0.002874	1.80
i080-011	0.002560	0.002861	1.12	0.003351	1.31	0.003865	1.51	0.004337	1.69	0.004155	1.62
i080-012	0.002591	0.002785	1.07	0.003232	1.25	0.004041	1.56	0.004196	1.62	0.004419	1.71
i080-013	0.002538	0.002671	1.05	0.003237	1.28	0.003835	1.51	0.004146	1.63	0.004203	1.66
i080-014	0.002555	0.002781	1.09	0.003299	1.29	0.004207	1.65	0.004199	1.64	0.004291	1.68
i080-015	0.002591	0.002739	1.06	0.003282	1.27	0.004214	1.63	0.004236	1.63	0.004278	1.65
i080-021	0.006018	0.006385	1.06	0.007223	1.20	0.010798	1.79	0.010955	1.82	0.011101	1.84
i080-022	0.005968	0.006299	1.06	0.007218	1.21	0.010847	1.82	0.010752	1.80	0.011032	1.85
i080-023	0.005981	0.006268	1.05	0.007249	1.21	0.010812	1.81	0.010870	1.82	0.011298	1.89
i080-024	0.005960	0.006256	1.05	0.007283	1.22	0.010876	1.82	0.010756	1.80	0.011281	1.89
i080-025	0.005947	0.006300	1.06	0.007165	1.20	0.010931	1.84	0.010864	1.83	0.011309	1.90
i080-031	0.001867	0.002050	1.10	0.002779	1.49	0.003051	1.63	0.003152	1.69	0.003222	1.73
i080-032	0.001782	0.001989	1.12	0.002622	1.47	0.003006	1.69	0.003080	1.73	0.003154	1.77
i080-033	0.001837	0.002112	1.15	0.002722	1.48	0.002972	1.62	0.003123	1.70	0.003270	1.78
i080-034	0.001799	0.002036	1.13	0.002542	1.41	0.002938	1.63	0.003213	1.79	0.003213	1.79
i080-035	0.001813	0.002041	1.13	0.002690	1.48	0.002990	1.65	0.003226	1.78	0.003136	1.73
i080-041	0.003279	0.003222	0.98	0.004108	1.25	0.005216	1.59	0.005059	1.54	0.005383	1.64
i080-042	0.003276	0.003413	1.04	0.003779	1.15	0.005217	1.59	0.005167	1.58	0.005380	1.64
i080-043	0.003299	0.003548	1.08	0.003684	1.12	0.005172	1.57	0.005336	1.62	0.005414	1.64
i080-044	0.003320	0.003604	1.09	0.004038	1.22	0.005256	1.58	0.005363	1.62	0.005251	1.58
i080-045	0.003336	0.003166	0.95	0.003781	1.13	0.004802	1.44	0.005145	1.54	0.005446	1.63
i080-101	0.001627	0.001877	1.15	0.002446	1.50	0.002760	1.70	0.002878	1.77	0.002864	1.76
i080-102	0.001691	0.001946	1.15	0.002471	1.46	0.002834	1.68	0.002892	1.71	0.002989	1.77
i080-103	0.001637	0.001906	1.16	0.002475	1.51	0.002801	1.71	0.002841	1.74	0.002881	1.76
i080-104	0.001631	0.001874	1.15	0.002481	1.52	0.002707	1.66	0.002898	1.78	0.002909	1.78
i080-105	0.001674	0.002050	1.22	0.002357	1.41	0.002851	1.70	0.002960	1.77	0.002994	1.79
i080-111	0.002679	0.002869	1.07	0.003491	1.30	0.004243	1.58	0.004296	1.60	0.004327	1.62
i080-112	0.002661	0.002766	1.04	0.003308	1.24	0.003882	1.46	0.004316	1.62	0.004401	1.65
i080-113	0.002710	0.002973	1.10	0.003260	1.20	0.004279	1.58	0.004403	1.62	0.004373	1.61
i080-114	0.002676	0.002987	1.12	0.003439	1.29	0.004214	1.57	0.004237	1.58	0.004351	1.63
i080-115	0.002650	0.002936	1.11	0.003338	1.26	0.004244	1.60	0.004339	1.64	0.004371	1.65
i080-121	0.006079	0.006386	1.05	0.007344	1.21	0.010982	1.81	0.010839	1.78	0.011202	1.84
i080-122	0.006053	0.006393	1.06	0.007395	1.22	0.010961	1.81	0.010994	1.82	0.011401	1.88
i080-123	0.006075	0.006348	1.04	0.007312	1.20	0.010702	1.76	0.010950	1.80	0.011468	1.89
i080-124	0.006081	0.006348	1.04	0.007322	1.20	0.010896	1.79	0.010744	1.77	0.011293	1.86
i080-125	0.006027	0.006323	1.05	0.007315	1.21	0.011073	1.84	0.010959	1.82	0.011369	1.89
i080-131	0.001902	0.002149	1.13	0.002823	1.48	0.003137	1.65	0.003251	1.71	0.003290	1.73
i080-132	0.001949	0.002145	1.10	0.002798	1.44	0.003134	1.61	0.003235	1.66	0.003293	1.69
i080-133	0.001934	0.002211	1.14	0.002699	1.40	0.003031	1.57	0.003284	1.70	0.003279	1.70
i080-134	0.001964	0.002196	1.12	0.002748	1.40	0.003027	1.54	0.003383	1.72	0.003280	1.67
i080-135	0.001970	0.002119	1.08	0.002783	1.41	0.003169	1.61	0.003231	1.64	0.003398	1.72
i080-141	0.003346	0.003595	1.07	0.003963	1.18	0.005126	1.53	0.005346	1.60	0.005531	1.65
i080-142	0.003383	0.003442	1.02	0.003886	1.15	0.005120	1.51	0.005270	1.56	0.005559	1.64
i080-143	0.003365	0.003323	0.99	0.003783	1.12	0.005167	1.54	0.005251	1.56	0.005829	1.73
i080-144	0.003379	0.003637	1.08	0.004108	1.22	0.005399	1.60	0.005350	1.58	0.005571	1.65
i080-145	0.003410	0.003436	1.01	0.004142	1.21	0.005225	1.53	0.005229	1.53	0.005484	1.61

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
i080-201	0.002435	0.002669	1.10	0.003212	1.32	0.003581	1.47	0.003763	1.55	0.003722	1.53
i080-202	0.002401	0.002644	1.10	0.003162	1.32	0.003494	1.46	0.003679	1.53	0.003670	1.53
i080-203	0.002437	0.002639	1.08	0.003312	1.36	0.003462	1.42	0.003743	1.54	0.003710	1.52
i080-204	0.002426	0.002599	1.07	0.003461	1.43	0.003565	1.47	0.003725	1.54	0.003742	1.54
i080-205	0.002382	0.002588	1.09	0.003230	1.36	0.003495	1.47	0.003623	1.52	0.003712	1.56
i080-211	0.003475	0.003703	1.07	0.004376	1.26	0.004857	1.40	0.005094	1.47	0.005151	1.48
i080-212	0.003495	0.003799	1.09	0.004243	1.21	0.005027	1.44	0.005016	1.44	0.005231	1.50
i080-213	0.003513	0.003748	1.07	0.004158	1.18	0.005037	1.43	0.005132	1.46	0.005154	1.47
i080-214	0.003500	0.003712	1.06	0.004176	1.19	0.005056	1.44	0.005021	1.43	0.005182	1.48
i080-215	0.003479	0.003758	1.08	0.004196	1.21	0.005051	1.45	0.005108	1.47	0.005158	1.48
i080-221	0.006769	0.007102	1.05	0.008127	1.20	0.011681	1.73	0.011703	1.73	0.012284	1.81
i080-222	0.006783	0.006985	1.03	0.008087	1.19	0.011748	1.73	0.011673	1.72	0.012110	1.79
i080-223	0.006780	0.007118	1.05	0.008164	1.20	0.011826	1.74	0.011717	1.73	0.012070	1.78
i080-224	0.006742	0.007107	1.05	0.008076	1.20	0.011802	1.75	0.011614	1.72	0.012000	1.78
i080-225	0.006749	0.007057	1.05	0.008086	1.20	0.011706	1.73	0.011630	1.72	0.012233	1.81
i080-231	0.002757	0.002924	1.06	0.003689	1.34	0.003896	1.41	0.004057	1.47	0.004069	1.48
i080-232	0.002775	0.002938	1.06	0.003579	1.29	0.003760	1.35	0.004035	1.45	0.004068	1.47
i080-233	0.002739	0.002930	1.07	0.003581	1.31	0.003973	1.45	0.004099	1.50	0.004143	1.51
i080-234	0.002688	0.002933	1.09	0.003462	1.29	0.003947	1.47	0.004122	1.53	0.004142	1.54
i080-235	0.002674	0.002853	1.07	0.003587	1.34	0.003745	1.40	0.004111	1.54	0.004044	1.51
i080-241	0.004187	0.004228	1.01	0.004926	1.18	0.005926	1.42	0.006184	1.48	0.006249	1.49
i080-242	0.004155	0.004277	1.03	0.004754	1.14	0.005892	1.42	0.005974	1.44	0.006152	1.48
i080-243	0.004125	0.004373	1.06	0.004749	1.15	0.005758	1.40	0.005981	1.45	0.006390	1.55
i080-244	0.004147	0.004432	1.07	0.004627	1.12	0.005999	1.45	0.006151	1.48	0.006298	1.52
i080-245	0.004154	0.004422	1.06	0.004617	1.11	0.005876	1.41	0.005980	1.44	0.006173	1.49
i080-301	0.003046	0.003201	1.05	0.004097	1.35	0.004070	1.34	0.004239	1.39	0.004157	1.36
i080-302	0.003007	0.003127	1.04	0.003793	1.26	0.004033	1.34	0.004146	1.38	0.004185	1.39
i080-303	0.003011	0.003224	1.07	0.003716	1.23	0.004126	1.37	0.004209	1.40	0.004267	1.42
i080-304	0.003025	0.003183	1.05	0.004050	1.34	0.004062	1.34	0.004271	1.41	0.004250	1.40
i080-305	0.003043	0.003222	1.06	0.004036	1.33	0.004111	1.35	0.004246	1.40	0.004230	1.39
i080-311	0.004143	0.004251	1.03	0.004899	1.18	0.005534	1.34	0.005571	1.34	0.005751	1.39
i080-312	0.004167	0.004421	1.06	0.005037	1.21	0.005606	1.35	0.005749	1.38	0.005734	1.38
i080-313	0.004103	0.004367	1.06	0.004980	1.21	0.005516	1.34	0.005604	1.37	0.005903	1.44
i080-314	0.004116	0.004344	1.06	0.004698	1.14	0.005618	1.36	0.005758	1.40	0.005868	1.43
i080-315	0.004111	0.004438	1.08	0.004835	1.18	0.005494	1.34	0.005685	1.38	0.005770	1.40
i080-321	0.007396	0.007761	1.05	0.008760	1.18	0.012145	1.64	0.012095	1.64	0.012715	1.72
i080-322	0.007410	0.007785	1.05	0.008691	1.17	0.012395	1.67	0.012394	1.67	0.012831	1.73
i080-323	0.007349	0.007721	1.05	0.008613	1.17	0.012214	1.66	0.012177	1.66	0.012685	1.73
i080-324	0.007358	0.007686	1.04	0.008612	1.17	0.011794	1.60	0.012211	1.66	0.012557	1.71
i080-325	0.007368	0.007711	1.05	0.008723	1.18	0.012444	1.69	0.012267	1.66	0.012717	1.73
i080-331	0.003435	0.003488	1.02	0.004148	1.21	0.004509	1.31	0.004717	1.37	0.004771	1.39
i080-332	0.003390	0.003541	1.04	0.004392	1.30	0.004488	1.32	0.004638	1.37	0.004744	1.40
i080-333	0.003377	0.003558	1.05	0.004158	1.23	0.004582	1.36	0.004692	1.39	0.004699	1.39
i080-334	0.003339	0.003529	1.06	0.004193	1.26	0.004589	1.37	0.004676	1.40	0.004731	1.42
i080-335	0.003643	0.003663	1.01	0.004166	1.14	0.004582	1.26	0.004790	1.31	0.004839	1.33
i080-341	0.004789	0.004997	1.04	0.005282	1.10	0.006701	1.40	0.006790	1.42	0.006954	1.45
i080-342	0.004811	0.004984	1.04	0.005540	1.15	0.006640	1.38	0.006623	1.38	0.006772	1.41
i080-343	0.004846	0.005061	1.04	0.005387	1.11	0.006613	1.36	0.006757	1.39	0.006966	1.44
i080-344	0.004816	0.005010	1.04	0.005397	1.12	0.006717	1.39	0.006838	1.42	0.006875	1.43
i080-345	0.004794	0.004936	1.03	0.005458	1.14	0.006552	1.37	0.006552	1.37	0.006673	1.39
ave	0.003627	0.003849	1.06	0.004474	1.23	0.005690	1.57	0.005798	1.60	0.005949	1.64

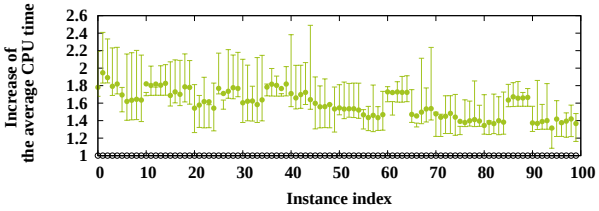


(a) degree centrality

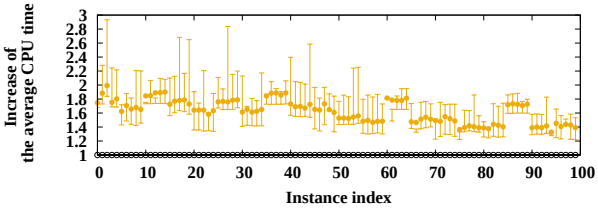
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality



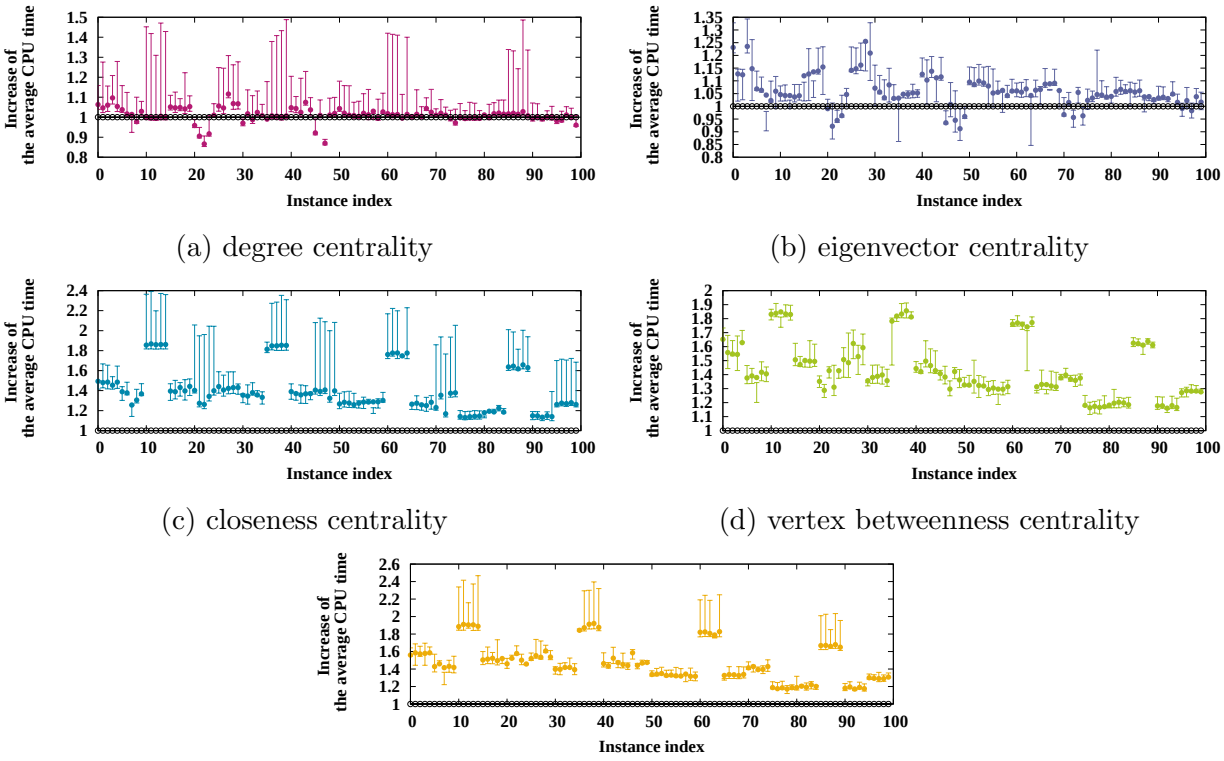
(e) edge betweenness centrality

Figure 3.48: Increase of CPU time from the original method (ADH, I080)

Table 3.41: CPU time [s] (ADH, I160)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
i160-001	0.007891	0.008392	1.06	0.009716	1.23	0.011786	1.49	0.013039	1.65	0.012310	1.56
i160-002	0.007543	0.007894	1.05	0.008504	1.13	0.011171	1.48	0.011751	1.56	0.011962	1.59
i160-003	0.007690	0.008153	1.06	0.008644	1.12	0.011416	1.48	0.011900	1.55	0.012095	1.57
i160-004	0.007606	0.008341	1.10	0.009396	1.24	0.011061	1.45	0.011749	1.54	0.012004	1.58
i160-005	0.007553	0.007960	1.05	0.008668	1.15	0.011215	1.48	0.012303	1.63	0.011976	1.59
i160-011	0.014691	0.015247	1.04	0.015696	1.07	0.020407	1.39	0.020215	1.38	0.020982	1.43
i160-012	0.014773	0.015017	1.02	0.015705	1.06	0.020344	1.38	0.020551	1.39	0.021567	1.46
i160-013	0.014958	0.015164	1.01	0.015618	1.04	0.018794	1.26	0.020630	1.38	0.021172	1.42
i160-014	0.014750	0.014419	0.98	0.015086	1.02	0.019254	1.31	0.020904	1.42	0.021106	1.43
i160-015	0.014943	0.015378	1.03	0.015826	1.06	0.020436	1.37	0.020998	1.41	0.021198	1.42
i160-021	0.037625	0.037714	1.00	0.039367	1.05	0.069795	1.86	0.068890	1.83	0.070901	1.88
i160-022	0.037673	0.037671	1.00	0.039272	1.04	0.070350	1.87	0.069241	1.84	0.071970	1.91
i160-023	0.037772	0.037601	1.00	0.039362	1.04	0.070227	1.86	0.069805	1.85	0.071939	1.90
i160-024	0.037641	0.037817	1.00	0.039024	1.04	0.070068	1.86	0.069012	1.83	0.071711	1.91
i160-025	0.037640	0.037731	1.00	0.039203	1.04	0.070075	1.86	0.068873	1.83	0.071098	1.89
i160-031	0.009530	0.010019	1.05	0.010673	1.12	0.013294	1.39	0.014356	1.51	0.014341	1.50
i160-032	0.009366	0.009808	1.05	0.010556	1.13	0.013011	1.39	0.013787	1.47	0.014173	1.51
i160-033	0.009328	0.009781	1.05	0.010590	1.14	0.013346	1.43	0.014001	1.50	0.014208	1.52
i160-034	0.009465	0.009859	1.04	0.010776	1.14	0.013222	1.40	0.014165	1.50	0.014153	1.50
i160-035	0.009246	0.009739	1.05	0.010673	1.15	0.013332	1.44	0.013819	1.49	0.014058	1.52
i160-041	0.024144	0.023147	0.96	0.023920	0.99	0.033859	1.40	0.032642	1.35	0.035236	1.46
i160-042	0.023966	0.021695	0.91	0.022092	0.92	0.030509	1.27	0.030870	1.29	0.036502	1.52
i160-043	0.023942	0.020742	0.87	0.022614	0.94	0.030210	1.26	0.034203	1.43	0.037774	1.58
i160-044	0.024029	0.022032	0.92	0.023146	0.96	0.032252	1.34	0.031485	1.31	0.036079	1.50
i160-045	0.024173	0.024411	1.01	0.025298	1.05	0.033810	1.40	0.034520	1.43	0.035240	1.46
i160-101	0.008585	0.009066	1.06	0.009798	1.14	0.012365	1.44	0.012940	1.51	0.013052	1.52
i160-102	0.008516	0.008909	1.05	0.009771	1.15	0.011978	1.41	0.012650	1.49	0.013224	1.55
i160-103	0.008478	0.009460	1.12	0.009849	1.16	0.012057	1.42	0.013758	1.62	0.013014	1.54
i160-104	0.008432	0.009006	1.07	0.010584	1.26	0.012063	1.43	0.012895	1.53	0.013551	1.61
i160-105	0.008502	0.009074	1.07	0.010280	1.21	0.012167	1.43	0.013539	1.59	0.013074	1.54
i160-111	0.015613	0.015147	0.97	0.016725	1.07	0.021148	1.35	0.021178	1.36	0.021821	1.40
i160-112	0.015578	0.015849	1.02	0.016454	1.06	0.020961	1.35	0.021498	1.38	0.021767	1.40
i160-113	0.015611	0.015415	0.99	0.016139	1.03	0.021537	1.38	0.021668	1.39	0.022140	1.42
i160-114	0.015657	0.016032	1.02	0.016975	1.08	0.021247	1.36	0.021882	1.40	0.022203	1.42
i160-115	0.015747	0.015871	1.01	0.016237	1.03	0.020983	1.33	0.021367	1.36	0.021945	1.39
i160-121	0.038928	0.038554	0.99	0.040181	1.03	0.070592	1.81	0.069378	1.78	0.071733	1.84
i160-122	0.038334	0.038606	1.01	0.040149	1.05	0.070825	1.85	0.069711	1.82	0.071751	1.87
i160-123	0.038319	0.038525	1.01	0.040125	1.05	0.070825	1.85	0.070247	1.83	0.073257	1.91
i160-124	0.038280	0.038383	1.00	0.040299	1.05	0.070961	1.85	0.071048	1.86	0.073526	1.92
i160-125	0.038247	0.038497	1.01	0.040241	1.05	0.070816	1.85	0.069307	1.81	0.071761	1.88
i160-131	0.010237	0.010723	1.05	0.011527	1.13	0.014232	1.39	0.014772	1.44	0.014973	1.46
i160-132	0.010274	0.010717	1.04	0.011334	1.10	0.014077	1.37	0.014602	1.42	0.014756	1.44
i160-133	0.010191	0.010435	1.02	0.011593	1.14	0.013834	1.36	0.015249	1.50	0.015535	1.52
i160-134	0.010280	0.011046	1.07	0.011429	1.11	0.014064	1.37	0.015055	1.46	0.015152	1.47
i160-135	0.010241	0.010617	1.04	0.011424	1.12	0.014028	1.37	0.014634	1.43	0.014880	1.45
i160-141	0.024637	0.022703	0.92	0.023038	0.94	0.034597	1.40	0.034665	1.41	0.035466	1.44
i160-142	0.024510	0.024731	1.01	0.024697	1.01	0.034109	1.39	0.033902	1.38	0.038850	1.59
i160-143	0.024685	0.021477	0.87	0.023336	0.95	0.034711	1.41	0.032019	1.30	0.035603	1.44
i160-144	0.024639	0.024902	1.01	0.022470	0.91	0.032607	1.32	0.035033	1.42	0.036189	1.47
i160-145	0.024590	0.025055	1.02	0.023602	0.96	0.034370	1.40	0.033490	1.36	0.036278	1.48

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
i160-201	0.012802	0.013354	1.04	0.014020	1.10	0.016225	1.27	0.016975	1.33	0.017129	1.34
i160-202	0.012778	0.013039	1.02	0.013875	1.09	0.016403	1.28	0.016933	1.33	0.017153	1.34
i160-203	0.012591	0.012716	1.01	0.013839	1.10	0.016065	1.28	0.017021	1.35	0.016986	1.35
i160-204	0.012741	0.012847	1.01	0.013909	1.09	0.016033	1.26	0.016848	1.32	0.016899	1.33
i160-205	0.012790	0.012844	1.00	0.013809	1.08	0.016295	1.27	0.016872	1.32	0.017019	1.33
i160-211	0.019901	0.020297	1.02	0.020938	1.05	0.025575	1.29	0.025754	1.29	0.026321	1.32
i160-212	0.019750	0.019991	1.01	0.020838	1.06	0.025404	1.29	0.025755	1.30	0.026077	1.32
i160-213	0.019612	0.020199	1.03	0.020840	1.06	0.025256	1.29	0.025422	1.30	0.026332	1.34
i160-214	0.019584	0.019536	1.00	0.020412	1.04	0.025234	1.29	0.025372	1.30	0.025796	1.32
i160-215	0.019675	0.020200	1.03	0.020852	1.06	0.025614	1.30	0.025854	1.31	0.025921	1.32
i160-221	0.041853	0.042585	1.02	0.044383	1.06	0.073735	1.76	0.073706	1.76	0.076213	1.82
i160-222	0.041773	0.042302	1.01	0.044094	1.06	0.074186	1.78	0.073918	1.77	0.076191	1.82
i160-223	0.041836	0.042396	1.01	0.044696	1.07	0.074339	1.78	0.073628	1.76	0.075584	1.81
i160-224	0.042663	0.042463	1.00	0.044472	1.04	0.074567	1.75	0.074294	1.74	0.076320	1.79
i160-225	0.041825	0.042417	1.01	0.044477	1.06	0.074265	1.78	0.074101	1.77	0.076408	1.83
i160-231	0.014538	0.014586	1.00	0.015495	1.07	0.018383	1.26	0.019096	1.31	0.019286	1.33
i160-232	0.014438	0.014652	1.01	0.015694	1.09	0.018395	1.27	0.019204	1.33	0.019295	1.34
i160-233	0.014513	0.014581	1.00	0.015794	1.09	0.018251	1.26	0.019280	1.33	0.019364	1.33
i160-234	0.014301	0.014926	1.04	0.015600	1.09	0.017876	1.25	0.018863	1.32	0.018956	1.33
i160-235	0.014343	0.014699	1.02	0.015233	1.06	0.018414	1.28	0.018821	1.31	0.019201	1.34
i160-241	0.028344	0.028631	1.01	0.027423	0.97	0.034867	1.23	0.039059	1.38	0.040069	1.41
i160-242	0.028043	0.028603	1.02	0.028464	1.02	0.037950	1.35	0.039168	1.40	0.040049	1.43
i160-243	0.028310	0.028603	1.01	0.027072	0.96	0.033074	1.17	0.038735	1.37	0.039440	1.39
i160-244	0.028145	0.027941	0.99	0.029681	1.05	0.038664	1.37	0.038253	1.36	0.039285	1.40
i160-245	0.028012	0.027230	0.97	0.026969	0.96	0.038610	1.38	0.038535	1.38	0.040013	1.43
i160-301	0.022970	0.022934	1.00	0.023504	1.02	0.026213	1.14	0.027122	1.18	0.027288	1.19
i160-302	0.023044	0.022889	0.99	0.023790	1.03	0.026062	1.13	0.026816	1.16	0.027094	1.18
i160-303	0.022623	0.022539	1.00	0.023668	1.05	0.025775	1.14	0.026541	1.17	0.026819	1.19
i160-304	0.022937	0.022889	1.00	0.023947	1.04	0.026230	1.14	0.026735	1.17	0.026874	1.17
i160-305	0.022877	0.022746	0.99	0.023708	1.04	0.026218	1.15	0.026794	1.17	0.027159	1.19
i160-311	0.030363	0.030704	1.01	0.031540	1.04	0.035814	1.18	0.035881	1.18	0.036074	1.19
i160-312	0.029896	0.029871	1.00	0.031632	1.06	0.035691	1.19	0.035751	1.20	0.035999	1.20
i160-313	0.030218	0.030755	1.02	0.032236	1.07	0.035876	1.19	0.036336	1.20	0.036037	1.19
i160-314	0.029986	0.030606	1.02	0.031731	1.06	0.036722	1.22	0.035935	1.20	0.036604	1.22
i160-315	0.030349	0.030821	1.02	0.032204	1.06	0.035964	1.19	0.035949	1.18	0.036460	1.20
i160-321	0.051751	0.052619	1.02	0.054650	1.06	0.084710	1.64	0.084087	1.62	0.086324	1.67
i160-322	0.051789	0.052821	1.02	0.054957	1.06	0.085174	1.64	0.083979	1.62	0.086521	1.67
i160-323	0.052265	0.053066	1.02	0.054200	1.04	0.084552	1.62	0.084195	1.61	0.086720	1.66
i160-324	0.051416	0.052845	1.03	0.053376	1.04	0.085227	1.66	0.084212	1.64	0.086316	1.68
i160-325	0.052002	0.052391	1.01	0.053382	1.03	0.084760	1.63	0.083766	1.61	0.085773	1.65
i160-331	0.024853	0.024672	0.99	0.025669	1.03	0.028538	1.15	0.029254	1.18	0.029349	1.18
i160-332	0.024824	0.024860	1.00	0.025711	1.04	0.028482	1.15	0.029223	1.18	0.029665	1.20
i160-333	0.025154	0.024941	0.99	0.025922	1.03	0.028524	1.13	0.029194	1.16	0.029484	1.17
i160-334	0.024951	0.024997	1.00	0.026157	1.05	0.028676	1.15	0.029410	1.18	0.029701	1.19
i160-335	0.024837	0.024935	1.00	0.025222	1.02	0.028308	1.14	0.028934	1.16	0.029193	1.18
i160-341	0.037874	0.037184	0.98	0.037547	0.99	0.047783	1.26	0.048147	1.27	0.049252	1.30
i160-342	0.037846	0.037274	0.98	0.038689	1.02	0.048238	1.27	0.048618	1.28	0.048976	1.29
i160-343	0.037540	0.037968	1.01	0.036893	0.98	0.047671	1.27	0.048147	1.28	0.048443	1.29
i160-344	0.037315	0.037297	1.00	0.038725	1.04	0.047735	1.28	0.047872	1.28	0.048130	1.29
i160-345	0.037690	0.036207	0.96	0.038306	1.02	0.047535	1.26	0.048129	1.28	0.049318	1.31
ave	0.023540	0.023634	1.00	0.024534	1.04	0.034302	1.46	0.034680	1.47	0.035665	1.52



