FAKULTÄT FÜR !NFORMATIK

Heuristic methods for solving two Generalized Network Problems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Anna Pagacz
Matrikelnummer 0426755

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer: Univ.-Prof. Dr. Günther R. Raidl
Mitwirkung: Univ.-Ass. Dr. Bin Hu

Wien, 22.02.2010 ______________________ ______________________
(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Erklärung zur Verfassung der Arbeit

Anna Pagacz
Nad Wilkowka 30
43-365 Wilkowice
Polen

„Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.“

Wien, 08.02.2010

Acknowledgments

First of all I want to thank my advisors Prof. Günther Raidl and Dr. Bin Hu for supervising my master thesis. I would like to especially thank Bin for all his patience and for providing so many interesting ideas and being a great help during the last months. My special thanks also belongs to my family, especially to my parents for believing in me and for all their support during the last years. Last but not least I want to thank Przemek for the great time we had together as well as for all his understanding and his support during the last years.

Zusammenfassung

Diese Arbeit setzt sich mit zwei kombinatorischen Optimierungsproblemen auseinander: das Problem des generalisierten minimalen knotenzweifachzusammenhängenden Netzwerks (GMVBCNP) und das Problem des generalisierten minimalen Spannbaums mit Gradbeschränkung (d-GMSTP). Beide Optimierungsprobleme sind NP-vollständig. Gegeben sind Graphen, deren Knoten in Cluster unterteilt sind. Das Ziel besteht jeweils darin, einen Teilgraphen mit minimalen Kosten zu finden, der genau einen Knoten von jedem Cluster verbindet und andere Zusatzbedingungen berücksichtigt. Beim d-GMSTP ist die Zusatzbedingung die Gradbeschränkung der Knoten. In der Praxis findet sich diese Problemstellung in der Telekommunikation wieder, wo Netzwerkknoten in mehrere Cluster unterteilt sind und auf Basis einer Baumarchitektur miteinander verbunden sind. Von jedem Cluster wird genau ein Knoten zum Rückgrat verbunden und durch die Gradbeschränkung wird die Transferqualität gewährleistet. Das GMVBCNP hingegen wird bei fehlertoleranten Backbone-Netzen angewendet. Um sicherzustellen, dass durch den Ausfall einer einzelnen Komponente andere Dienste nicht beeinflusst werden, müssen die Verbindungen redundant sein. Diese Arbeit stellt zwei Lösungsansätze für das d-GMSTP vor. Ein Ansatz basiert auf variable Nachbarschaftssuche (VNS), bei der verschiedene Arten von Nachbarschaftsstrukturen komplementär arbeiten und dadurch die Effizienz bei der Zusammenarbeit maximiert wird. Ein anderer Ansatz basiert auf einen memetischen Algorithmus (MA). Das GMVBCNP wird in dieser Arbeit ebenfalls mit einem memetischen Algorithmus (MA) gelöst. Dabei werden für die Zusammensetzung der Knoten zwei verschiedene Ansätze betrachtet. AuSSerdem werden mit Hilfe von Graph-Reduzierungstechniken, die den Suchraum signifikant verkleinern, lokale Verbesserungen erzielt. Beide Problemstellungen wurden auf euklidischen Instanzen mit bis zu 442 Knoten getestet.

Abstract

This thesis examines two combinatorial optimization problems: the Generalized Degree Constrained Minimum Spanning Tree Problem (d-GMSTP) and the Generalized Minimum Vertex Bi-connected Network Problem (GMVBCNP). Both problems are NP- hard. Given a clustered graph where nodes are partitioned into clusters, the goal is to find a minimal cost subgraph containing exactly one node from each cluster and satisfying other constraints. For the d-GMSTP the subgraph has to fulfill degree constraint. It plays an important role in telecommunication areas where network nodes are divided into clusters and they need to be connected via tree architecture using exactly one node per cluster and satisfying degree constraint for transfer quality. The GMVBCNP can be applied to the design of survivable backbone networks that should be fault tolerant to the single component outage. In order to ensure that the failure of a single service vertex would not lead to disconnection of other services, redundant connections need to be created. For solving the d-GMSTP two approaches are proposed: Variable Neighborhood Search (VNS) which uses different types of neighborhoods, which work in complementary ways to maximize the collaboration efficiency and a Memetic Algorithm (MA) involving local improvement. For solving the GMVBCNP a Memetic Algorithm (MA) is proposed. Two different population management approaches are considered, as well as local improvement involving graph reduction technique that reduces the search space significantly. Both problems are tested on Euclidean instances with up to 442 nodes.

Contents

List of Algorithms

List of Figures

1 Introduction

This thesis attacks two optimization problems: **Generalized Degree Constrained Minimum Spanning Tree Problem (d-GMSTP)** and **Generalized Minimum Vertex Bi-connected Network Problem (GMVBCNP)**. Both problems are located in combinatorial optimization areas and are classified as NP- hard. The solution of both problems can be applied in the real world, in the area of network design especially when one considers the design of a large-capacity backbone network connecting many individual networks. Generalized Degree Constrained Minimum Spanning Tree problem plays an important role in telecommunication area where network nodes are divided into clusters and they need to be connected via a tree architecture using exactly one node per cluster, however degree constraint must not be violated. Generalized Minimum Vertex Bi-connected Network Problem can be applied in a design of survivable backbone networks that should be fault tolerant to the single component outage.

The **Generalized Degree Constrained Minimum Spanning Tree (d-GMST)** problem is an extension of the classical Minimum Spanning Tree Problem, however the degree constraint is considered as well. In this problem the minimum spanning tree is searched and none of its vertices has degree grater than $d \geq 2$. The problem is defined as follow.

Consider an undirected, weighted complete graph $G = (V, E, c)$ with node set V, edge set E, edge cost function $c : E \rightarrow \mathbb{R}^{+}$. The node set V is partitioned into r pairwise disjointed clusters V_1,V_2, ...,V_r containing $n_{1,}n_{2,}...n_r$ nodes respectively. A spanning tree of a graph is a cycle- free subgraph connecting all nodes. A solution to the d-GMST problem defined on G is a graph $S = (P, T)$ with $P = \{p_1, p_2, \ldots, p_r\} \subseteq V$ containing exactly one node from each cluster, $p_i \in V_i$ for all $i = 1, 2, .., r$ and $T \subseteq P \times P \subseteq E$ being the tree spanning nodes P. Lets assign numerical costs $c_{u,v} \geq 0$ to each edge $(u, v) \in T$ then the Generalized Minimum Spanning Tree is a spanning tree with minimum total edge cost

$$C(T) = \sum_{(u,v)\in T} c(u, v).$$

In the degree constrained d-GMST problem an additional constraint called degree $deg(u)$ of every vertex $u \in P$ is considered. The degree is the number $deg(u) \leq d$, $\forall u \in P$, where d is a given upper bound of edges adjacent to u. Thus the **Generalized Degree Constrained Minimum Spanning Tree** is the tree that fulfills the degree constraint and simultaneously minimizes the total edge cost. Figure 3.1 shows an example for a solution to d-GMST.

Similar to the Generalized Minimum Spanning Tree problem, the **Generalized Minimum Vertex Bi-connected Network Problem (GMVBCNP)** is an extension of the classical Minimum Vertex Bi-connectivity Network Problem. Nowadays network's reliability and survivability are very

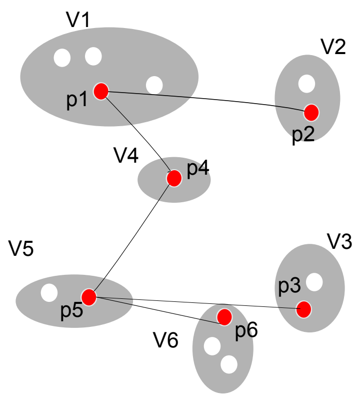


Figure 1.1: Example for d-GMST problem, where degree $d = 3$

crucial not only in telecommunication or IT industry, but also in many other industrial areas and it is not acceptable that the failure of a single service vertex could lead to disconnection of other vertices. Therefore a redundant connection needs to be created in order to provide alternative connections between the rest of service vertices. The graph theory describes this type of network robustness by means of vertex connectivity. A $k-connected$ network, $k > 2$, is said to be survivable because currently the new technologies provide good reliable solutions. Therefore the probability of a second failure before the first one is repaired can be kept very small. Thus this thesis focuses on the case of reliable networks which are 2-connected.

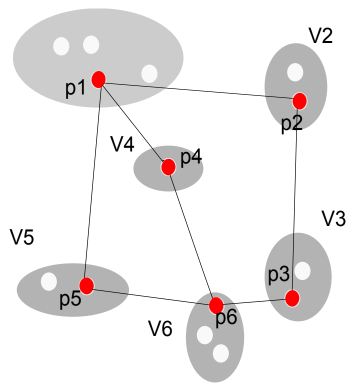


Figure 1.2: Example for a solution to GMVBCNP.

To define **GMVBCNP** we can link directly to Generalized Minimum Spanning Tree Problem (GMSTP) by requiring a vertex bi-connected graph, that includes cycles, instead of having a spanning tree. Consider an undirected, weighted, complete graph $G = (V, E, c)$ with node set V, edge set E, edge cost function $c : E \rightarrow \mathbb{R}^+$. The node set V is partitioned into r pairwise disjoined clusters V_1,V_2, ...,V_r containing n_1,n_2,...n_r, nodes respectively. The vertex bi-connected graph is a subgraph that connects all nodes and the failure of a single vertex will not disconnect the graph.

A solution to GMVBCNP defined on G is a subgraph $S = (P, T), P = \{p_1, .., p_n\} \subseteq V$ connecting exactly one node from each cluster, i.e. $p_i \in V_i, \forall i = 1, ..r$ and containing no *cut nodes*. A cut node is a node whose removal would disconnect a graph. Let us assign numerical costs $c_{u,v} \geq 0$ to each edge $(u, v) \in T$, then cost of such a vertex bi-connected graph are its total edge costs i.e.

$$C(T) = \sum_{(u,v) \in T} c(u, v),$$

and the objective is to identify a feasible solution with minimum costs. Figure 1.2 shows an example for a solution to GMVBCNP.

1.1 Graph theory

Undirected graph

This section describes an undirected graph. Please consider that the definitions given here can be different from some in the literature, but mostly the differences are slight. The aim of presenting this theory here is to establish the notation and to introduce the terms used in this work.

An undirected graph G is a pair (V, E), where V is a finite set and E is a binary relation in V. The set V is called the vertex set of G and its elements are called vertices. In an undirected graph $G = (V, E)$, the edge set E consists of unordered pairs of vertices. That is, an edge is a set $\{u, v\}$, where $u, v \in V$ and $u \neq v$. The notation (u, v) is used for an edge; the notations (u, v) and (v, u) are considered to be the same edge. In an undirected graph self-loops are forbidden, and so every edge consists of exactly two distinct vertices. If (u, v) is an edge in an undirected graph $G = (V, E)$, we say that (u, v) is incident to vertex u and v and vertex v is adjacent to vertex u. When the graph is undirected, the adjacency relation is symmetric. The degree deg of vertex in an undirected graph is the number of edges incident to it. A path of length k from a vertex u to a vertex u in a graph $G = (V, E)$ is a sequence $\langle v_o, v_1, \ldots, v_r \rangle$ of vertices such that $u = v_o$, $u' = v_r$ and $(v_{i-1}, v_i) \in E$ for $i = 1, 2, ..., r$. The length of the path is the number of edges in the path. A path is simple if all vertices in the path are distinct. An undirected graph is connected if every pair of vertices is connected by a path. We say that a graph $G' = (V', E')$ is a subgraph of $G = (V, E)$ if $V' \in V$ and $E' \in E$.

Graph connectivity

If for any two nodes $\{u, v\} \in V$ of graph $G = (V, E)$ an $[i, j]$ -path exists, the graph is said to be connected, otherwise it is disconnected. A maximal connected subgraph of G is a component of G.

If G is connected and $G \setminus S$ is disconnected, where S is a set of vertices or set of edges, then we say that S separates G. For more details please refer to [22].

Definition 1 [k-Connectivity] A graph G is vertex (edge) k-connected ($k \geq 2$) if it has at least $k + 2$ vertices and no set of $k - 1$ vertices (edges) separates it. The maximal value of k for which *a connected graph G is k - connected is the connectivity of G. For $k = 2$, graph G is called biconnected.*

If S is a vertex set such that $G \setminus S$ has more connected components than G, set S is called an articulation set. If $S = v$, the vertex v is called *articulation* or *cut vertex.*

The below theory represents fundamentals of graph (vertex) connectivity:

Definition 2 [Menger's theorem] A graph $G = (V, E)$ is k- vertex connected (k - edge connected) if for each pair v, u of distinct vertices , G contains at least k vertex- disjointed (edge disjointed) $[v, u]$-paths. Note:While vertex k-connectivity implies edge k-connectivity the reverse does not generally hold.

As the redundant edge we denote edge that can be easily removed from the solution without violating the vertex-biconnectivity feature. As a minimum vertex biconnected network we denote a graph that contains no cut nodes and no redundant edges.

The block cut graph

Referring to [23], a block is denoted as maximal subgraph of a graph G that is already vertex biconnected. If the graph is vertex biconnected the whole graph represents one block. If any two blocks of G share at least one node, then this node is called the cut point of graph G and its removal would disconnect graph G into at least two components.

A block cut tree $T = (V_T, E_T)$ is an undirected tree that represents relationships between blocks and cut points in graph G. Figure 1.3 illustrates the block cut tree, where two types of vertices are presented: cut vertices and block vertices. Each cut vertex in graph G is represented by corresponding cut node in V_T, and each maximal vertex biconnected block of graph G is represented by means of a unique block - vertex in V_T.

The block vertex is represented by all vertices of the corresponding block in G, except the ones which are cut points.

1.2 Dynamic Programming

In computer science this method is usually used to tackle problems which are solvable in polynomial time. The method described here is very helpful when solving a complex problems by breaking them down into sub-problems in a recursive manner. The space of the subproblems must be small, which means that any recursive algorithm that is solving the problem should be able to solve the same sub-problems time after time without generating any new sub-problems. The solutions to the original problem can be obtained either in top-down approach or bottom-up approach, which tries to solve the sub-problems first and use their solutions to build-on and finally obtain solutions to bigger sub-problems. For more details please refer to [29] and [3].

1.3 Metaheuristics and population based methods

The term metaheuristic has been introduced by Glover in 1986 [10], however for the time being there is no commonly accepted definition. Nevertheless we can outline here some common properties that characterize the metaheuristics:

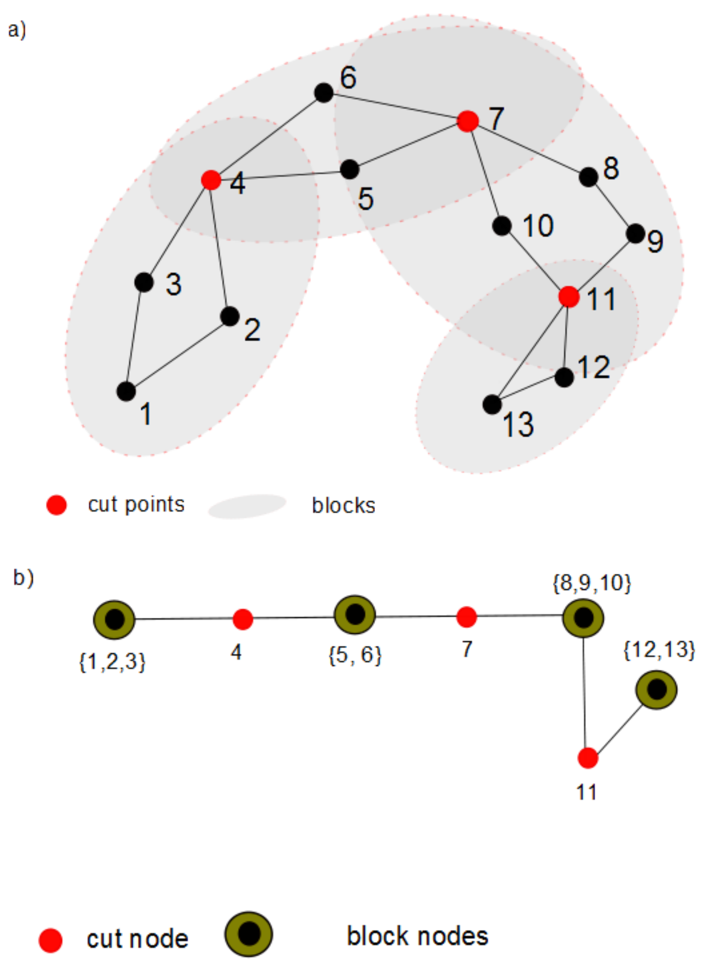


Figure 1.3: a) Graph $G = (V, E)$ which is not vertex biconnected, b) the corresponding block tree representing block nodes and cut nodes of graph G

- strategies that guide the search process
- exploring the search space in efficient way in order to find an optimal or near-optimal solution
- approximate and usually non-deterministic algorithms
- non-problem specific
- the abstract level is permitted by a basic concept.

The above characteristics guide us to the conclusion that the following (but not restricted to) classes of algorithms can be rated as metaheuristics: Ant Colony Optimization, Evolutionary Computation including Genetic Algorithms, Iterated Local Search, Simulated Annealing and Tabu Search. Generally metaheuristics are high level strategies which use different methods to explore

Algorithm 1 Evolutionary Algorithm

```
$t$←$0$
Initialization ($P$(0))
Evaluate ($P$)
repeat
    Selection ($P(t)$)
    Recombination ( $P'(t)$)
    Mutate ( $P'(t)$)
    Evaluate ($P'$)
    $P(t{+}1)$←Replacement ($P$(t), $P'$)
    $t$ ←$t{+}1$
until  termination condition met
```

the search spaces, simultaneously keeping the balance between diversification and intensification. This thesis focuses on Evolutionary Algorithms.

The population based methods like Evolutionary Algorithms (EA) deal with a set of solutions in every iteration. Such methods can be concisely characterized as computational methods of evolutionary processes. Evolutionary Algorithms use operators of recombination or crossover to create new individuals, and may adapt mutation operator also, which causes a self-adaptation of individuals. In order to gain good results by running EA, the *intensification* and *diversification* strategies should be applied. It has been proved by many applications that by using an improvement mechanism to increase the fitness of individuals is quite beneficial. Such EA, that applies a local search algorithm to every individual are often called *Memetic Algorithms*. By using a population of individuals it ensures the exploration of search space, but in order to quickly identify the good areas in the search space local search techniques need to be involved. However this can lead to premature convergence towards sub-optimal solutions, which is the one of the major difficulties of EA's. A simply way to diversify the population is involvement of mutation operators. A more sophisticated way can be the application of population management techniques, described in chapter 2.9. Algorithm 1 describes the general idea of EA's. The phrase "Evolutionary Algorithms" denotes a family of parallel, randomized search optimization heuristics which share the following features:

- population P of solutions
- individuals (parents) are selected from this population and are mated to form new individuals (children)
- children are possibly mutated to introduce diversity into the population and avoid ending up with a population of very similar or even identical solutions.

An initialization of the population can be performed by means of a random procedure or heuristics can be involved. The main condition must be fulfilled – the diversity of a beginning (initial) population of chromosomes should be achieved. Each member of the population is characterized by a fitness value. This fitness determines the chances of the individuals survival and a member with a lower fitness has a smaller chance of survival and of being chosen for mating (and thus to have its genes reproduced). The net effect of survival of the fittest is that the average fitness of the population increases with each generation. By allowing mutation, the diversity of the population is increased and new (maybe better) attributes can be created. This simulation of evolution allows the principle of survival of the fittest to be applied to optimization problems. The goal is to find a population member with a very high fitness level, corresponding to a perhaps an optimal solution to the problem.

Steady state Evolutionary Algorithm As a steady state sort of algorithm is used to solve GMVBCNP, it is briefly introduced in here. Steady state EA [13] [11] is an algorithm with successive population replacement. Strictly speaking there is no offspring population. Instead offspring are generated gradually replacing the worst individual(s) immediately so that the population size is kept constant.

The *elitism strategy* can be used as well in order to improve the results. In this case, the best individual at generation $k+1$ (the father or the mother) is maintained in the next generation if its child has a performance inferior to that of its parent. Without elitism, the best results can be lost during the selection, mutation and crossover operations. In case of global elitism, each individual in the population of generation $k+1$ can replace its parent of generation k, if it has a performance superior than him. In this case, at a generation $k+1$, the individuals are better than the individuals at generation k.

Convergence The main problem of an EA is the premature convergence. The fitness of the best and the average individual in each generation increases towards a global optimum. Convergence is the progression towards increasing uniformity. A gene is said to have converged when a high percentage of the population share the same value. The population is said to have converged when all of the genes have converged. As the population converges, the average fitness will approach that of the best individual. To avoid premature convergence the sort of population management can be involved, which will control population diversity. Such a method is introduced in Chapter 2.9.

1.4 Variable Neighborhood Search

Variable Neighborhood Search (VNS) is a metaheuristic [2] that explicitly applies a strategy based on dynamical change of neighborhood structures. The change of neighborhoods leads to exploration of local optima but as well to escape from these valleys in order to reach under-explored areas to find even better results [21]. The VNS main cycle is composed basically of three steps: shaking, local search and move. The VNS can be combined with Variable Neighborhood Descent (VND) that applies a best improvement local search to find s local minima. The basic idea of VNS is to improve VND by means of shaking function that jumps to a random new solution among the neighbors of the current solution. This approach provides the possibility of escaping from local optima and valleys containing them. The Algorithm 2 presents the pseudocode of general VNS used further for solving the d-GMSTP.

Algorithm 2 Schema of Variable Neighborhood Search

```
generate initial solution $S$
repeat
    $k$ =1
    while $k < k_{max}$ do
        generate a random $S'$ from one of the neighborhood $N_k$ of $S$ // shaking phase
        $l$ =1
        while $l < l_{max}$ do
            find the best neighbor $S''$ of $N_l(S')$
            if $f(S'') < f(S')$ then
                $S' = S''$
                $l$ =1
            else
                $l = l$+1
        if $f(S') < f(S)$ then
            $S = S'$
            $k$ =1
        else
            $k = k$+1
until termination condition met
```

2 The Generalized Minimum Vertex Biconnected Network Problem

2.1 Problem formulation

The considered Generalized Minimum Vertex Biconnected Network Problem (GMVBCNP) is defined as follow. We consider a complete, undirected weighted graph $G = (V, E, c)$ with node set V, edge set E and edge cost function $c : E \rightarrow \mathbb{R}^{+}$. The node set V is partitioned into r pairwise disjointed clusters: $V_1, ...V_r$, $\bigcup_{i=1}^{r} V_i = V$, $V_i \cap V_j = 0\ \forall i,\ j = 1, ..., r; i \neq j$.

A solution to GMVBCNP defined on G (Figure 2.1) is a subgraph $S = (P, T)$, $P = \{p_1, .., p_r\} \subseteq V$ connecting exactly one node from each cluster, i.e. $p_i \in V_i, \forall i = 1, ..r$ and containing no cut nodes. A cut node is a node whose removal would disconnect a graph.

The costs of such a vertex biconnected network are its total edge costs i.e. C(T)=$\sum_{(u,v)\in T} c(u, v)$, and the objective is to identify a feasible solution with minimum costs. The feasible solution should be a *redundant edges* free subgraph. As the redundant edges we denote edges that can be easily removed from the solution without violating the vertex-biconnectivity feature.

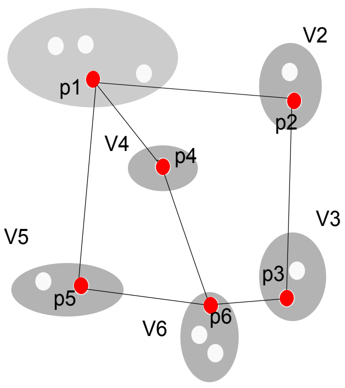


Figure 2.1: Example for a solution to GMVBCNP.

In order to solve the The Generalized Minimum Biconnected Network Problem, a problem specific operators have been implemented. In this section the general idea of an implemented operators is presented.

2.2 Previous work

For the time being there are not too many literatures addressing the Generalized Minimum Vertex Biconnected Network Problem. Eswaran and Tarjan [4] were the first to investigate *vertex biconnectivity augmentation problem for graphs*(V2AUG). By means of a reduction of the Hamiltonian cycle they proved that the decision problem for V2AUG is NP-complete. Watanabe and Nakamura [37] proved that minimum-cost augmentation for edge or vertex k-connectivity is NP-hard, for any $k \geq 2$. Some preliminary results of running memetic algorithm for the Generalized Minimum Vertex Biconnected Network Problem can be found in [17]. The vertex biconnectivity as well as the edge biconnectivity problem has been attacked by Ljubic in [22]. Because the Generalized Minimum Vertex Biconnected Network Problem and the Generalized Minimum Edge Biconnected Network Problem (GMEBCNP) are strongly related it is worth to mention that Leitner in [21] proposed VNS approach for solving the GMEBCNP. He proposed different types of neighborhood structures addressing particular properties as spanned nodes and the edges between them.

2.3 Memetic Algorithm

After performing tests with a steady state evolutionary algorithm, the results were still not satisfactory, so I decided to use a memetic algorithm. Memetic algorithms are a hot topic nowadays, and they were successful in many optimization problems. The difference between Memetic Algorithm (MA) and EA [13][11] is that MA actually exploits all available knowledge about the problem under study. As mentioned in [25] MA exploits problem knowledge by involving hybridization, that evaluates to the use of pre-existing heuristics, preprocessing data reduction rules, approximation, local search techniques and specialized recombination operators etc. The important feature of MA is the use of adequate representation to the problem being tackled. Those both allow creation of a highly efficient tool, for solving difficult optimization problems. MA's are considered as population based metaheuristics, and deal with the population of individuals, similar to EA's. However opposite to EA's Memetic Algorithms they couple with an individual learning procedure capable of performing local refinements.

The Algorithm 3 pseudocode presents the involved MA for solving the GMVBCNP. For selection, the standard tournament selection with tournament size of two is applied. The designed framework for MA is based of the steady state EA, however it involves both local improvements with graph reduction as well as the population management technique. The applied operators and approaches are described in the next subsections.

2.4 Initial solutions

All initial solutions are created as Hamiltonian cycles. In the considered problem Hamiltonian cycles represents feasible solutions, because removal of a single node disconnects a cycle and creates a path and therefore all nodes are still connected. Each solution is represented by means of vector *solution.data* which stores the indices of nodes chosen from clusters, and adjacent list that represents connections between clusters - in other words it represents edges of a clustered graph.

Algorithm 3 Memetic Algorithm for the GMVBCNP

```
create random initial population $P$
repeat
    select two parental solutions $S_1 \wedge S_2 \in P$
    create a new solution $S_N$ by crossover on $S_1 \wedge S_2$with probability $p_{cross}$
    mutate a new solution $S_N$ with probability $p_{mut}$
    if $edgemanagement = true$ then
        check the percentage of covered edges in population $P$
        change $edgemanagementstrategyparameters$ adequately
    locally improve $S_N$ with probability $p_{imp}$
    if $delta\, managenent = true$ then
        repeat
            mutate $S_N$
        until $S_N$ will satisfy condition for addition
    update diversity parameter $\triangle$
until no new better solution found in last $l$ iterations
```

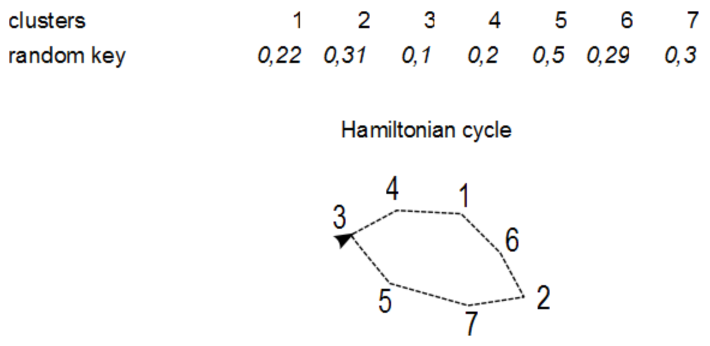
Algorithm 4 Create Initial Solution S

```
for $i = 1, \ldots, r$ do
    $w_i\,(V_i)$ = random [0,1)
sort $w_i\,(V_i)$ descending
create adjacent list *adjlist*

for $i = 1, \ldots, r$ do
    random node $n_i$ from cluster $V_i$
    add node $n_i$ to $solution.data$
```

A method based on random keys [1] is used to create the initial solutions. In this method, to each gene a random number w_i , $i = 1, \ldots, r$ drawn uniformly from $[0, 1)$ is assigned (Figure 2.2). A single gene represents a cluster. To decode the chromosome we visit nodes in ascending order of their genes. Nodes that should be early in the tour tend to evolve genes closer to 0 and those that come later tend to evolve genes closer to 1. In the second step we need to choose randomly a node from each cluster. The nodes are stored then in solution.data, and connections between clusters are stored in an adjacent list. Pseudocode of Algorithm 4 represents the idea of a creation of initial solutions.

The time complexity for sorting the w_r random values is $O(r\, log\, r)$. Randomizing r keys takes $O(r)$ and the time complexity for selection of r nodes is $O(r)$. It leads to the $O(r\, log\, r)$ time complexity for initialization of a single solution.

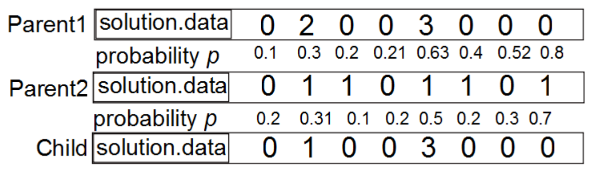


clusters	1	2	3	4	5	6	7
random key	0,22	0,31	0,1	0,2	0,5	0,29	0,3

Figure 2.2: Hamiltonian cycle created by means of random keys

2.5 Crossover operators

A crossover operator should be designed with the aim to provide the highest possible heritability, i.e. an offspring should have as many common features to its parents as possible. There were many ideas for crossover operators, however after running test instances and measuring efficiency as well as time complexity I decided to use only two of them. The general aim for both of them was to inherit as many common edges and nodes as possible from parental solutions. The common step for both: *one point crossover* and *greedy crossover* is assignment to each gene probability p. Figure 2.3 illustrates an example of building a new descendant solution. To each parental gene probability $p \in [0, 1)$ is assigned and then genes from parental solutions with higher probability assigned on the specific gene position are inherited. In the presented example the genes from parents: 2,2,1,1,1,1,1,1 are transferred to the new solution respectively. This method guarantees that nodes common for both parents will be always inherited. The next step in both crossover operators refers to the creation of an adjacent list that stores connections between clusters. This is described separately in the following subsections for each operator.



Parent1	solution.data	0	2	0	0	3	0	0	0
	probability *p*	0.1	0.3	0.2	0.21	0.63	0.4	0.52	0.8
Parent2	solution.data	0	1	1	0	1	1	0	1
	probability *p*	0.2	0.31	0.1	0.2	0.5	0.2	0.3	0.7
Child	solution.data	0	1	0	0	3	0	0	0

Figure 2.3: The heritage of common nodes

One-Point Crossover operator

This crossover operator is based on a random choice of parental edges, which creates a new offspring solution $S_N = (P_N, T_N)$. To the parental edges of solution S_l with lower connection cost, the higher probability p is assigned. Assignment of probability can simulate the heritage of more edges from the solution with lower cost, however please note that inherited edges from S_l would not necessarily be edges with the lowest costs in that solution. The pseudocode of this crossover is presented by Algorithm 5. The new created solution might be an infeasible one, therefore a specially designed method needed to be run. A modified DFS algorithm, presented in section 2.7, determines whether or not the new solution is vertex biconnected. Repair methods can introduce some redundant edges, so it is necessary to evaluate the solution and remove the unnecessary edges. Let us consider and an example presented below. We create a new offspring individual $S_N = (P_N, T_N)$ from two parental solutions $S_1 = (P_1, T_1), S_2 = (P_2, T_2)$ presented on Figure 2.4a. In the first step node p_i, $i = 1, \ldots, r$ from each cluster of solutions S_1 or S_2 is inherited, as presented on Figure 2.4. This leads to the solution which will posses the common nodes for both parents. In this case common nodes $\{p_2,\, p_3\}$ are included in the new solution and the rest of the nodes $\{p_1,\, p_4,\, p_5,\, p_6\}$ are inherited in a random way. In the second step a new solution based on parental edges is built. Lets assume the solution S_1 has lower connection cost than solution S_2. Defining the probability $prob \in [x,\, 1)$, where $x \in (0.5,\, 0.99]$ and assigning it to S_1 it will favor the edges from the solution with lower connection cost. The idea is very simple: we start with cluster V_1, random the probability p and compare it with $prob \in [x,\, 1)$. If $p < prob$ then inherit edges from parent S_1, alternatively from parent S_2. The steps b, c, d, e on Figure 2.4 build the new solution S_N. Figure 2.4e presents a complete solution, where all clusters $\{V_1, V_2, V_3, V_4, V_5, V_6\}$ were visited and edges adjacent to certain clusters were inherited. In order to check if the graph is vertex biconnected the Modified DFS is run, described in next chapter, to determine if there are any cut vertices. In the presented example the new solution S_N does not posses cut nodes, so we can start determining if there are any redundant edges.