(a) degree centrality

(b) eigenvector centrality

(c) closeness centrality

(d) vertex betweenness centrality

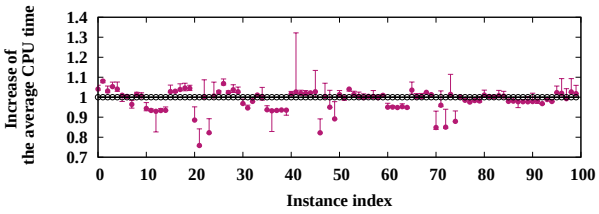
(e) edge betweenness centrality

Figure 3.49: Increase of CPU time from the original method (ADH, I160)

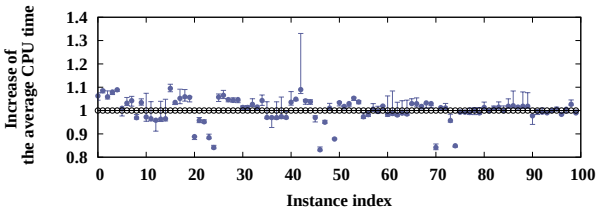
Table 3.42: CPU time [s] (ADH, I320)

	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
i320-001	0.044484	0.046286	1.04	0.047273	1.06	0.059947	1.35	0.063906	1.44	0.063096	1.42
i320-002	0.044835	0.048453	1.08	0.048653	1.09	0.059428	1.33	0.064189	1.43	0.063909	1.43
i320-003	0.044015	0.045397	1.03	0.046621	1.06	0.059047	1.34	0.062354	1.42	0.061966	1.41
i320-004	0.044720	0.047086	1.05	0.048280	1.08	0.060239	1.35	0.062973	1.41	0.064346	1.44
i320-005	0.044661	0.046473	1.04	0.048639	1.09	0.060296	1.35	0.062092	1.39	0.062660	1.40
i320-011	0.096802	0.097773	1.01	0.097750	1.01	0.117700	1.22	0.120959	1.25	0.123612	1.28
i320-012	0.096047	0.096441	1.00	0.099065	1.03	0.119997	1.25	0.122318	1.27	0.124462	1.30
i320-013	0.095344	0.091941	0.96	0.099367	1.04	0.112454	1.18	0.122146	1.28	0.123214	1.29
i320-014	0.095829	0.097239	1.01	0.092906	0.97	0.119903	1.25	0.120886	1.26	0.122774	1.28
i320-015	0.095840	0.096422	1.01	0.099101	1.03	0.109810	1.15	0.118970	1.24	0.122414	1.28
i320-021	0.270273	0.255113	0.94	0.262801	0.97	0.525324	1.94	0.513252	1.90	0.531367	1.97
i320-022	0.270598	0.252664	0.93	0.261084	0.96	0.522373	1.93	0.512877	1.90	0.529692	1.96
i320-023	0.271530	0.252406	0.93	0.260281	0.96	0.520650	1.92	0.511330	1.88	0.529299	1.95
i320-024	0.270567	0.252550	0.93	0.260565	0.96	0.517931	1.91	0.511910	1.89	0.533109	1.97
i320-025	0.270251	0.252557	0.93	0.260758	0.96	0.521227	1.93	0.507952	1.88	0.528391	1.96
i320-031	0.055517	0.057096	1.03	0.060784	1.09	0.071082	1.28	0.074116	1.34	0.074922	1.35
i320-032	0.055662	0.057327	1.03	0.057609	1.03	0.071439	1.28	0.073796	1.33	0.074723	1.34
i320-033	0.055800	0.057940	1.04	0.058698	1.05	0.071830	1.29	0.074745	1.34	0.075093	1.35
i320-034	0.056336	0.058931	1.05	0.059664	1.06	0.073488	1.30	0.074703	1.33	0.075415	1.34
i320-035	0.055976	0.058471	1.04	0.059136	1.06	0.072032	1.29	0.074838	1.34	0.075010	1.34
i320-041	0.194672	0.172431	0.89	0.172889	0.89	0.244508	1.26	0.256813	1.32	0.281644	1.45
i320-042	0.194884	0.147785	0.76	0.186639	0.96	0.254791	1.31	0.259336	1.33	0.270935	1.39
i320-043	0.194716	0.194949	1.00	0.185312	0.95	0.244629	1.26	0.256591	1.32	0.281043	1.44
i320-044	0.194503	0.159988	0.82	0.171925	0.88	0.210847	1.08	0.250084	1.29	0.268606	1.38
i320-045	0.194555	0.195706	1.01	0.163720	0.84	0.255578	1.31	0.261195	1.34	0.270799	1.39
i320-101	0.048980	0.050279	1.03	0.051799	1.06	0.063711	1.30	0.068091	1.39	0.067716	1.38
i320-102	0.048580	0.051908	1.07	0.051747	1.07	0.063344	1.30	0.067168	1.38	0.069867	1.44
i320-103	0.049455	0.050615	1.02	0.051780	1.05	0.063028	1.27	0.065783	1.33	0.066054	1.34
i320-104	0.050325	0.052194	1.04	0.052529	1.04	0.064595	1.28	0.067539	1.34	0.068940	1.37
i320-105	0.050460	0.051843	1.03	0.052834	1.05	0.065048	1.29	0.068031	1.35	0.067695	1.34
i320-111	0.100650	0.097495	0.97	0.101984	1.01	0.115421	1.15	0.124189	1.23	0.129344	1.29
i320-112	0.100602	0.095204	0.95	0.101951	1.01	0.123764	1.23	0.124088	1.23	0.126457	1.26
i320-113	0.099744	0.097671	0.98	0.102390	1.03	0.122259	1.23	0.123855	1.24	0.125510	1.26
i320-114	0.100819	0.101894	1.01	0.102345	1.02	0.123397	1.22	0.124811	1.24	0.126245	1.25
i320-115	0.100526	0.100894	1.00	0.104825	1.04	0.126375	1.26	0.126029	1.25	0.126134	1.25
i320-121	0.272316	0.255254	0.94	0.264256	0.97	0.519433	1.91	0.514552	1.89	0.528964	1.94
i320-122	0.273098	0.254616	0.93	0.264963	0.97	0.522574	1.91	0.509911	1.87	0.533513	1.95
i320-123	0.272725	0.254694	0.93	0.264513	0.97	0.521376	1.91	0.512007	1.88	0.530322	1.94
i320-124	0.272064	0.254668	0.94	0.264922	0.97	0.518199	1.90	0.513131	1.89	0.529411	1.95
i320-125	0.272657	0.254970	0.94	0.264655	0.97	0.521897	1.91	0.509811	1.87	0.530013	1.94
i320-131	0.061309	0.062365	1.02	0.063497	1.04	0.076408	1.25	0.078950	1.29	0.079673	1.30
i320-132	0.059829	0.061407	1.03	0.062740	1.05	0.074097	1.24	0.077472	1.29	0.078181	1.31
i320-133	0.060781	0.061962	1.02	0.066238	1.09	0.076856	1.26	0.078267	1.29	0.079023	1.30
i320-134	0.061708	0.062904	1.02	0.064266	1.04	0.076620	1.24	0.079339	1.29	0.081852	1.33
i320-135	0.060410	0.061717	1.02	0.062671	1.04	0.075831	1.26	0.077790	1.29	0.079725	1.32
i320-141	0.197044	0.202426	1.03	0.191057	0.97	0.260027	1.32	0.263556	1.34	0.273543	1.39
i320-142	0.197053	0.162016	0.82	0.163880	0.83	0.256851	1.30	0.264614	1.34	0.272128	1.38
i320-143	0.197381	0.198052	1.00	0.187537	0.95	0.255526	1.29	0.260687	1.32	0.271935	1.38
i320-144	0.197292	0.187436	0.95	0.199362	1.01	0.251880	1.28	0.263418	1.34	0.273011	1.38
i320-145	0.197086	0.175709	0.89	0.173006	0.88	0.248258	1.26	0.257221	1.31	0.271202	1.38

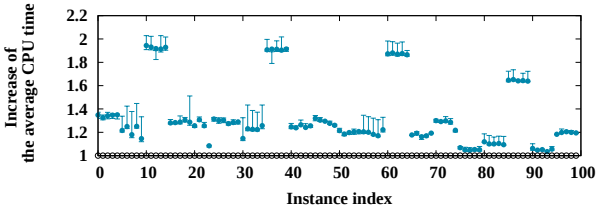
	wc	d_v		e_v		c_v		b_v		b_e	
name	ave	ave	inc	ave	inc	ave	inc	ave	inc	ave	inc
i320-201	0.068385	0.069292	1.01	0.070702	1.03	0.083061	1.21	0.085009	1.24	0.084764	1.24
i320-202	0.067834	0.067641	1.00	0.069022	1.02	0.080408	1.19	0.084186	1.24	0.084528	1.25
i320-203	0.069398	0.072218	1.04	0.071462	1.03	0.083126	1.20	0.085626	1.23	0.086395	1.24
i320-204	0.068652	0.069610	1.01	0.072282	1.05	0.082586	1.20	0.085287	1.24	0.085730	1.25
i320-205	0.068006	0.068273	1.00	0.070519	1.04	0.082056	1.21	0.083943	1.23	0.085067	1.25
i320-211	0.119050	0.118778	1.00	0.115896	0.97	0.143312	1.20	0.143565	1.21	0.144961	1.22
i320-212	0.118371	0.118887	1.00	0.116316	0.98	0.142139	1.20	0.142817	1.21	0.144086	1.22
i320-213	0.119155	0.119594	1.00	0.119917	1.01	0.140843	1.18	0.141320	1.19	0.144501	1.21
i320-214	0.118232	0.117969	1.00	0.117780	1.00	0.138388	1.17	0.141994	1.20	0.143504	1.21
i320-215	0.118650	0.119764	1.01	0.120924	1.02	0.144937	1.22	0.145055	1.22	0.147055	1.24
i320-221	0.286831	0.272523	0.95	0.282172	0.98	0.537259	1.87	0.527806	1.84	0.543993	1.90
i320-222	0.286623	0.272398	0.95	0.283007	0.99	0.538901	1.88	0.526810	1.84	0.547126	1.91
i320-223	0.286933	0.272155	0.95	0.281472	0.98	0.536488	1.87	0.528939	1.84	0.546378	1.90
i320-224	0.286469	0.272896	0.95	0.283234	0.99	0.537265	1.88	0.528730	1.85	0.545566	1.90
i320-225	0.287178	0.272520	0.95	0.282922	0.99	0.535486	1.86	0.527368	1.84	0.548012	1.91
i320-231	0.078413	0.081210	1.04	0.080754	1.03	0.092324	1.18	0.095231	1.21	0.096443	1.23
i320-232	0.078826	0.079011	1.00	0.081136	1.03	0.093926	1.19	0.095415	1.21	0.097011	1.23
i320-233	0.080715	0.080999	1.00	0.082138	1.02	0.093755	1.16	0.097328	1.21	0.098050	1.21
i320-234	0.080381	0.082348	1.02	0.082999	1.03	0.094037	1.17	0.098073	1.22	0.098609	1.23
i320-235	0.078341	0.079365	1.01	0.080672	1.03	0.093408	1.19	0.095401	1.22	0.096786	1.24
i320-241	0.210925	0.178965	0.85	0.177502	0.84	0.274237	1.30	0.269341	1.28	0.285390	1.35
i320-242	0.212115	0.203623	0.96	0.214741	1.01	0.274073	1.29	0.276843	1.31	0.284788	1.34
i320-243	0.211153	0.179490	0.85	0.213464	1.01	0.274107	1.30	0.277030	1.31	0.286416	1.36
i320-244	0.211576	0.214605	1.01	0.202572	0.96	0.272154	1.29	0.278978	1.32	0.286660	1.35
i320-245	0.212304	0.186630	0.88	0.180244	0.85	0.258409	1.22	0.278144	1.31	0.285667	1.35
i320-301	0.184505	0.184868	1.00	0.183154	0.99	0.197029	1.07	0.198093	1.07	0.203389	1.10
i320-302	0.187823	0.184964	0.98	0.186483	0.99	0.196985	1.05	0.200324	1.07	0.200853	1.07
i320-303	0.190435	0.185786	0.98	0.188832	0.99	0.199170	1.05	0.202598	1.06	0.203730	1.07
i320-304	0.189703	0.186529	0.98	0.190168	1.00	0.199494	1.05	0.205813	1.08	0.204398	1.08
i320-305	0.187638	0.184158	0.98	0.185713	0.99	0.197144	1.05	0.202307	1.08	0.202728	1.08
i320-311	0.234461	0.237008	1.01	0.237555	1.01	0.262175	1.12	0.258772	1.10	0.259659	1.11
i320-312	0.236700	0.237480	1.00	0.236383	1.00	0.260455	1.10	0.259868	1.10	0.259616	1.10
i320-313	0.233546	0.234023	1.00	0.235789	1.01	0.256966	1.10	0.256084	1.10	0.256161	1.10
i320-314	0.236468	0.238304	1.01	0.239865	1.01	0.261004	1.10	0.258478	1.09	0.259504	1.10
i320-315	0.235855	0.237196	1.01	0.235749	1.00	0.258000	1.09	0.256415	1.09	0.257756	1.09
i320-321	0.389554	0.381441	0.98	0.396700	1.02	0.641596	1.65	0.641011	1.65	0.651904	1.67
i320-322	0.386673	0.379123	0.98	0.394640	1.02	0.638822	1.65	0.634423	1.64	0.648123	1.68
i320-323	0.391279	0.382390	0.98	0.397112	1.01	0.642020	1.64	0.635985	1.63	0.651838	1.67
i320-324	0.391039	0.382260	0.98	0.398190	1.02	0.642555	1.64	0.636166	1.63	0.652904	1.67
i320-325	0.391058	0.381992	0.98	0.398291	1.02	0.640899	1.64	0.636640	1.63	0.649695	1.66
i320-331	0.198663	0.194321	0.98	0.194192	0.98	0.210834	1.06	0.215181	1.08	0.218088	1.10
i320-332	0.198813	0.194936	0.98	0.198063	1.00	0.208134	1.05	0.213915	1.08	0.215778	1.09
i320-333	0.200863	0.194432	0.97	0.199412	0.99	0.211189	1.05	0.214344	1.07	0.214962	1.07
i320-334	0.199860	0.197580	0.99	0.197942	0.99	0.206764	1.03	0.213526	1.07	0.216739	1.08
i320-335	0.200876	0.196592	0.98	0.200833	1.00	0.212002	1.06	0.215372	1.07	0.216267	1.08
i320-341	0.316194	0.323558	1.02	0.318220	1.01	0.374145	1.18	0.382560	1.21	0.392235	1.24
i320-342	0.319130	0.325455	1.02	0.313862	0.98	0.382325	1.20	0.379344	1.19	0.389555	1.22
i320-343	0.315670	0.313375	0.99	0.317413	1.01	0.380250	1.20	0.386403	1.22	0.394508	1.25
i320-344	0.314519	0.322808	1.03	0.322919	1.03	0.377008	1.20	0.384019	1.22	0.392103	1.25
i320-345	0.315165	0.320814	1.02	0.312012	0.99	0.376443	1.19	0.383572	1.22	0.392365	1.24
ave	0.170122	0.165067	0.97	0.168065	0.99	0.240628	1.41	0.242239	1.42	0.248726	1.46



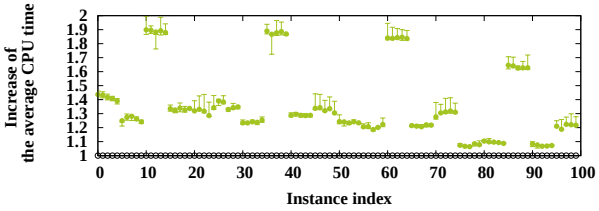
(a) degree centrality



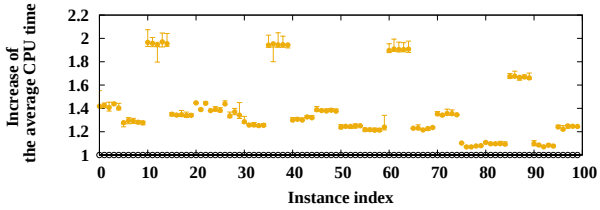
(b) eigenvector centrality



(c) closeness centrality



(d) vertex betweenness centrality



(e) edge betweenness centrality

Figure 3.50: Increase of CPU time from the original method (ADH, I320)

Next, we investigate the tradeoff between the increase in CPU time and the decrease in average gap. It is natural that the computation time increases as the average gap become smaller. Thus, the balance of these aspects is important for evaluating the performance of algorithms.

The tradeoff between the increase in CPU time and the decrease in average gaps for benchmark problem is shown in Figs. 3.51–3.53, and we composed them in Fig. 3.54. In Figs. 3.51–3.54, the horizontal axis is the increase in CPU time from the original method, and the vertical axis is the decrease in average gap, where d_v is the degree centrality, e_v is the eigenvector centrality, c_v is the closeness centrality, b_v is the vertex betweenness centrality, and b_e is the edge betweenness centrality. Thus, plots in the top left indicate a small increase in computation time and large decrease in average gap, i.e., using network centralities shows good performance.

From Fig. 3.51, it is revealed that using network centralities decreases the average gap for almost all instances because almost all plots are in the positive area. From Fig. 3.51 (d), (e), and (f), the average gap decreases by approximately 40% with double the computation time. These instances are the last five instances of each benchmark problem set. These instances have the largest number of terminals. Thus, using the network centrality, especially betweenness centrality shows good performance.

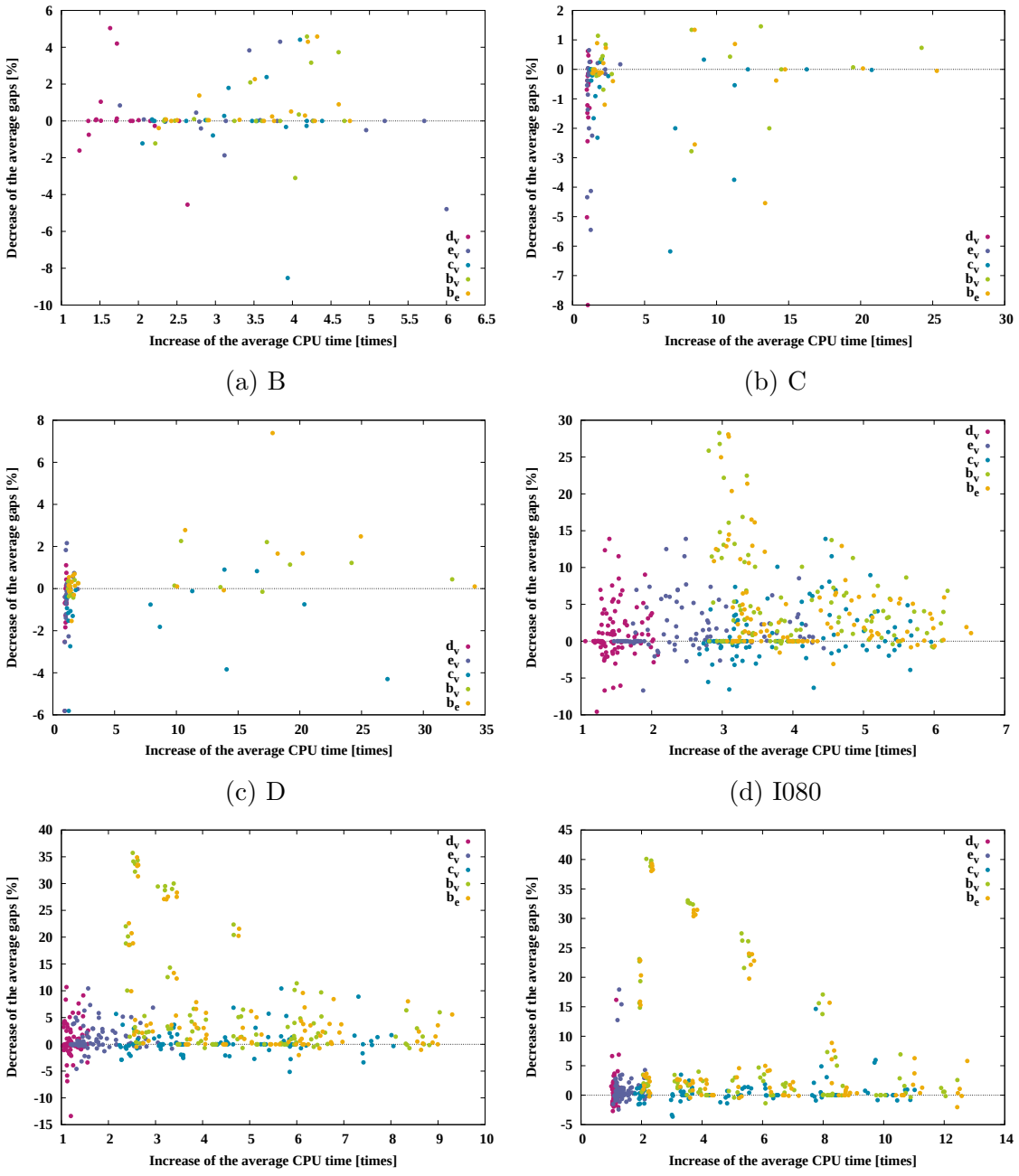
From Fig. 3.52, it is revealed that using network centralities decreases the average gap for almost all instances because almost all plots are in the positive area. From Fig. 3.52 (d), (e), and (f), the average gap decreases by approximately 40% with double the computation time. This trend is the same as the results of SPH. These instances are the last five instances of each benchmark problem set. These instances have the largest number of terminals. Thus, using the network centrality, especially betweenness centrality shows good performance.

From Fig. 3.53, it is revealed that using network centralities decreases the average gap for almost all instances because almost all plots are in the positive area. The increase of the computation time by using the network centrality is at most twice the computation time of the original ADH. From Fig. 3.53 (c), the average gap decreases at most 30% with 1.1 times the computation time of the original ADH. This trend differs from the results of SPH and DNH.

Figure 3.54 shows the composed figures of Figs. 3.51, 3.52, and 3.53. From Figs. 3.54 (a) and (b), the increase of the computation time of using degree centrality is very small, on the other hand, the increase of the computation time of using betweenness centralities is large. However, the decrease of the average gap of betweenness centrality is very large and the average gap is improved almost all instances. The computation time of SPH and DNH is still short unless combined with the network centrality. From Fig. 3.54 (c), the distribution of plots are different from Figs. 3.54 (a) and (b). This is because the range of the increase in the computation time of ADH is tighter than that of SPH and DNH.

From these results, we found that using network centralities has a good balance between

the increase of the computation time and the decrease of the average gap. Additionally, using the network centrality is effective for the instances that have a large number of edges and terminals because the computation time of construction methods depends on them.