Algorithm 5 One-Point Crossover

```
input: $S_1 = (P_1, T_1), S_2 = (P_2, T_2)$
for $i = 1, \ldots,$ number of clusters do
    random $p$

    if $p < probability$ then
        add to $P_N$ node from $P_1$ of cluster i and to $T_N$ all edges from $T_1$ adjacent to cluster i
    else
        add to $P_N$ node from $P_2$ of cluster i and to $T_N$ all edges from $T_2$ adjacent to cluster i
ensure vertex biconnectivity of $S_N$
remove redundant edges of $S_N$
```

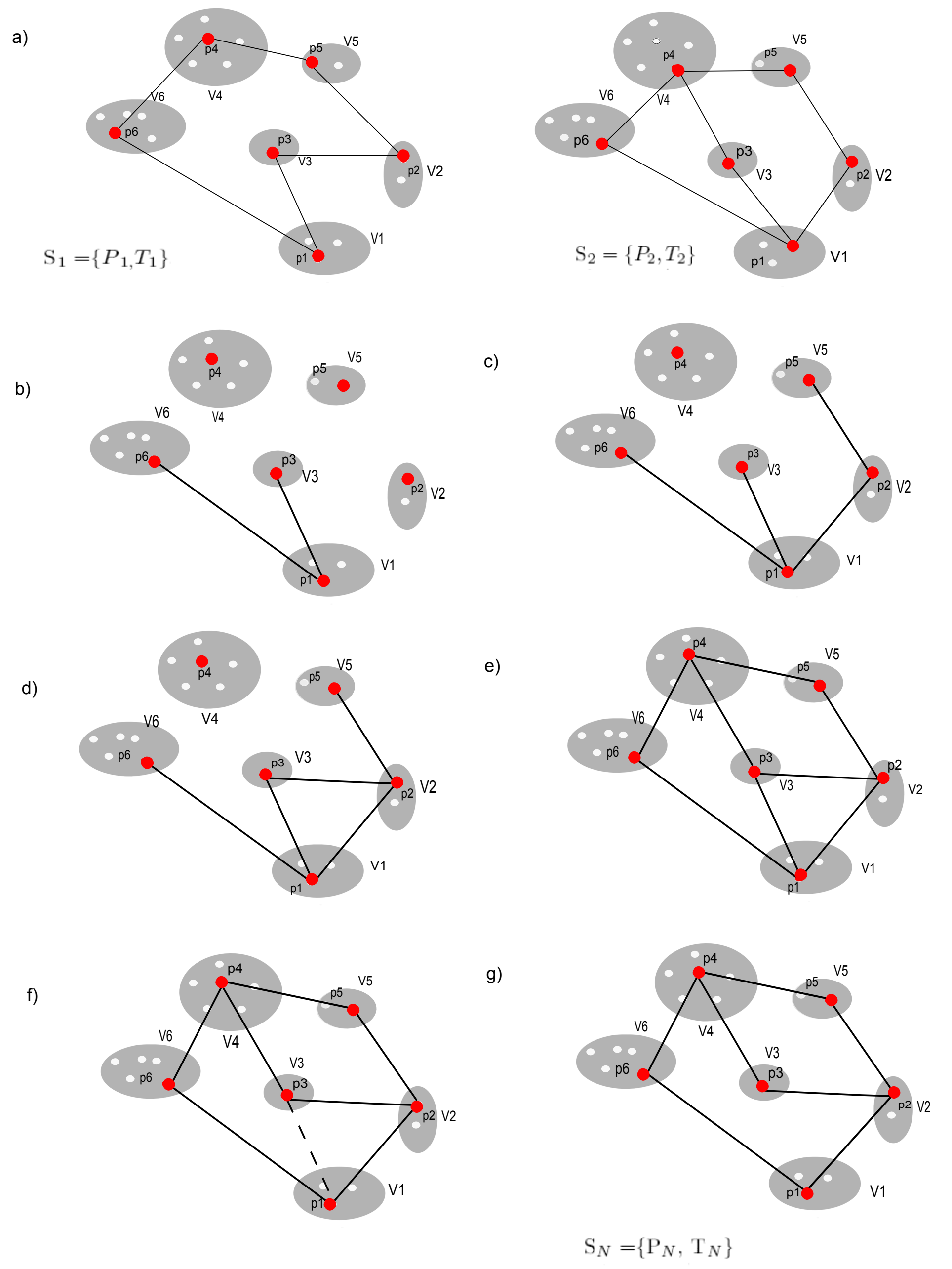


Figure 2.4: One-Point crossover

It is easy to notice that edges $\{p_1, p_2\}, \{p_3, p_4\}, \{p_3, p_2\}, \{p_1, p_3\}$ can be considered as redundant, so the new solution does not present a minimum vertex-biconnected graph. How to determine the edges for removal is described with more details in chapter *Repair procedure*. Considering the presented example, by removing an edge $\{p_1, p_3\}$ a minimum vertex-biconnected

Algorithm 6 Greedy Crossover Operator

```
input: $S_1 = (P_1, T_1), S_2 = (P_2, T_2)$
for $i = 1, \ldots, r$ do
    $S_N$= $S_N$+$\{p_i \in P_1 \vee$ p$_i \in P_2\}$
T$_N$ = T$_1$∩T$_2$
sort $E_N$ by alfa or beta or gamma

for each edge $e(p, v)$ in $T_N$ do
    if edge $e(u, v)$ is removable $\wedge$ graph $S_N$ is vertex biconnected then
        $T_N = T_N \setminus e(u, v)$
```

graph is archived. Considering only the time complexity of pure crossover operator, without calling the repair method it would be $O(r)$ for inheriting the nodes and edges, and $O(r + m)$ for checking if the graph is vertex biconnected, which gives $O(m)$ in the worst case. Further considering $O(l)$ for connecting possible blocks and $O(m^2)$ for repair method, the upper bound for crossover operator is $O(m^2)$.

Greedy Crossover Operator

This crossover operator inherits the properties from both parents in two steps as well. First I determine which nodes p_i , $i = 1, \ldots, r$ will be inherited from the clusters, and then I determine which edges will build the new solution $S_N = (P_N, T_N)$.

The basic idea of this crossover operator is to inherit firstly all edges from both parents $T_N = T_1 \cup T_2$, which denotes that the new solution probably has many redundant edges. In the next step all edges T_N are considered in a particular order:

- *alfa* - decreasing costs
- *beta* - decreasing perturbated costs $c'(p_i, p_j) \cdot \rho$, where ρ is uniformly distributed random value [0.5,...,1.0]
- *gamma* - random order

and repair procedure is called. This procedure tests if the chosen edges are redundant and removes them if applicable from T_N. If all redundant edges are removed, then the archived solution can be considered as a minimum vertex-biconnected.

The pseudocode of Greedy Crossover Operator is presented by Algorithm 6.

Considering only the time complexity of a pure crossover operator, without repair method a new solution is achieved in the worst case $O(r)$. Further considering $O(m^2)$ for repair method, the upper bound for crossover operator is $O(m^2)$.

2.6 Mutation Operators

Mutation is applied to each child individually after crossover. It randomly alters each gene with a small probability. The traditional view presents crossover as the more important of the two techniques for rapidly exploring a search space. Mutation provides a small amount of random search, and ensures that all solutions in the search space have the possibility of being examined. The designed mutation operators were inspired by the neighbor's structures presented in [21].

Simple Node Exchange Mutation Operator

This mutation operator is based on exchange of exactly one node p_i from cluster V_i to a new node p'_i from the same cluster. Therefore if $I = \{p_j \in P \,|\, (p_i, p_j) \in T\}$ is the set of nodes incident to p_i in $S = (P, T)$, these nodes will be incident to p'_i in the new solution $S' = (P', T')$ with $P = P \setminus \{p_i\} \cup \{p'{}_i\}$, p_i, $p'{}_i \in V_i$, $p_i \neq p'_i$ and $T' = T \setminus \{(p_i, p) \,|\, p \in I\} \cup \{(p'_i, p) \,|\, p \in I\}$.

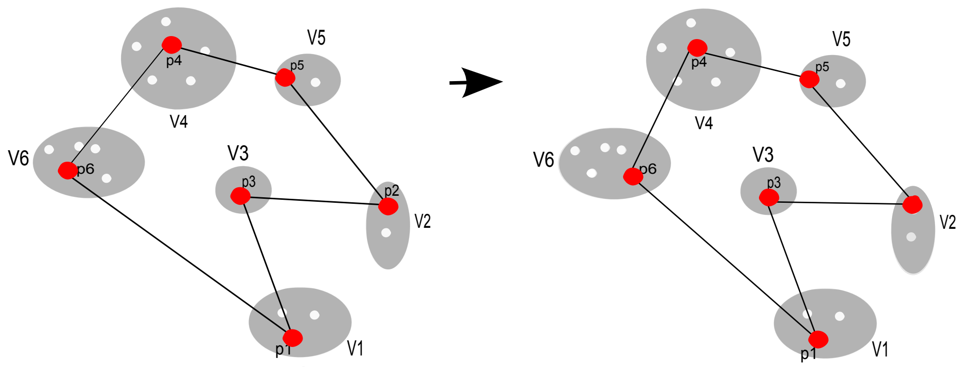


Figure 2.5: Simple Node Exchange Mutation Operator

On Figure 2.5 the node within clusterV_6 has been exchanged. The Algorithm 7 presents the pseudocode of Simple Node Exchange Mutation Operator. The considered complexity time is constant $O(1)$.

Algorithm 7 Simple Node Exchange Mutation Operator

```
input: $G = (V, E)$
$p_i$= used node of cluster $V_i$
$p'_i$= random node from cluster $V_i \setminus p_i$

if $p_i \neq p'_i$ then
    save current solution
```

Edge Augmentation Mutation Operator

This mutation operator adds the new edge. This operation extends graph $G = (V,\, E)$ with an edge $e(u,\, v) \in E$. So we are receiving the new solution $S' = (P', T' \cup \{e\})$. The resulting graph is certainly not minimal, because it includes a redundant edge, so the connection cost is not minimal, however the graph is vertex biconnected. Therefore, there is at least one edge that can be removed without violating the vertex biconnectivity property of solution $S' = (P', T' \cup \{e\})$.

The Algorithm 8 represents the pseudocode of Edge Augmentation Mutation Operator.

Algorithm 8 Edge Augmentation Mutation Operator

```
input: $G = (V, E)$

repeat
    random edge $e_1$
    random edge $e_2$
until $(e_1 = e_2)$
if ( $cost\, e_1 < cost\, e_2$) then
    $E \cup \{e_1\}$
else
    $E \cup \{e_2\}$
call repair procedure ( $S' = (P, T')$)
```

After the introduction of an new edge the *repair* procedure is called, in order to examine and remove redundant edges. In the example shown in Figure 2.6, the initial solution is augmented by *e* $(p_1,\ p_3)$ which leads to S' where at least one edge out of $E' = \{\, (p_1, p_2), (p_2, p_3), (p_3, p_6), (p_1, p_6)\,\}$ can be removed. During the optimization process, (p_3, p_6) and (p_1, p_2) are removed. The time complexity of this mutation operator equals to the complexity of the repair method, that is $O(m^2)$ as an upper bound.

Two Nodes Swap Mutation Operator

This mutation operator is based on the rearrangement of nodes between two clusters. This leads to the exchange of some adjacent edges between two clusters. Swapping p_i and p_j $(p_i \in V_i,\, p_j \in V_j, i \neq j)$ is defined as follows.

Consider a solution $S = (P, T)$ and let $I_i = \{p \in P \,|\, (p_i, p) \in T\,\}$ be the set of nodes incident to p_i, and $I_j = \{p \in P \,|\, (p_j,\, p) \in T\,\}$ the set of nodes incident to p_j in S. The exchange operation transforms $S = (P, T)$ into a new solution $S' = (P, T')$ with $T' = T \setminus \cup \{(p_i, p) | \, p \in P_j\} \cup \{(p_j, p) | \, p \in P_i\}$. In other words all edges that were incident to p_i are incident to p_j, and vice versa.

The example of an exchange operation between two clusters is presented on Figure 2.7. The adjacent edges of nodes p_4 and p_5 are exchanged. The pseudocode of this mutation operator is presented by Algorithm 9. As in some cases this mutation operator can introduce redundant edges, repair method examines edges against removal possibility. The time complexity of this mutation operator equals to the complexity of the repair method, what is $O(m^2)$ as an upper bound.

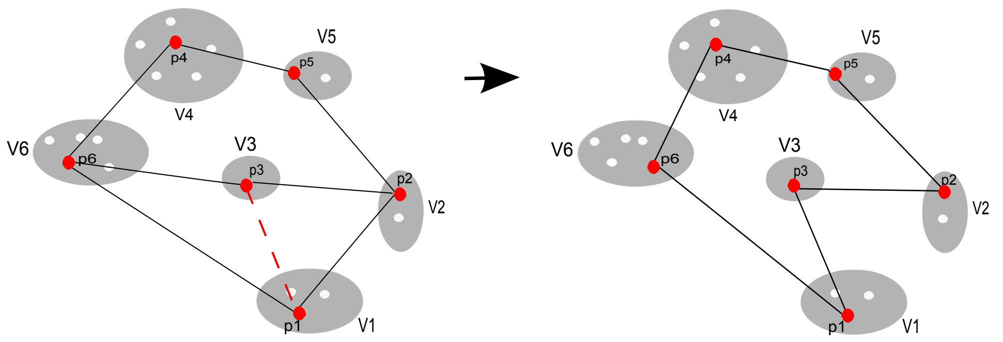


Figure 2.6: Edge Augmentation Mutation Operator

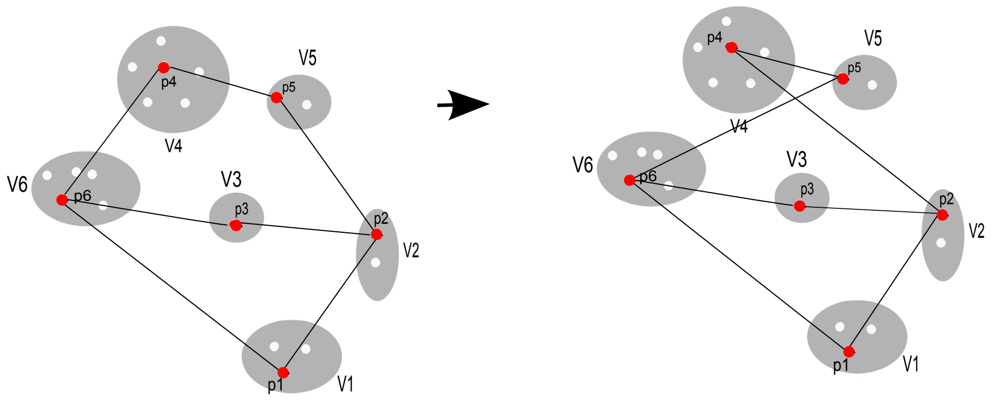


Figure 2.7: Two Nodes Swap Mutation Operator

2.7 Cut Nodes Detection and repair procedure

In order to deal with feasible solutions a special algorithm, inspired by R.Sedgewick [34] that detects cut nodes has been implemented. The main conclusion regarding graph biconnectivity is: ˋThe graph is biconnected if and only if there are at least two different paths connecting each pair of vertices". To identify the cut vertices depth first search (DFS) algorithm can be used. However in GMVBCNP some small modifications of DSF are necessary, which is described further.

Algorithm 9 Two Nodes Swap Mutation Operator

```
input: $G = (V, E)$
repeat
    random cluster $V_1$
    random cluster $V_2$
until $V_1 \neq V_2$
swap adjacent edges of cluster $V_1$ and $V_2$
call repair procedure ( $S' = (P, T')$)
```

Modified DFS

In the considered problem, the classical DFS would not discover the cut nodes. Therefore some small modification has been done. The general idea of cut nodes detection is as follows:

Let $G = (V,\ E)$ be a connected, undirected graph.

We can find all articulation points in a graph using depth-first search, so let T be a depth-first search tree.

Theorem 1. The root of T is an articulation point if it has two or more children.

Proof: If the root has less than two children, then deleting the root does not disconnect T so the root is not an articulation point. If the root has two or more children, then because there are no cross edges (G is undirected), every path from one child of the root to another contains the root. Therefore, the root is an articulation point.

Theorem 2. A non-root vertex v of T is an articulation point if it has at least one child w and there is no back edge from a descendant of w to a proper ancestor of v.

Proof: If there is no back edge from a descendant of w to a proper ancestor of v, then because there are no cross edges, every path from w to the parent of v contains v.

Lets consider the following function: $low[v] = min\{d[v]\} \cup [\{d[x] : (u, x)$ *is a back edge from some descendant* u of $v\}]$.

If $low[v]$ is computed for each vertex v, then v is an articulation point if $low[w] \geq d[v]$ for some child w of v in T.

There is a back edge from a descendant of a child w of v to a proper ancestor of v if $low[w] \geq d[v]$.

If there is a back edge from a descendant to a proper ancestor x of v, then $low[w] \leq d[v]$. Since x is a proper ancestor of v, then $d[x] < d[v]$ so $low[v] < d[v]$. If $low[w] \leq d[v]$, then there is a back edge from some descendant u of v to a vertex x with $d[x] < d[v]$. Thus, x was discovered before v; x is a proper ancestor of v.

In implemented DFS (Algorithm 10) a node u is a *cut vertex*, for every child v of u, if there is no back edge from v to a node higher in the tree than u [18]. If there is no way to visit other nodes in the graph, that are in the decedent tree of u, without passing through u, such a node is *cut vertex*.

Thus, for each node in DFS traversal, $dfsnum(v)$ and $low(v)$ is calculated (Figure 11). As $dfsnum(v)$ we denote the number defining the order of visited nodes by DFS traversal. The definition of $low(v)$ is the lowest $dfsnum$ of any node that is either in the DFS subtree rooted at v or connected to a node in that subtree by a back edge. Then, in DFS, if there are no more

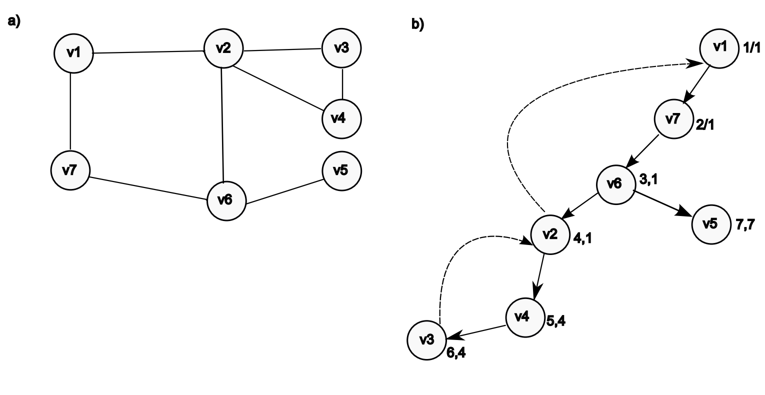


Figure 2.8: DFS Algorithm :*a* Graph with two cut vertices $v2$ and $v6$ **b)** depth first tree representing the graph on figure *a*; the numbers next to the nodes represents $dfsnum$ and low

nodes to visit, the values of low are backed up and updated as only returned from each recursive call.

The stack has been used to trace back the recursive calls. When an $edge(u, x)$ is processed - either by a recursive call on vertex x from vertex u, or (u, x) is back edge, that edge is put to a stack. Later, if u is identified as cut vertex, then all edges from the top of the stack down to (u, x) are the edges of one biconnected component. So, the edges are popped out of the stack until the top of the stack is (u, x). Those edges belong to a biconnected component.

The running time of the algorithm seems to depend relatively more on the number of edges than on the number of vertices. That is because, if a graph has more edges incident to each vertex, the algorithm needs more works in each call of the recursive function to decompose the graph's structure. The algorithm runs at linear time and it can be observed that the complexity of overall performance is $O(r + m)$ in the worst case, where r is the number of vertices and m is the number of edges in a graph.

Connecting block vertices

When creating a new offspring by means of one of the above crossover operators, or when applying mutation to an individual, the final solution S' after those operations can be infeasible or not minimal. This can be caused either by introducing redundant edges to solution S or by violating the vertex biconnectivity property. So some solutions S' need to be repaired. Therefore additional repair procedure has been implemented.

When the crossover operator is called, a new offspring is created mostly from parental edges. Unfortunately for some reasons, in order to create a feasible solution using only parental edges would be not sufficient. At the end of the crossover operation the method *checkSolutionV2Connectivity* is called. This method uses the modified DFS algorithm 10 to identify the prospective cut vertices and block vertices. To repair the solution S' including cut vertices v_c, one node from each block vertex B_i is selected in random way (Algorithm 11) and a new edge

Algorithm 10 Articulation DFS

```
input: $\forall$ v$_i$.depth = -1 , $i$= 1, ..., n

DFS ($v$)

$v$.depth = increase($dfsCounter$)

$v$.low = $v$.depth
forall $e(v,x)$ do
    if $x$.dfsnum = -1 then
        $x$.dfslevel = increase($v$.dfslevel)
        $v$.ChildNumber = increase($v$.ChildNumber)
        push $e(v,x)$ on $stack$
        DFS($x$)
        $v$.low = min($v$.low, $x$.low)
        if $v$.dfsnum = 1 then
            if $v$.ChildNumber$\geq 2$ then
                add $v$ to articPointList
            repeat
                add $stack$.pop to bccEdgeList
            until $stack$.top$\neq(v,x)$
        else
            if $x$.low $\geq v$.dfsnum then
                add $v$ to $v$.articPointList
                repeat
                    add $stack$.pop to bccEdgeList
                until $stack$.top$\neq(v,x)$
    else
        if $x$.dfslevel $<$ $v$.dfslevel - 1 then
            $v$.low = min($v$.low, $x$.dfsnum)
            push $edge(v,x)$ on $stack$
```

$e(u,v)$ is added to solution S'. Of course the cut vertices are excluded during the random search. When all block vertices B_i are connected together the solution is feasible in respect of vertex biconnectivity, but not in respect of minimal connection cost, because adding new edges during solution repair can introduce redundant edges. In order to connect l block vertices B_i, $l-1$ edges are needed to make the solution feasible in respect of vertex biconnectivity property. In the worst case we achieve feasible solution in $O(l)$, where l is the number of blocks in a graph.

Figure 2.9a presents the solution S'that contains one cut vertex p_5 and two cut blocks containing nodes B_2= $\{p_4, p_5, p_6\}$ and B_2= $\{p_1, p_2, p_3, p_5\}$. To make the solution feasible the new edge $e(p_3, p_4)$ has been added, see Figure 2.9b. The solution S' is a feasible one, but now there exist redundant edges, which should be removed. This is performed by the *repair* procedure.

Algorithm 11 Algorithm for connecting block in solution S'

```
input: $S' = (P', T')$
DFS ( $S'$)

if $cut\_vertex$ exist between blocks $B_i$ and $B_j$ then
    for $i = 1, \ldots, l$ do
        random node $u$ from $B_i$
        random node $v$ from $B_j$
        add edge $e(u, v)$ to $T'$
```

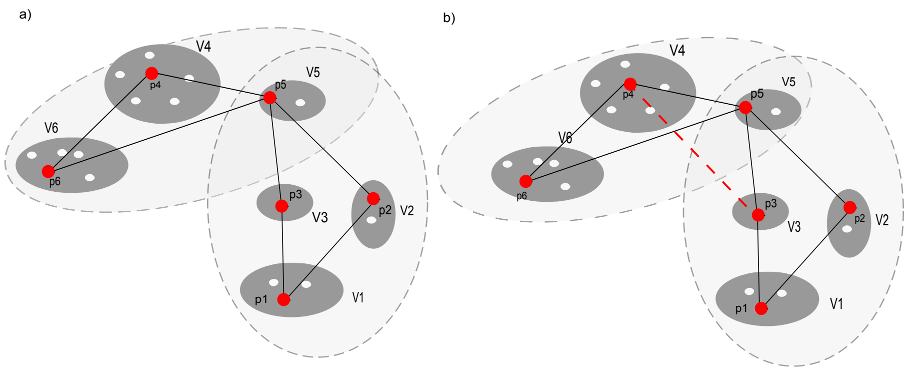


Figure 2.9: Repairing the infeasible solution by connecting blocks of solution S'

Repair procedure

The mutation, crossover or connection of blocks can introduce redundant edges. The solution $S_N = (P_N, T_N)$ can be feasible in respect of vertex biconnectivity but the connection cost would be not minimal. In such cases the redundant edges should be identified and removed. In order to minimize the search area, not all edges are checked against redundancy, but only the edges $(p_i, p_j) \in T_N$, where $deg(p_i) > 2 \wedge deg(p_j) > 2$ are considered. Inspired by [14], to minimize the connection cost, before the edges are examined against its removal possibility, all edges adjacent to nodes with degree $deg > 2$ are put in particular order:

- *alfa* - decreasing costs
- *beta* - decreasing perturbated costs $c'(p_i,\, p_j) \cdot \rho$, where$\rho$ is uniformly a distributed random value [0.5,...,1.0]
- *gamma* -random order.

To each edge (p_i, p_j) a weight w_c is assigned. What is obvious, is the *alfa* strategy emphasizes on the intensification, whereas *gamma* favours diversification. Each time the *repair method* is called, one of the above sorting criterion is selected. Some initial values were set up (*alfa, beta, gamma*) = (0.6, 0.2, 0.2), however they change dynamically during the execution of the memetic algorithm. The values are strongly connected with the population management procedure, which determines how diverse the current population is. If the diversity factor *div* will be too small or too high, then the *gamma* is increased and the *alfa* decreased respectively.

The search for redundant edges starts with the edge with the highest weight w_c, determined by factor *alfa, beta* or *gamma*. In Figure 2.10a it can be seen that solution *S'* includes three potential redundant edges $e_1(p_3,\, p_5)$, $e_2(p_3,\, p_4)$ and $e_3(p_4,\, p_5)$, because the degree of each node $deg(p_i) > 2,\ i \in \{3,4,5\}$ is more than two. Assume that weight $w_c(e_1) > w_c(e_2) > w_c(e_3)$, so edge $e_1(p_3,\, p_5)$ is removed from solution S', if and only if it will not introduce a cut vertex. This is checked by running Modified DFS which examines if the edge can be removed. Next the edge $e_2(p_3,\, p_4)$ will not be examined, because degree of node $deg(p_3) = 2$ and removal of this edge would violate the vertex bi-connectivity property. The last potential edge is $e_3(p_4,\, p_5)$. As removal of this edge will not introduce cut vertex it is removed. The minimum vertex bi-connected solution is achieved and presented in Figure 2.10b. The worst case is when all nodes have degree $deg > 2$. In order to sort the edges we need $O(m\ log\ m)$, for examining each edge against removal in the worst case we need $O(m(n+m))$. In general it gives an upper bound of $O(m^2)$

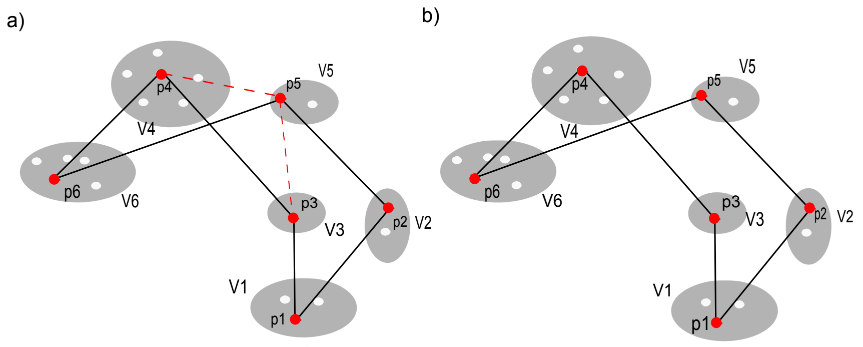


Figure 2.10: Removing redundant edges by the repair procedure.

2.8 Local improvement

Graph reduction

The graph reduction technique has been introduced and successfully applied to the GMEBCNP in [21] and [14]. The motivation is to reduce the search space for some neighborhood structures on which the local improvement procedures are based on.

It is generally not possible to derive an optimal selection of spanned nodes in polynomial time when a global structure S^g is given. However, this task becomes feasible once the spanned nodes in a few specific clusters are fixed. Based on the global structure, it can be distinguished between *branching clusters* that have a degree greater than two, and *path clusters* that have a degree of two. Note that there are no clusters with degree one, since this would violate the biconnectivity constraint.

Once the spanned nodes within all branching clusters are fixed, it is possible to efficiently determine optimal selection of nodes for the path clusters: By computing the shortest path between two nodes of branching clusters which are connected by path clusters, optimal spanned nodes can be obtained.

Formally, for any global structure $S^g = \langle V^g, T^g \rangle$, we can define a *reduced global structure* $S^g_{red} = \langle V^g_{red}, T^g_{red} \rangle$. V_{red} denotes the set of branching clusters, i.e. $V^g_{red} = \{V_i \in V^g \mid \deg(V_i) \geq 3\}$. T^g_{red} consists of edges which represent sequences of path clusters connecting these branching clusters, i.e. $T^g_{red} = \{(V_a, V_b) \mid (V_a, V_{k_1}), (V_{k_1}, V_{k_2}), \ldots, (V_{k_{l-1}}, V_{k_l}), (V_{k_l}, V_b) \in T^g \wedge V_a, V_b \in V^g_{red} \wedge V_{k_i} \notin V^g_{red}, \ \forall i = 1, \ldots, l\}$. Note that S^g_{red} is in general a multi-graph that can contain multiple edges corresponding to multiple paths in S^g between two nodes. Figure 2.11 shows an example for applying graph reduction on the global structure S^g of Figure 1.2 . V_2 and V_3 are branching clusters while all others are path clusters. Corresponding to the reduced global structure $S^g_{red} = \langle V^g_{red}, T^g_{red} \rangle$ we can define a *reduced graph* $G_{red} = \langle V_{red}, E_{red} \rangle$ with the nodes representing all branching clusters $V_{red} = \{v \in V_i \mid V_i \in V^g_{red}\}$ and edges between any pair of nodes whose clusters are adjacent in the reduced global structure, i.e. $(i, j) \in E_{red} \Leftrightarrow (V_i, V_j) \in T^g_{red}, \forall i \in V_i, j \in V_j$. Each such edge (i, j) corresponds to the shortest path connecting i and j in the subgraph of G represented by the reduced structure's edge (V_i, V_j), and (i, j) therefore gets assigned this shortest path's costs.

When fixing the spanned nodes in V^g_{red}, the costs of the corresponding solution S with optimally chosen nodes in path clusters can be efficiently determined by using the precomputed shortest path costs stored with the reduced graph's edges. Decoding the corresponding solution, i.e. making the optimal spanned nodes within path clusters explicit, is done by choosing all nodes lying at the shortest paths corresponding to used edges from E_{red}.

For details on how the graph reduction can be efficiently implemented, please refer to [21].

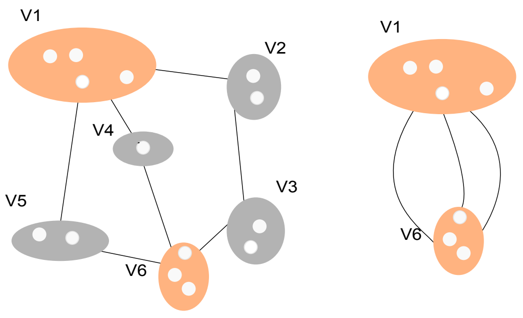


Figure 2.11: Example for applying graph reduction on a global structure.

As a matter of fact, all solutions considered for local improvement are edge-minimal since they are derived from the crossover operator. Hence they consist of $O(r)$ edges only. The overall time complexity is bounded by $O(r \cdot d_{\max}^3)$, with $d_{\max}$ being the maximum number of nodes within a single cluster.

Node Optimization Neighborhood

This neighborhood structure [17] emphasizes the selection of the spanned nodes in the branching clusters while not modifying the global structure. When V_{red} is the set of branching clusters in a current solution S, the *Node Optimization Neighborhood* (NON) consists of all solutions S' that differ from S by exactly one spanned node of a branching cluster. A move within NON is accomplished by changing $p_i \in V_i \in V_{red}$ to $p'_i \in V_i, p_i \neq p'_i,\ i \in \{1, \ldots, r\}$. By using the graph reduction technique, spanned nodes of path clusters are computed in an optimal way. Algorithm 12 shows NON in detail.

Algorithm 12 Node Optimization (solution S)

```
compute reduced structure $S^g_{red} = \langle V^g_{red}, T^g_{red} \rangle$
forall $V_i, V_j \in V^g_{red} \wedge V_i \neq V_j$ do
    forall $u \in V_i \neq p_i$ do
        change used node $p_i$ of cluster $V_i$ to $u$
        forall $v \in V_j$ do
            change used node $p_j$ of cluster $V_j$ to $v$
            if current solution better than best then
                save current solution as best
            restore initial solution
restore and return best solution
```

Updating the objective value for a considered neighbor can be done in $O(d_{\max})$ and $O(r)$ neighbors are to be considered. However, applying a graph reduction in advance adds $O(r \cdot d_{\max}^3)$ to the time complexity. This is also the overall time complexity of NON.

If V_{red} is empty, then the global structure is a round trip and all spanned nodes can be determined by using the shortest path calculation analogously to the generalized traveling salesman problem [16] [25] and NON is not searched at all.

Cluster Re-Arrangement Neighborhood

With this neighborhood structure I try to optimize a solution with respect to the arrangement of the clusters. Given a solution S with its global structure $S^g = \langle V^g, T^g \rangle$, let $adj(V_a)$ and $adj(V_b)$ be the sets of adjacent clusters of V_a and V_b in S^g, respectively. Moving from S to a neighbor solution S' in the *Cluster Re-Arrangement Neighborhood* (CRAN) means to swap these sets of adjacent clusters, resulting in $adj(V'_a) = adj(V_b)$ and $adj(V'_b) = adj(V_a)$ with V'_a and V'_b being the clusters in S' corresponding to V_a and V_b in S, respectively. S' can be further improved by using the shortest path calculations to re-choose the spanned nodes in the path clusters. Since doing this after each move is relatively time-expensive, the graph reduction is used again to enhance the performance. Whenever the arrangement of two path clusters is swapped, it is possible to only apply incremental updates on the paths that contain them. However, if at least one of these clusters is a branching cluster, the graph reduction procedure must be completely re-applied as the structure of the whole solution graph may change. The pseudocode is given in Algorithm 13.

Algorithm 13 Cluster Re-Arrangement (solution S)

```
compute reduced structure $S^g_{red} = \langle V^g_{red}, T^g_{red} \rangle$
for $i = 1, \ldots, r-1$ do
    for $j = i+1, \ldots, r$ do
        swap adjacency lists of nodes $p_i$ and $p_j$
        if $V_i$ or $V_j$ is a branching cluster then
            recompute reduced solution $S^g_{red} = \langle V^g_{red}, T^g_{red} \rangle$
        else
            if $V_i$ and $V_j$ belong to the same reduced path $\mathcal{P}$ then
                update $\mathcal{P}$ in $S^g_{red}$
            else
                update the path containing $V_i$ in $S^g_{red}$
                update the path containing $V_j$ in $S^g_{red}$
        if current solution better than best then
            decode and save current solution as best
        restore initial solution and $S^g_{red}$
restore and return best solution
```

2.9 Population Management

There is not much literature about population management techniques. Some theoretical ideas can be found in [35]. It has been observed that the quality of metaheuristics is the result of cooperation between $intensification$ and $diversifiation$ strategies. The use of the memetic algorithm can upset the balance between those two factors. Although crossover and mutation operators decrease and increase diversity, involvement of local improvement can lead to premature convergence. So the question is how to prevent the premature convergence and how to explore the search space exhaustive as possible. The already known diversity measure involved in Genetic Algorithms in the context of fitness sharing, crowding etc. can be applied, however I used two new population management techniques. The test results presented in the thesis proves the success of an implemented strategy that actively controls the diversity of the population.