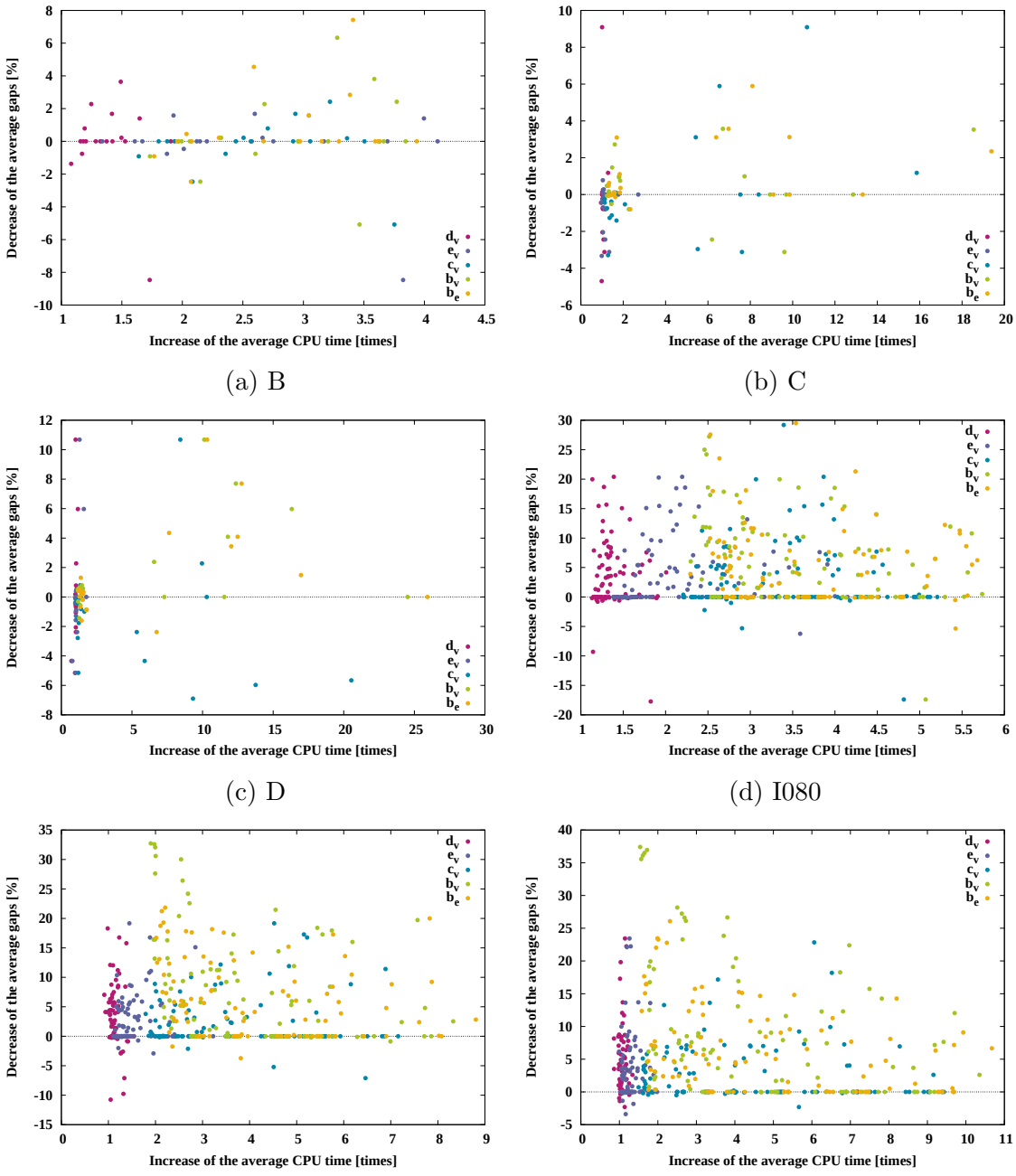


(a) B (b) C

(c) D (d) I080

(e) I160 (f) I320

Figure 3.51: The trade off of the increase of the CPU time and the decrease of the average gaps (SPH)

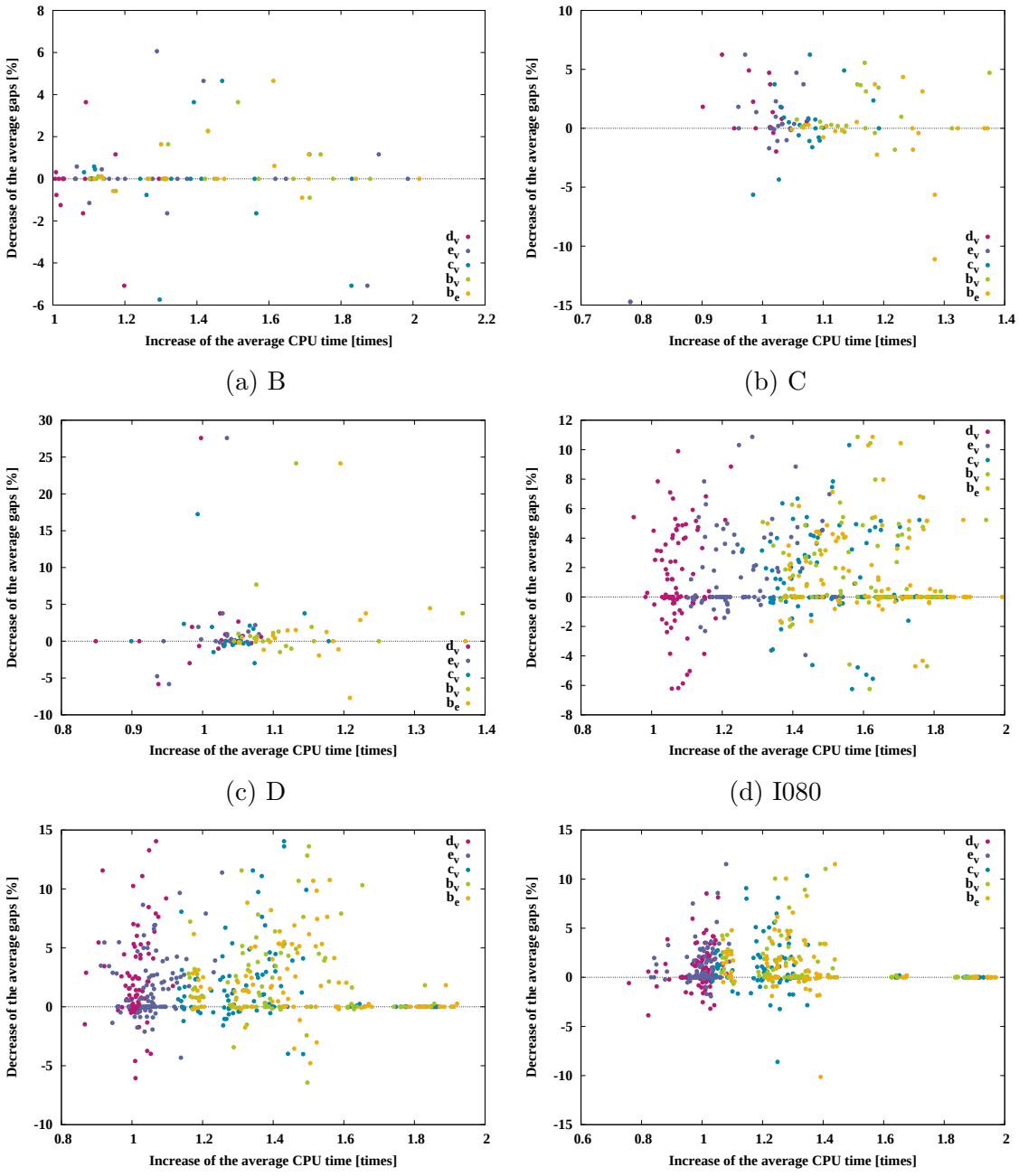


(a) B (b) C

(c) D (d) I080

(e) I160 (f) I320

Figure 3.52: The trade off of the increase of the CPU time and the decrease of the average gaps (DNH)

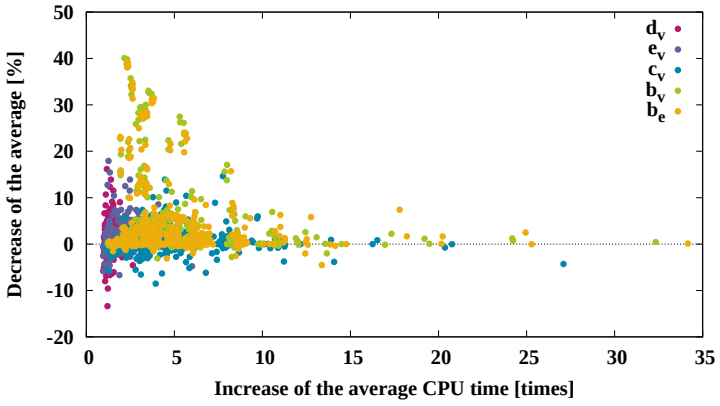


(a) B (b) C

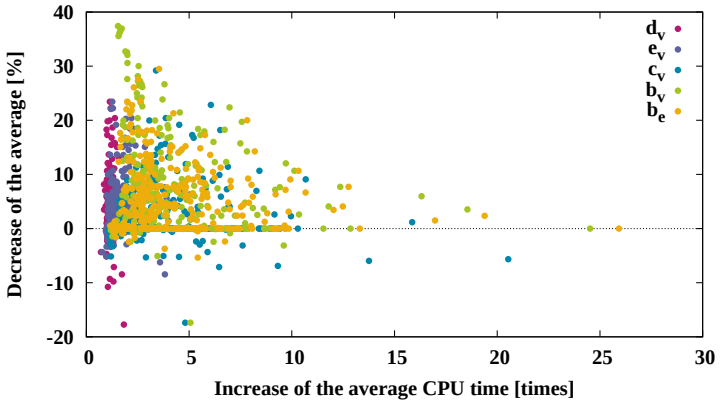
(c) D (d) I080

(e) I160 (f) I320

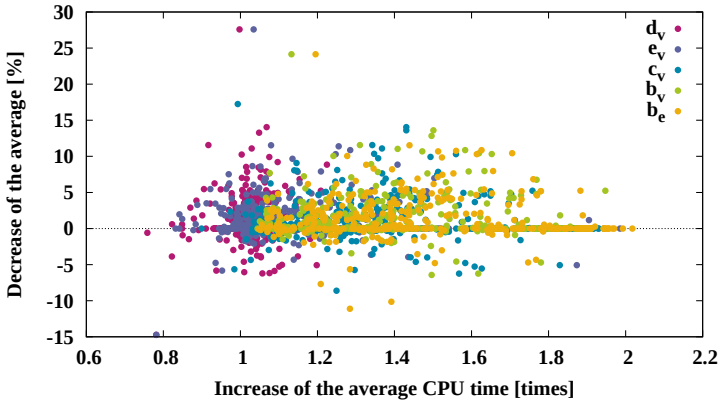
Figure 3.53: The trade off of the increase of the CPU time and the decrease of the average gaps (ADH)



(a) SPH



(b) DNH



(c) ADH

Figure 3.54: The trade off of the increase of the CPU time and the decrease of the average gaps

3.6 Discussion of Chapter 3

We evaluated the performance of three construction methods combined with five network centralities. In detail, we used the shortest path heuristic, distance network heuristic, and average distance heuristic as the construction methods and the degree, eigenvector, closeness, vertex betweenness, and edge betweenness centralities as the network centralities. Thus, we evaluated 18 combinations in total. Also, we used six benchmark problems in SteinLib. The benchmark problem sets B, C, and D are random graphs with randomly assigned edge weights from 1 to 10. In addition, the densities of their edges are sparse at less than 10%. On the other hand, the benchmark problem sets I080, I160, and I320 are random graphs with incidentally distributed edge weights; the weight of an edge between terminals is approximately 300, the weight of an edge between a terminal and a non-terminal is approximately 200, and the weight of an edge between non-terminals is approximately 100. In addition, some instances have a high density of the edges, including the complete graph. For ease of comparison, we summarize the average gaps of 18 combinations in Table 3.43–3.45.

First, we evaluate the performance of the shortest path heuristic combined with network centralities because the shortest path heuristic is one of the most popular construction methods for Steiner trees. In the case of the shortest path heuristic, the average gaps of using network centralities are smaller than that of the original shortest path heuristic for almost all instances. The average gap of all instances of the original shortest path heuristic is 9.70. However, that of using the degree centrality is 9.63, eigenvector centrality is 9.35, closeness centrality is 9.79, vertex betweenness centrality is 6.46, and edge betweenness centrality is 6.54. Thus, we can arrange the network centralities in order of effectiveness as follows: vertex betweenness, edge betweenness, eigenvector, degree, and closeness centrality.

The reason why the betweenness centrality shows good performance is it uses the flow in the graph. The high betweenness vertices and edges frequently appear on all of the shortest paths, and these become relay points of the flow in the graph. The minimum Steiner tree may minimize the total flow between terminals. Thus, using the betweenness centrality shows good performance to solve the Steiner tree problem in graphs.

The results of using the vertex and edge betweenness centralities are different. For example, the vertex betweenness centrality of an edge connecting vertices with small and large vertex betweenness centrality is larger than its edge betweenness centrality. If one vertex of an edge frequently appears on the shortest path but the other vertex does not frequently appear on the shortest path, the edge connecting these vertices does not frequently appear on the shortest path. Thus, the edge betweenness centrality of the edge close to the smaller value of the vertex betweenness centrality of the two vertices to which the edge connects.

The use of the eigenvector and degree centralities also shows good performance. The eigenvector centrality treats vertices adjacent to important vertices as important. Thus, these vertices are also important to find minimum Steiner trees. On the other hand, the degree

Table 3.43: Summary of the average gaps of the SPH combined with network centralities.

name	wc	d_v	e_v	c_v	b_v	b_e
B	1.59	1.40	1.49	1.71	1.05	0.80
C	2.18	3.30	3.21	3.08	2.15	2.46
D	3.16	3.94	3.70	4.44	2.82	2.32
I080	14.58	13.44	12.86	13.61	9.74	10.13
I160	17.44	16.92	16.50	17.08	11.09	11.22
I320	19.27	18.76	18.36	18.85	11.93	12.33
ave	9.70	9.63	9.35	9.79	6.46	6.54

Table 3.44: Summary of the average gaps of the DNH combined with network centralities.

name	wc	d_v	e_v	c_v	b_v	b_e
B	2.15	2.18	2.42	2.37	1.74	1.39
C	5.18	5.38	5.84	4.99	4.75	4.03
D	5.86	5.72	5.86	7.13	4.31	4.38
I080	18.58	15.45	15.02	15.64	12.60	13.52
I160	22.32	19.96	20.01	20.13	14.60	16.48
I320	25.25	22.05	22.40	22.46	16.58	18.91
ave	13.22	11.79	11.93	12.12	9.10	9.79

Table 3.45: Summary of the average gaps of the ADH combined with network centralities.

name	wc	d_v	e_v	c_v	b_v	b_e
B	1.11	1.31	0.83	1.25	0.70	0.61
C	3.96	3.39	3.76	3.60	2.70	4.47
D	4.70	3.37	3.27	3.54	2.72	3.24
I080	2.87	1.76	1.50	1.65	1.44	1.25
I160	3.81	1.86	2.18	2.06	1.88	1.79
I320	3.02	2.30	1.93	2.14	1.84	1.91
ave	3.25	2.33	2.25	2.37	1.88	2.21

centrality treats vertices directly connecting many vertices as important. These vertices may play an important role in making minimum Steiner trees.

However, the use of the closeness centrality increases the average gap from the original shortest path heuristic. The closeness centrality uses only the shortest distance information from a vertex to all other vertices. In a random graph, the sum of the shortest distances from a given vertex to all other vertices is similar for any given vertex. Thus, the closeness centrality is not effective to distinguish important vertices to find minimum Steiner trees.

We evaluated the performance of the proposed method by using other construction methods: the distance network heuristic and average distance heuristic. For the distance network heuristic, the average gaps of using network centralities are smaller than that of the original distance network heuristic for almost all instances. The average gap of all instances of the original distance network heuristic is 13.22. However, that of degree centrality is 11.79, eigenvector centrality is 11.93, closeness centrality is 12.12, vertex betweenness centrality is 9.10, and edge betweenness centrality is 9.79. Thus, we can arrange the network centralities in order of effectiveness as follows: vertex betweenness, edge betweenness, degree, eigenvector, and closeness centrality. Comparing these results with those of the shortest path heuristic, the general trend is the same (only the order of the eigenvector and degree centralities switches).

In the case of using the average distance heuristic, the average gaps of using network centralities are smaller than the one of the original average distance heuristic for almost all instances. The average gap of all instances of the original distance network heuristic is 3.25. However, that of the degree centrality is 2.33, eigenvector centrality is 2.25, closeness centrality is 2.37, vertex betweenness centrality is 1.88, and edge betweenness centrality is 2.21. The average gaps are much smaller than those of the shortest path and distance network heuristics because the computation cost of the average distance heuristic is larger approximately twice as a large. Thus, we can arrange the network centralities in order of effectiveness as follows: vertex betweenness, edge betweenness, eigenvector, degree, and closeness centrality. This trend is the same as that for the shortest path heuristic.

To evaluate the performance of approximation methods, computation time is also important. Thus, we evaluated the balance between the increase in CPU time and the decrease in average gap in Fig. 3.54.

The trends of the shortest path heuristic and distance network heuristic are very similar. These methods decrease the average gap by approximately 40% within twice the CPU time of the original construction methods for some instances: i320-341, i320-342, i320-343, i320-344, and i320-345. These instances have a common feature in that density of terminals is high (25%). In contrast, these methods failed to find smaller weight Steiner trees than the original construction method for some instances: d01, d06, d11, and d16. In fact, the densities of terminals of these instances are very low (0.5%). From these results, the construction method combined with network centrality is efficient when the density of terminals is high.

On the other hand, for the average distance heuristic, the average gap decreases by approximately 30% for the instance d11. This result implies that using the average distance heuristic is effective for instances where the shortest path and distance network heuristics show poor performance.

In addition, using the average distance heuristic reduces the CPU time for some instances. An example of this phenomenon is shown in Fig. 3.55. The vertical axis indicates the CPU time of the average distance heuristic (not including the CPU time for calculating network centralities) and the horizontal axis indicates the frequency of 50 trials. From Fig. 3.55, the CPU time of the average distance heuristic combined with network centralities is shorter than that of the original average distance heuristic. The mechanism of this phenomenon is still unclear, but using network centralities may realize an efficient construction process for Steiner trees.

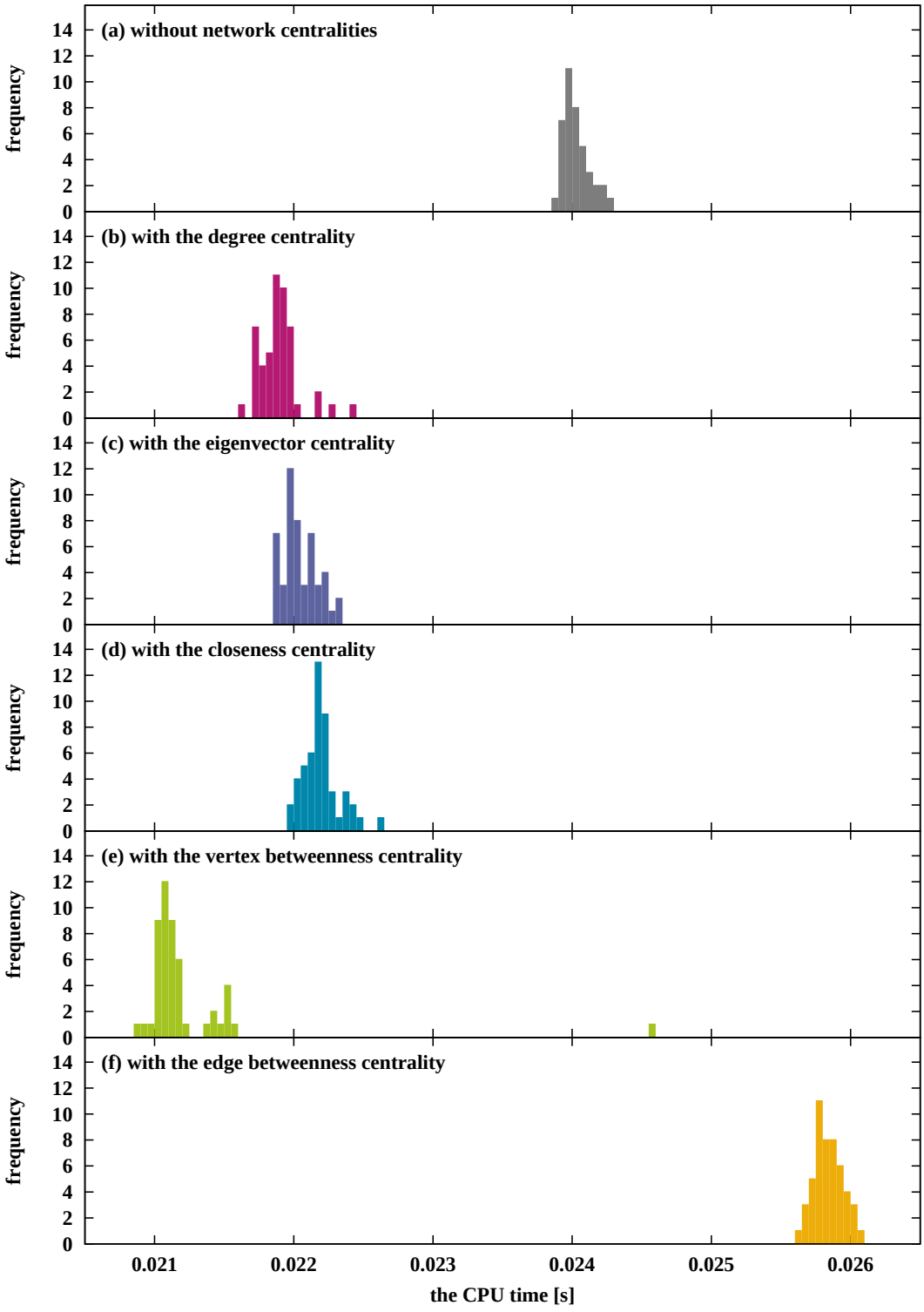


Figure 3.55: Difference in CPU time distribution with the ADH (i160-044).

3.7 Summary of Chapter 3

The approximation method is one of the practical approaches to solve $\mathcal{NP}$-hard combinatorial optimization problems. There are two approaches to the approximation method: construction method and improvement method. The construction method priorities finding a feasible solution as soon as possible. On the other hand, the improvement method priorities finding the best solution possible. Both methods are closely related, so the development of both methods is desired. From this viewpoint, in this chapter, we focus on the construction method to solve the Steiner tree problem in graphs.

Many construction methods have been proposed to solve the Steiner tree problem in graphs; however, the shortest path heuristic, distance network heuristic, and average distance heuristic are the most popular. These construction methods use the shortest paths between arbitrary vertices and a minimum spanning tree to find a Steiner tree because these are special cases of the Steiner tree problem and easy to solve. If the number of terminals is two, the Steiner tree problem in graphs becomes the shortest path problem, and it can be solved optimally in reasonable time using the Dijkstra algorithm. Similarly, if all vertices are terminals, the Steiner tree problem in graphs becomes the minimum spanning tree problem, and it can be solved optimally in reasonable time using the Kruskal algorithm. If there are many shortest paths between arbitrary vertices and many minimum spanning trees exist in a graph, these methods can be selected arbitrarily. This arbitrariness sometimes leads to large weight Steiner trees. If the selected shortest paths share many edges, the total weight of the Steiner tree becomes smaller than the sum of the shortest distances, and we can obtain small-weight Steiner trees. However, if the selected shortest paths do not share edges, the total weight of the Steiner tree is not reduced and we cannot obtain small-weight Steiner trees.

To overcome this issue, we propose using the network centrality to distinguish the vertices and edges that can share the edges included in the shortest paths selected by the construction methods to solve the Steiner tree problem in graphs. Network centrality is an important factor in complex network theory. Network centrality measures the importance of vertices and edges in a graph. The vertices and edges that are important in the graph can also be expected to be important in the construction of small-weight Steiner trees. Thus, we use network centrality to distinguish which vertices and edges should be included in the Steiner tree.

The construction methods to solve the Steiner tree problem in graphs use the shortest path between arbitrary vertex pairs. To distinguish which shortest path can find small-weight Steiner trees, we propose using not only shortest path information but also network centrality information to select the shortest path. However, enumerating all of the shortest paths between arbitrary vertex pairs and calculating the sum of the network centrality of edges included in each shortest path has high computation cost. To suppress the increase in

computation cost, we propose using a new edge weight: a weighted sum of the original edge weight and network centrality. We expect that calculating the shortest path with this new edge weight has the same effect as enumerating all of the shortest paths between arbitrary vertex pairs and selecting one of them that maximizes the sum of the network centrality of included edges. Note that the objective of the shortest path problem is finding a path that minimizes the sum of the edge weights of included edges; however, the value of network centrality is large where vertices and edges are important. To overcome this contradiction, we use the reciprocal of the network centrality to calculate the new edge weight.

Many network centralities are defined on vertices, not edges. However, we want to use the network centrality of edges. Thus, we treat the average value of the network centralities of both vertices of each edge as the network centrality of edges. Besides, many network centralities have been proposed. We select the most popular network centralities: degree, eigenvector, closeness, and betweenness centralities.

In this chapter, we use three construction methods (shortest path heuristic, distance network heuristic, and average distance heuristic) and four network centralities (degree, eigenvector, closeness, and betweenness centralities). Because the betweenness centrality can be defined for both vertices and edges, we use both centralities and call them vertex betweenness centrality and edge betweenness centrality, respectively. Thus, we propose $3 \times 5 = 15$ combinations. Then, we compare the performance of these methods to that of the original methods. To evaluate the performance of the approximation methods, the gap and computation time are the most important aspects. Thus, we evaluated the average gap and CPU time for solving benchmark instances. We used six benchmark problems: B, C, D, I080, I160, and I320 in SteinLib, which is one of the most popular benchmark libraries for the Steiner tree problem in graphs. Each benchmark problem consists of some instances; we used 358 instances in total. The instances in the benchmark problem sets B, C, and D are sparse random graphs with randomly assigned edge weights from 1 to 10. The instances in the benchmark problem sets I080, I160, and I320 are random graphs with incidentally assigned edge weights; the weight of an edge between terminals is approximately 300, the weight of an edge between a terminal and a non-terminal is approximately 200, and the weight of an edge between non-terminals is approximately 100. Our proposed method has a scaling parameter α. We tune this parameter per instance by pre-numerical experiments.

From the results of numerical experiments, it is revealed that using network centralities reduces the average gap for almost all instances. In particular, the vertex and edge betweenness centralities show the best performance for all construction methods. This is because the betweenness centrality is calculated based on the flow in the graph. The minimum Steiner tree is considered as a tree that connects all terminals with the minimum flow. Thus, the inclusion of vertices and edges that become relay points of the flow contributes to making a small-weight Steiner tree. On the other hand, using the closeness centrality is not efficient to find a small-weight Steiner tree. This is because the closeness centrality does not consider

the structure of the graph. In a random graph, the average distance from a given vertex to all other vertices is very similar for any given vertex. Thus, the closeness centrality cannot distinguish vertices and edges that should be included in the Steiner tree.

In our proposed method, the computation time becomes longer than that of the original methods because it must calculate of the network centrality of all vertices and edges. From this viewpoint, we also evaluate the increase in CPU time from the original methods. The results of numerical experiments indicate that using network centralities reduces the average gap by at most 40% within twice the CPU time of the original methods for some instances. In particular, using network centralities is effective for instances with a high density of terminals.

For the average distance heuristic, the computation time of using network centralities is shorter than the computation time not using network centralities for some instances. The mechanism of this is still unclear; however, using network centrality may contribute to efficient selection of the shortest path among construction processes.

From these results, we found the followings:

- Using betweenness centrality combined with any construction method is the most effective way to find small-weight Steiner trees.
- The CPU time of our proposed method is longer than that of the original method because the proposed method must calculate the network centrality; however, the CPU time of our proposed method is still reasonable.

Thus, network centrality can be applied to other types of Steiner tree problems, such as the constrained Steiner tree problem and prize-collecting Steiner tree problem.

Chapter 4

Improvement method using chaotic neurodynamics

4.1 Background of Chapter 4

Improvement methods are applied to a solution obtained by a construction method. Improvement methods can be divided into two types: local search and meta-heuristic. The local search method tries to find neighborhood solutions by applying a small change to an original solution. Thus, we can improve solutions by repeating moving to the best solution from neighborhoods. However, this method is trapped at a local minimum solution and fails to find good solutions. To overcome this demerit, meta-heuristics control local searches by escaping from local minima.

There are two popular local search methods to solve the Steiner tree problem in graphs: Steiner vertex search and key path search. Steiner vertex search makes neighborhood solutions based on adding or removing non-terminal vertices. Thus, we can make at most $|V \setminus T|$ Steiner trees. However, this number includes not only feasible solutions but also infeasible solutions. Thus, the number of feasible solutions may smaller than $|V \setminus T|$ On the other hand, key path search finds neighborhood solutions based on dividing trees and reconnecting trees.

To control these local searches, many meta-heuristics have already been used, such as the genetic algorithm, tabu search, and greedy randomized adaptive search procedure. In 1997, a meta-heuristic that uses chaotic neurodynamics to solve the traveling salesman problem was proposed by Hasegawa, Ikeguchi, and Aihara [12]. This method was called the *chaotic search*, and it shows good performance. After that, the chaotic search was applied to solve many combinatorial optimization problems, such as the traveling salesman problem [12, 13, 14, 15], quadratic assignment problem [16, 17, 18], vehicle routing problem [19, 20], packet routing problem [21, 22, 23, 24, 25, 26, 27, 28], and motif extraction problem [29, 30].

From this viewpoint, we investigate the performance of the chaotic search to solve the

Steiner tree problem in graphs. In addition, the chaotic search can convert to the tabu search by changing its parameters. Thus, we compared the performance of the chaotic search with the tabu search.

4.2 Steiner vertex neighborhood

We present an algorithm for finding neighborhood solutions of the Steiner tree problem in graphs or neighborhood Steiner trees. This algorithm is called the *Steiner vertex neighborhood*. A Steiner vertex is a non-terminal vertex included in a Steiner tree. A solution of the Steiner tree problem in graphs or a Steiner tree is represented by a set of vertices. This means that if we have an optimal vertex set, we can generate an optimal solution or optimal Steiner tree by its minimum spanning tree on a subgraph induced in an input graph G by the optimal vertex set.

From this viewpoint, if V_S is a set of vertices of a solution or a Steiner tree S, we can easily define a neighborhood solution S' by adding or removing an arbitrary non-terminal vertex v from V_S, such as

$$V_{S'} = \{V_S \setminus \{v\} \mid v \in V_S\} \cup \{V_S \cap \{v\} \mid v \notin V_S\}, \tag{4.1}$$

where $V \setminus T$ is a set of non-terminals.

Selecting an arbitrary non-terminal vertex for addition or removal from V_S takes $\mathcal{O}(1)$ time. Then, making a subgraph and its minimum spanning tree can be processed simultaneously by using the Kruskal algorithm, i.e., sorting all edges E in ascending order at first and then making a minimum spanning tree with edges in an induced subgraph. The computation cost of sorting E is $\mathcal{O}(|E| \log |E|)$ by using the quick sort. The computation cost of making a minimum spanning tree is $\mathcal{O}(|E|)$ by using the Kruskal algorithm. The computation cost of evaluating whether an edge is included in a induced subgraph is $\mathcal{O}(1)$. Therefore, the total computation cost of finding a neighborhood solution is $\mathcal{O}(|E| \log |E|)$.