Delta Population Management

This strategy especially puts emphasis on the quality of the solution S_N and on the diversity of the whole population after adding the new solution to population P. In order to check if the new solution S_N sufficiently diverses the population P, a parameter $dist$ called distance has been defined, which for any pair of solutions determines their relative distance (or similarity). The common failure is to measure the objective function space instead of the *solution space*. When designing the distance measure it is necessary to consider the problem to be solved and the encoding of solutions as well. Designing distance measure independently from the problem would not provide a desired results [35]. In considering the GMVBCNP the Hamming distance has been applied to measure the diversity between two solutions. Of course another technique, like Minkowsky distance could be applied as well. By defining the distance measure that can calculate the distance between any two solutions, we achieve

$$dist_P = min_{S_i \in P} dist(S_k, S_i)$$

where S_k is the distance of the given solution to the population. As mentioned above the Hamming distance has been involved for distance measurement, so first the vector solution.data is examined and if the values on the relevant vector's index differs then I increase $dist_P$ by 1. In the next step the adjacent lists are compared. I increase the $dist_P$ by 1 respectively only if the values at certain indices are different.
Of course the solution that has a small distance to another solution would not contribute much to the diversity of the whole population. Therefore a *diversity parameter* $\triangle$ has been introduced, in order to control if a certain solution can be inserted to the population or if it should be evolved further to fulfill a given criteria. Assume the quality of S_k is sufficient, it can be added if the below formula holds:

$$dist_P(S_k) = min_{S_i \in P} dist(S_k, S_i) \geq \triangle.$$

As can be seen in Algorithm 3, if the quality of the solution does not allow the insertion operation into the population, the solution is mutated until the distance between solutions is sufficient. However it was observed that for some instances it takes a long time to evolve the solution that has sufficient distance to the population, so it was decided to introduce a benchmark, that stops the mutation after n runs, and the solution is simply added to population. Parameter $\triangle$ allows to control of the diversity of the population, however the value of that parameter strongly affects the quality of the population. Setting it to low would decrease and setting it to high would increase diversity too quickly. In case only the solutions with a large distance will be introduced, after a few iterations it could lead to the population consisting of very different solutions, which is not the scope. The diversity parameter should be neither too large nor too small, its value should be well balanced. It can be changed dynamically during the execution of the algorithm. The initial value of $\triangle$ is set to 10% of the clusters' number, but during the running time the parameter is decreased or increased, depending on the population diversity that is examined by the second technique *edge-percentage population management.*

When performing tests, the population size equal to 100 individuals was used. In order to save computation time, when *delta Population Management* is called, solution is compared with

10% - 20% of population size. Of course this percentage can be changed but one has to be aware of the increasing time complexity.

Edge-Percentage Population Management

The second implemented technique examines the percentage of the covered edges in the population P among all possible edges. In order to calculate the percentage of all the edges covered in population the matrix *global matrix* M_g has been declared. It stores all edges covered by the population. As only the population is created and population management is involved the *global matrix* M_g is updated. To save the computation time, it is initialized only once, before the first generation will run. Each time the new edge (u, v) is introduced or deleted from the single solution S, the *global matrix* M_g is affected by deletion or addition of this edge (u, v). The percentage $edgpercent$ of covered edges is calculated every k iterations and compared with the input parameter *percent*. If the $edgepercent < percent$, then parameters $(alfa, beta, gamma)$, used in repair method, are set to (0.2, 0.2, 0.6) to increase population diversity. If $edgepercent \geq percent$ which means that diversity reaches a certain limit then $(alfa, beta, gamma)$ are set to (0.5, 0.3, 0.2). The value of parameter $percent$ changes dynamically during the execution of the algorithm, because after each *k* iterations more edges are covered, so the parameter $percent$ should change accordingly. In turn the parameter $edgepercent$ assists in defining the value of the diversity parameter $\triangle$. This leads to the assumption that both techniques complete on each other, however they can be used independently as well.

2.10 Computational results for GMVBCNP

The MA was tested on Euclidean TSPlib instances with geographical center clustering [6, 8]. They contain 137 to 431 nodes partitioned into 28 to 87 clusters. The number of nodes per cluster varies. All experiments have been performed on 2× Dual-Core AMD Opteron Processor 2214 with 8GB RAM. The program is written in C++ and gcc version 4.1.3 is used. In order to compute average values and standard deviations, 30 independent runs have been performed for each instance. The population of the MA consists of 100 solutions. As stopping criterion I use:

- $tcgen$, denoting number of generations for termination according to convergence
- $ttime$, denoting time limit for termination
- $tgen$, denoting number of generations until termination.

The probability of mutation operator was set to p_{mut}=0.6. As no reference values are available, the results of different settings of MA for GMVBCNP are compared to each other. I have run MA for different parameter settings, testing behavior of algorithm for different local improvement probabilities p_{locim} as well as for different types of population management approaches. All solutions, generated by initialization, crossover and mutation operators, as well as local improvement methods fulfill vertex bi-connectivity constraint.

Test instances

I tested my MA on large Euclidean TSPlib instances with geographical clustering. Geographical clustering is performed as follow [8]: first the r center nodes are chosen which are located as far as possible from each other. This is obtained by selecting the first center randomly, the second center as farthest node from the first center, the third center as the farthest node from the set of the two first centers, and so on. In next step the clustering is performed by assigning to its nearest center node the remaining nodes. The considered instances are listed in Table 2.1 and the values of the columns denote names of the instances, number of nodes, number of edges and number of clusters.

Results of test instances for different p_{locimp} and pop_{mngt} settings

For each setting of algorithm, the tables show the objective values for the average values of the final solutions $\overline{C(T)}$ found during 30 runs, their standard deviations and the average CPU-time $\overline{time}$ required for the search process. In order to examine the algorithm effectiveness, different settings for p_{locimp} has been used. The results are given in Table 2.2. It is easy to notice that using p_{locimp}=0.2 significantly improves results, in comparison to those gained without local improvement. The results presented in Table 2.2 show as well that increasing the value of p_{locimp} improves the final outcome. This could mean that designed genetic operators are weaker comparing to local improvement operators. However genetic operators for this problem do not strongly concentrate on creation of individuals with minimum total connection cost. Both crossover and mutation rather focus on producing solutions undiscovered so far, which are improved by local

Table 2.1: TSPLib instances with geographical clustering [6]. Number of nodes vary for each cluster.

Instance Name	$\|V\|$	$\|E\|$	r
gr137	137	9316	28
kroa150	150	11175	30
d198	198	19503	40
krob200	200	19900	40
gr202	202	20301	41
ts225	225	25200	45
pr226	226	25425	46
gil262	262	34191	53
pr264	264	34716	54
pr299	299	44551	60
lin318	318	50403	64
rd400	400	79800	80
fl417	417	86736	84
gr431	432	92665	87

improvement. They might turn out to be better than the best ones found in the previous generations. Therefore in designed MA, local improvement approach plays a crucial role for improvement of results and its p_{locimp} that affects the outcome. This is proved by results in Table 2.4, where increase of computation time for MA without local improvement still provides worse outputs than those of MA with p_{locimp}. Returning to results in Table 2.2, an increase of p_{locimp}, affects the computation time significantly. Figures 2.12 and 2.13 present the relative results of $\overline{C(T)}$ and $\overline{time}$ for individual instances, where p_{locimp}=1.0 equals 100%.

Figure 2.12 clearly shows that results obtained with parameter p_{locimp}=1.0 are better by about 1% - 4.5% from those with p_{locimp}=0.2. However, on the other hand diagram on Figure 2.13 shows that computation time can be twice as large for p_{locimp}=1.0. In order to find the balance between the running time and achieved results, p_{locimp}=0.2 seems to be a good choice. The comparison of diagrams presented on Figures 2.14 , 2.15, 2.16, 2.17, 2.18, 2.19 proves that results gained with p_{locimp}=0.2 are the most optimal when using fast runs. Even when computation time was increased with 50% the results for p_{locimp}=1.0 were still worse. The poor results, gained with p_{locimp}=1.0 comparing to the results gained with p_{locimp}=0.2 within the same computation time, are caused by the expensive local improvement approaches. So it is possible to obtain better results with lower local improvement probability using short running times as for the higher value of p_{locimp}. Table 2.2 presents outcomes gained with termination condition tcgen 200, however using ttime as termination condition and results presented on Figures 2.14 , 2.15, 2.16, 2.17, 2.18, 2.19 proves legitimacy for p_{locimp}=0.2 as a balanced choice.

The next set of tests was performed with different settings of population management approaches. Both approaches provide better results in comparison to MA without population management, see results in Table 2.3. The best results were obtained when involving both population management types, however edge management is highly time consuming because each operation

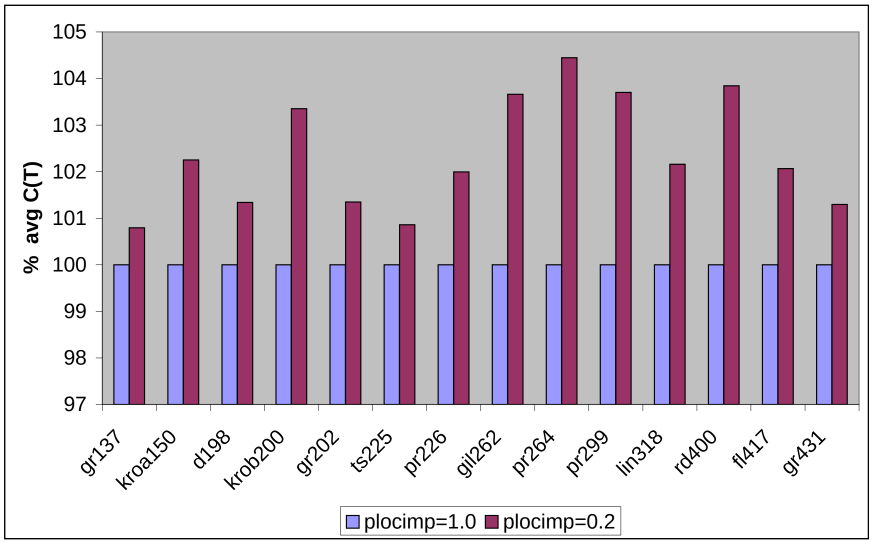


Figure 2.12: Relative results of $\overline{C(T)}$ on TSPlib instances for p_{locimp}=1.0 (without pop_{mngt}) and p_{locimp}=0.2 (with delta pop_{mngt}),where p_{locimp}=1.0 equals 100%

for insertion or deletion of a single edge must be reflected in global edge matrix and frequency matrix, that stores the frequency of edge occurrences in the population set. The computation time for edge management gets doubled for small instances and tripled for large instances in comparison to those for delta management, what presents Figure 2.20. Therefore from the computation time perspective the delta management seems to be more efficient.

Please note here that in delta management the new created solution is compared only with 10% of population's individuals (delta percent parameter) and on the basis of obtained distance value the new solution is either added to population or mutated until the termination condition is fulfilled. The increase of percentage value (delta percent) would lead to higher computation time, nevertheless 10% of trials is sufficient for delta management. Please refer to Figure 2.21. Considering the compromise between best found solutions and computation time, the best parameter setting is delta population management with 10% as population comparison parameter and local improvement set to p_{locimp}=0.2. The MA was tested for different termination conditions as well. The used $tcgen$ = 200 is sufficient for algorithm to convergence. Naturally the increased computation time leads to better results, what is obvious, as for $ttime$ and $tgen$ termination conditions in Table 2.5. On Figure 2.22 the behavior of objective function for different p_{locimp} settings can be seen. The increased value of p_{locimp} leads to better results in the begin generations, however at some point objective value flirts around the same approximate values for p_{locimp}=1,

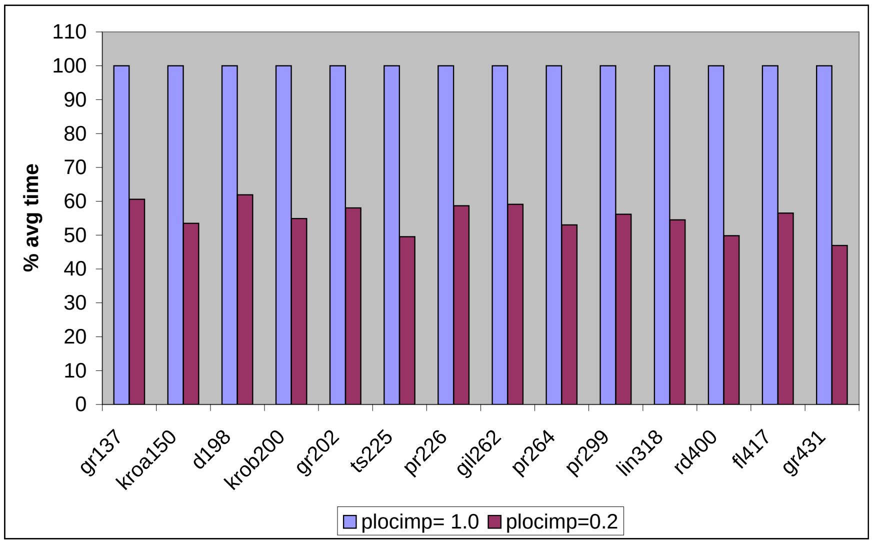


Figure 2.13: Relative results of $\overline{time}$ on TSPlib instances for p_{locimp}=1.0 (without pop_{mngt}) and p_{locimp}=0.2 (with delta pop_{mngt}), where p_{locimp}=1.0 equals 100%

p_{locimp}=0.6, p_{locimp}=0.2. Finally, as the Figure 2.22b proves, for short runs the results obtained with p_{locimp}=0.2 are better than results obtained with p_{locimp}=1. As no reference results are available, it was not possible to compare the obtained and presented results here with different ones.

Table 2.2: Results of MA: different p_{locimp}, delta pop_{mngt}, popsize 100, p_{mut}=0.6.

Instance	p_{locimp}=0			p_{locimp}=0.2			p_{locimp}=0.6			p_{locimp}=1.0		
Name	$\overline{C(T)}$	std dev	$\overline{time}$	$\overline{C(T)}$	std dev	$\overline{time}$	$\overline{C(T)}$	std dev	$\overline{time}$	$\overline{C(T)}$	std dev	$\overline{time}$
gr137	476.5	17.8	1.5	444.4	4.7	2	441.3	2.5	2.9	**440.8**	1.7	3.9
kroa150	12994.5	691.1	2.1	11809.5	289.0	2.3	11619.8	150.3	3.6	**11601.8**	90.7	4.7
d198	11708.1	407.8	3.8	10840.6	115.4	5.2	10753.6	102.9	7.6	**10730.6**	97.9	9.3
krob200	15153.4	655.9	3.9	13896.5	325.1	4.5	13573.2	192.4	7.2	**13531.6**	167.5	9.2
gr202	355.7	11.0	3.4	323.1	4.9	4.7	320	2.3	7.3	**319.7**	2.5	9.1
ts225	78642.1	3100.0	4.7	70296.9	754.5	5.4	69910.2	307.4	8.9	**69768.5**	332.9	13
pr226	75712.8	3632.2	4.3	66939.4	1186.9	5.4	66687.1	966.5	7.4	**66126.3**	980.8	9.9
gil262	1351.7	102.3	6.7	1132.3	27.2	10.4	1108.3	23.2	15.3	**1099.1**	18.9	20.7
pr264	36185.4	1799.4	7	32031.5	511.1	10.5	31409.1	847.7	17.4	**30860.8**	816.1	23.2
pr299	29313.8	2361.7	9.3	23801.5	682.3	14.1	23313.6	442.2	22.2	**23016.5**	325.0	28.1
lin318	28387	1927.6	9.7	21811	396.2	15.1	21562.3	373.3	24.8	**21368.9**	189.5	31.9
rd400	9678.5	689.5	15.8	7142.8	137.5	30.5	6958.8	122.5	51.2	**6887.7**	114.8	70.4
fl417	13307	844.6	18.5	10277.4	201.4	27.7	10156.3	198.4	41.5	**10129.1**	177.7	49
gr431	1716.6	120.4	18.7	1306.1	13.2	36.6	1296	10.4	60.5	**1290.4**	8.7	81.9

Table 2.3: Results of MA: different types of pop_{mngt}, popsize 100, p_{mut}=0.6, p_{locimp}**=0.2**.

Instance	without pop_{mngt}			delta pop_{mngt}			delta & edge pop_{mngt}		
Name	$\overline{C(T)}$	std dev	$\overline{time}$	$\overline{C(T)}$	std dev	$\overline{time}$	$\overline{C(T)}$	std dev	$\overline{time}$
gr137	445.8	6.3	1.8	**444.4**	4.7	2	**444.4**	5.2	3.9
kroa150	11898.2	418.2	2.3	11809.5	289.0	2.3	**11618.3**	110.2	5.3
d198	10836.2	125.1	4.7	10835.2	115.4	5.2	**10798.4**	124.9	12.7
krob200	13811.9	362.1	4.8	13896.5	325.1	4.5	**13658.4**	197.7	10.8
gr202	326.5	6.1	4.4	323.1	4.9	4.7	**321.9**	4.1	10.3
ts225	70520.5	765.2	5.5	70296.9	754.5	5.4	**70092.1**	517.6	14.4
pr226	**66215.1**	1247.7	4.8	66939.4	1186.9	5.4	66852.2	816.9	16.1
gil262	1145.2	34.3	9.2	**1132.3**	27.2	10.4	1140.7	31.3	26.8
pr264	31765.4	940.9	9.9	32031.5	511.1	10.5	**31661.1**	760.6	29.6
pr299	24258.9	964.7	12.6	23801.5	682.3	14.1	**23727.3**	532.6	37.7
lin318	21917.6	514.2	14.3	21811	396.2	15.1	**21733.0**	358.1	42.2
rd400	7273.1	190.2	28.9	7142.8	137.5	30.5	**7081.8**	149.1	82.3
fl417	**10193**.5	291.9	26.7	10277.4	201.4	27.7	10212.1	181.2	104.8
gr431	1309.2	17.5	34.6	**1306.1**	13.2	36.6	1308.2	12.5	111.9

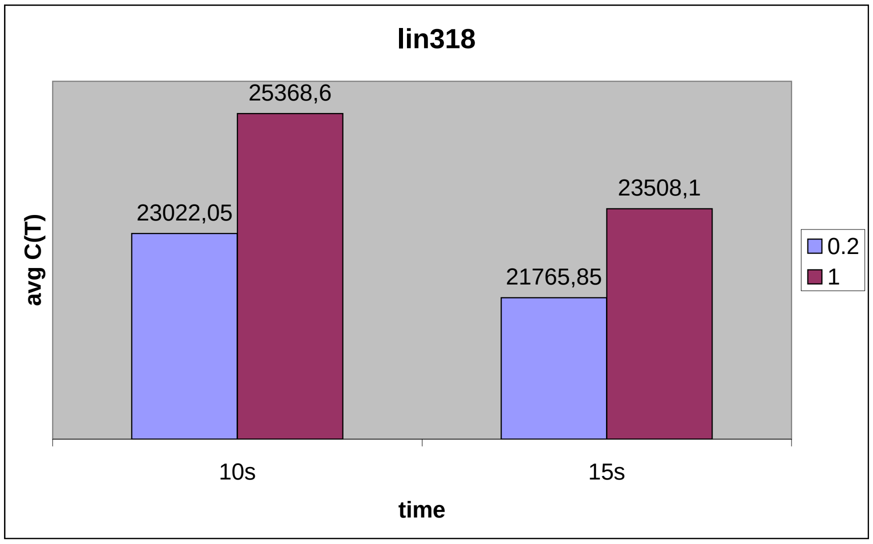


Figure 2.14: $\overline{C(T)}$ for lin318 with different p_{locimp} settings and termination condition $ttime$.

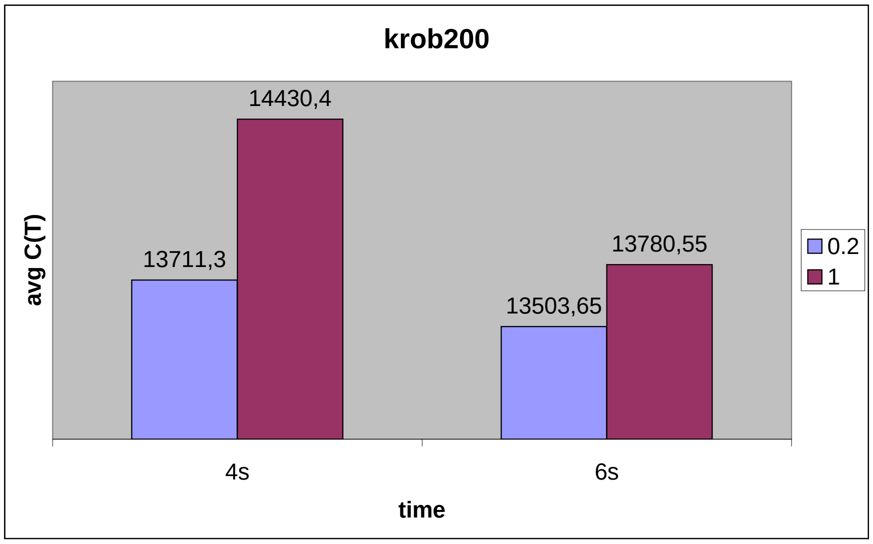


Figure 2.15: $\overline{C(T)}$ for krob200 with different p_{locimp} settings and termination condition $ttime$.

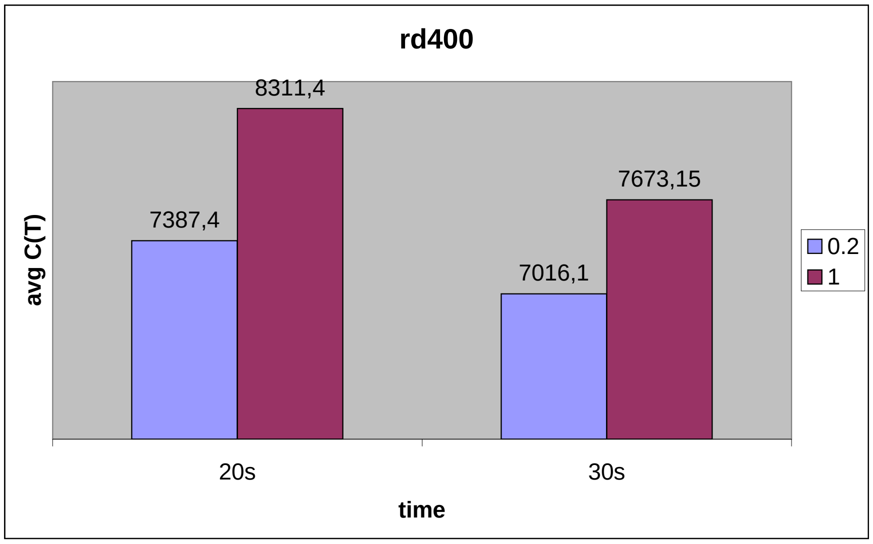


Figure 2.16: $\overline{C(T)}$ for rd400 with different p_{locimp} settings and termination condition $ttime$.

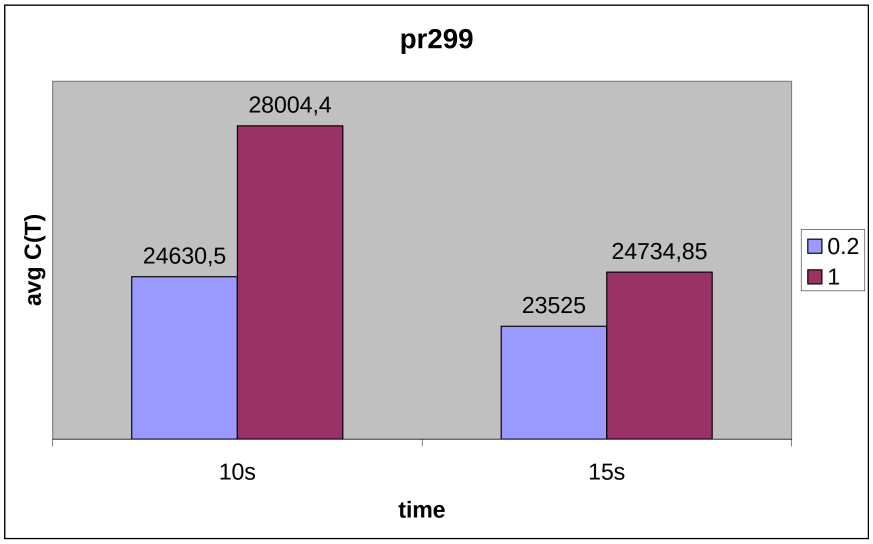


Figure 2.17: $\overline{C(T)}$ for pr299 with different p_{locimp} settings and termination condition $ttime$.

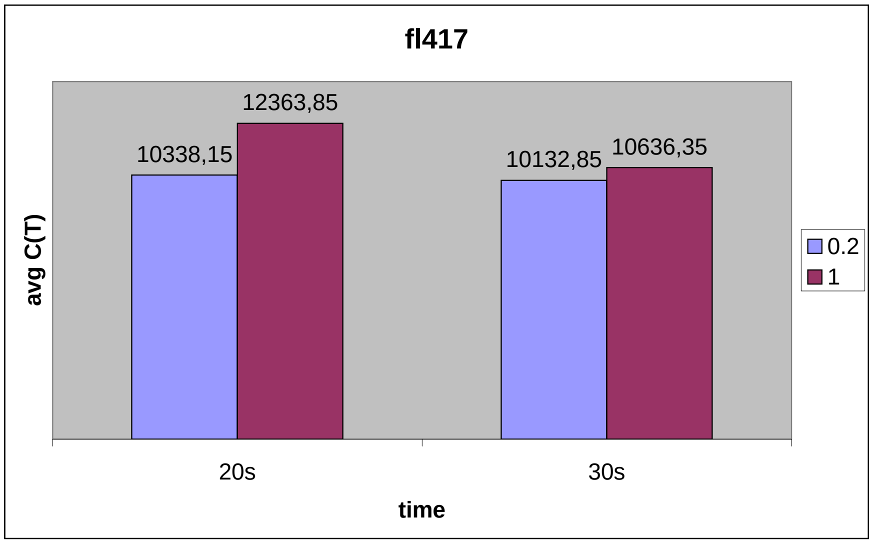


Figure 2.18: $\overline{C(T)}$ for fl417 with different p_{locimp} settings and termination condition $ttime$

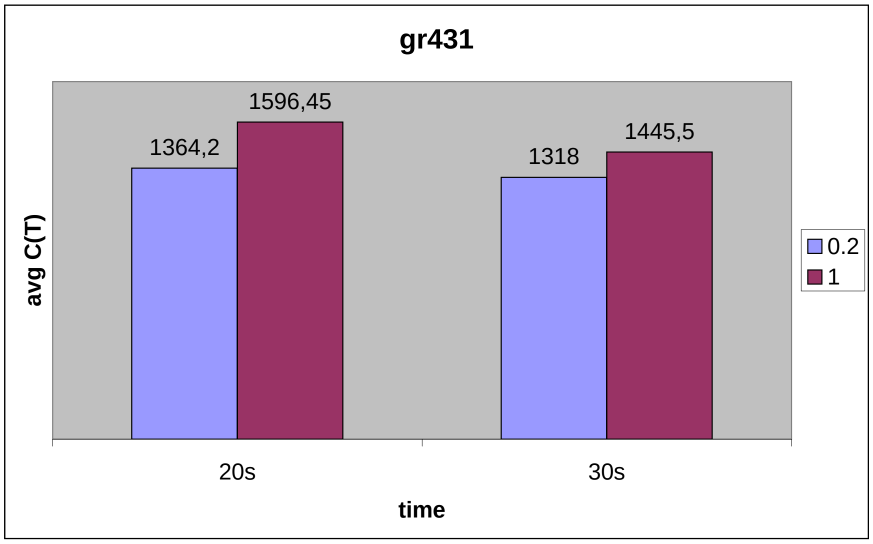


Figure 2.19: $\overline{C(T)}$ for gr431 with different p_{locimp} settings and termination condition $ttime$.

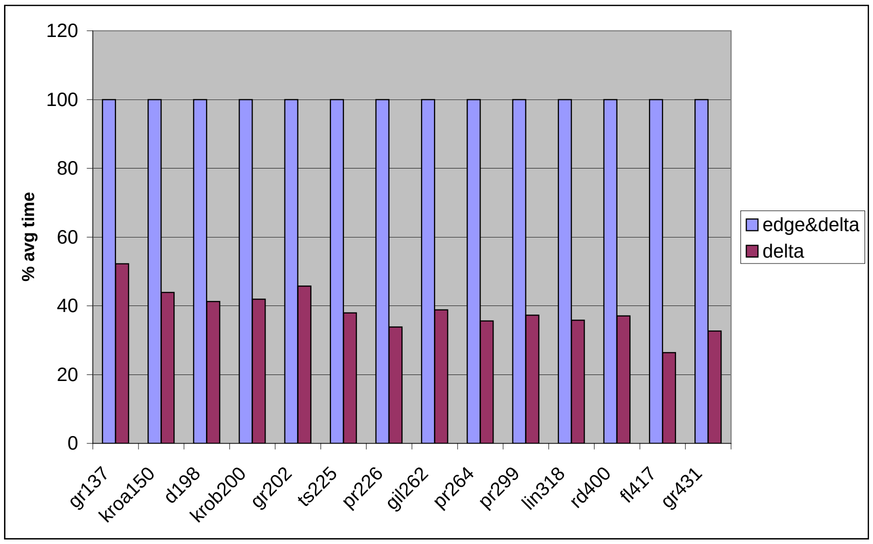


Figure 2.20: Relative results of $\overline{time}$ on TSPlib instances for p_{locimp}=0.2 with delta and edge pop_{mngt}, where $edge\&delta$ equals 100%

Table 2.4: Results on MA: with differentp_{locimp}, **without** pop_{mngt}, popsize 100, p_{mut}=0.6

Instance	p_{locimp}=0.0, time=60s			p_{locimp}=0.0, tcgen=200			p_{locimp}=0.2, tcgen=200		
Name	$\overline{C(T)}$	std dev	time	$\overline{C(T)}$	std dev	$\overline{time}$	$\overline{C(T)}$	std dev	$\overline{time}$
gr137	452.5	7.6	60	476.5	16.9	1.5	**445.8**	6.3	1.8
kroa150	12088.9	539.7	60	13113.5	881.5	2	**11898.2**	418.2	2.3
d198	11108.6	174.1	60	11734	308.0	4.1	**10836.2**	125.1	4.7
krob200	14020.4	434.1	60	15498	1125.5	3.7	**13811.9**	362.1	4.8
gr202	332.4	7.0	60	354.6	9.8	3.3	**326.5**	6.1	4.4
ts225	72616	1977.0	60	80461.6	4594.0	4.7	**70520.5**	765.2	5.5
pr226	68318.4	1424.5	60	76582.5	3620.2	4.3	**66215.1**	1247.7	4.8
gil262	1177.6	29.5	60	1358.9	74.6	6.4	**1145.2**	34.3	9.2
pr264	32693	655.5	60	36226.8	2075.0	7.2	**31765.4**	940.9	9.9
pr299	25119.9	743.1	60	29379.9	2312.7	9.2	**24258.9**	964.7	12.6
lin318	23457.4	801.7	60	27429.2	1928.5	10	**21917.6**	514.2	14.3
rd400	7799.7	246.9	60	9385.3	502.8	15.2	**7273.1**	190.2	28.9
fl417	10794.8	405.0	60	13080.9	823.8	16.4	**10193.5**	291.9	26.7
gr431	1395.6	40.6	60	1693	89.5	17.3	**1309.2**	17.5	34.6

Table 2.5: Results on GMVBCNP, MA with different termination conditions, popsize 100, p_{mut}=0.6, p_{locimp}=0.2, delta pop_{mngt}

Instance	MA with tgen				MA with tcgen			MA with time		
Name	$tgen$	$\overline{C(T)}$	std dev	$\overline{time}$	$\overline{C(T)}$	std dev	$\overline{time}$	$time$	$\overline{C(T)}$	std dev
gr137	2000	441.7	2.5	2.6	444.4	4.7	2	100s	**441.5**	3.2
kroa150	2000	**11596.5**	113.9	3.6	11809.5	289.0	2.3	100s	11602.2	182.4
d198	2000	10739.3	88.4	6.4	10835.2	115.4	5.2	200s	**10724.1**	83.2
krob200	4000	13483.9	198.7	8	13896.5	325.1	4.5	200s	**13428.2**	198.6
gr202	4000	321.9	3.6	8.4	323.1	4.9	4.7	200s	**321.5**	3.5
ts225	4000	**69909.9**	372.8	10.9	70296.9	754.5	5.4	200s	70037.7	398.8
pr226	4000	66161.9	619.9	7.8	66939.4	1186.9	5.4	200s	**65644.3**	842.8
gil262	4000	1102.9	20.5	15.5	1132.3	27.2	10.4	200s	**1098.9**	19.0
pr264	4000	31205.2	726.9	15.6	32031.5	511.1	10.5	200s	**30995.5**	701.9
pr299	6000	23119.5	386.4	23.3	23801.5	682.3	14.1	300s	**23010.3**	497.5
lin318	6000	21450.7	252.9	25.6	21811	396.2	15.1	300s	**21432.7**	196.9
rd400	6000	6941.4	95.7	52	7142.8	137.5	30.5	300s	**6893**	127.2
fl417	10000	**9955.3**	163.4	55.4	10277.4	201.4	27.7	600s	10006.8	178.5
gr431	10000	1288.7	8.2	67.7	1306.1	13.2	36.6	600s	**1287.8**	7.7

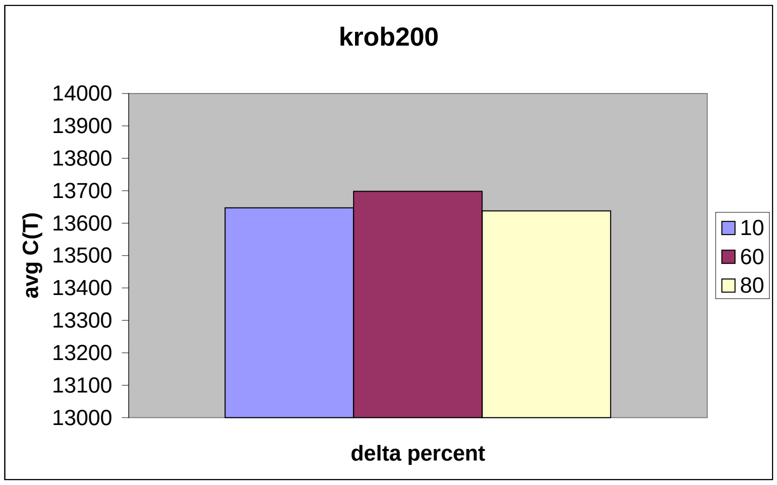


Figure 2.21: $\overline{C(T)}$ for krob200 with different $deltapercent$ settings and termination condition $tcgen$=200.