The pseudo code of the Steiner vertex neighborhood is shown in Fig. 4.1. In Fig. 4.1, a function SortInAscendingOrder sorts the array in ascending order, a function InducedSubgraph returns the induced subgraph of the input graph and its subset of vertices, a function Group returns the group index of the vertex, and a function MergeGroup merges groups of these vertices. Edges in E are sorted in ascending order in line 6. A vertex set of the neighborhood solution S', $V_{S'}$, is made in line 7. An induced subgraph of the input graph G of a vertex set $V_{S'}$, G', is made in line 8. The loop from lines 9 to 16 is repeated $|E|$ times. If an edge (i, j) is included in an edge set of the induced subgraph E', we try to insert this edge to the neighborhood solution. If inserting this edge does not make any loops in the neighborhood solution or both side vertices of this edge belong to different groups, we insert this edge to the neighborhood solution in line 12. Then, we merge groups of both side vertices of the inserted edge in line 13. This procedure can be implemented easily by using the union-find data structure. Finally, the Steiner vertex neighborhood returns S' in line 17.

Unfortunately, S' sometimes does not satisfy the requirements of a Steiner tree. For example, removing a Steiner vertex results in a disconnected induced subgraph, and an added Steiner vertex forms a single element disconnected component. In such cases, we treat these solutions as *infeasible* and ignore them.

```
   /* Definition */
 1 v ∈ V \ T: the arbitrary selected non-terminal;
 2 w: the weight function of edges;
 3 G = (V, E): the input graph;
 4 S = (V_S, E_S): the current solution;
 5 S' = (V_S', E_S'): the neighborhood solution;
   /* Make a neighborhood solution */
 6 SortInAscendingOrder(E, w);
 7 V_S' ← {V_S \ {v} | v ∈ V_S} ∪ {V_S ∪ {v} | v ∉ V_S};
 8 G' = (V', E') ← InducedSubgraph(G, w, V_S');
 9 for each edge (i, j) ∈ E in ascending order of w do
10     if (i, j) ∈ E' then
11         if Group(i) ≠ Group(j) then
12             E_S' ← E_S' ∪ {(i, j)};
13             MergeGroups(i, j);
14         end
15     end
16 end
17 return S'
```

Figure 4.1: Pseudo code of the Steiner vertex neighborhood.

Examples of the Steiner vertex neighborhood are shown in Fig. 4.2. In Fig. 4.2, circles indicate vertices and lines indicate edges; red circles indicate terminals, bold lines indicate edges included in a Steiner tree, and dotted lines indicate edges not included in an induced subgraph. Let Fig. 4.2 (a) be the current Steiner tree. Its objective function value is nine. If the Steiner vertex neighborhood is applied to a pink colored vertex in Fig. 4.2 (a), we can obtain the Steiner tree in the Fig. 4.2 (b). Its objective function value is ten because the weight of the newly inserted edge is unity. Then, if the Steiner vertex neighborhood is applied to a blue colored vertex in Fig. 4.2 (b), we can obtain the Steiner tree in Fig. 4.2 (c). Its objective function value is improved to eight because the weight of the removed edge is two. However, if the Steiner vertex neighborhood is applied to an orange colored vertex in Fig. 4.2 (c), we cannot find a feasible solution because there are no edges to connect the left top terminal in the induced subgraph.

The Steiner vertex neighborhood can make at most $|V \setminus T|$ neighborhood solutions. If we repeat replacing a neighborhood solution that improves the objective function value, we may reach a better solution than the initial solution. This method is called the *local search method*. However, the local search method almost finds near optimal solutions because it gets trapped at local optima. To escape from such local optima, metaheuristics are usually used.

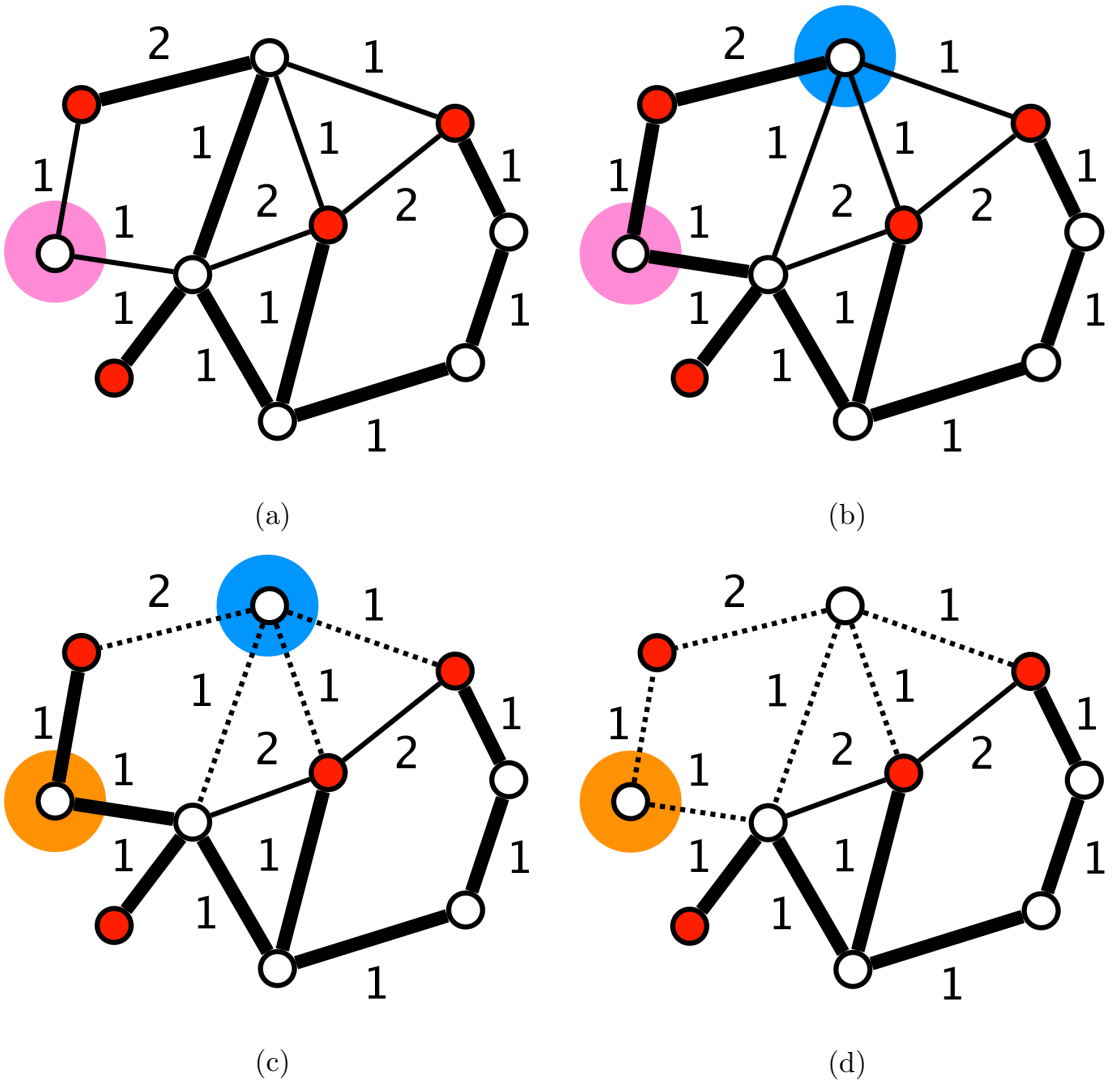


Figure 4.2: Examples of the Steiner vertex neighborhood. Circles indicate vertices and lines indicate edges; red circles indicate terminals, bold lines indicate edges included in a Steiner tree, and dotted lines indicate edges not included in an induced subgraph. Let (a) be the current Steiner tree (objective function value is nine). If the Steiner vertex neighborhood is applied to a pink colored vertex in (a), we can obtain the (b) Steiner tree (objective function value is ten). Then, if the Steiner vertex neighborhood is applied to a blue colored vertex in (b), we can obtain the (c) Steiner tree (objective function value is improved to eight). However, if the Steiner vertex neighborhood is applied to an orange colored vertex in (c), we cannot find a feasible solution because there are no edges to connect the left top terminal in the induced subgraph.

4.3 Improvement method using chaotic neurodynamics

Given a feasible solution or a feasible Steiner tree, we can easily improve it by a local search method. However, a local search method gets trapped at local minima and fails to find global minimum solutions in almost all cases. The chaotic search [12] uses chaotic neurodynamics [11] to escape from such local minima. The most important features of chaotic neurodynamics are as follows: (1) a neuron fires when its internal state exceeds a threshold, (2) the internal states of neurons are updated by an input and its current internal state, and (3) if a neuron fires, this neuron cannot fire for a while.

The chaotic search has been applied to several $\mathcal{NP}$-hard combinatorial optimization problems, such as the traveling salesman problem [12, 13, 14, 15], quadratic assignment problem [16, 17, 18], vehicle routing problem [19, 20], packet routing problem [21, 22, 23, 24, 25, 26, 27, 28], and motif extraction problem [29, 30]. Thus, it is expected that the chaotic search also shows good performance in solving the Steiner tree problem in graphs.

In the chaotic search, firing of a chaotic neuron corresponds to each neighborhood solution and decides whether movement to the corresponding neighborhood solution should be carried out or not. In other words, the movement to the corresponding neighborhood solution is carried out only when the corresponding neuron fires.

In this thesis, we use the Steiner vertex neighborhood to find neighborhood solutions. Thus, we prepare $|V \setminus T|$ neurons. Unfortunately, we cannot directly apply the chaotic search proposed by Hasegawa et al. in 1997 [12] because we use $|V \setminus T|$ neurons, however, they used $|V|^2$ neurons. Hasegawa et al. also proposed the chaotic search that neurons correspond to vertices (not edges) in 2002 [17]. Thus, we introduce this new chaotic search.

A neuron fires when its internal state exceeds a threshold. The internal state of neurons is defined as the sum of these equations:

$$\xi_i(t+1) = \beta \Delta_i(t), \tag{4.2}$$

$$\zeta_i(t+1) = -\alpha \sum_{d=0}^{t-1} k^d x_i(t-d) + \theta, \tag{4.3}$$

$$\eta_i(t+1) = -W \sum_{l=1}^{|V \setminus T|} x_l(t) + W. \tag{4.4}$$

Equation (4.2) express the input to the ith neuron at time t. In Eq. (4.2), β is a scaling parameter and $\Delta_i(t)$ is an amount of improvement in the movement to the neighborhood solution, which is obtained by applying the Steiner vertex neighborhood to the ith non-terminal vertex, namely, $\Delta_i(t) = \{F(t) - F_i(t)\}/n$, where $F(t) = \sum_{e \in E_{S(t)}} w(e)z(e)$ is the weight of the current solution at time t, $F_i(t) = \sum_{e \in E_{S_i(t)}} w(e)z(e)$ is the weight of the neighborhood solution obtained by applying the Steiner vertex neighborhood to the ith non-terminal vertex at time t, $E_{S(t)}$ is the set of edges included in the current solution at time

t, $E_{S_i(t)}$ is the set of edges included in the ith neighborhood solution at time t, and n is a normalization parameter. In other words, Eq. (4.2) works to reflect the improvement of the objective function value in the internal state of neurons. If $\Delta_i(t)$ is positive, the internal state of the neuron becomes positive and large. By using Eq. (4.2), a simple local search is realized.

Equation (4.3) express the refractory effect. It prevents the firing of neurons that fired in recent iterations. In other words, the refractory effect contributes to escape from local minima. In Eq. (4.3), α ($\alpha > 0$) is a scaling parameter, k ($0 < k < 1$) is a decay factor, $x_i(t)$ is the output of the ith neuron at time t (described later), and θ ($\theta > 0$) is a bias.

Equation (4.4) expresses the input from other neurons. In Eq. (4.4), W is the connection weight between neurons. This equation regulates the firing rates. If many neurons fire in the previous iteration, $\eta_i(t+1)$ in Eq. (4.4) becomes negative, which prevents the firing of neurons. On the other hand, if no neurons fired in the previous iteration, $\eta_i(t+1)$ in Eq. (4.4) becomes positive, which increases the firing rates.

Then, the output of the ith chaotic neuron is defined by Eq. (4.5).

$$x_i(t+1) = f\{\xi_i(t+1) + \zeta_i(t+1) + \eta_i(t+1)\}, \tag{4.5}$$

where $f(x) = 1/(1+\mathrm{e}^{-x/\epsilon})$ is a sigmoidal function and ϵ is a steepness parameter. Thus, the range of outputs of neurons is $[0, 1]$. From this viewpoint, if $x_i(t+1) \geq 0.5$, the ith chaotic neuron fires at time $t+1$ and the current solution is replaced with the ith neighborhood solution. Therefore, x_i keeps the history of firing of the ith neuron. Here, firing means the addition or removal of the ith vertex from the solution or the Steiner tree.

In the chaotic search, the internal states of neurons are asynchronously updated in random order. Thus, t in Eqs. (4.2)–(4.5) expresses the local time of the ith neuron. Unfortunately, the Steiner vertex neighborhood finds not only feasible solutions but also infeasible solutions. If a neuron corresponds to an infeasible neighborhood solution, the internal state of the neuron is not updated and the current internal state is left. Thus, neurons corresponding to infeasible neighborhood solutions cannot fire. On the other hand, if the corresponding neighborhood solution is feasible, the internal state of the neuron is updated by Eqs. (4.2)–(4.4). If the output (calculated by Eq. (4.5)) is larger than 0.5, a neuron fires and the current solution is replaced by its corresponding neighborhood solution immediately. Namely, the current solution could be updated more than once before all neurons are updated once.

The computation cost is one of the important aspects in evaluating the performances of algorithms. If we simply implement the calculation of the refractory effect $\zeta_i(t+1)$, it takes $\mathcal{O}(t-1)$ where t is the current time, because it is calculated by the summation of the history

of outputs of the neuron. However, it can be transformed as follows:

$$\zeta_i(t+1) = -\alpha \sum_{d=0}^{t-1} k^d x_i(t-d) + \theta, \tag{4.6}$$

$$= -\alpha\{k^0 x_i(t) + k^1 x_i(t-1) + k^2 x_i(t-2) + \cdots + k^{t-1} x_i(1)\} + \theta, \tag{4.7}$$

$$= -\alpha x_i(t) + k\{-\alpha x_i(t-1) - \alpha k x_i(t-2) - \cdots - \alpha k^{t-2} x_i(1)\} + \theta. \tag{4.8}$$

If we consider $\zeta_i(t)$ (at previous time),

$$\zeta_i(t) = -\alpha \sum_{d=0}^{t-2} k^d x_i(t-1-d) + \theta, \tag{4.9}$$

$$= -\alpha\{k^0 x_i(t-1) + k^1 x_i(t-2) + k^2 x_i(t-3) + \cdots + k^{t-1} x_i(1)\} + \theta, \tag{4.10}$$

$$= -\alpha x_i(t-1) - \alpha k x_i(t-2) - \alpha k^2 x_i(t-3) - \cdots - \alpha k^{t-2} x_i(1) + \theta. \tag{4.11}$$

Thus, substituting Eq. (11) into Eq. (8),

$$\zeta_i(t+1) = -\alpha x_i(t) + k\{\zeta_i(t) - \theta\} + \theta, \tag{4.12}$$

$$= -\alpha x_i(t) + k\zeta_i(t) + \theta(1-k). \tag{4.13}$$

Therefore, we can calculate $\zeta_i(t+1)$ only from five scalars and do not need to memorize the history of the neuron outputs. From Eq. (13), the computation cost of the refractory effect $\zeta_i(t+1)$ can be reduced to $\mathcal{O}(1)$.

In addition, if we simply implement the calculation of the mutual connection $\eta_i(t+1)$, it takes $\mathcal{O}(|V \setminus T|)$ because it is calculated by the summation of outputs of neurons $x_j(t)$ ($j =$ 1, 2, ..., $|V \setminus T|$). However, this summation of outputs of all neurons can be updated by subtracting $x_j(t)$ from and adding $x_j(t+1)$ to the current summation of outputs of neurons because asynchronous updating is used in the chaotic search. The computation cost of this process is $\mathcal{O}(1)$; thus, the computation cost of $\eta_i(t+1)$ is $\mathcal{O}(1)$. By using these techniques, the bottleneck of the computation cost of updating a neuron is making a neighborhood solution, which takes $\mathcal{O}(|E| \log |E|)$.

The pseudo code of the chaotic search is shown in Fig. 4.3. In Fig. 4.3, a function RandomNumber returns a random number between the input values by using the mersenne twister, a function ShortestPathHeuristic returns a feasible solution by using the shortest path heuristic [35], a function SteinerVertesNeighoborhood returns a feasible or infeasible solution by using the Steiner vertex neighborhood, a function ObjectiveFunction returns an objective function value of the input solution, and a function SigmoidalFunction returns the output of the sigmoidal function. The iteration counter t, the sum of the output of all neurons r, the initial value of the refractory effect of each neuron and the initial value of the output of each neuron are initialized in lines 1 to 4. An initial solution S is found in line 5 by using the shortest path heuristic [35]. Then, the initial solution is substituted with the

best solution S^* in line 6. The loop from lines 7 to 26 is repeated for $t_{\max}$ times. In each iteration, all neurons are updated in random order. A neighborhood solution S' is made in line 9 by using the Steiner vertex neighborhood at the vertex v. If the obtained neighborhood solution is feasible, the internal state of the neuron v is updated in lines 11 to 16. To reduce the computation cost, the sum of the output of all neurons r is updated by subtracting the output of the neuron v from r before updating it (in line 14) and adding the output of the neuron v to r after updating it (in line 16). If the neighborhood solution is the best solution ever found, the best solution and current solution are replaced by the neighborhood solution in lines 18 and 19. Otherwise, if the internal state of the updated neuron v exceeds the threshold, the current solution is replaced with the neighborhood solution in line 21. After updates have been attempted on all neurons, the iteration counter t is incremented in line 25.

The refractory effect of the chaotic search is similar to the tabu effect of the tabu search. Thus, we can easily transform the chaotic search into the tabu search by changing these points:

- In Eq. (4.2), set $\beta = 1$.
- In Eq. (4.3), set $\alpha \to \infty$, $k = 1$, $\theta = 0$, and replace $t - 1$ with a tabu tenure s.
- In Eq. (4.5), ignore η_i.
- Neurons are updated in arbitrary order synchronously.
- The output of only the neuron with the largest internal state is unity and the others are zero.

From this viewpoint, we compared the performance of the chaotic search and tabu search transformed by the chaotic search.

```
   /* Definition */
 1 t: the iteration count;
 2 r: the sum of the output of all neurons;
 3 S: the current solution;
 4 S*: the best solution;
 5 S': the neighborhood solution;
 6 ξ: the vector of the gain effect of all neurons;
 7 ζ: the vector of the refractory effect of all neurons;
 8 η: the vector of the mutual connection of all neurons;
 9 x: the vector of the output of all neurons;
   /* Initialization */
10 ζ[1, ..., |V \ T|] ← 0;
11 x[1, ..., |V \ T|] ← RandomNumber(0, 1);
12 r ← Σ_{j=1}^{|V\T|} x[j];
13 t ← 1;
   /* Find an initial solution */
14 S ← ShortestPathHeuristic(V, E, w);
15 S* ← S;
16 while t ≤ limit do
17     for each v ∈ V \ T do
           /* Find the neighborhood solution v */
18         S' ← SteinerVertexNeighborhood(V_S, {v}, E, w);
19         if S' is a feasible solution then
               /* Update the internal state of the neuron v */
20             ξ[v] ← β {ObjectiveFunction(S) − ObjectiveFunction(S')};
21             ζ[v] ← −αx[v] + kζ[v]θ(1 − k);
22             η[v] ← −wr + w;
23             r ← r − x[v];
24             x[v] ← SigmoidalFunction(ξ[v] + ζ[v] + η[v]);
25             r ← r + x[v];
               /* Update the best and current solutions */
26             if ObjectiveFunction(S') < ObjectiveFunction(S*) then
27                 S* ← S';
28                 S ← S';
29             else if x[v] > threshold then
30                 S ← S';
31             end
32         end
33     end
34     t ← t + 1;
35 end
```

Figure 4.3: Pseudo code of the chaotic search to solve the Steiner tree problem in graphs.

4.4 Performance evaluation

4.4.1 Experimental conditions

We compared the performance of the chaotic search to that of the tabu search. We used the benchmark problem sets B, C, I080, and I160 in SteinLib [61]. We construct an initial solution by using the shortest path heuristic [35], which constructs a wide variety of Steiner trees by changing the root terminal. On the other hand, we find neighborhood solutions by using the Steiner vertex neighborhood, which finds neighborhood solutions by addition or removal of a non-terminal vertex from the current solution.

The most biggest difference between the chaotic search and tabu search is how to update a current solution or how to decide which neurons are fired. In the chaotic search, a neuron fires when its internal state exceeds a threshold. Thus, many neurons could fire during an iteration. Here, we define an iteration as updating all neurons once. In contrast to the chaotic search, in the tabu search, only the neuron that has the largest internal state fires during an iteration. If a neuron fires, the current solution is replaced with the corresponding neighborhood solution.

To avoid overlook of the best solution among the search, we introduce an aspiration criterion [12]. If a neighborhood solution has the smallest objective function value during the search, the current solution is forced to move to this neighborhood solution even if the corresponding neuron does not fire.

These meta-heuristics require a terminate condition. Thus, the upper limit of the iterations is set to $5,000$. This terminate condition is fair in terms of the number of neighborhood solutions produced. Under this terminate condition, both the chaotic search and the tabu search try to find neighborhood solutions $5,000|V \setminus T|$ times. The computation cost of finding a neighborhood solution is $\mathcal{O}(|E| \log |E|)$. This is larger than the computation cost of moving a solution $\mathcal{O}(|E|)$. Thus, the computation cost of finding neighborhood solutions occupies the computation cost of an iteration. From this viewpoint, we set the same upper limit of iterations for the chaotic search and tabu search. However, in the chaotic search, many moving solutions could occur during an iteration. Thus, the computation time of the chaotic search may be longer than that of the tabu search.

4.4.2 Parameter tuning

Meta-heuristics have many parameters that must be tuned carefully. Parameters control the balance between the intensification and diversification of the search. Intensification means searching around good solutions found in previous iterations. It is thought that good solutions share some parts. Thus, searching around good solutions contributes to finding better solutions. However, it is not guaranteed that the best solution ever found is a globally good solution. Therefore, we should search for a variety of solutions. From this viewpoint,

diversification is also introduced.

In the case of the tabu search, there is a parameter called tabu tenure s. If s is too short, intensification is too strong and cannot escape from local minima. On the other hand, if s is too long, diversification is too strong, which prevents the intensification of the search. Thus, a tabu tenure must be tuned to an appropriate value.

In the case of the chaotic search, there are six parameters: a scaling parameter of the gain effect β, a scaling parameter of the refractory effect α, a decay factor of the refractory effect k, a bias θ, a connection weight W, and a steepness parameter of the sigmoidal function ϵ. These parameters control the balance between the gain effect, refractory effect, and mutual connection. If the gain effect is too large, the chaotic search becomes a local search and fails to escape from local minima. If the refractory effect is too large, moving to good solutions is prevented. If the mutual connection is too large, firing of neurons becomes synchronized. Thus, parameters must be tuned to appropriate value.

We set parameters of both meta-heuristics by parameter tuning in pre-numerical experiments, and the results are shown in Table 4.1.

Table 4.1: Selected parameters for the tabu search and chaotic search

	tabu search	chaotic search					
name	s	β	α	k	θ	W	ϵ
B	14	1.5	1	0.95	1	0.0	0.001
C	10	1.5	1	0.92	0.5	0.01	0.001
I080	12	0.8	1	0.9	1	0.005	0.001
I160	10	0.89	1	0.9	1	0.005	0.001

The tabu search has one integer parameter, tabu tenure s. Thus, we can tune the performance of the tabu search by changing s and selecting the best s. On the other hand, the chaotic search has six parameters, β, α, k, θ, W, and ϵ. Among these parameters, β and α are most important because they decide the strength of the gain effect and refractory effect, respectively. In this thesis, we fixed $\alpha = 1$ and tuned β. At first, we set $\beta = 1$, $\alpha = 1$, $k = 0.95$, $\theta = 1$, $W = 0$, and $\epsilon = 0.001$ and observed the firing rates. After we found a parameter set with which at least one neuron fires throughout $5,000$ iterations for all instances of a benchmark problem set, we observed the time series of the gap and the raster plot. If the gap becomes large during the search for an instance, the firing rate is too high for that instance. In this case, we increase β, k, and W or decrease θ and ϵ to decrease the firing rate. On the other hand, if the firing rate is low and variety of the firing pattern of neurons is small, we decrease β, k, and W or increase θ and ϵ to increase the firing rate.

Figure 4.4 shows examples of the time series of the gap and raster plot when θ is changed. The raster plot is used to visualize when and which neurons were fired. The horizontal axis

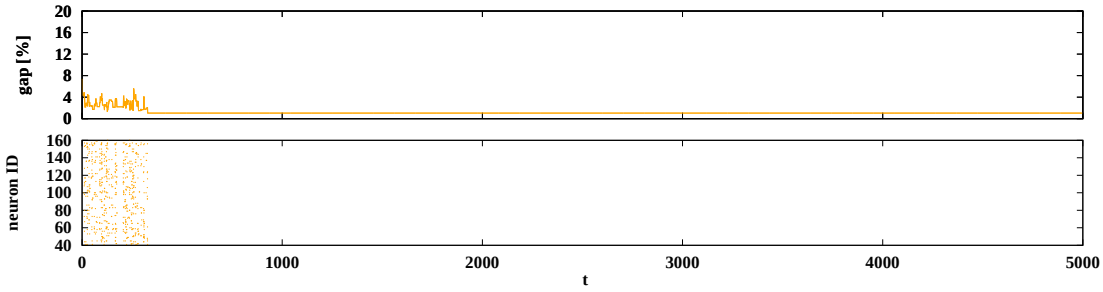
indicates the time, and the vertical axis indicates the neuron index. Each plot represents a neuron firing at that time. In Fig. 4.4 (a), the gap does not change from $t = 20$ to $5,000$ because the number of firing neurons is small. In this case, we changed parameters to increase the firing rate. In contrast, in Fig. 4.4 (b), the gap does not become small because of high firing rate. In this case, we changed parameters to decrease the firing rate. Figure 4.4 (c) shows the time series of the gap and raster plot when the firing rate is set to a suitable value. In this case, the chaotic search finds an optimal solution. These results show that although the performance of the chaotic search depends on the parameter set, it can be controlled by changing parameter sets while observing firing rate and firing pattern.

To solve the Steiner tree problem in graphs, we introduce a normalization parameter n. This is also an important parameter to keep the balance between intensification and diversification of the search. We use the maximum edge weight as n for the benchmark problem sets B and C because the maximum value of the gain by applying the Steiner vertex neighborhood equals the maximum edge weight. We use the median of edge weights as n instead of the maximum edge cost for the benchmark problem sets I080 and I160 because the value of the maximum edge weight differs between whether the edge between terminals exists or not. In Table 4.1, parameter values are shown with the above-mentioned parameter tuning method.

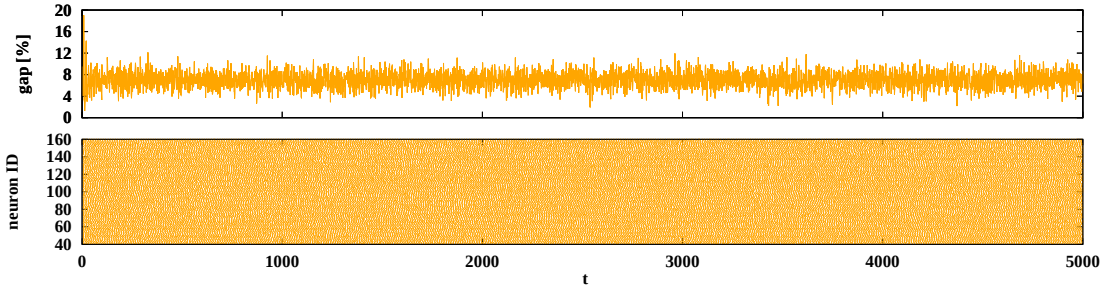
4.4.3 Experimental results

The results of the numerical experiments are shown in Tables 4.2–4.5. In Tables 4.2–4.5, $|V|$ is the total number of vertices, $|E|$ is the total number of edges, $|T|$ is the total number of terminals of instances, gap is the gap of the objective function value of the obtained solution from the optimum solution, # of opt is the number of optimum solutions found during the search, time is the CPU time to terminate the search, and firing rate is the average firing rate.

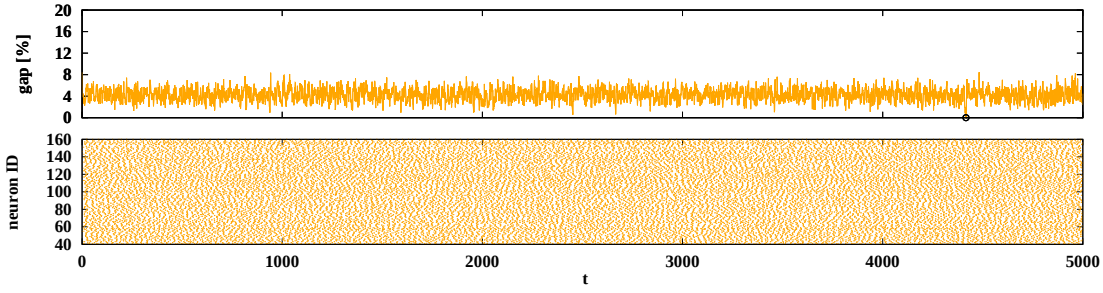
Gap is defined as the difference between the objective function value of the obtained solution and the optimal solution normalized by the optimal solution [%], namely $\{F(S) - F(S^*)\}/F(S^*) \cdot 100[\%]$, where $F(\cdot)$ is the objective function value of the solution or the Steiner tree $\cdot$, S is the obtained solution, and S^* is the optimum solution. Thus, small gaps indicate good performance. # of opt is the number of optimal solutions found during the search. If a meta-heuristic offers good solutions frequently during the search, it could be a practical indicator for generally unknown optimal solutions. We increment it when the objective function value of the current solution equals the objective function value of the optimum solution at the end of an iteration. Thus, the maximum value of # of opt is the same as the upper limit of the iterations, $5,000$. Time is the CPU time from inputting an instance to outputting the best solution. Both the chaotic search and tabu search were implemented with the programming language C using the gcc compiler (ver. 4.2.1). All



(a) $\theta = 0.7$ and average firing rate is 0.11%



(b) $\theta = 2$ and average firing rate is 17.6%



(c) $\theta = 1$ and average firing rate is 7.4%

Figure 4.4: Search process of the chaotic search when solving i160-345 with different firing rates. In each figure, the upper plot is the time series of the gap and the lower plot is a raster plot. Each raster plot indicates when and which neuron fired. Parameters are set to $\beta = 0.89$, $\alpha = 1$, $k = 0.9$, $W = 0.005$, $\epsilon = 0.001$, and $\theta =$ (a) 0.7, (b) 2, and (c) 1. (a) Searching process of the chaotic search when the firing rate is low. In this case, the search does not proceed. (b) Searching process of the chaotic search when the firing rate is high. In this case, the diversification of the search is strong, and the gap does not reach a small value. (c) Searching process of the chaotic search when the firing rate is a suitable value. In this case, the search finds an optimal solution (shown with a black open circle).

numerical experiments were performed on an iMacPro (2017) with 3.2 GHz Intel Xeon W processor and 64 GB 2666 MHz DDR4 random access memory. We also show the firing rates to compare the computation times.

We tried to obtain solutions 50 times per instance because the chaotic search and tabu search include randomness (random selection of a root vertex and random order of neuron updates). Thus, we compared the average value of the gap of the best solution for these 50 trials.

In Tables 4.2–4.5, bold numerals indicate better results; the smaller the gaps, the larger the number of optimal solutions found and the shorter the computation time. We also investigate the firing rates. The firing rate of the tabu search is $1/|V \setminus T|$ because only one neuron with the largest internal state fires. To compare with the firing rate of the chaotic search, we show the firing rate of the tabu search in Tables 4.2–4.5.

From Table 4.2, the average gaps are 0.000%, which indicates that both meta-heuristics can obtain optimal solutions for all instances of the benchmark problem set B. The average number of optimal solutions found by the tabu search is 16.900 and that of the chaotic search is 272.923. Thus, the average number of optimal solutions found by the chaotic search is larger than that of the tabu search. The average calculation time of the tabu search is 0.627 [s] and that of the chaotic search is 0.619 [s]. Thus, the calculation time of the chaotic search and the tabu search are approximately the same. The average firing rate of the tabu search is 1.444% and that of the chaotic search is 2.066%. Thus, the difference in average firing rate of the tabu search and chaotic search is small. These results indicate that the chaotic search performs better than the tabu search for the benchmark problem set B.

From Table 4.3, the average gap of the tabu search is 0.032% and that of the chaotic search is 0.105%. Thus, the average gap of the chaotic search is larger than that of the tabu search. The tabu search found optimal solutions for 19 instances and the chaotic search found optimal solutions for 14 instances. The average number of optimal solutions for the tabu search is 773.688 and that for the chaotic search is 4.387. Thus, the average number of optimal solutions of the chaotic search is smaller than that of the tabu search. The average calculation time of the tabu search is 46.767 [s] and that of the chaotic search is 80.772 [s]. Thus, the average calculation time of the chaotic search is longer than that of the tabu search. The average firing rate of the tabu search is 0.200% and that of the chaotic search is 1.906%. Thus, the average firing rate of the chaotic search is larger than that of the tabu search. This difference causes the difference in calculation time of the tabu search and chaotic search. These results indicate that the chaotic search shows poor performance compared with the tabu search for the benchmark problem set C.

From Table 4.4, the average gap of the tabu search is 0.032% and that of the chaotic search is 0.000%. Thus, the average gap of the chaotic search is smaller than that of the tabu search. The tabu search found optimal solutions for 99 instances and the chaotic search found optimal solutions for all instances. The average number of optimal solutions of the

tabu search is 17.151 and that of the chaotic search is 37.110. Thus, the average number of optimal solutions of the chaotic search is larger than that of the tabu search. The average calculation time of the tabu search is 1.494 [s] and that of the chaotic search is 1.413 [s]. Thus, the average calculation time of the chaotic search is shorter than that of the tabu search, but the difference is small. The average firing rate of the tabu search is 1.250% and that of the chaotic search is 5.551%. Thus, the average firing rate of the chaotic search is larger than that of the tabu search. However, this difference does not affect the calculation time because $|V|$ is small. These results indicate that the chaotic search performs better than the tabu search for the benchmark problem set I080.

From Table 4.5, the average gap of the tabu search is 0.102% and that of the chaotic search is 0.002%. Thus, the average gap of the chaotic search is smaller than that of the tabu search. The tabu search found optimal solutions for 83 instances and the chaotic search found optimal solutions for 98 instances. The average number of optimal solutions of the tabu search is 14.133 and that of the chaotic search is 40.133. Thus, the average number of optimal solutions of the chaotic search is larger than that of the tabu search. The average calculation time of the tabu search is 7.452 [s] and that of the chaotic search is 7.549 [s]. Thus, the calculation time of the tabu search is shorter than that of the chaotic search, but the difference is small. The average firing rate of the tabu search is 0.625% and that of the chaotic search is 4.074%. Thus, the average firing rate of the chaotic search is larger than that of the tabu search. However, this difference does not affect the calculation time because $|V|$ is small. These results indicate that the chaotic search performs better than the tabu search for the benchmark problem set I160.

Table 4.2: Results of the tabu search and the chaotic search for the benchmark problem set B

				Gap [%]		# of opt		Time [s]		Firing rate [%]	
name	$\|V\|$	$\|E\|$	$\|T\|$	TS	CS	TS	CS	TS	CS	TS	CS
b01	50	63	9	**0.000**	**0.000**	11.240	**281.780**	0.293	**0.250**	2.000	1.435
b02	50	63	13	**0.000**	**0.000**	5.160	**694.720**	0.266	**0.250**	2.000	1.159
b03	50	63	25	**0.000**	**0.000**	2.420	**2457.340**	0.266	**0.260**	2.000	0.974
b04	50	100	9	**0.000**	**0.000**	4.660	**82.100**	0.322	**0.310**	2.000	2.486
b05	50	100	13	**0.000**	**0.000**	27.820	**63.500**	**0.338**	0.341	2.000	2.874
b06	50	100	25	**0.000**	**0.000**	49.380	**396.180**	0.339	**0.330**	2.000	2.396
b07	75	94	13	**0.000**	**0.000**	14.920	**38.580**	0.497	**0.491**	1.333	1.934
b08	75	94	19	**0.000**	**0.000**	8.580	**120.720**	0.527	**0.515**	1.333	1.715
b09	75	94	38	**0.000**	**0.000**	3.380	**387.820**	0.511	**0.501**	1.333	1.498
b10	75	150	13	**0.000**	**0.000**	6.540	**27.940**	**0.633**	0.646	1.333	2.814
b11	75	150	19	**0.000**	**0.000**	8.460	**33.780**	**0.610**	0.611	1.333	2.945
b12	75	150	38	**0.000**	**0.000**	19.840	**47.080**	**0.632**	0.640	1.333	2.175
b13	100	125	17	**0.000**	**0.000**	6.180	**29.360**	0.809	**0.750**	1.000	1.406
b14	100	125	25	**0.000**	**0.000**	5.340	**8.300**	0.923	**0.901**	1.000	1.708
b15	100	125	50	**0.000**	**0.000**	11.280	**220.880**	0.920	**0.890**	1.000	1.215
b16	100	200	17	**0.000**	**0.000**	0.960	**5.560**	1.197	**1.136**	1.000	3.023
b17	100	200	25	**0.000**	**0.000**	**50.520**	2.920	**1.104**	1.201	1.000	3.161
b18	100	200	50	**0.000**	**0.000**	**67.520**	14.060	**1.108**	1.114	1.000	2.270
ave				**0.000**	**0.000**	16.900	**272.923**	0.627	**0.619**	1.444	2.066

Table 4.3: Results of the tabu search and the chaotic search for the benchmark problem set C

				Gap [%]		# of opt		Time [s]		Firing rate [%]	
name	$\|V\|$	$\|E\|$	$\|T\|$	TS	CS	TS	CS	TS	CS	TS	CS
c01	500	625	5	**0.000**	**0.000**	26.180	**30.040**	2.583	**2.497**	0.200	0.169
c02	500	625	10	**0.000**	**0.000**	**42.320**	23.220	3.181	**3.132**	0.200	0.173
c03	500	625	83	**0.000**	**0.000**	**9.600**	0.000	**13.439**	14.054	0.200	0.582
c04	500	625	125	0.649	**0.093**	**0.000**	**0.000**	**17.475**	18.194	0.200	0.393
c05	500	625	250	**0.000**	**0.000**	**34.960**	1.220	**20.156**	20.913	0.200	0.639
c06	500	1000	5	**0.000**	**0.000**	**16.900**	4.420	**3.450**	3.632	0.200	0.139
c07	500	1000	10	**0.000**	**0.000**	**36.520**	16.700	**4.699**	5.090	0.200	0.209
c08	500	1000	83	**0.000**	0.196	**186.920**	0.000	**17.655**	19.530	0.200	0.894
c09	500	1000	125	**0.000**	0.141	**158.900**	0.000	**22.654**	24.251	0.200	1.057
c10	500	1000	250	**0.000**	0.091	**66.420**	0.000	**26.681**	27.229	0.200	1.282
c11	500	2500	5	**0.000**	**0.000**	**4.560**	3.060	**10.741**	33.666	0.200	2.661
c12	500	2500	10	**0.000**	**0.000**	**11.680**	2.240	**14.065**	33.862	0.200	2.635
c13	500	2500	83	**0.000**	**0.000**	**373.660**	0.020	**39.129**	54.460	0.200	2.919
c14	500	2500	125	**0.000**	**0.000**	**1053.820**	0.080	**44.421**	59.569	0.200	2.833
c15	500	2500	250	**0.000**	**0.000**	**4999.780**	4.100	**48.496**	53.245	0.200	2.186
c16	500	12500	5	**0.000**	**0.000**	**6.940**	1.520	**63.335**	237.116	0.200	4.461
c17	500	12500	10	**0.000**	**0.000**	**35.540**	0.160	**71.298**	233.455	0.200	4.443
c18	500	12500	83	**0.000**	0.885	**2983.020**	0.000	**139.045**	274.847	0.200	3.995
c19	500	12500	125	**0.000**	0.685	**449.360**	0.000	**165.387**	279.820	0.200	3.715
c20	500	12500	250	**0.000**	**0.000**	**4976.680**	0.960	**207.441**	216.872	0.200	2.732
ave				**0.032**	0.105	**773.688**	4.387	**46.767**	80.772	0.200	1.906

Table 4.4: Results of the tabu search and the chaotic search for the benchmark problem set I080

				Gap [%]		# of opt		Time [s]		Firing rate [%]	
name	$\|V\|$	$\|E\|$	$\|T\|$	TS	CS	TS	CS	TS	CS	TS	CS
i080-001	80	120	6	**0.000**	**0.000**	3.840	**34.280**	**0.410**	0.442	1.250	3.327
i080-002	80	120	6	**0.000**	**0.000**	2.320	**39.600**	**0.356**	0.406	1.250	3.610
i080-003	80	120	6	**0.000**	**0.000**	10.360	**222.680**	0.406	**0.391**	1.250	3.616
i080-004	80	120	6	**0.000**	**0.000**	0.460	**38.620**	**0.438**	0.447	1.250	3.999
i080-005	80	120	6	**0.000**	**0.000**	17.540	**18.960**	**0.423**	0.452	1.250	3.586
i080-011	80	350	6	**0.000**	**0.000**	3.580	**12.480**	**0.720**	0.764	1.250	6.336
i080-012	80	350	6	**0.000**	**0.000**	0.960	**5.680**	**0.702**	0.759	1.250	5.859
i080-013	80	350	6	**0.000**	**0.000**	1.780	**32.380**	**0.751**	0.758	1.250	5.737
i080-014	80	350	6	**0.000**	**0.000**	1.040	**37.120**	**0.722**	0.755	1.250	6.073
i080-015	80	350	6	**0.000**	**0.000**	2.740	**9.020**	**0.729**	0.747	1.250	6.013
i080-021	80	3160	6	**0.000**	**0.000**	11.920	**25.640**	3.969	**3.463**	1.250	7.331
i080-022	80	3160	6	**0.000**	**0.000**	6.280	**32.680**	4.011	**3.447**	1.250	7.225
i080-023	80	3160	6	**0.000**	**0.000**	**70.600**	32.080	3.968	**3.459**	1.250	7.345
i080-024	80	3160	6	**0.000**	**0.000**	11.360	**65.340**	3.901	**3.461**	1.250	7.331
i080-025	80	3160	6	**0.000**	**0.000**	**133.060**	42.780	3.944	**3.453**	1.250	7.262
i080-031	80	160	6	**0.000**	**0.000**	4.940	**126.480**	**0.488**	0.502	1.250	4.290
i080-032	80	160	6	**0.000**	**0.000**	0.140	**10.560**	**0.572**	0.604	1.250	4.324
i080-033	80	160	6	**0.000**	**0.000**	6.100	**17.100**	**0.522**	0.543	1.250	4.145
i080-034	80	160	6	**0.000**	**0.000**	2.840	**24.900**	**0.475**	0.501	1.250	4.155
i080-035	80	160	6	**0.000**	**0.000**	0.940	**15.420**	**0.490**	0.543	1.250	4.372
i080-041	80	632	6	**0.000**	**0.000**	1.880	**59.640**	1.093	**1.087**	1.250	6.588
i080-042	80	632	6	**0.000**	**0.000**	0.560	**60.380**	**1.086**	1.090	1.250	6.574
i080-043	80	632	6	**0.000**	**0.000**	0.600	**46.000**	**1.061**	1.079	1.250	6.441
i080-044	80	632	6	**0.000**	**0.000**	1.260	**8.880**	**1.065**	1.104	1.250	6.590
i080-045	80	632	6	**0.000**	**0.000**	0.700	**16.920**	**1.066**	1.091	1.250	6.431
i080-101	80	120	8	**0.000**	**0.000**	0.660	**7.480**	**0.488**	0.490	1.250	3.280
i080-102	80	120	8	**0.000**	**0.000**	1.580	**4.260**	**0.490**	0.501	1.250	3.605
i080-103	80	120	8	**0.000**	**0.000**	3.300	**7.040**	**0.566**	0.592	1.250	3.661
i080-104	80	120	8	3.218	**0.000**	0.000	**50.340**	**0.496**	0.496	1.250	3.576
i080-105	80	120	8	**0.000**	**0.000**	4.020	**691.860**	0.399	**0.355**	1.250	2.950
i080-111	80	350	8	**0.000**	**0.000**	**4.460**	1.660	**0.805**	0.841	1.250	5.855
i080-112	80	350	8	**0.000**	**0.000**	2.300	**17.360**	**0.793**	0.848	1.250	5.840
i080-113	80	350	8	**0.000**	**0.000**	3.200	**40.360**	**0.782**	0.819	1.250	6.437
i080-114	80	350	8	**0.000**	**0.000**	4.100	**47.740**	**0.753**	0.790	1.250	5.983
i080-115	80	350	8	**0.000**	**0.000**	7.800	**27.500**	**0.749**	0.816	1.250	5.886
i080-121	80	3160	8	**0.000**	**0.000**	0.880	**40.460**	4.044	**3.529**	1.250	7.273
i080-122	80	3160	8	**0.000**	**0.000**	**39.300**	25.000	3.913	**3.530**	1.250	7.377
i080-123	80	3160	8	**0.000**	**0.000**	**46.660**	18.460	4.046	**3.550**	1.250	7.412
i080-124	80	3160	8	**0.000**	**0.000**	**150.800**	85.440	4.155	**3.554**	1.250	7.375
i080-125	80	3160	8	**0.000**	**0.000**	**34.660**	11.840	4.017	**3.543**	1.250	7.411
i080-131	80	160	8	**0.000**	**0.000**	0.680	**81.140**	0.545	**0.542**	1.250	4.259
i080-132	80	160	8	**0.000**	**0.000**	5.360	**18.900**	**0.480**	0.528	1.250	4.631
i080-133	80	160	8	**0.000**	**0.000**	10.560	**24.200**	**0.520**	0.551	1.250	4.771
i080-134	80	160	8	**0.000**	**0.000**	7.160	**15.560**	**0.531**	0.534	1.250	4.327
i080-135	80	160	8	**0.000**	**0.000**	2.400	**37.680**	**0.451**	0.477	1.250	4.488
i080-141	80	632	8	**0.000**	**0.000**	1.140	**8.520**	**1.115**	1.168	1.250	6.696
i080-142	80	632	8	**0.000**	**0.000**	1.020	**20.540**	**1.120**	1.173	1.250	6.562
i080-143	80	632	8	**0.000**	**0.000**	1.220	**8.980**	**1.132**	1.174	1.250	6.596
i080-144	80	632	8	**0.000**	**0.000**	1.900	**14.460**	**1.136**	1.172	1.250	6.542
i080-145	80	632	8	**0.000**	**0.000**	1.500	**26.460**	**1.150**	1.164	1.250	6.483

				Gap [%]		# of opt		Time [s]		Firing rate [%]	
name	$\|V\|$	$\|E\|$	$\|T\|$	TS	CS	TS	CS	TS	CS	TS	CS
i080-201	80	120	16	**0.000**	**0.000**	8.680	**90.920**	**0.594**	0.600	1.250	3.729
i080-202	80	120	16	**0.000**	**0.000**	2.240	**10.960**	**0.505**	0.534	1.250	3.744
i080-203	80	120	16	**0.000**	**0.000**	11.740	**49.180**	**0.560**	0.578	1.250	3.967
i080-204	80	120	16	**0.000**	**0.000**	0.780	**28.720**	0.691	**0.657**	1.250	3.380
i080-205	80	120	16	**0.000**	**0.000**	33.200	**92.060**	0.544	**0.543**	1.250	3.377
i080-211	80	350	16	**0.000**	**0.000**	1.600	**12.060**	**0.960**	1.001	1.250	5.988
i080-212	80	350	16	**0.000**	**0.000**	1.700	**9.460**	**0.959**	1.001	1.250	6.012
i080-213	80	350	16	**0.000**	**0.000**	1.840	**8.780**	**0.950**	0.996	1.250	6.135
i080-214	80	350	16	**0.000**	**0.000**	2.000	**4.720**	**0.970**	1.011	1.250	6.154
i080-215	80	350	16	**0.000**	**0.000**	3.060	**10.240**	**0.964**	1.018	1.250	6.147
i080-221	80	3160	16	**0.000**	**0.000**	**16.080**	8.840	4.447	**3.898**	1.250	7.375
i080-222	80	3160	16	**0.000**	**0.000**	13.580	**27.940**	4.390	**3.907**	1.250	7.370
i080-223	80	3160	16	**0.000**	**0.000**	3.400	**13.680**	4.462	**3.898**	1.250	7.393
i080-224	80	3160	16	**0.000**	**0.000**	**86.280**	10.840	4.518	**3.868**	1.250	7.377
i080-225	80	3160	16	**0.000**	**0.000**	**185.780**	19.380	4.389	**3.891**	1.250	7.350
i080-231	80	160	16	**0.000**	**0.000**	4.840	**13.280**	**0.740**	0.757	1.250	4.251
i080-232	80	160	16	**0.000**	**0.000**	1.620	**6.880**	**0.700**	0.724	1.250	4.477
i080-233	80	160	16	**0.000**	**0.000**	1.500	**35.040**	**0.662**	0.681	1.250	4.490
i080-234	80	160	16	**0.000**	**0.000**	2.500	**25.180**	**0.596**	0.644	1.250	4.965
i080-235	80	160	16	**0.000**	**0.000**	5.520	**20.820**	**0.667**	0.706	1.250	4.405
i080-241	80	632	16	**0.000**	**0.000**	**1.980**	1.260	**1.296**	1.357	1.250	6.528
i080-242	80	632	16	**0.000**	**0.000**	0.200	**5.240**	**1.275**	1.324	1.250	6.429
i080-243	80	632	16	**0.000**	**0.000**	0.700	**7.020**	**1.277**	1.322	1.250	6.562
i080-244	80	632	16	**0.000**	**0.000**	0.980	**7.520**	**1.260**	1.308	1.250	6.362
i080-245	80	632	16	**0.000**	**0.000**	0.880	**7.760**	**1.318**	1.358	1.250	6.545
i080-301	80	120	20	**0.000**	**0.000**	9.640	**26.400**	**0.611**	0.618	1.250	3.685
i080-302	80	120	20	**0.000**	**0.000**	4.080	**17.560**	**0.683**	0.692	1.250	3.861
i080-303	80	120	20	**0.000**	**0.000**	26.800	**41.320**	**0.630**	0.639	1.250	3.616
i080-304	80	120	20	**0.000**	**0.000**	63.240	**323.580**	0.571	**0.568**	1.250	3.549
i080-305	80	120	20	**0.000**	**0.000**	**27.840**	15.920	**0.720**	0.740	1.250	3.677
i080-311	80	350	20	**0.000**	**0.000**	0.760	**21.440**	**0.991**	1.033	1.250	5.803
i080-312	80	350	20	**0.000**	**0.000**	2.660	**12.780**	**0.980**	1.025	1.250	5.931
i080-313	80	350	20	**0.000**	**0.000**	2.300	**35.320**	**0.992**	1.034	1.250	5.777
i080-314	80	350	20	**0.000**	**0.000**	7.120	**7.480**	**0.986**	1.037	1.250	5.838
i080-315	80	350	20	**0.000**	**0.000**	4.740	**37.680**	**0.990**	1.030	1.250	5.766
i080-321	80	3160	20	**0.000**	**0.000**	**61.880**	22.580	4.634	**4.111**	1.250	7.146
i080-322	80	3160	20	**0.000**	**0.000**	20.680	**20.720**	4.584	**4.035**	1.250	7.130
i080-323	80	3160	20	**0.000**	**0.000**	**150.880**	7.340	4.654	**4.064**	1.250	7.139
i080-324	80	3160	20	**0.000**	**0.000**	**69.540**	15.440	4.586	**4.095**	1.250	7.089
i080-325	80	3160	20	**0.000**	**0.000**	**184.120**	36.300	4.657	**4.118**	1.250	7.109
i080-331	80	160	20	**0.000**	**0.000**	16.720	**40.720**	**0.717**	0.737	1.250	4.455
i080-332	80	160	20	**0.000**	**0.000**	4.520	**13.620**	**0.719**	0.733	1.250	4.627
i080-333	80	160	20	**0.000**	**0.000**	0.860	**16.480**	**0.770**	0.782	1.250	4.299
i080-334	80	160	20	**0.000**	**0.000**	**16.140**	12.720	**0.751**	0.777	1.250	4.663
i080-335	80	160	20	**0.000**	**0.000**	4.560	**24.800**	**0.700**	0.729	1.250	4.573
i080-341	80	632	20	**0.000**	**0.000**	2.080	**20.860**	**1.346**	1.383	1.250	6.183
i080-342	80	632	20	**0.000**	**0.000**	1.480	**4.720**	**1.364**	1.416	1.250	6.274
i080-343	80	632	20	**0.000**	**0.000**	1.080	**25.100**	**1.359**	1.391	1.250	6.176
i080-344	80	632	20	**0.000**	**0.000**	3.920	**11.520**	**1.357**	1.391	1.250	6.218
i080-345	80	632	20	**0.000**	**0.000**	0.340	**1.980**	**1.359**	1.420	1.250	6.187
ave				0.032	**0.000**	17.151	**37.110**	1.494	**1.413**	1.250	5.551

Table 4.5: Results of the tabu search and the chaotic search for the benchmark problem set I160