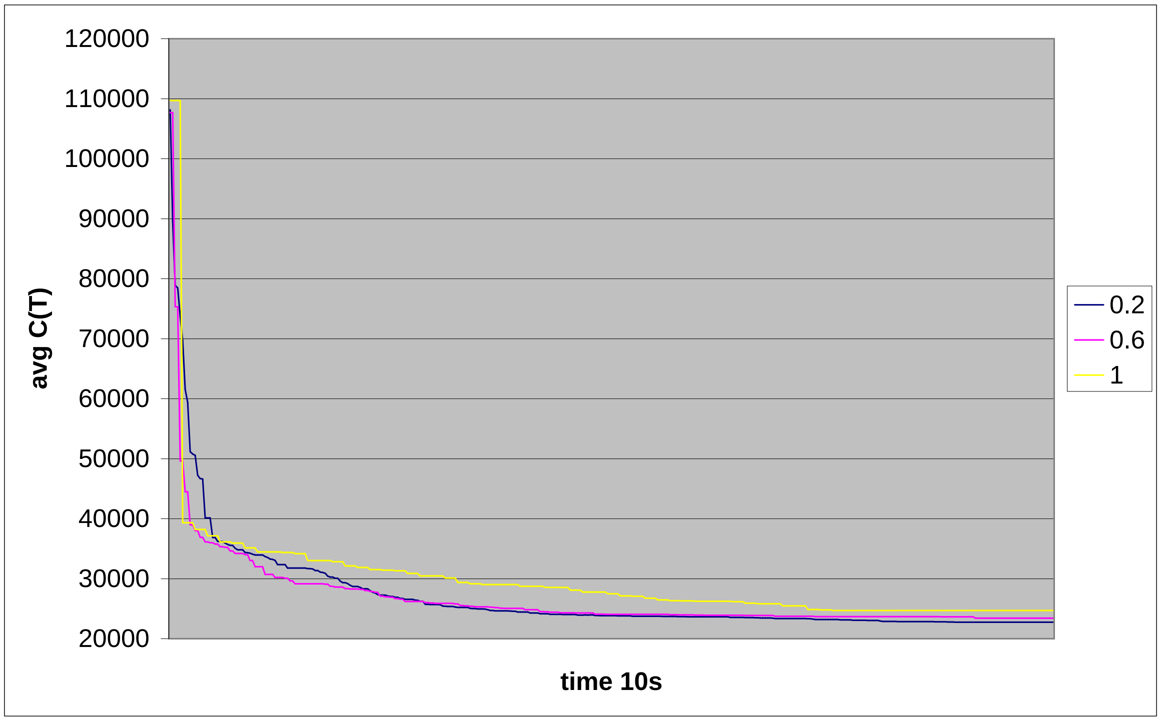


Figure 2.22: The behavior of $\overline{C(T)}$ in 20 runs, for lin318 with different p_{locimp} settings and termination condition $ttime$ 10s.

3 The Generalized Degree Constrained Minimum Spanning Tree Problem

3.1 Problem formulation

The **Generalized Degree Constrained Minimum Spanning Tree (d-GMST)** problem is an extension of the classical Minimum Spanning Tree Problem. In this problem the generalized minimum spanning tree is searched and none of its vertices is allowed to have degree grater than $d_{max} \geq 2$. The problem is defined as follow.

Consider an undirected, weighted complete graph $G = (V, E, c)$ with node set V, edge set E, edge cost function $c : E \to \mathbb{R}^+$. The node set V is partitioned into r pairwise disjointed clusters V_1,V_2,...,V_r containing $n_{1,}n_{2,}...n_r$ nodes respectively. A spanning tree of a graph is a cycle- free subgraph connecting all nodes. A solution to the d-GMST problem defined on G, Figure 3.1, is a graph $S = (P,T)$ with $P = \{p_1, p_2, \ldots, p_r\} \subseteq V$ containing exactly one node from each cluster, $p_i \in V_i$ for all $i = 1, 2, .., r$ and $T \subseteq P \times P \subseteq E$ being the tree spanning nodes P. In the d-GMST problem an additional constraint called degree $deg(u)$ of every vertex $u \in P$ is considered. The degree is the number $deg(u) \leq d_{max} \wedge u \in P$, where d_{max} is a given upper bound of edges adjacent to u. Thus the **Generalized Degree Constrained Minimum Spanning Tree** is the tree that fulfills the degree constraint and simultaneously minimizes the total edge cost. Lets assign numerical costs $c_{u,v} \geq 0$ to each edge $(u, v) \in T$, where $deg(u) \leq d_{max}$ and $deg(v) \leq d_{max}$ then the Generalized Degree Constrained Minimum Spanning Tree is a spanning tree with minimum total edge cost

$$C(T) = \sum_{(u,v)\in T} c(u, v).$$

3.2 Previous work

The Generalized Minimum Spanning Tree Problem (GMSTP) as well as the Degree Constrained Minimum Spanning Tree Problem (d-MSTP) were studied already and a lot of researches have been performed for both of them. In this thesis I focused on the Generalized Degree Constrained Minimum Spanning Tree Problem (d-GMSTP), which to the best of my knowledge has not been yet addressed in the literature. As d-GMSTP is a conjunction of GMSTP and d-MSTP this chapter briefly introduces a previous works related to both of them.

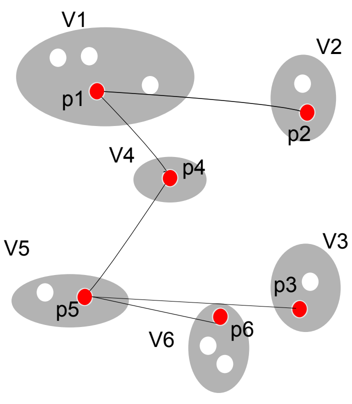


Figure 3.1: Example for d-GMST problem, where degree $d = 3$

The GMST problem was first proposed by Myung, Lee and Tcha [26] and they proved that it is NP-hard and they provided Integer Linear Programming formulation. Feremans, Labbe and Laporte provided further formulations in [7] and investigated all eight ILPs. Feremans in [6] proposed exact solution algorithms based on branch and cut approach, while Pop, Kern and Still in [28] proposed approximation algorithms and heuristics. Furthermore Pop [29] utilized underlying idea of his MIP formulation in a Simulated Annealing approach which allows solving larger instances. To solve more general and larger instances several metaheuristics has been suggested by Ghosh [9]. In his paper he studied performance of neighborhood searches based on tabu search and variable neighborhood search (VNS). He implemented and compared Tabu Search that incorporates recently based memory and aspiration rules, Tabu Search (TS2) that incorporates frequency based memory, reduced Variable Neighborhood Descent, VNS that combines deterministic and stochastic changes of neighborhoods and Variable Neighborhood Decomposition search (VNDS) that does not look at the whole neighborhood of investigated problem only does an exhaustive search on the subset of clusters. Comparing this approaches Ghosh concluded that TS2 and VNDS perform best on the average. Golden, Raghavan and Stanojevic [12] presented two heuristic search techniques - local search and a genetic algorithm. They proposed as well a simple lower and upper bound heuristics for GMST problem. Hu, Leitner and Raidl [15] proposed VNS approach which uses three different types of neighborhoods. Two of them work in complementary ways in order to maximize search effectiveness whereas for the third they applied Mixed Integer Programming (MIP) to optimize local parts within candidate tree's solution.

The application of d-MST in the design of electrical circuits has been pointed out by Narula and Ho [27]. They proposed primal and dual heuristic procedures and a branch and bound algorithm. The branch and bound algorithm which makes use of an edge elimination procedure based on edge exchange has been proposed by Savelsbergh and Volgenant [33]. Volgenant [36] proposed also edge exchanges used in branch and bound algorithm based on Lagrangian relaxation. Ribeiro and Souza [32] proposed a VNS based on dynamic neighborhood model and using a variable neighborhood descent iterative improvement algorithm for local search. Knowles and Corne introduced in [19] a novel tree construction algorithm called the Randomized Principal Method (RPM) which builds degree constrained trees of low cost from solution vectors. They applied RPM in three iterative methods: simulated annealing, multistart hillclimbing and genetic algorithm. Another several heuristics including simulated annealing and genetic algorithms has been presented by Krishnamoorthy, Ernst and Sharaiha [20]. They proposed as well a method based on problem space search (PSS), which as they concluded works best over all the problems. An efficient Evolutionary Algorithm proposed by Raidl [30] produces only a feasible candidate solutions and provides substantially stronger locality than most previous approaches. Raidl and Julstrom [31] proposed representation of spanning trees in EAs directly as sets of their edges and developed initialization, recombination, and mutation operators for this representation. The operators offer locality, heritability, and computational efficiency. The adapting edges procedure presented further in this thesis has been inspired by [5]. Feketea, Khullerb, Klemmsteina, Raghavacharic and Young considered degree constraint for a graph and proposed modification of a given spanning tree representing an unfeasible solution by using adoptions to meet the degree constraint. A novel network-flow based algorithm has been introduced as well.

3.3 Neighborhood Structures

This section introduces description and basis of common approaches for both VNS and MA implemented for the Generalized Degree Constrained Minimum Spanning Tree Problem (d-GMSTP). When considering the d-GMSTP, it is possible to divide it into two subproblems:

- compute the solution based on selection of nodes from the clusters
- compute the solution based on the connections between clusters.

The subproblems above define two different strategies, their corresponding solution representations and their neighborhood structures. However the structures proposed in [21] has been modified to satisfy degree constraint considered in this thesis.

n-Node Exchange Neighborhood

The idea of this approach is to first select one node per cluster [9]. When the nodes are selected we can apply either the classical Kruskal Algorithm and adapting edges procedure (AEP) afterwards to fix the nodes violating degree constraint or the degree constrained Kruskal Algorithm (d-Kruskal) producing feasible solutions representing the Generalized Degree Constrained Minimum Spanning Tree on selected nodes. The details of both approaches AEP and d-Kruskal are presented in the next section. The solution is stored as vector $P = \{p_1, p_2, \ldots, p_r\}$ where r denotes the number of clusters and p_i denotes the node in cluster V_i, $i = 1, ..., r$.

This representation is a basis for the used neighborhood structure called 1-Node Exchange Neighborhood (1-NEN), that was originally proposed by Ghosh [9] and successfully involved by [21]. In this neighborhood we can determine all $|V|$ solutions derived from exchange of node in vector P, that is precisely for one cluster V_i the node p_i is replaced by a different node p'_i of the same cluster. Since a single MST can be computed in time $O(r^2)$, e.g. by Prim the evaluation of the whole neighborhood to find the best neighboring solution can be derived in $O(|V| \cdot r^2)$ time when involving d-Kruskal or using the classical Kruskal Algorithm and AEP the upper bound equals $O(|V| \cdot r^2 \cdot m\, log\, m)$, where m is the number of all possible edges of the graph G.

The NEN structure can be generalized by simultaneously replacing $n \geq 2$ nodes. In this thesis the version with exchange of two nodes (2-NEN) is also used as neighborhood structure. However this structure considers only the pairs of clusters that are adjacent in current solution and is the most expensive one. Its complete evaluation takes a lot of time for large instances, therefore I terminate its exploration after certain time limit. It returns the so-far best solution instead of the best solution. For more details concerning those structures please refer to [21].

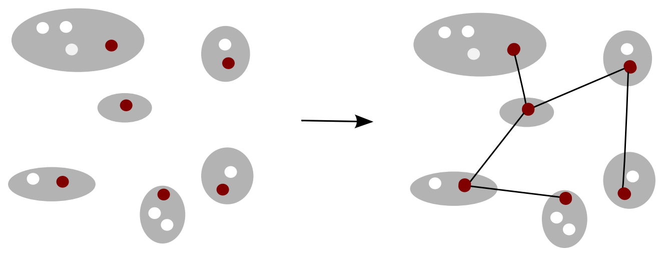


Figure 3.2: Finding solution by approach proposed by Ghosh.

Figure 3.2 presents the example of approach proposed by Ghosh: Given the selected nodes, one of the approaches, the classical Kruskal with AEP or d-Kruskal algorithm determines the most suitable edges in order to find d-GMST.

Edge Exchange

This neighborhood is based on so-called global graph therefore its formal definition is introduced here. Given a clustered graph $G = (V, E)$ the *global graph* denoted by $G^g = (V^g, E^g)$ consists of nodes corresponding to clusters in G, i.e. $V^g = V_1, V_2, ..., V_r$ and edge set $E^g = \{(V_i, V_j) \mid \exists (u, v) \in E \wedge u \in V_i \wedge v \in V_j\}$. Each global connection (V_i, V_j) represents all edges $\{(u, v) \in E \mid u \in V_i \wedge v \in V_j\}$ of graph G.

Pop [29] has shown that the reverse process to NEN can be used: starting from the spanning tree with given selected global connections of so-called "global graph" it is possible to determine optimal nodes. The set of possible solutions is quite huge, however the search space can be limited by implementation of dynamic programming. The solution in this neighborhood is represented by the set of edges of global graph. The basic idea of this neighborhood is to remove one of the solution's edges, that divides the spanning tree into two subtrees and in the next step to introduce a new edge connecting subtrees in minimum cost way. As in this thesis the Generalized Degree Constrained Minimum Spanning Tree Problem is considered the new global edge (V_i, V_j) can be introduced to solution S' only if $deg(V_i) < d_{max} \wedge deg(V_j) < d_{max}$. The size of this neighborhood depends basically on the number of edges $O(r)$ that can be removed and it requires $O(r^2)$ moves in order to exploit all possible ways of reconnecting the resulting two components. In order to evaluate all neighbors in an efficient way, incremental dynamic programming has been applied. The whole dynamic programming is performed only once at the beginning and all costs are kept. The data are updated incrementally only for each considered move and the values of cluster V_i are recalculated only if it gets a child, loses a child or the costs of its successor change. The pseudocode of Algorithm 14 presents the neighborhood of Edge Exchange (EEN) that involves dynamic programming. The main idea is to root the global spanning tree at an arbitrary cluster

$V_{root} \in V^g$ and to direct all edges towards the leafs. Then the tree is traversed in recursive depth-first manner and the minimum costs for the subtree rooted at V_k are calculated for each cluster $V_k \in V^g$ and for each node $v \in V_k$ when v is the node to be connected from V_k. Let us consider global graph $G^g = \langle V^g, E^g \rangle$ consisting of nodes corresponding to clusters in G, $V^g = \{V_1, V_2, ..., V_r\}$ and edge set $E^g = \{(V_i, V_j) \mid \exists (u, v) \in E \wedge u \in V_i \wedge v \in V_j\}$. Let us consider now a spanning tree $S^g = \langle V^g, T^g \rangle$ on this global graph. This tree represents the set of all feasible generalized spanning trees on G which contains for each edge $(V_a, V_b) \in T^g$ a corresponding edge $(u, v) \in E$ with $u \in V_a \wedge v \in V_b \wedge a \neq b$. Then we can determine the minimum costs of a subtree by following recursion:

$$C(T^g, V_k, v) = \begin{cases} 0 \;\; if\, V_k\, is\, a\, leaf \\ \sum_{V_l \in Succ(V_k)} min_{u \in V_l} \{c(v, u) + C(T^g, V_l, u)\} \; otherwise \end{cases}$$

where $Succ(V_k)$ identifies the set of all successors of V_k in T^g. For more details please refer to [15].

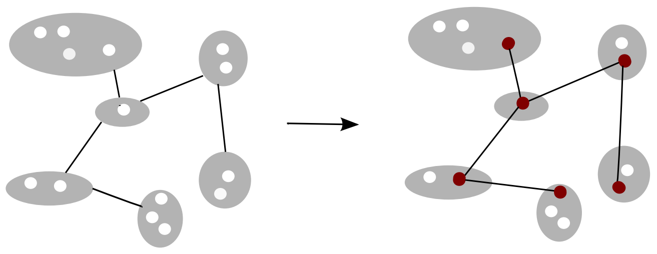


Figure 3.3: Finding solution by approach proposed by Pop.

Figure 3.3 presents the approach proposed by Pop: Given the so-called "global graph", the optimal nodes are determined.

3.4 Adapting edges and degree constraint Kruskal

In this thesis I consider the degree constraint for Generalized Minimum Spanning Tree Problem, therefore it was necessary to use an algorithm that examines if all nodes u in the solution $S = (P, T)$ have degree $deg(u) \leq d_{max}$, $\forall u \in P$, where d_{max} is a given upper bound of global edges adjacent to u.