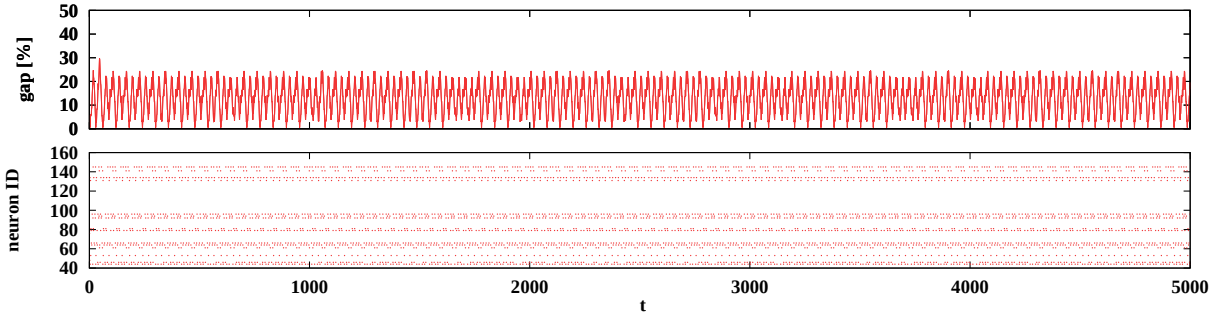
				Gap [%]		# of opt		Time [s]		Firing rate [%]	
name	$\|V\|$	$\|E\|$	$\|T\|$	TS	CS	TS	CS	TS	CS	TS	CS
i160-001	160	240	7	**0.000**	**0.000**	0.400	**132.540**	1.054	**0.875**	0.625	0.365
i160-002	160	240	7	**0.000**	**0.000**	46.320	**295.140**	0.792	**0.686**	0.625	0.461
i160-003	160	240	7	**0.000**	**0.000**	0.260	**361.140**	0.821	**0.682**	0.625	0.231
i160-004	160	240	7	**0.000**	**0.000**	15.560	**1600.760**	0.809	**0.720**	0.625	0.397
i160-005	160	240	7	**0.000**	**0.000**	8.340	**103.860**	0.959	**0.838**	0.625	0.116
i160-011	160	812	7	**0.000**	**0.000**	0.660	**17.900**	**1.983**	2.262	0.625	4.086
i160-012	160	812	7	**0.000**	**0.000**	2.980	**8.180**	**2.047**	2.310	0.625	3.665
i160-013	160	812	7	**0.000**	**0.000**	3.220	**41.940**	**1.922**	2.184	0.625	3.890
i160-014	160	812	7	**0.000**	**0.000**	0.780	**4.760**	**2.134**	2.352	0.625	4.013
i160-015	160	812	7	**0.000**	**0.000**	**2.860**	2.620	**2.079**	2.353	0.625	3.952
i160-021	160	12720	7	**0.000**	**0.000**	**210.440**	9.240	22.637	**22.593**	0.625	5.663
i160-022	160	12720	7	**0.000**	**0.000**	**22.600**	2.400	23.801	**22.875**	0.625	5.652
i160-023	160	12720	7	**0.000**	**0.000**	**19.060**	17.220	23.239	**22.612**	0.625	5.630
i160-024	160	12720	7	**0.000**	**0.000**	**13.640**	3.040	23.610	**22.589**	0.625	5.653
i160-025	160	12720	7	**0.000**	**0.000**	**21.640**	4.680	23.708	**22.723**	0.625	5.626
i160-031	160	320	7	**0.000**	**0.000**	1.200	**45.940**	**1.074**	1.144	0.625	1.962
i160-032	160	320	7	**0.000**	**0.000**	0.140	**4.380**	**1.045**	1.064	0.625	1.270
i160-033	160	320	7	**0.000**	**0.000**	0.120	**8.440**	1.261	**1.192**	0.625	1.738
i160-034	160	320	7	**0.000**	0.144	**19.180**	0.000	**1.040**	1.121	0.625	1.478
i160-035	160	320	7	**0.000**	**0.000**	0.640	**7.600**	1.109	**1.019**	0.625	1.412
i160-041	160	2544	7	**0.000**	**0.000**	0.320	**1.200**	**5.753**	5.921	0.625	5.262
i160-042	160	2544	7	**0.000**	**0.000**	0.720	**5.700**	**5.641**	5.868	0.625	5.212
i160-043	160	2544	7	**0.000**	**0.000**	1.960	**2.040**	**5.616**	5.880	0.625	5.325
i160-044	160	2544	7	**0.000**	**0.000**	0.940	**16.380**	**5.576**	5.896	0.625	5.200
i160-045	160	2544	7	**0.000**	**0.000**	2.620	**4.160**	**5.718**	6.014	0.625	5.013
i160-101	160	240	12	**0.000**	**0.000**	11.120	**114.980**	1.157	**1.106**	0.625	0.910
i160-102	160	240	12	**0.000**	**0.000**	1.580	**49.680**	1.293	**1.216**	0.625	0.731
i160-103	160	240	12	**0.000**	**0.000**	5.500	**256.940**	1.130	**1.025**	0.625	0.714
i160-104	160	240	12	**0.000**	**0.000**	0.880	**1.500**	1.100	**1.056**	0.625	0.223
i160-105	160	240	12	**0.000**	**0.000**	28.240	**540.860**	1.045	**1.026**	0.625	1.052
i160-111	160	812	12	**0.000**	**0.000**	**13.340**	3.520	**2.328**	2.579	0.625	4.529
i160-112	160	812	12	**0.000**	**0.000**	**0.760**	0.280	**2.573**	2.801	0.625	4.520
i160-113	160	812	12	**0.000**	**0.000**	**1.980**	0.520	**2.413**	2.713	0.625	4.671
i160-114	160	812	12	0.067	**0.000**	0.000	**0.380**	**2.550**	2.746	0.625	4.582
i160-115	160	812	12	**0.000**	**0.000**	**2.820**	0.340	**2.361**	2.650	0.625	4.571
i160-121	160	12720	12	**0.000**	**0.000**	**1.840**	0.640	24.373	**23.298**	0.625	5.966
i160-122	160	12720	12	**0.000**	**0.000**	**196.460**	6.920	23.844	**23.459**	0.625	5.938
i160-123	160	12720	12	**0.000**	**0.000**	**34.740**	2.340	23.846	**23.229**	0.625	5.951
i160-124	160	12720	12	**0.000**	**0.000**	**101.800**	2.000	23.881	**23.259**	0.625	5.942
i160-125	160	12720	12	**0.000**	**0.000**	**15.180**	2.660	23.904	**23.323**	0.625	5.933
i160-131	160	320	12	**0.000**	**0.000**	13.620	**14.220**	**1.376**	1.515	0.625	2.376
i160-132	160	320	12	**0.000**	**0.000**	0.060	**7.700**	**1.323**	1.374	0.625	2.776
i160-133	160	320	12	2.092	**0.000**	0.000	**8.860**	**1.426**	1.565	0.625	2.314
i160-134	160	320	12	0.548	**0.000**	0.000	**16.660**	**1.221**	1.342	0.625	2.566
i160-135	160	320	12	**0.000**	**0.000**	1.500	**5.960**	**1.374**	1.477	0.625	2.271
i160-141	160	2544	12	**0.000**	**0.000**	1.020	**3.560**	**5.952**	6.265	0.625	5.501
i160-142	160	2544	12	**0.000**	**0.000**	0.380	**0.440**	**5.970**	6.365	0.625	5.525
i160-143	160	2544	12	**0.000**	**0.000**	1.020	**3.740**	**5.936**	6.384	0.625	5.549
i160-144	160	2544	12	0.192	**0.000**	0.000	**0.040**	**5.869**	6.321	0.625	5.463
i160-145	160	2544	12	**0.000**	**0.000**	**0.260**	0.120	**5.945**	6.416	0.625	5.497

				Gap [%]		# of opt		Time [s]		Firing rate [%]	
name	$\|V\|$	$\|E\|$	$\|T\|$	TS	CS	TS	CS	TS	CS	TS	CS
i160-201	160	240	24	**0.000**	**0.000**	16.360	**24.520**	**1.614**	1.654	0.625	2.290
i160-202	160	240	24	**0.000**	**0.000**	1.280	**184.960**	1.605	**1.588**	0.625	1.344
i160-203	160	240	24	**0.000**	**0.000**	0.640	**15.700**	**1.668**	1.783	0.625	2.141
i160-204	160	240	24	**0.000**	**0.000**	2.220	**4.340**	**1.560**	1.718	0.625	2.299
i160-205	160	240	24	1.474	**0.000**	0.000	**0.660**	**1.518**	1.643	0.625	1.930
i160-211	160	812	24	**0.000**	**0.000**	**1.340**	0.240	**3.124**	3.375	0.625	4.980
i160-212	160	812	24	**0.000**	**0.000**	**0.240**	0.020	**3.112**	3.350	0.625	5.071
i160-213	160	812	24	**0.000**	**0.000**	0.040	**0.160**	**3.071**	3.355	0.625	5.044
i160-214	160	812	24	0.245	**0.000**	0.000	**0.020**	**3.109**	3.387	0.625	4.686
i160-215	160	812	24	0.870	**0.000**	0.000	**0.620**	**3.142**	3.372	0.625	5.169
i160-221	160	12720	24	**0.000**	**0.000**	**136.960**	0.180	25.542	**25.339**	0.625	6.411
i160-222	160	12720	24	**0.000**	**0.000**	**110.500**	2.000	25.289	**25.086**	0.625	6.358
i160-223	160	12720	24	**0.000**	**0.000**	**138.760**	0.480	25.203	**25.034**	0.625	6.355
i160-224	160	12720	24	**0.000**	**0.000**	**8.640**	0.780	25.371	**25.136**	0.625	6.365
i160-225	160	12720	24	**0.000**	**0.000**	**5.380**	0.300	25.441	**25.147**	0.625	6.430
i160-231	160	320	24	0.901	**0.000**	0.000	**1.920**	**1.674**	1.954	0.625	3.564
i160-232	160	320	24	**0.000**	**0.000**	**1.080**	1.020	**1.789**	1.978	0.625	3.059
i160-233	160	320	24	1.325	**0.000**	0.000	**2.280**	**1.952**	2.117	0.625	2.813
i160-234	160	320	24	0.167	**0.000**	0.000	**2.800**	**1.836**	2.057	0.625	2.552
i160-235	160	320	24	**0.000**	**0.000**	0.080	**0.660**	**1.977**	2.168	0.625	3.088
i160-241	160	2544	24	**0.000**	**0.000**	**3.280**	0.140	**6.492**	6.968	0.625	5.722
i160-242	160	2544	24	**0.000**	**0.000**	**0.280**	0.020	**6.487**	6.990	0.625	5.830
i160-243	160	2544	24	**0.000**	**0.000**	0.020	**0.160**	**6.416**	6.852	0.625	5.676
i160-244	160	2544	24	0.532	**0.000**	0.000	**0.040**	**6.560**	7.091	0.625	5.749
i160-245	160	2544	24	**0.000**	**0.000**	**3.960**	0.120	**6.575**	7.116	0.625	5.836
i160-301	160	240	40	**0.000**	**0.000**	**7.780**	1.500	**1.977**	2.228	0.625	2.922
i160-302	160	240	40	**0.000**	**0.000**	**13.560**	1.880	**2.054**	2.221	0.625	2.667
i160-303	160	240	40	0.542	**0.000**	0.000	**1.400**	**1.867**	2.085	0.625	2.619
i160-304	160	240	40	**0.000**	**0.000**	3.900	**4.300**	**2.095**	2.288	0.625	2.798
i160-305	160	240	40	0.306	**0.000**	0.000	**0.760**	**1.880**	2.076	0.625	2.685
i160-311	160	812	40	**0.000**	**0.000**	**1.640**	0.220	**3.610**	3.904	0.625	5.218
i160-312	160	812	40	**0.000**	**0.000**	0.020	**0.040**	**3.735**	4.000	0.625	5.121
i160-313	160	812	40	**0.000**	**0.000**	**1.200**	0.140	**3.722**	4.065	0.625	5.157
i160-314	160	812	40	0.145	**0.000**	0.000	**0.160**	**3.756**	3.999	0.625	4.993
i160-315	160	812	40	**0.000**	**0.000**	**1.220**	0.080	**3.653**	3.868	0.625	5.178
i160-321	160	12720	40	**0.000**	**0.000**	**12.160**	0.180	**27.624**	27.971	0.625	6.432
i160-322	160	12720	40	**0.000**	**0.000**	**25.780**	0.560	**27.591**	27.765	0.625	6.422
i160-323	160	12720	40	**0.000**	**0.000**	**13.620**	1.040	**27.531**	27.686	0.625	6.394
i160-324	160	12720	40	**0.000**	**0.000**	**14.140**	0.360	**27.549**	27.833	0.625	6.491
i160-325	160	12720	40	**0.000**	**0.000**	**15.300**	0.400	**27.383**	28.026	0.625	6.388
i160-331	160	320	40	0.077	**0.000**	0.000	**2.260**	**2.265**	2.472	0.625	3.575
i160-332	160	320	40	0.139	**0.000**	0.000	**0.060**	**2.366**	2.558	0.625	3.775
i160-333	160	320	40	**0.000**	**0.000**	**31.480**	1.120	**2.099**	2.315	0.625	3.826
i160-334	160	320	40	**0.000**	**0.000**	3.060	**5.760**	**2.131**	2.293	0.625	3.383
i160-335	160	320	40	0.623	**0.000**	0.000	**0.460**	**2.317**	2.552	0.625	3.276
i160-341	160	2544	40	**0.000**	**0.000**	**1.920**	0.180	**7.284**	7.800	0.625	5.595
i160-342	160	2544	40	**0.000**	0.048	**0.060**	0.000	**7.170**	7.671	0.625	5.611
i160-343	160	2544	40	**0.000**	**0.000**	**1.200**	0.140	**7.249**	7.530	0.625	5.513
i160-344	160	2544	40	**0.000**	**0.000**	**0.300**	0.000	**7.231**	7.390	0.625	5.672
i160-345	160	2544	40	**0.000**	**0.000**	**3.240**	0.020	**7.316**	7.748	0.625	5.570
ave				0.102	**0.002**	14.133	**40.133**	**7.452**	7.549	0.625	4.074

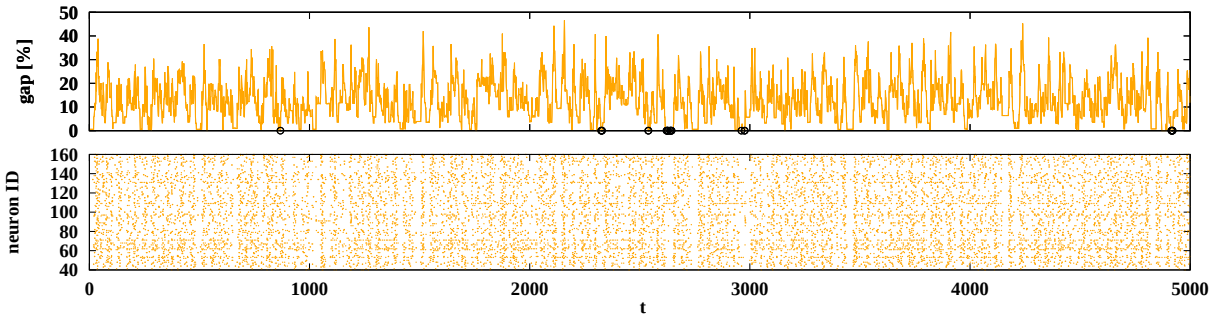
Next, we compare the searching processes of the chaotic search and the tabu search. To clarify the searching process, we take the time series of the gap and raster plots.

The searching processes of the chaotic search and the tabu search when solving i160-134 are shown in Fig. 4.5. In each figure, the upper part shows the time series of the gap, while the lower part shows the raster plots. From Fig. 4.5 (a), the tabu search does not reach an optimal solution and the same neurons fired repeatedly. On the other hand, from Fig. 4.5 (b), the chaotic search reaches the optimal solution 42 times, and almost all neurons fire at least once. These results indicate that the chaotic search can search a wide variety of solutions effectively.

The searching processes of the chaotic search and tabu search when solving i160-034 are shown in Fig. 4.6. From Fig. 4.6 (a), the tabu search reaches the optimal solution 45 times, and neuron firing continues during the search. However, from Fig. 4.6 (b), the gap of the chaotic search does not move after $t = 2,748$ because no neurons fire after $t = 2,748$. These results indicate that the performance of the chaotic search depends on the parameter set.

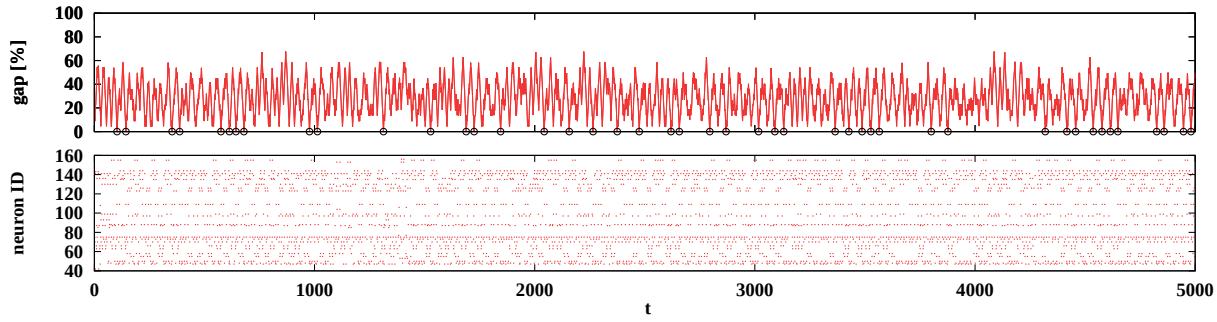


(a) Tabu search

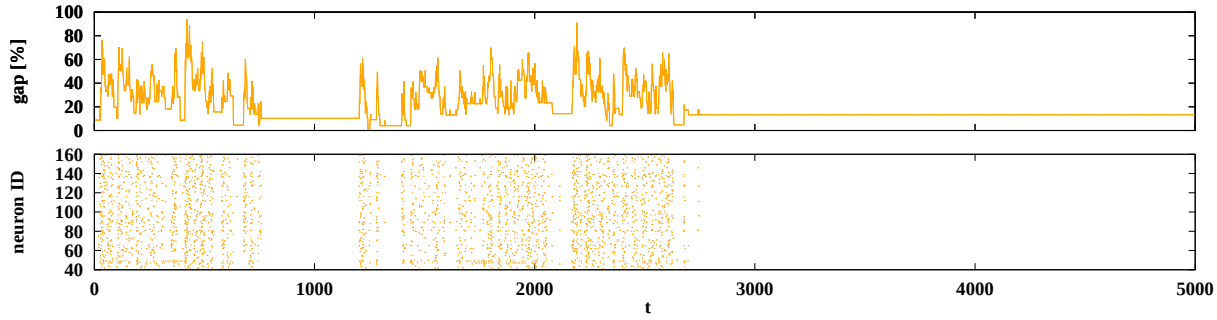


(b) Chaotic search

Figure 4.5: Example of searching processes of the (a) tabu search and (b) chaotic search when solving i160-134. In (a), $s = 10$. In (b), $\beta = 0.89, \alpha = 1, k = 0.9, \theta = 1$, $W = 0.005$, and $\epsilon = 0.001$. Black circles indicate when the search finds optimal solutions. The results of gaps show that the chaotic search finds optimal solutions 42 times during $5,000$ iterations. However, the tabu search does not reach an optimal solution. The raster plots show that a wider variety of firing is observed in the chaotic search than in the tabu search.



(a) Tabu search



(b) Chaotic search

Figure 4.6: Example of searching processes of the (a) tabu search and (b) chaotic search when solving i160-034. In (a), $s = 10$. In (b), $\beta = 0.89, \alpha = 1, k = 0.9, \theta = 1$, $W = 0.005$, and $\epsilon = 0.001$. Black circles indicate when the search finds optimal solutions. The results of gaps show that the tabu search finds optimal solutions 45 times during $5,000$ iterations. However, the chaotic search does not reach an optimal solution. The raster plots show that no neurons fire during $t = 760$ to $1,201$ and $t = 2,747$ to 5000 in the chaotic search. This is an example of a chaotic search with poor performance.

4.4.4 Comparison with other meta-heuristics

Many meta-heuristics have been applied to solve the Steiner tree problem in graphs. Thus, we compare the performances of some popular meta-heuristics with the chaotic search by using the benchmark problem set C. We introduce three meta-heuristics: genetic algorithm [42], tabu search [40, 41], and greedy randomized adaptive search procedure (GRASP) [57].

A genetic algorithm searches a solution based on the principles derived from the dynamics of natural population genetics. In a genetic algorithm, each solution is expressed as a chromosome. Then, new solutions are made by applying crossover, mutation, and inversion operations. The solution of the Steiner tree problem in graphs is represented by a set of vertices included in the Steiner tree. The fitness of any solution is computed as its minimum spanning tree in an induced subgraph by a set of vertices included in the Steiner tree. Esbensen [64] represented each gene by a pair of variables, one representing the number of genes (vertices) and the second the allele of the gene (whether it is included or not).

A tabu search searches a solution following a steepest descent mildest ascent approach, a solution which has the largest improvement or the smallest worsening is selected as the next solution. To avoid selecting the same solutions repeatedly, i.e., cycling, the tabu search prohibits some exchanges for a while. Gendreau et al. [65] presented a tabu search with diversification. Starting from a certain number of feasible solutions constructed by the randomized shortest path heursitic [35], they apply a tabu search on all these solutions and record the best result. Then, a diversification mechanism is applied. They consider two approaches to diversify the current solution: one is removing vertices included in the current solution for a long time and the other is including vertices not included in the current solution for a long time. If there are no improvements for a certain number of iterations, they apply a path diversification; a path between two vertices i and j is removed from a given solution S where $d_S(i,j) - d(i,j)$ is largest for all vertices $i, j \in V_S$ ($d_S(i,j)$ denotes the shortest distance between i and j in the current solution S).

GRASP searches a solution in two phases: constructing a feasible solution and then performing a local search. In the construction phase, feasible solutions are built using construction methods, such as the shortest path heuristic [35], distance network heuristic [36], and average distance heuristic [37]. Here, to obtain a variety of solutions, we introduce randomness to construction methods. For example, in the case of the shortest path heuristic, we can obtain a variety of solutions by randomly selecting a vertex to be added to the tree in the next iteration. In the local search phase, a constructed solution is improved by applying a local search. GRASP for the Steiner tree problem in graphs was implemented by [56]. They use the distance network heuristic as the construction method and the Steiner vertex search as the local search method.

Table 4.6 shows a comparison of the best gaps of the chaotic search (CS), genetic algorithm (GA) [64], tabu search (TS) [65], and GRASP [56] for the benchmark problem set C.

From Table 4.6, the chaotic search finds the optimum solutions for 14 instances, the genetic algorithm finds the optimum solution for all (20) instances, the tabu search finds the optimum solutions for 18 instances, and GRASP finds the optimum solutions for 18 instances.

The chaotic search shows poor performance for this benchmark problem set because the appropriate parameters are different for different edge densities. If we tune parameters for solving low density instances, its firing rate is too low for solving high density instances. To overcome this issue, we will develop an automatic parameter tuning method.

Table 4.6: Comparison of the best gaps [%] of four metaheuristics

name	$\|V\|$	$\|E\|$	$\|T\|$	CS	GA [64]	TS [65]	GRASP [56]
c01	500	625	5	**0.00**	**0.00**	**0.00**	**0.00**
c02	500	625	10	**0.00**	**0.00**	**0.00**	**0.00**
c03	500	625	83	**0.00**	0.03	**0.00**	**0.00**
c04	500	625	125	0.09	0.01	**0.00**	**0.00**
c05	500	625	250	**0.00**	**0.00**	**0.00**	**0.00**
c06	500	1000	5	**0.00**	**0.00**	**0.00**	**0.00**
c07	500	1000	10	**0.00**	**0.00**	**0.00**	**0.00**
c08	500	1000	83	0.20	**0.00**	**0.00**	**0.00**
c09	500	1000	125	0.14	0.06	0.14	**0.00**
c10	500	1000	250	0.09	**0.00**	**0.00**	**0.00**
c11	500	2500	5	**0.00**	**0.00**	**0.00**	**0.00**
c12	500	2500	10	**0.00**	**0.00**	**0.00**	**0.00**
c13	500	2500	83	**0.00**	0.66	**0.00**	**0.00**
c14	500	2500	125	**0.00**	0.12	0.31	**0.00**
c15	500	2500	250	**0.00**	**0.00**	**0.00**	**0.00**
c16	500	12500	5	**0.00**	6.36	**0.00**	**0.00**
c17	500	12500	10	**0.00**	**0.00**	**0.00**	**0.00**
c18	500	12500	83	0.89	1.15	**0.00**	2.66
c19	500	12500	125	0.69	0.68	**0.00**	0.68
c20	500	12500	250	**0.00**	**0.00**	**0.00**	**0.00**
average				0.11	0.45	**0.02**	0.17

4.5 Discussion of Chapter 4

We evaluate the performance of the chaotic search by comparing it with that of the tabu search. We used the benchmark problem sets B, C, I080, and I160 in SteinLib. The parameters of the tabu search and chaotic search are tuned to appropriate values per benchmark problem set by pre-numerical experiments. We summarize the results of the numerical experiments in Table 4.7. In Table 4.7, gap means the average gap from the optimum solutions, time means the average CPU time, # of opt means the number of optimum solutions found during the search. The bottleneck of the computation cost for the tabu search and chaotic search is finding neighborhood solutions. Thus, we compare the performances of the tabu search and chaotic search for finding the same number of neighborhood solutions. To realize this, we set an upper limit on the number of iterations as $5,000$. In other words, we finish the search when all neurons have attempted to update 5,000 times or neighborhood solutions are found $5,000|V \setminus T|$ times.

Table 4.7: Summary of the results obtained by the tabu search and chaotic search.

	Gap [%]		Time [s]		# of opt	
name	TS	CS	TS	CS	TS	CS
B	0.00	0.00	1.444	2.066	16.90	272.92
C	0.03	0.11	46.767	80.772	773.69	4.39
I080	0.03	0.00	1.494	1.413	17.15	37.11
I160	0.10	0.00	7.452	7.549	14.13	40.13
average	0.04	0.03	14.289	22.95	205.47	88.64

From Table 4.7, both the tabu search and the chaotic search always find optimum solutions for the benchmark problem set B. The CPU time of the chaotic search is longer than that of the tabu search because the total numbers of firings and moving solutions is different. In the tabu search, only one neuron fires during one iteration. On the other hand, in the chaotic search, up to $|V \setminus T|$ neurons fires during one iteration because neurons fire whenever their output is above the threshold value. However, the computation cost of moving solutions is smaller than that of finding neighborhood solutions. The total number of optimum solutions found during the chaotic search is much larger than that during the tabu search. Frequent hits of good solutions is very important when solving real-world problems, i.e., when the optimum solution is unclear. From this aspect, the chaotic search significantly outperforms the tabu search for the benchmark problem set B.

For the benchmark problem set C, the average gap of the chaotic search is larger than that of the tabu search. One of the reasons why the chaotic search shows poor performance is that the parameter tuning is very difficult for C. C has 20 instances that have different

numbers of edges, terminals, and of course, structures. If we set parameters for low firing rates, the chaotic search shows good performance for instances with a small number of edges; however, it shows poor performance for instances with a large number of edges. Thus, we set parameters for mid-range firing rates. If we tune parameters per instance (not per problem set), the performance of the chaotic search may improve. The CPU time of the chaotic search is approximately twice that of the tabu search. This is because the firing rate of the chaotic search is very high for instances with a small number of edges, and the total number of moving solutions becomes larger than that in the tabu search. The total number of optimum solutions found during the chaotic search is much smaller than that during the tabu search. This is because the chaotic search cannot find optimum solutions for almost all instances.

For the benchmark problem sets I080 and I160, the chaotic search always finds optimum solutions; however, the tabu search fails to find optimum solutions for some instances. Besides, the CPU times of the chaotic search and tabu search are similar. In particular, in the case of I080, the CPU time of the chaotic search is shorter than that of the tabu search. The total number of optimum solutions found by the chaotic search is larger than that by the tabu search. From these results, the chaotic search shows better performance than the tabu search for I080 and I160.

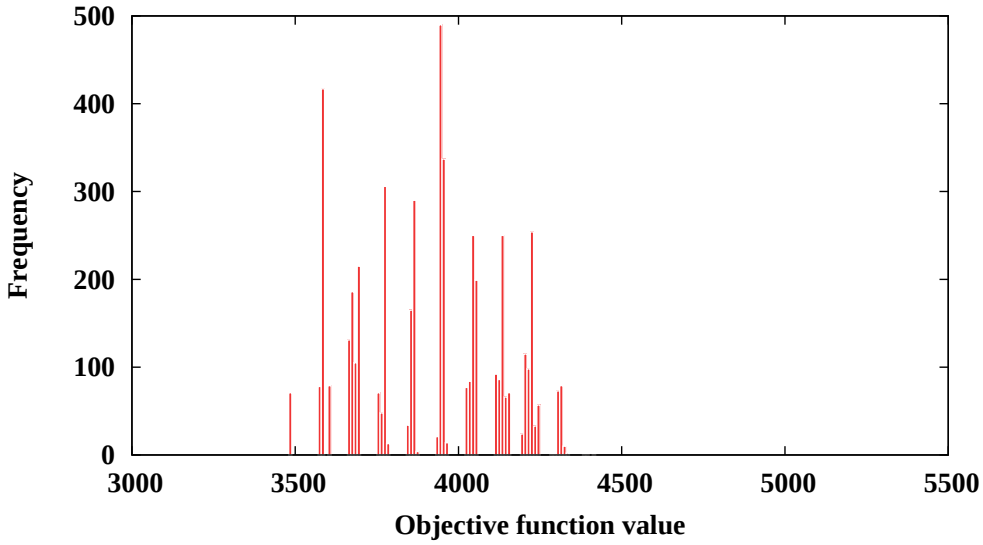
Benchmark problem sets I080 and I160 are designed to disable the pre-processing; in these problem sets edges larger than the shortest distance between both vertices are removed to reduce the input size. From this perspective, these problems are more difficult than the benchmark problem sets B and C. In spite of this, the chaotic search always finds optimum solutions for them. This result indicates that the chaotic search shows good performance without pre-processing.

One of the reasons why the chaotic search shows good performance is that it searches a wide variety of solutions. In general, the meta-heuristics show good performance when the balance between intensification and diversification fits the instance. Intensification refers to searching around good solutions that have already been found. On the other hand, diversification refers to searching mainly in areas that have not yet been searched. Thus, they are contradictory. To compare the strength of diversification of the chaotic search with that of the tabu search, we investigate the distribution of the objective function value of the solutions found during the search. Figures 4.7 and 4.8 show these distribution. It should be noted that in Fig. 4.8 (b), the maximum frequency was $2,301$, the second maximum was 444, and the third maximum was 129. There are no neuron fired during $t = 760$ to $1,201$ (441 iterations) and $t = 2,747$ to $5,000$ ($2,253$ iterations). In these periods, the current solution was not updated and its objective function value was not changed too. This is the reason why the first and second maximum frequencies become very large (the same objective function value is kept for these iterations). Thus, we ignore them and set the range of the vertical axis from 0 to 350. From Fig. 4.7, it can be seen that the chaotic search found a wide variety of solutions than the tabu search. The maximum value of the objective function value found is $4,412$ in

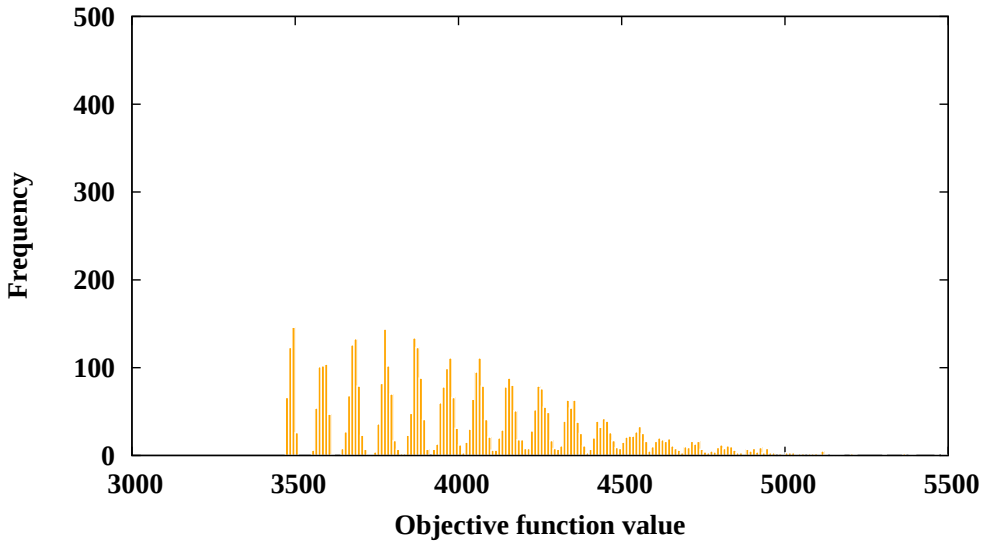
the tabu search and $5,490$ in the chaotic search. Thus, strong diversification may contribute to find good solutions for the Steiner tree problem in graphs. In Fig. 4.8, it can be seen that the chaotic search also found a wide variety of solutions than the tabu search. However, the chaotic search repeatedly searching the same solution. This is because no neurons fire during $t = 760$ to $1,201$ and $t = 2,747$ to $5,000$. The current solution is not updated during these periods, thus, the objective function value is also not changed during these periods. From these results, strong diversification may contribute to finding good solutions for the Steiner tree problem in graphs.

We also compare the performance of the chaotic search with other meta-heuristics: the genetic algorithm [64], tabu search [65], and greedy randomized adaptive search procedure (GRASP) [56]. Note that these meta-heuristics are carefully tuned to solve benchmark instances, i.e., applying pre-processing, including a special diversification mechanism, etc. In contrast, our chaotic search is very simple.

From Table 4.6, the average gap of the chaotic search is smaller than that of the genetic algorithm and GRASP. On the other hand, the average gap of the tabu search is smaller than that of the chaotic search. However, the chaotic search found a smaller gap solution than the tabu search for instance c14. This result indicates that the chaotic search can outperform the tabu search when it is sophisticated i.e., applies pre-processing, includes a special diversification mechanism, etc.

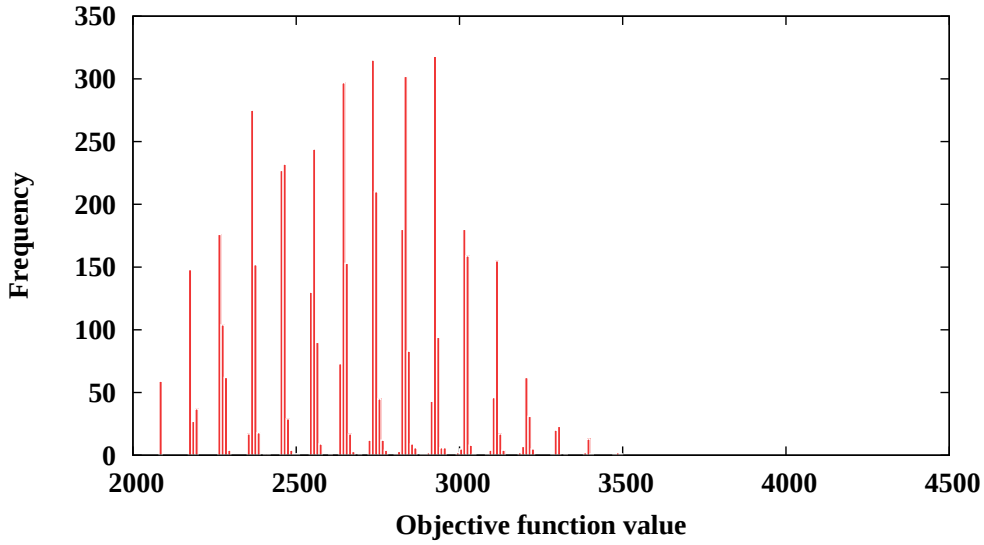


(a) Tabu search

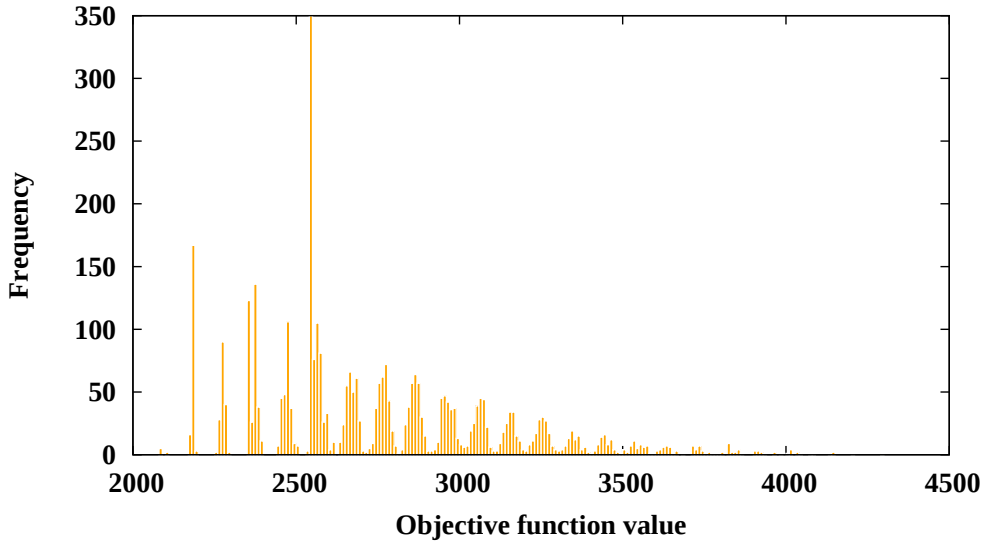


(b) Chaotic search

Figure 4.7: Example of distribution of found solutions of the (a) tabu search and (b) chaotic search when solving i160-034. In (a), $s = 10$. In (b), $\beta = 0.89, \alpha = 1, k = 0.9, \theta = 1$, $W = 0.005$, and $\epsilon = 0.001$. This is an example of a chaotic search with good performance.



(a) Tabu search



(b) Chaotic search

Figure 4.8: Example of distribution of found solutions of the (a) tabu search and (b) chaotic search when solving i160-034. In (a), $s = 10$. In (b), $\beta = 0.89, \alpha = 1, k = 0.9, \theta = 1$, $W = 0.005$, and $\epsilon = 0.001$. This is an example of a chaotic search with poor performance. The search has not proceed during $t = 760$ to $1,201$ and $t = 2,747$ to $5,000$. Thus, the first and second maximum frequencies was 2301 and 444, respectively. However, frequencies less than third is very small. To visualize these small values, we set the range of the vertical axis from 0 to $5,000$.

4.6 Summary of Chapter 4

The improvement method is another practical approach to solving $\mathcal{NP}$-hard combinatorial optimization problems. There are two types of improvement methods: local search method and meta-heuristics. The local search method is specialized for specific problems. For solving the Steiner tree problem in graphs, the Steiner vertex neighborhood and key-path neighborhood are frequently used to find neighborhood solutions, and the local search method is realized by selecting the best solution among neighborhoods. Unfortunately, repeating the local search method converges to local optima. To escape from such undesirable local optima, meta-heuristics are used. Meta-heuristics control the application of the local search methods. Many meta-heuristics have been applied to solve the Steiner tree problem in graphs, such as the genetic algorithm, tabu search, and greedy randomized adaptive search procedure. Meanwhile, efficient meta-heuristics have been proposed to solve the traveling salesman problem, which is a classical and well-studied NP-hard combinatorial optimization problem proposed [12, 13, 14, 15]. This method uses chaotic neurodynamics [11] to find good solutions, i.e., chaotic search. After that, the chaotic search was applied to solve other $\mathcal{NP}$-hard combinatorial optimization problems, such as the quadratic assignment problem [16, 17, 18], vehicle routing problem [19, 20], packet routing problem [21, 22, 23, 24, 25, 26, 27, 28], and motif extraction problem [29, 30], and it has shown good performance for them. However, the chaotic search has not been used to solve the Steiner tree problem in graphs yet. Therefore, we apply this method to solve the Steiner tree problem in graphs.

The chaotic search uses chaotic neurodynamics to find good solutions. The most important points to model chaotic neurodynamics are that (1) neurons fire when the sum of the inputs surpasses the threshold and (2) after a neuron fires, this neuron cannot fire for a while. Additionally, Hasegawa *et al.* [12] introduced a mechanism to tune firing rates using the history of the firing of all neurons. In their implementation, neurons correspond to application of the local search to edges. Thus, $|V|^2$ are used. The firing rate is tuned by using the sum of the self feedback and the sum of output of neurons that correspond to edges connecting the same city. However, in our implementation, neurons correspond to the local search for vertices not edges. Thus, we cannot directly apply this firing rate tuning method. On the other hand, Hasegawa *et al.* also proposed the chaotic search that neurons correspond to application of the local search to vertices in 2002 [17]. Thus, we introduce this firing rate tuning method.

The biggest difference between the problems that have already solved by the chaotic search and the Steiner tree problem in graphs is that the local search method produces not only feasible solutions but also infeasible ones. Thus, if the local search method produces infeasible solutions, we ignore them.

The chaotic search we use to solve the Steiner tree problem in graphs has six parameters: scaling parameter of the gain effect β, scaling parameter of the refractory effect α, decay

factor of the refractory effect k, bias of the refractory effect θ, connection weight of neurons W, and steepness parameter of the sigmoidal function ϵ. Note that the α and β must be balanced. Thus, we fix one of them and tune the other; we fix α to 1 and tune β. For ease of tuning β, we normalize the gain or the improvement of the local search by the maximum edge weight for the benchmark problem sets B and C and by the median of edge weights for the benchmark problem sets I080 and I160, respectively. This is because the expected value of the gain using the Steiner vertex neighborhood is at most the maximum edge weight. However, in the benchmark problem sets I080 and I160, the maximum edge weight depends on whether there is an edge between terminals. Thus, we use the median of edge weights instead of the maximum edge weight.

If the refractory effect of the chaotic search is very strong and is valid for a limited time, this effect is equivalent to the tabu effect of the tabu search. Thus, we use the tabu search for comparison with the chaotic search.

The results of numerical experiments show that the chaotic search and the tabu search always find optimal solutions for the benchmark problem set B. On the other hand, the chaotic search shows poor performance compared to the tabu search for the benchmark problem set C. One of the reasons why the chaotic search shows poor performance is that parameter tuning is very difficult for this benchmark problem set. In these numerical experiments, we use one parameter set for one benchmark problem set. Despite using the same parameter set, the average firing rate varies between instances. Thus, we adjust parameters so that many instances can be solved with small gaps. If we tune parameters per instance, the chaotic search may show better performance.

For the benchmark problem sets I080 and I160, the chaotic search always finds optimum solutions; however, the tabu search fails to find optimum solutions for some instances. One of the reasons why the chaotic search shows better performance than the tabu search is that the chaotic search searches a wide variety of solutions. We investigate the variety of solutions found by the chaotic search and the tabu search as shown in Figs. 4.7 and 4.8. From Figs. 4.7 and 4.8, it can be seen the distribution of the chaotic search is wider than that of the tabu search. Thus, for solving these benchmark problems, strong diversification of the search is effective and the chaotic search can realize it.

We also compared the average gaps for the benchmark problem set C of the chaotic search to that of the genetic algorithm [64], tabu search [65], and greedy randomized adaptive search procedure (GRASP) [56]. These methods include the pre-processing and diversification mechanism. It is revealed that the chaotic search shows better performance than the genetic algorithm and GRASP. However, the chaotic search shows poor performance than the tabu search. This is because our chaotic search is very simple; not including the pre-processing and diversification mechanism. Thus, if we introduce them, the performance of the chaotic search may improve.

The chaotic search that we used to solve the Steiner tree problem in graphs is very simple;

thus, it has no mechanism to tune firing rates automatically. Our future work will introduce a sophisticated mechanism to tune firing rates and an automatic parameter tuning method for the chaotic search to solve the Steiner tree problem in graphs. Additionally, we will introduce pre-processing and post-processing phases.

Chapter 5

Conclusion

5.1 General conclusion

In this thesis, we aim to develop efficient algorithms to solve the Steiner tree problem in graphs. The Steiner tree problem in graphs is a classical combinatorial optimization problem, and it plays an important role in developing both exact and approximation algorithms. The Steiner tree problem in graphs is easy to understand but difficult to solve, and it has attracted many researchers. The input of the Steiner tree problem in graphs is an undirected graph $G = (V, E)$, where V is a set of vertices and E is a set of edges, and a terminal set $T \subseteq V$. The Steiner tree is a subgraph that spans T and does not have any loops. The objective of the Steiner tree problem in graphs is to find a Steiner tree that minimizes the sum of the weights of the edges included in it. The Steiner tree problem in graphs is an $\mathcal{NP}$-hard combinatorial optimization problem [33]. Thus, at present, many researchers believe that there is no polynomial-time algorithm that can solve this problem exactly and optimally. However, this problem lies in various real-world problems, such as designing very-large-scale integration, multicast trees, and pipelines. The input size of such real-world problems is usually very large, and applying exponential time algorithms to solve such large problems is not practical. It should be noted that in the real world, it is often sufficient to obtain an approximate solution that is close to optimal rather than an exact optimal. Thus, many approximation algorithms have been applied to solve the Steiner tree problem in graphs.

Approximation methods can be roughly divided into two types: construction and improvement. Construction methods focus on finding a feasible solution from an input instance as quickly as possible. The merit of using a construction method is that it can obtain ample solutions very fast. However, we sometimes want a sophisticated solution even if its computation time is long. In this case, we use an improvement method. Improvement methods can be divided into two types: local search and meta-heuristic. The local search receives an instance and an initial solution; then, it repeats producing neighborhood solutions and moving one of them. The local search has been developed independently for each problem because

it must generate neighborhood solutions according to the characteristics of the problem. We can find a better solution than the initial solution by using the local search. Unfortunately, the local search gets trapped at local minima and sometimes cannot find good solutions. To overcome this issue, meta-heuristics, which control local searches, are used. The merit of using meta-heuristics is that they can escape from undesirable local minima and continue the search. We must set some termination conditions when we use meta-heuristics. This is because meta-heuristics do not converge to local minima, unlike the local search. Popular termination conditions are as follows: after a certain time, after a certain number of iterations, and after a certain number of iterations since the last update of the best solution. The construction method cannot generate ample solutions for almost all cases, and the improvement method requires an initial solution generated by the construction method. Therefore, both the construction and improvement methods are important approaches to solve combinatorial optimization problems. From this viewpoint, we tried to develop efficient construction and improvement methods to solve the Steiner tree problem in graphs in this thesis.

If there are two terminals, the Steiner tree problem is the same as the shortest path problem. On the other hand, if all vertices are terminals, the Steiner tree problem is the same as the minimum spanning tree problem. Thus, the shortest path and minimum spanning tree are the results of special patterns of the Steiner tree problem in graphs. From this viewpoint, the shortest path and minimum spanning tree are frequently used to find a Steiner tree. The shortest path heuristic (SPH) [35], distance network heuristic (DNH) [36], and average distance heuristic (ADH) [37] are popular methods to find a small-weight Steiner tree, and these all use the shortest path and minimum spanning tree. However, these methods sometimes find a large-weight Steiner tree because they do not consider the existence of multiple shortest paths between arbitrary vertex pairs. If there are multiple shortest paths between arbitrarily vertex pairs, these construction methods arbitrary select one of them to find a Steiner tree. If selected paths share as many common edges as possible, the weight of the obtained Steiner tree becomes smaller than the sum of the shortest distances of the selected paths. On the other hand, if selected paths do not share common edges, the weight of the obtained Steiner tree equals the sum of the shortest distances of the selected paths. Thus, the performances of these construction methods are unstable for instances that have multiple shortest paths between arbitrary vertex pairs. A typical example of such instances is designing very-large-scale integration circuits. Circuits usually consist of vertical and horizontal lines on a plane. In this case, the graph becomes a grid, and a grid graph has multiple shortest paths between arbitrary vertex pairs. To improve the performance of these construction methods for such instances, we propose using network centrality [7] to distinguish which paths should be selected to find a small-weight Steiner tree.

Network centrality measures the importance of vertices and edges in a network. The words “graph” and “network” are usually used interchangeably. Network centrality is one of the major topics of complex network theory, which was born around 2000. However,

network centrality was proposed around 1950. Thus, various network centralities have been proposed over the past 70 years. We expected that important vertices and edges in a network play an important role in finding small-weight Steiner trees or there is network centrality to distinguish important vertices and edges to find small-weight Steiner trees. From this viewpoint, we proposed introducing network centrality to solve the Steiner tree problem in graphs. The popular construction methods to find Steiner trees use the shortest path between arbitrary vertex pairs. Thus, we want to enumerate all shortest paths between arbitrary vertex pairs, calculate the sum of the network centrality of vertices and edges included in each path, and select the one with the largest network centrality. Unfortunately, enumerating all shortest paths has a large computation cost. Thus, instead of enumerating all shortest paths, we select the shortest path that minimizes the weighted sum of the original edge weight and its network centrality. Note that important edges in a network have a large network centrality, and we want to include such edges as much as possible. However, the shortest path problem is a minimization problem. To overcome this contradiction, we use the reciprocal of the network centrality when calculating the weighted sum. The strength of each term is adjusted by a scaling parameter. By using this new edge weight, we can obtain the shortest path that may minimize the sum of the original edge weight and maximize the sum of its network centrality in reasonable time. Among the various network centralities, we selected the degree [8], eigenvector [9], closeness [10], and betweenness centralities [8] because they are very popular. Among them, only the betweenness centrality is defined for both vertices and edges. Thus, we use the betweenness centrality of vertices (vertex betweenness centrality) and edges (edge betweenness centrality) in this thesis.

The performance of exact algorithms is compared using the time and space complexity because exact algorithms return the same solution. However, there is no guarantee that approximation algorithms return the same solution. Thus, we must compare the performance from various aspects, not only in time and space complexity but also in the quality of returned solutions, stability of returned solutions, ease of implementation, scalability, etc. For ease of comparison between approximation methods, benchmark instances are used. We can directly compare the performances of algorithms by solving the same instances. As the library of benchmark instances of the Steiner tree problem in graphs, SteinLib has been maintained since 2001. Thus, we use benchmark instances published in SteinLib to evaluate the performances of algorithms. The most important points to evaluate the performances of algorithms are the quality of returned solutions and computation time. From this viewpoint, we use the average gap from the optimum solutions and CPU time to compare the performances of algorithms. In SteinLib, the objective function value of the optimum solution or known best solution is listed. Thus, we calculate the average gaps from them.

We investigated the performances of the eighteen combinations of three construction methods and five network centralities: SPH (without network centrality), SPH combined with degree centrality, SPH combined with eigenvector centrality, SPH combined with close-

ness centrality, SPH combined with vertex betweenness centrality, SPH combined with edge betweenness centrality, DNH (without network centrality), DNH combined with degree centrality, DNH combined with eigenvector centrality, DNH combined with closeness centrality, DNH combined with vertex betweenness centrality, DNH combined with edge betweenness centrality, ADH (without network centrality), ADH combined with degree centrality, ADH combined with eigenvector centrality, ADH combined with closeness centrality, ADH combined with vertex betweenness centrality, and ADH combined with edge betweenness centrality.

The results of numerical experiments indicate that using network centralities contributes to decreasing the objective function value for almost all instances. In particular, using the vertex and edge betweenness centralities showed the best performance when combined with any construction method. The reason why the betweenness centrality shows good performance is that it measures the flow in the graph. The minimum Steiner tree is expected to minimize the total flow between terminals. Thus, high betweenness centrality vertices and edges contribute to finding a small-weight Steiner tree. The results of using the vertex and edge betweenness centralities are different. This is because the value of edge betweenness centrality depends on the vertex betweenness centrality of the smaller vertex (because each edge connects two vertices). On the other hand, we use the average value of the vertex betweenness centralities of both vertices as the centrality of an edge when using the vertex betweenness centrality. Thus, the latter is larger than the former. This difference may cause different results. In contrast, using the closeness centrality shows poor performance among all five network centralities. This is because the closeness centrality ignores the structure of the graph. The closeness centrality is calculated as the sum of the shortest distance from a vertex to all other vertices. Benchmark instances that we used are random graphs; thus, the sum of shortest distances from a given vertex to all other vertices is similar for any given vertex. Therefore, the closeness centrality cannot distinguish important vertices and edges to find a small-weight Steiner tree.

One of the demerits of using network centrality is an increase in computation time. The computation costs of the eigenvector, closeness, and betweenness centralities of all vertices and edges are larger than those of SPH and DNH. The computation cost directly affects computation time. Thus, we compare the CPU time of the original (not using network centrality) and proposed (using network centrality) methods. The results of numerical experiments revealed that the proposed method successfully reduces the average gap by up to 40% from the original method when the CPU time is twice that of the original method for some instances using SPH and DNH. These instances have a common feature: the density of terminals is high (25%). Thus, the proposed method may be efficient for instances with a high density of terminals. Besides, for ADH, the CPU time of the proposed method is shorter than that of the original method for some instances unless the calculation of the network centrality phase is added. By comparing the CPU times of ADH, it is revealed that the CPU time of

ADH of the proposed method is shorter than that of the original ADH. The reason for this phenomenon is still unclear; however, using network centrality may affect the construction process of ADH.

The major findings of the construction method combined network centralities are as follows: (1) The use of network centrality, especially betweenness centrality, contributes to finding small-weight Steiner trees. (2) The computation time clearly increases when using network centrality. However, the use of network centrality reduces the average gap by up to 40% from the original methods for some benchmark instances within twice the CPU time of the original methods.

In Chapter 4, we try to apply the chaotic search, which is an efficient meta-heuristic that uses chaotic neurodynamics to escape from local minima to solve the Steiner tree problem in graphs. The chaotic search was first proposed by M. Hasegawa, T. Ikeguchi, and K. Aihara in 1997, to solve the traveling salesman problem [12]. The traveling salesman problem is also a classical and well-studied $\mathcal{NP}$-hard combinatorial optimization problem. The chaotic search combined with a popular local search method called the 2-opt shows better performance than the random search. After that, the chaotic search was applied to solve other $\mathcal{NP}$-hard combinatorial optimization problems, such as the quadratic assignment problem [16, 17, 18], vehicle routing problem [19, 20], packet routing problem [21, 22, 23, 24, 25, 26, 27, 28], and motif extraction problem [29, 30]. The chaotic search also shows good performance for the above problems.

In the chaotic search, selecting the next solution from the neighborhood of the current solution is determined by chaotic neurodynamics. The most important points for reproducing chaotic neurodynamics are (1) neurons fire when their output is over a threshold and (2) neurons do not fire again for a while after firing. The latter feature is called the refractory effect. Hasegawa *et al.* additionally introduced an adjustment mechanism for the firing rates using the history of the firing of all neurons because the firing rate directly affects the performance of the chaotic search. The firing rate reflects the balance between intensification and diversification of the search. In general, it is thought that meta-heuristics show the best performance when the balance between intensification and diversification of the search fits the instance. Here, intensification means focusing on searching regions around good solutions already found because it is expected that good solutions exist around other good solutions. On the other hand, diversification means focusing on jumping from one search region to other regions to avoid being trapped at local minima. Thus, a low firing rate corresponds to strong intensification and a high firing rate corresponds to strong diversification. Therefore, it is important to set an appropriate firing rate during the search. However, introducing such a sophisticated mechanism causes an increase in the number of parameters that must be tuned. Parameter tuning is a significant problem when using meta-heuristics. For ease of parameter tuning, we use a simple mechanism to adjust firing rates instead of a sophisticated one; we use the weighted sum of the outputs of neurons. If many neurons were fired at the previous

iteration, the term works to decreasing the internal state of neurons at the next iteration.

The chaotic search we used to solve the Steiner tree problem in graphs has six parameters: scaling parameter of the gain effect β, scaling parameter of the refractory effect α, decay factor of the refractory effect k, bias of the refractory effect θ, connection weight of neurons W, and steepness parameter of the sigmoidal function ϵ. Note that the α and β must be balanced. Thus, we can fix one of them and tune the other; we fix α to 1 and tune β. For ease of tuning β, we normalize the gain or the improvement of the local search by the maximum edge weight for the benchmark problem sets B and C and by the median of edge weights for the benchmark problem sets I080 and I160. This is because the expected value of the gain using the Steiner vertex neighborhood is at most the maximum edge weight. However, in the benchmark problem sets I080 and I160, the maximum edge weight depends on whether there is an edge between terminals. Thus, we use the median of edge weights instead of the maximum edge weight. We tune parameters by pre-numerical experiments. First, we set the parameters to arbitrary values. Then, we observe the time series of the objective function value of the current solution and firing patterns. If the range of the objective function value of the current solution is large and many neurons fire simultaneously, we regulate parameters to decrease the firing rate because these features indicate too strong diversification. We can decrease the firing rate by increasing β, decreasing k, and increasing θ. On the other hand, if the range of the objective function value of the current solution is small and few neurons fire simultaneously, we regulate parameters to increase the firing rate because these features indicate too strong intensification. We can increase the firing rate by decreasing β, increasing k, and decreasing θ.

If the refractory effect of the chaotic search becomes very strong and it is valid for a limited time, this effect is equivalent to the tabu effect of the tabu search. The tabu effect works by avoiding moving to solutions that were recently searched. From this viewpoint, the chaotic search can easily convert to the tabu search by setting the scaling parameter of the refractory effect α to ∞, introducing a validity period referring to the history of firing, and only the neuron with the largest internal state fires during one iteration. Therefore, we use this tabu search for comparison with the performance of the chaotic search. There are many aspects to evaluate the performance of approximation algorithms. We use the average gap from the optimum solutions, CPU time, and the number of optimum solutions found during the search. The last one was proposed by us. In real-world problems, the optimal solution is usually unclear. In such situations, a frequent hit of good solutions becomes one of the practical indicators of goodness of algorithms. Thus, we introduce this aspect to evaluate the performances of the chaotic search and tabu search.

The meta-heuristics require a termination condition. We set the upper limit of iterations to $5,000$ as a termination condition. In other words, the searching solution is terminated when each neuron has been updated $5,000$ times. Thus, we make the total number of generated neighborhood solutions the same. The reason why we use this termination condition is that

the bottleneck of computation cost lies in the generation of neighborhood solutions. Note that the chaotic search and tabu search employ different moving strategies: the immediate moving strategy and best moving strategy, respectively. In the immediate moving strategy, no more than $|V \setminus T|$ neurons fire during one iteration. On the other hand, in the best moving strategy, only one neuron fires during one iteration. Thus, the total number of moving solutions is different between them. This difference is due to the difference in CPU time.

However, the computation costs to update neurons of the chaotic search and tabu search are the same. There are two differences between the tabu search and chaotic search: how the refractory effect is decreased and how to decide the firing of neurons. First, we consider the refractory effect of the chaotic search and tabu search. In the tabu search, if a neuron fires, the refractory effect of this neuron becomes infinite. Then, this refractory effect is canceled after a predetermined number (parameter) of iterations. Thus, the computation cost of the refractory effect of a neuron of the tabu search is $\mathcal{O}(1)$. On the other hand, in the case of the chaotic search, the refractory effect is decreased exponentially. We can calculate it by keeping only the current refractory effect. Thus, the computation cost of the refractory effect of a neuron of the chaotic search is $\mathcal{O}(1)$. Secondly, we consider the firing of neurons. In the tabu search, only the neuron with the highest internal state is fired. Thus, the best moving strategy is employed. The internal state is the weighted sum of the improvement of the objective function value and refractory effect. On the other hand, in the chaotic search, neurons with larger internal states than the threshold value are fired. Thus, an immediate moving strategy is employed. In our numerical experiments, we set the terminating condition for neuron updates to $5,000$. This indicates that the total number of firings of the chaotic search is larger than that of the tabu search. However, the total number local searches applied is the same. Therefore, our numerical experiments are conducted fairly.

The results of numerical experiments revealed that the chaotic search and tabu search always find optimum solutions for benchmark problem set B. The computation time of the chaotic search is approximately twice that of the tabu search. This is because the number of moving solutions is different between them. However, the total number of generated neighborhood solutions is the same. In this sense, it is a fair comparison. On the other hand, for benchmark problem set C, the average gap of the chaotic search is larger than that of the tabu search. One of the reasons why the chaotic search shows poor performance for benchmark problem set C is that parameter tuning is very difficult. In our numerical experiments, we use the same parameter set for each benchmark problem. Thus, we use the same parameter set to solve all instances in benchmark problem set C. However, the average firing rate with a parameter set differs per instance. The average firing rate directly affects the performance of the chaotic search. Thus, we set parameters that can produce as many instances as possible with low gaps. If we set parameters per instance, the performance of the chaotic search may improve. For the benchmark problem sets I080 and I160, the chaotic

search finds smaller gap solutions than the tabu search. These instances disturb the pre-processing by removing edges if there is a smaller weight shortest path between both vertices of the edge. These results indicate that the chaotic search shows good performance for such instances.

We also analyzed the searching processes of the chaotic search and tabu search to understand why the chaotic search shows better performance than the tabu search for benchmark problem sets I080 and I160. We observed the time series of the objective function value and firing patterns. The chaotic search allows moving solutions at most $|V \setminus T|$ times during one iteration. Thus, we use the smallest objective function value among moved solutions during one iteration as the objective function value of that iteration. It is clarified that the chaotic search successfully finds optimum solutions more frequently than the tabu search. These results indicate that the solutions obtained by the chaotic search may be good for such instances.

The major findings of the improvement method using chaotic neurodynamics to solve the Steiner tree problem in graphs are as follows: (1) The chaotic search also shows good performance, but when it is combined with the local search, it returns not only feasible solutions but also infeasible solutions. (2) The chaotic search shows good performance for benchmark instances that aim to disturb the pre-processing techniques.

In this thesis, we tried two approaches to solve the Steiner tree problem in graphs. One is the construction method combined with network centrality and the other is the improvement method using chaotic neurodynamics. Both construction and improvement methods are closely related, and the development of both methods is strongly desired.

First, we highlighted the demerits of three popular construction methods for finding a Steiner tree: these algorithms sometimes find a large-weight Steiner tree because they ignore the existence of multiple shortest paths between arbitrary vertex pairs. To overcome this issue, we proposed using the network centrality to distinguish the shortest path that maximizes the sum of the network centrality of vertices and edges included in a path. These shortest paths are expected to cause shared the common edges. We selected the degree, eigenvector, closeness, and betweenness centralities combined with SPH, DNH, and ADH. The results of numerical experiments indicate that using betweenness centrality contributes to finding small-weight Steiner trees. One of the demerits of our proposed method is an increase in computation time because our proposed method must calculate the network centrality of all vertices and edges. However, the increase in the CPU time is relatively small and still practical.

Next, we applied the chaotic search, which is a meta-heuristic using chaotic neurodynamics to solve the Steiner tree problem in graphs. The chaotic search is an efficient meta-heuristic, and has been applied to solve various $\mathcal{NP}$-hard combinatorial optimization problems; however, it has not been applied to solve the Steiner tree problem in graphs before. Thus, we used the chaotic search to solve the Steiner tree problem in graphs. The chaotic

search was combined with local search methods that always generate feasible solutions. Unfortunately, the local search method to solve the Steiner tree problem in graphs generates not only feasible solutions but also infeasible solutions. Thus, we ignore neurons that correspond to infeasible solutions at that time. The results of numerical experiments revealed that the chaotic search also shows good performance to solve the Steiner tree problem in graphs. In particular, the chaotic search shows significant performances for instances that disturb pre-processing.

The major findings of this thesis can be summarized as follows: (1) We proposed using network centrality to solve the Steiner tree problem in graphs. We experimentally showed that using network centrality contributes to finding small-weight Steiner trees. In particular, betweenness centrality showed the best performance. Besides, the decrease in the gap becomes large when the increase in the computation time is approximately twice that of the original (not using network centrality) construction methods. These results indicate that using network centrality will also show good performance for other types of Steiner tree problems in graphs, such as the constrained Steiner tree problem and prize-collecting Steiner tree problem. (2) We investigated the performance of the chaotic search to solve the Steiner tree problem in graphs. The chaotic search successfully found smaller weight Steiner trees than the tabu search. The reason why the chaotic search shows good performance is that it frequently hits optimum solutions, especially for instances made to disturb pre-processing.

5.2 Future work

In Chapter 3, we mentioned that ADH combined with network centralities decreases the computation time for some instances. Our future works will clarify the reason why the computation time becomes short in such cases and apply this method to solve other types of Steiner tree problems, such as the constrained Steiner tree problem and prize-collecting Steiner tree problem.

In Chapter 4, we also mentioned a manual parameter tuning method for realizing the effective chaotic search introduced in pre-numerical experiments and showed that a key factor to realize such an effective search is a balance between search intensification and diversification. This balance could be realized by setting an appropriate firing rate and firing pattern for the chaotic neurons. Thus, it is an important issue to develop an automatic algorithm for controlling these parameters by using the firing rates and firing patterns of the chaotic neurons to solve the Steiner tree problem in graphs effectively.

Acknowledgments

The study presented in this dissertation has been carried out at Tokyo University of Science under the direction and the guidance of Professor Tohru Ikeguchi of Tokyo University of Science during 2017–2020. Therefore, I would like to take this opportunity to thank all those who helped me and provided me encouragements with accomplishing this work.

First of all, I would like to express my heartfelt gratitude to my supervisor Professor Tohru Ikeguchi. He welcomed me when I became a member of Ikeguchi Laboratory from the doctoral program. Based on my dream of becoming an academic staff in the future, he enthusiastically taught me a lot of things required to people who live as an academic staff such as the attitude as a researcher, how to conduct research, how to teach juniors, how to run a laboratory, how to obtain research funds, how to conduct collabolation, the attitude as an educator, the method of teaching students through lectures, and so on. I was able to become a fellow of the Japan Society for the Promotion of Science and received the IEICE Academic Encouragement Award in my doctoral course. I would like to show the deepest appreciation to Professor Tohru Ikeguchi. As of the time of writing this acknowledgment, more than 850 daily reports have been exchanged six days a week for three years. There were days when the research progressed and days when the research did not progress, and I was worried about what to write in the daily report of the day I did not progress. I was greatly encouraged by the reply from Professor Tohru Ikeguchi. Besides, he often conducted a lunch meeting with me. It was a valuable time to talk about various things. My deepest appreciation was goes to Professor Tohru Ikeguchi.

I would like to thank Associate professor Takayuki Kimura of Nippon Institute of Technology, an advisor in the undergraduate course and master course, taught me the importance, depth, and fun of combinatorial optimization from my undergraduate days to the present. Even after I left Kimura Laboratory and became a member of Ikeguchi Laboratory, I had many discussions about my research with Associate professor Takayuki Kimura. I often consult with him when I only gathered negative experimental data. He warmly encouraged me and broaden my vision. He examined the experimental data from various angles and found a better perspective. I would like to thank Associate professor Takayuki Kimura.

I would like to thank Professor Kenya Jin'no of Tokyo City University, an official advisor in the master's course. He always pointed out the weak points of my research. When there

is a time limit for questions and answers such as in academic society, we had discussions until after the session. Discussions with him always guided my research in a natural and fair direction. I would like to thank Professor Kenya Jin'no.

I wish to express my appreciation to Professor Yukinobu Taniguchi, Professor Hitoshi Watanabe, Professor Hiroyuki Yashima, Professor Takako Akakura, Associate professor Yoshiko Ikebe, and Professor Mikio Hasegawa for the fair and significant examination as my doctorate dissertation committee at Tokyo University of Science.

Moreover, I am grateful to Professor Yoshihiko Horio of Tohoku University, Professor Masaharu Adachi of Tokyo Denki University, Professor Hiroaki Kurokawa of Tokyo University of Technology, Professor Hiroo Sekiya of Chiba University, Associate professor Kantaro Fujiwara of the University of Tokyo, Associate professor Takafumi Matsuura of Nippon Institute of Technology, Lecturer Hideyuki Kato of Oita University, Assistant professor Yutaka Shimada of Saitama University, Assistant professor Kaori Kuroda of Toyo University, Assistant professor Nina Sviridova of Tokyo University of Science, Lecturer Ryota Nomura of Kagoshima Immaculate Heart University, Assistant professor Yoshikazu Yamanaka of Utsunomiya University, and Assistant professor Yoshitaka Itoh of Hokkaido University of Science for continuing advice and encouragement.

I wish to thank all members of Ikeguchi Laboratory, Kimura Laboratory, and participants of the nonlinear workshop for a warm and interesting research environment that I have enjoyed very much. This work was supported by Japan Society of the Promotion Science KAKENHI Grant Number JP18J10671. Finally, I owe my deepest gratitude to my parents who helped and encouraged me greatly and gave all kinds of assistance to facilitate to reach my educational goal and make this study a great success under the most comfortable atmosphere.

Bibliography

[1] 久保幹雄, 組合せ最適化とアルゴリズム. 共立出版株式会社, 2000.

[2] A. B. Kahng and G. Robins, *On optimal interconnections for VLSI*. Springer Science & Business Media, 1994.

[3] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos, "Multicast routing for multimedia communication," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 286–292, 1993.

[4] E. K. Donkoh, S. K. Amponsah, and K. F. Darkwah, "Optimal pipeline connection for the west african gas pipeline project," *Research Journal of Applied Sciences, Engineering and Technology*, vol. 3, no. 2, pp. 67–73, 2011.

[5] N. L. Biggs, E. K. Lloyd, and R. J. Wilson, *Graph Theory, 1736-1936*. Oxford University Press, 1986.

[6] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, pp. 440–442, 1998.

[7] S. Wasserman and K. Faust, *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[8] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social Networks*, vol. 1, no. 3, pp. 215–239, 1978.

[9] P. Bonacich, "Factoring and weighting approaches to status scores and clique identification," *The Journal of Mathematical Sociology*, vol. 2, no. 1, pp. 113–120, 1972.

[10] A. Bavelas, "Communication patterns in task-oriented groups," *The Journal of the Acoustical Society of America*, vol. 22, no. 6, pp. 725–730, 1950.

[11] K. Aihara, T. Takabe, and M. Toyoda, "Chaotic neural networks," *Physics Letters A*, vol. 144, no. 6–7, pp. 333–340, 1990.

[12] M. Hasegawa, T. Ikeguchi, and K. Aihara, "Combination of chaotic neurodynamics with the 2-opt algorithm to solve traveling salesman problems," *Physical Review Letters*, vol. 79, no. 12, pp. 2344–2347, 1997.

[13] M. Hasegawa, T. Ikeguchi, and K. Aihara, "Solving large scale traveling salesman problems by chaotic neurodynamics," *Neural Networks*, vol. 15, no. 2, pp. 271–283, 2002.

[14] S. Motohashi, T. Matsuura, T. Ikeguchi, and K. Aihara, "The Lin-Kernighan algorithm driven by chaotic neurodynamics for large scale traveling salesman problems," in *Artificial Neural Networks – ICANN 2009* (C. Alippi, M. Polycarpou, C. Panayiotou, and G. Ellinas, eds.), pp. 563–572, Springer Berlin Heidelberg, 2009.

[15] S. Motohashi, T. Matsuura, and T. Ikeguchi, "Chaotic search method using the Lin-Kernighan algorithm for traveling salesman problems," in *Proceedings of International Symposium on Nonlinear Theory and its Applications*, pp. 144–147, 2008.

[16] M. Hasegawa, T. Ikeguchi, and K. Aihara, "Exponential and chaotic neurodynamical tabu searches for quadratic assignment problems," *Control and Cybernetics*, vol. 29, no. 3, pp. 773–788, 2000.

[17] M. Hasegawa, T. Ikeguchi, K. Aihara, and K. Itoh, "A novel chaotic search for quadratic assignment problems," *European Journal of Operational Research*, vol. 139, no. 3, pp. 543–556, 2002.

[18] H. Ohnishi, Y. Shimada, K. Fujiwara, and T. Ikeguchi, "Chaotic neurodynamical search with small number of neurons for solving QAP," *Nonlinear Theory and Its Applications, IEICE*, vol. 8, no. 3, pp. 255–265, 2017.

[19] T. Hoshino, T. Kimura, and T. Ikeguchi, "Two simple local searches controlled by chaotic dynamics for vehicle routing problems with time windows," in *Proceedings of Metaheuristic International Conference*, 2007.

[20] 星野聖, 木村貴幸, and 池口徹, "時間枠制約付き配送計画問題に対するカオスダイナミックスを用いたメタヒューリスティック解法," 電子情報通信学会論文誌 A, vol. J90–A, no. 5, pp. 431–441, 2007.

[21] T. Kimura and T. Ikeguchi, "An optimum strategy for dynamic and stochastic packet routing problems by chaotic neurodynamics," *Integrated Computer-Aided Engineering*, vol. 14, no. 4, pp. 307–322, 2007.

[22] T. Kimura and T. Ikeguchi, "A new algorithm for packet routing problems using chaotic neurodynamics and its surrogate analysis," *Neural Computing and Applications*, vol. 16, no. 6, pp. 519–526, 2007.

[23] T. Kimura, H. Nakajima, and T. Ikeguchi, "A packet routing method for complex networks by a stochastic neural network," *Physica A: Statistical Mechanics and its Applications*, vol. 376, pp. 658–672, 2007.

[24] T. Kimura, T. Hiraguri, and T. Ikeguchi, "An effective routing algorithm with chaotic neurodynamics for optimizing communication networks," *American Journal of Operations Research*, vol. 2, no. 3, pp. 348–356, 2012.

[25] K. Kimura, T. Kimura, T. Hiraguri, and K. Jin'no, "Effective method for wind and solar power grid systems based on recurrent neural networks," *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol. 18, no. 6, pp. 1034–1043, 2014.

[26] T. Kimura, T. Takamizawa, K. Kimura, and K. Jin'no, "Neural-based routing strategy with transmission information for complex communication networks," *Nonlinear Theory and Its Applications, IEICE*, vol. 6, no. 2, pp. 263–274, 2015.

[27] T. Kimura, T. Takamizawa, and T. Matsuura, "Neural-based routing method for alleviating congestion in complex networks," *American Journal of Operations Research*, vol. 6, no. 4, pp. 343–354, 2016.

[28] Y. Morita and T. Kimura, "An improved routing algorithm using chaotic neurodynamics for packet routing problems," *Nonlinear Theory and Its Applications, IEICE*, vol. 9, no. 1, pp. 95–106, 2018.

[29] T. Matsuura and T. Ikeguchi, "Chaotic motif sampler: detecting motifs from biological sequences by using chaotic neurodynamics," *Nonlinear Theory and Its Applications, IEICE*, vol. 1, no. 1, pp. 207–220, 2010.

[30] T. Matsuura, K. Numata, and T. Ikeguchi, "Searching characteristics of chaotic neurodynamics for combinatorial optimization," *Nonlinear Theory and Its Applications, IEICE*, vol. 3, no. 4, pp. 573–585, 2012.

[31] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[32] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, 1956.

[33] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations, The IBM Research Symposia Series* (R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, eds.), pp. 85–103, Boston, MA: Springer US, 1972.

[34] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*. Courier Corporation, 2001.

[35] H. Takahashi and A. Matsuyama, "An approximate solution for the Steiner problem in graphs," *Mathematica Japonica*, vol. 24, no. 6, pp. 573–577, 1990.

[36] L. Kou, G. Markowsky, and L. Berman, "A fast algorithm for Steiner trees," *Acta Informatica*, vol. 15, no. 2, pp. 141–145, 1981.

[37] V. J. Rayward-Smith, "The computation of nearly minimal Steiner trees in graphs," *International Journal of Mathematical Education in Science and Technology*, vol. 14, no. 1, pp. 15–23, 1983.

[38] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM computing surveys*, vol. 35, no. 3, pp. 268–308, 2003.

[39] S. Kirkpatrick, C. D. Gelatt, Jr, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[40] F. Glover, "Tabu search—part I," *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190–206, 1989.

[41] F. Glover, "Tabu search—part II," *ORSA Journal on Computing*, vol. 2, no. 1, pp. 4–32, 1990.

[42] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, no. 2, pp. 65–85, 1994.

[43] K. Helsgaun, "An effective implementation of the Lin–Kernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, no. 1, pp. 106–130, 2000.

[44] É. D. Taillard and K. Helsgaun, "POPMUSIC for the travelling salesman problem," *European Journal of Operational Research*, vol. 272, no. 2, pp. 420–429, 2019.

[45] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, vol. 69, no. 2, p. 026113, 2004.

[46] M. Fujita, T. Kimura, and T. Ikeguchi, "Solving the Steiner tree problem in graphs by chaotic search," *Nonlinear Theory and Its Applications, IEICE*, vol. 11, no. 1, pp. 90–108, 2020.

[47] E. N. Gilbert and H. O. Pollak, "Steiner minimal trees," *SIAM Journal on Applied Mathematics*, vol. 16, no. 1, pp. 1–29, 1968.

[48] M. R. Garey, R. L. Graham, and D. S. Johnson, "The complexity of computing Steiner minimal trees," *SIAM Journal on Applied Mathematics*, vol. 32, no. 4, pp. 835–859, 1977.

[49] L. R. Foulds and R. L. Graham, "The Steiner problem in phylogeny is NP-complete," *Advances in Applied Mathematics*, vol. 3, no. 1, pp. 43–49, 1982.

[50] A. Z. Zelikovsky, "An 11/6-approximation algorithm for the network Steiner problem," *Algorithmica*, vol. 9, no. 5, pp. 463–470, 1993.

[51] P. Berman and V. Ramaiyer, "Improved approximations for the Steiner tree problem," *Journal of Algorithms*, vol. 17, no. 3, pp. 381–408, 1994.

[52] M. Karpinski and A. Zelikovsky, "New approximation algorithms for the Steiner tree problems," *Journal of Combinatorial Optimization*, vol. 1, no. 1, pp. 47–65, 1997.

[53] S. Hougardy and H. J. Prömel, "A 1.598 approximation algorithm for the Steiner problem in graphs," in *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'99, pp. 448–453, Society for Industrial and Applied Mathematics, 1999.

[54] G. Robins and A. Zelikovsky, "Improved Steiner tree approximation in graphs," in *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'00, pp. 770–779, Society for Industrial and Applied Mathematics, 2000.

[55] K. A. Dowsland, "Hill-climbing, simulated annealing and the Steiner problem in graphs," *Engineering Optimization*, vol. 17, no. 1–2, pp. 91–107, 1991.

[56] S. L. Martins, M. G. C. Resende, C. C. Ribeiro, and P. M. Pardalos, "A parallel grasp for the Steiner tree problem in graphs using a hybrid local search strategy," *Journal of Global Optimization*, vol. 17, no. 1, pp. 267–283, 2000.

[57] T. A. Feo and M. G. C. Resende, "A probabilistic heuristic for a computationally difficult set covering problem," *Operations Research Letters*, vol. 8, no. 2, pp. 67–71, 1989.

[58] L. Luyet, S. Varone, and N. Zufferey, "An ant algorithm for the Steiner tree problem in graphs," in *Applications of Evolutionary Computing* (M. Giacobini, ed.), pp. 42–51, Springer Berlin Heidelberg, 2007.

[59] A. Kapsalis, V. J. Raywad-Smith, and G. D. Smith, "Solving the graphical Steiner tree problem using genetic algorithms," *Journal of Operational Research Society*, vol. 44, no. 4, pp. 397–406, 1993.

[60] J. André, S. Auray, J. Brac, D. De Wolf, G. Maisonnier, M.-M. Ould-Sidi, and A. Simonnet, "Design and dimensioning of hydrogen transmission pipeline networks," *European Journ al of Operational Research*, vol. 229, no. 1, pp. 239–251, 2013.

[61] T. Koch, A. Martin, and S. Voß, "SteinLib: An updated library on Steiner tree problems in graphs," in *Steiner Trees in Industry* (X. Z. Cheng and D.-Z. Du, eds.), pp. 285–325, Boston, MA: Springer US, 2001.

[62] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM*, vol. 34, no. 3, pp. 596–615, 1987.

[63] U. Brandes, "A faster algorithm for betweenness centrality," *The Journal of Mathematical Sociology*, vol. 25, no. 2, pp. 163–177, 2001.

[64] H. Esbensen, "Computing near-optimal solutions to the Steiner problem in a graph using a genetic algorithm," *Networks*, vol. 26, no. 4, pp. 173–185, 1995.

[65] M. Gendreau, J.-F. Larochelle, and B. Sansó, "A tabu search heuristic for the Steiner tree problem," *Networks*, vol. 34, no. 2, pp. 162–172, 1999.

List of Papers

"Referred papers"

1. M. Fujita, T. Kimura, and K. Jin'no, "An Effective Construction Algorithm for the Steiner Tree Problem Based on Edge Betweenness," *Journal of Signal Processing, The Research Institute of Signal Processing,* vol. 20, issue 4, pp. 145–148, 2016.
2. M. Fujita, T. Kimura, and T. Ikeguchi, "Solving the Steiner tree problem in graphs by chaotic search," *Nonlinear Theory and Its Applications, IEICE,* vol. 11, no. 1, pp. 90–108, 2020.

"International conferences (referred)"

3. M. Fujita, T. Kimura, and K. Jin'no, "An Improved KMB Algorithm Using Edge Betweenness for Steiner Tree Problems," in *Proceedings of the 4th Korea-Japan Joint Workshop on Complex Communication Sciences (KJCCS 2015),* J11, Nozawa onsen, Japan, Jan. 2016.
4. M. Fujita, T. Kimura, and K. Jin'no, "An Effective Construction Algorithm for Steiner Tree Problem Based on Edge Betweenness," in *Proceedings of the 2016 RISP International Workshop on Nonlinear Circuits, Communications and Signal Processing (NCSP 2016),* pp. 530–533, Honolulu, USA, Mar. 2016.
5. M. Fujita, T. Kimura, and K. Jin'no, "A Construction Method for Steiner Tree Problem Using Betweenness Centrality," in *Proceedings of the 2016 International Symposium on Nonlinear Theory and its Applications (NOLTA 2016),* pp. 124–127, Yugawara, Japan, Nov. 2016.
6. M. Fujita, T. Kimura, and T. Ikeguchi, "Solving the Steiner Tree Problem in Graphs by Chaotic Neural Network using Key Path Neighborhood," in *Proceedings of the 12th Metaheuristics International Conference (MIC 2017),* pp. 834–836, Barcelona, Spain, Jul. 2017.
7. M. Fujita, T. Kimura, Kantaro Fujiwara and T. Ikeguchi, "Evaluation of the performance of the chaotic neural network for solving the Steiner tree problem in graphs

with incidence costs," in *Proceedings of the 2017 International Symposium on Nonlinear Theory and its Applications (NOLTA 2017),* pp. 712–715, Cancun, Mexico, Dec. 2017.

8. M. Fujita, T. Kimura, Kantaro Fujiwara and T. Ikeguchi, "Solving the Steiner Tree Problem in Graphs Using the Key-Path Based Neighborhood with the kth Shortest Path," in *Proceedings of the 2018 International Symposium on Nonlinear Theory and its Applications (NOLTA 2018),* pp. 61–64, Tarragona, Spain, Sep. 2018.
9. M. Fujita, T. Kimura, and T. Ikeguchi, "The Relationship Between Average Firing Rates and Performance of the Chaotic Search for Solving the Steiner Tree Problem in Graphs," in *Proceedings of the 2019 International Symposium on Nonlinear Theory and its Applications (NOLTA 2019),* pp. 505–508, Kuala Lumpur, Malaysia, Dec. 2019.

"Technical reports"

10. 藤田実沙, 木村貴幸, 神野健哉, "タブーサーチを用いたシュタイナー木問題の解法," 電子情報通信学会技術研究報告, vol. 115, no. 14, pp. 49–52, 香川県社会福祉総合センター, 2015 年 4 月.
11. 藤田実沙, 木村貴幸, 神野健哉, "媒介中心性を考慮したシュタイナー木構築法," 情報処理学会研究報告, vol. 2016–AL–158, no. 20, 石川県教育会館, 2016 年 6 月.
12. 藤田実沙, 木村貴幸, 池口徹, "異なる不応性を有するニューラルネットワークによるグラフ的シュタイナー木問題の解探索性能の比較," 電子情報通信学会技術研究報告, vol. 118, no. 243, NLP2018–81, pp. 51–56, 東北大学 青葉山キャンパス, 2018 年 10 月.
13. 藤田実沙, 木村貴幸, 池口徹, "ネットワーク中心性を使用した Shortest Path Heuristic," 電子情報通信学会技術研究報告, vol. 119, no. 19, NLP2019–8, pp. 41–46, ホルトホール大分, 2019 年 5 月.

"Domestic conferences"

14. 藤田実沙, 木村貴幸, 神野健哉, "Edge Betweenness を考慮したシュタイナー木構築法," *2016* 年 電子情報通信学会 *NOLTA* ソサイエティ大会講演論文集, B–9, 東京理科大学 葛飾キャンパス, 2016 年 6 月.
15. 藤田実沙, 木村貴幸, 神野健哉, "近接中心性を考慮したシュタイナー木構築法," *2016* 年 電子情報通信学会 ソサイエティ大会, N–1–6, 北海道大学 札幌キャンパス, 2016 年 9 月.
16. 藤田実沙, 木村貴幸, 藤原寛太郎, 池口徹, "キーパス近傍に基づいたカオスニューラルネットワークによるグラフ的シュタイナー木問題の解法," *2017* 年 電子情報通信学会 *NOLTA* ソサイエティ大会講演論文集, NLS-10, 中京大学 名古屋キャンパス, 2017 年 6 月.
17. 藤田 実沙, 木村 貴幸, 藤原 寛太郎, 池口 徹, "頂点に基づく局所探索法を使用したカオスニューラルネットワークによるグラフ的シュタイナー木問題の解法," *2017* 年 電子情

報通信学会 ソサイエティ大会, N–1–13, 東京都市大学 世田谷キャンパス, 2017 年 9 月.
18. 藤田実沙, 木村貴幸, 藤原 寛太郎, 池口徹, “K 番目の最短経路を使用したグラフ的シュタイナー木構築法,” *2018* 年 電子情報通信学会 総合大会, N–1–20, 東京電機大学, 2018 年 3 月.
19. 藤田実沙, 木村貴幸, 藤原寛太郎, 池口徹, “グラフ的シュタイナー木問題に対する複数の最短経路を使用した局所探索法,” *2018* 年 電子情報通信学会 *NOLTA* ソサイエティ大会講演論文集, A–4, 京都テルサ, 2018 年 6 月.
20. 藤田実沙, 木村貴幸, 池口徹, “グラフ的シュタイナー木問題に対するタブーサーチとカオスサーチの探索の多様性について,” *2019* 年 電子情報通信学会 総合大会, ANS–1–3, 早稲田大学 西早稲田キャンパス, 2019 年 3 月.
21. 藤田実沙, 木村貴幸, 池口徹, “枝媒介中心性を使用した Shortest Path Heuristic,” *2019* 年 電子情報通信学会 *NOLTA* ソサイエティ大会講演論文集, A–17, アオーレ長岡, 2019 年 6 月.