In overall the degree of each cluster is stored in a vector $degree$ separate one for each single solution. The degree value is updated each time when an edge is removed or added to the solution. Before an edge is introduced to create a feasible solution, it is examined if it will not exceed the degree constraint and if it will not introduce a cycle. I propose two approaches for solving the d-GMSTP: Adapting Edges Procedure (AEP) and degree constrained Kruskal (d-Kruskal) algorithm.

Algorithm 14 Edge Exchange Neighborhood with degree constraint

```
forall global edges $(V_i, V_j) \in T^g$ do
    remove $(V_i, V_j)$
    $M_1$ =list of clusters in component $K_1^g$ containing $V_i$
    $M_2$ =list of clusters in component $K_2^g$ containing $V_j$
    forall $V_k \in M_1$ do
        root $K_1^g$ at $V_k$
        forall $V_l \in M_2$ do
            root $K_2^g$ at $V_l$
            if $degree\,(V_k) < d_{max} \wedge degree\,(V_l) < d_{max}$ then
                add $(V_k, V_l)$
                use incremental dynamic programming to compute the solution and the objective
                value
            if current solution better then best then
                save current solution as best
            remove $(V_k, V_l)$
restore and return best solution
```

Adapting Edges Procedure (AEP)

The pseudocode of Algorithm 15 presents the adaptation edge heuristic, that fixes the nodes violating the degree constraint. The effectiveness of this method has been tested in conjunction with memetic algorithm, called further a-MA. It is run after the classical Kruskal algorithm, if any node in the solution S' violates degree constraint.

Before the search for a suitable edge that can be added is performed I create the list of all possible edges $ALLEDGES$, sorted by cost ascending, which can be introduced into solution. The list includes the set of $E = \{(u, v) \mid \exists\,(u, v) \in E \wedge u \in V_i \wedge v \in V_j\} \wedge i = 1, ..., r \wedge j = 1, ...r \wedge i \neq j$. As an input the $LIST$ containing all nodes violating the degree constraint is created as well. It is performed only once at the beginning of adapting edge procedure as any node that violates degree constraint will not be introduced by AEP. Once the input lists are created we start removing the edges with highest connection cost that are adjacent to node $u \in LIST$. When an edge (u, v) is removed the tree is divided into two components: K_1 containing u and K_2 containing v respectively. In order to connect the components we try to introduce the edge (k, l) with the lowest connection cost satisfying following constraints:

- the edge (k, l) must not exist in current solution (1),
- k and l must belong to different components (2),
- $degree\,(k) < d_{max} \wedge degree\,(l) < d_{max}$ (3).

Let us consider an example presented by solution S on Figure 3.4. The solution S posses one node in cluster V_1 that violates degree constraint and we need to remove one of the edges adjacent to this node. Let us assume that an edge (u, v), $u \in V_1 \wedge v \in V_2$ has the highest

Algorithm 15 Adapting Edges Procedure (AEP)

```
input : $LIST$ =vector storing all nodes that violates degree constraint
$ALLEDGES$ =list of all possible edges sorted by cost ascending
$t = 0$

forall nodes in $LIST$ do
    if $degree\,(u) > d_{max}, u \in LIST$ then
        remove edge $(u, v)$ adjacent to $u$ with the highest connection cost
        update $degree\,(u)$ and $degree\,(v)$
        $M_1$ =list of nodes in component $K_1^g$ containing $u$
        $M_2$ =list of nodes in component $K_2^g$ containing $v$
        repeat
            examine edge $(k, l) \in ALLEDGES[t]$
            if edge does not exist in current solution then
                if $k \in M_1$ and $l \in M_2$ then
                    if $degree\,(k) < d_{max}$ and $degree\,(l) < d_{max}$ then
                        add $(k, l)$
                        update $degree\,(k)$ and $degree\,(l)$
                        edge can be added $= true$
            $t++$
        until edge can be added $= false$
```

connection cost among the edges adjacent to node u. By removing the edge (u, v) we split the tree into two components: K_1 and K_2. Let us assume that the edge (v, l), $v \in V_2 \wedge l \in V_3$ has the lowest connection cost to merge components and satisfies the constraints (1)-(3). Therefore we say that node l adopts neighbor of node u and the final solution S' is the feasible one. AEP is generally called after crossover and initialization, as well as after local improvement in a-MA, if any node of solution S' violates degree constraint. As the list of all edges is created and sorted by their cost the procedure requires $O(m\,log\,m)$ time in the worst case, where m denotes edges of the set $ALLEDGES$.

Degree Constrained Kruskal Algorithm (d-Kruskal)

The proposed degree constrained Kruskal algorithm, Algorithm 16, builds the GMST considering simultaneously the degree constraint. An edge (u, v) is added to the tree if it does not violate the degree constraint. In some cases the set T_k of edges provided by 1-NEN or 2-NEN can be insufficient in order to construct the d-GMST, therefore it is necessary to introduce an edge from $F = E \setminus T_k$. The neighborhoods provide the set of edges created after replacement of the node p_i with node p'_i. Removing p_i and all its incident edges from the solution S forms a graph consisting of $l \geq$ components $K_1, ..., K_l$. So, the set T_k consists of:

- the edges of S after removing p_i and its incident edges,
- all edges (p'_i, p_j) with $j = 1, ..., r \wedge i \neq j$,

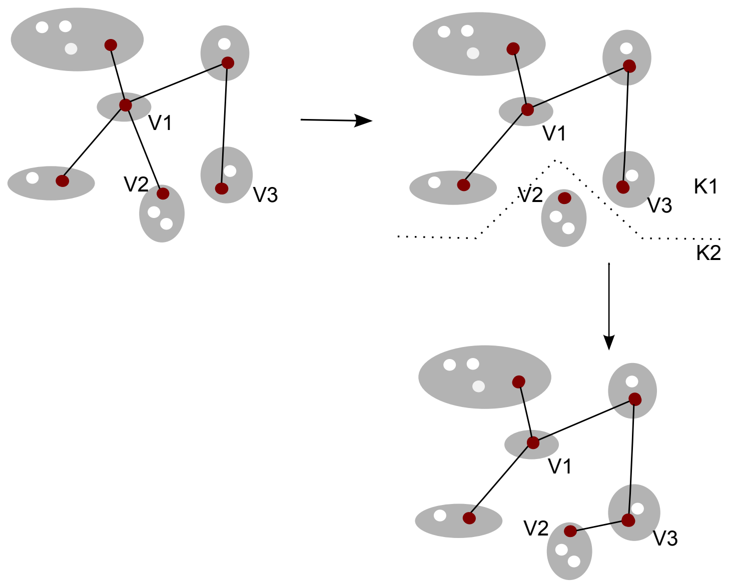


Figure 3.4: Adapting edges for d-GMST with d_{max}=3.

- the shortest edges between pairs of the components $K_1, ..., K_l$.

If the subset (p'_i, p_j) with $j = 1, ..., r \wedge i \neq j$ contains more edges with lower cost, they will be sequentially introduced to solution. The structure will be shuffled and it will be not possible to create a complete solution from the remaining subset of edges because of degree constraint.

To inherit as many edges as possible from T_k the proposed d-Kruskal algorithm assigns to each edge the $weight$ based on its cost and parameter $\beta \in (0, 0.1)$ that leads to a stronger bias towards edges from set T_k. To all edges $(V_i, V_j) \in T_k$ the $weight$ =cost$\cdot\beta$ is assigned, whereas the edges of F obtain $weight$ =cost. In the next step the edges are sorted by the $weight$ in ascending order and the d-GMST is built. The computational effort of the procedure is $O(m\, log\, m)$.

3.5 Variable Neighborhood Search

Initial solutions

The greedy Minimum Distance Heuristic (MDH) used by Ghosh [9] has been applied for initialization of solutions when running VNS. In MDH the nodes with the lowest connection cost to all nodes of other clusters are used, and the tree is spanned on those nodes, forming the minimum spanning tree, Algorithm 17. Using d-Kruskal algorithm it yields to $O(|V|^2 + r^2 \log r^2)$ time complexity.

Algorithm 16 degree constrained Kruskal (d-Kruskal)

```
input : $T = \varnothing$, $S = \varnothing$
$i$ =1
$ALLEDGES$ =list of all possible edges sorted by $weight$ ascending

forall nodes in $ALLEDGES$ do
    repeat
        if $deg\,(u) < d_{max} \wedge deg\,(v) < d_{max} \wedge (u \wedge v\,not\,conencted\,in\,T))$ then
            $T = T \cup (u, v)$
            union the sets containing $u$ and $v$
            update $degree\,(u)$ and $degree\,(v)$
        $i++$
    until $S$ has more than one set
```

Algorithm 17 Initialization of solutions in VNS for the d-GMST

```
for $i = 1, ..., r$ do
    choose node $p_i \in V_i$ with minimal $\sum_{v \in V \setminus V_{\{i\}}} c(p_i, v)$
calculate d-GMST by means of d-Kruskal on nodes $P = \{p_1, ..., p_r\}$
```

Shaking

The goal of the shaking is to perturb the solution S and to provide a good starting point S' for the local search [2]. During the local search the set of neighborhood structures is used, but the search is not restricted only to this set. At the end of local search the new solution S' is compared with S, and in case it is better it replaces S and the algorithm starts again with k =1, where k_{max} is the size of shaking. Otherwise, k is incremented and the shaking is performed. Therefore the shaking phase introduces a diversification of the search. For shaking, see Algorithm 18, I use two approaches: 1-NEN and EEN. In 1-NEN shaking starts with four moves, whereas in EEN starts with exchange of five edges, because some edges selected for insertion might be discarded due to degree infringements.

VNS Framework

In this thesis the general VNS schema with VND as local improvement is used. During VND approach it is alternated between 1-NEN, 2-NEN and EEN in the order described by Algorithm 19. This sequence has been proposed by Leitner [21] and determined according to computational complexity of searching the neighborhoods. The Algorithm 19 presented in this section considers the degree constraint. Inside EEN an edge (V_k, V_l) connecting two components K_1^g and K_2^g is introduced only if $degree(V_k) < d_{max} \wedge degree(V_l) < d_{max}$. In 1-NEN, 2-NEN the d-Kruskal recalculates the spanning tree of the new solution S'.

Algorithm 18 Shake

```
input : S, k - size

for $i = 1, ..., k$ do
    remove random edge $(V_i, V_j) \in T^g$ diving tree into components $K_1^g$ and $K_2^g$
    if $deg(V_k) < d_{max} \wedge deg(V_l) < maxdegree$ then
        insert random edge $(V_k, V_l)$ connecting $K_1^g$ and $K_2^g$
    else
        insert edge $(V_i, V_j)$
    use dynamic programming to recalculate used nodes
for $i = 1, ..., k + 2$ do
    randomly change the used node $p_i$ of randomly chosen cluster $V_i$
recalculate MST by d-Kruskal
```

3.6 Memetic Algorithm

The results obtained by running MA for GMVBCNP were satisfactory so I decided to test d-GMST using memetic algorithm as well. However for d-GMSTP I distinguish two kinds of MA, which differ from the used approach fixing the nodes violating degree constraint: *a-MA* involves AEP after crossover and local improvement, see Algorithm 20 whereas *k-MA* builds the new offspring by means of d-Kruskal algorithm, see Algorithm 21.

For selection, the standard tournament selection with tournament size of two is applied. The designed framework for both MA's is based on the steady state EA and it involves local improvement technique to improve the quality of solutions. The applied operators and approaches are described in the next subsections.

Initial solutions

The initial population is created in random order to provide as much diversity as possible. Each solution includes one random node per cluster. Once the coincidental nodes are selected in a-MA the classical Kruskal algorithm is applied in order to build a tree spanned on selected nodes. Because some of the nodes can violate degree constraint AEP is called in order to make the solutions feasible. On the other hand in k-MA once the nodes are selected the d-Kruskal creates the initial individuals. However both approaches creates a feasible initial population of individuals.

Crossover operator

For recombination a simple uniform crossover operator has been implemented. For each gene it is individually decided from which parent it will be inherited. Once the genes are selected, similar to initialization either classical Kruskal and next AEP or d-Kruskal is performed to connect selected nodes in feasible way forming d-GMST.

Algorithm 19 VND for the d-GMST

```
$l$=1
repeat
    switch $l$ do
        case 1 : 1-NEN
            for $i = 1, ..., r$ do
                forall $p'_i \in V_i \setminus p_i$ do
                    change used node $p_i$ of cluster $V_i$ to $p'_i$
                    recalculate MST by d-Kruskal
                    if current solution $S'$ is better than the best then
                        save $S'$ as the best
        case 2 : EEN
            call edge_exchange()
        case 3 : 2-NEN
            forall clusters $V_i$ and $V_j$ adjacent in current solution do
                forall $p'_i \in V_i \setminus p_i$ and $p'_j \in V_j \setminus p_j$ do
                    change used node $p_i$ of cluster $V_i$ to $p'_i$
                    change used node $p_j$ of cluster $V_j$ to $p'_j$
                    recalculate MST by d-Kruskal
                    if current solution $S'$ is better than the best then
                        save $S'$ as the best
            restore the best solution
    if solution improved then
        $l$ =1
    else
        $l = l$+1
until $l > 3$
```

Mutation Operator

The mutation operator swaps chosen nodes within clusters. The number of nodes that is exchanged is determined each time randomly. See Figure 3.5a where nodes within clusters V_2 and V_3 were exchanged. However for some solutions swapping the node leads to exchange of an edge as well, as the tree is recalculated and spanned on currently selected nodes with minimal connection cost, see Figure 3.5b.

Local improvement

The local improvement is applied with probability p_{locim}. Because the used neighborhood structures: 1-NEN and 2-NEN are very time consuming the value of p_{locim} should be kept as small as possible. For local improvement three other parameters are involved:

- $locim_{best}$, determining the use of local improvement whenever new best solution is found

- ls_{prob}, probability determining if the new found solution should be improved by local search
- $locim_{startgen}$, determining after which generation the local improvement should be used.

Usually in the begin stage the MA results are improved without involvement of local search. However after some number of generations the results are not bettered as fast as at the beginning, applying methods like local search explores the search space more efficiently. As mentioned above the applied neighborhoods are time consuming, therefore in order to save some computation time, the local improvement is applied after $locim_{startgen}$ generation.

Solution Archive

MA creates new solutions from already known solutions, therefore with increasing number of generations the solutions get more and more similar or even identical. The simplest method to avoid this issue is to compare the new solution to all solutions in the current population, however once the individual is withdrawn from population, the information about its existence is lost as well. Here comes up the idea of an archive creation, implemented by Wolf [38]. He proposed a complete solution archive for GAs that effectively transforms duplicates into similar so far unconsidered candidate solutions. I use this concept for MA as well. Each time a new solution

Algorithm 20 Memetic Algorithm for the d-GMST using AEP (a-MA)

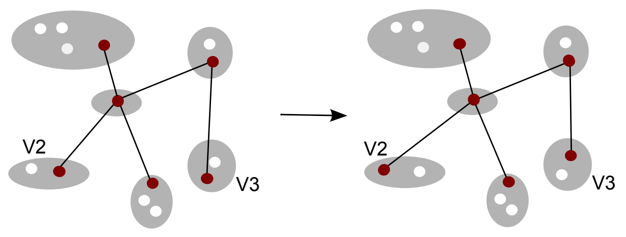
```
create random initial population $P$
AEP($P$)
repeat
    select two parental solutions $S_1 \wedge S_2 \in P$
    create a new solution $S_N$ by crossover on $S_1 \wedge S_2$
    AEP($S_N$)
    mutate a new solution $S_N$ with probability $p_{mut}$
    locally improve $S_N$ with probability $p_{imp}$
    AEP($S_N$)
    replace one parental solution by $S_N$
until termination condition
```

Algorithm 21 Memetic Algorithm for the d-GMST using d-Kruskal (k-MA)

```
create random initial population $P$ using d-Kruskal

repeat
    select two parental solutions $S_1 \wedge S_2 \in P$
    create a new solution $S_N$ by crossover on $S_1 \wedge S_2$ using d-Kruskal
    mutate a new solution $S_N$ with probability $p_{mut}$
    locally improve $S_N$ with probability $p_{imp}$ and calculate $S_N$ by d-Kruskal
    replace one parental solution by $S_N$
until termination condition
```

a)



b)

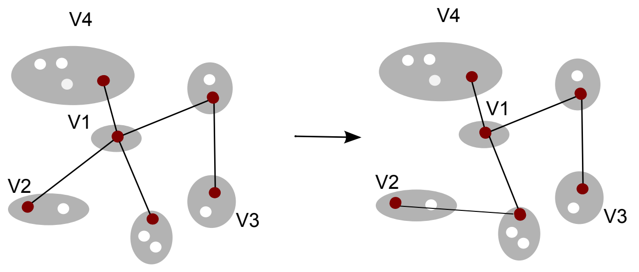


Figure 3.5: Mutation of d-GMST, with $d_{max} = 4$.

Algorithm 22 Mutation operator for the d-GMST Memetic Algorithm

```
$k$ =number of clusters which nodes are swapped
**for** $i = 1, ..., k$ **do**
    randomly change the used node $p_i$ of a random cluster $V_i$
recalculate d-MST
```

is generated I check if it was considered before. In case it was, it will be transformed. All archive relevant operations like insert, find and transform require $O(r)$ time, where r is the number of clusters.

3.7 Computational results for d-GMSTP

For the d-GMSTP Euclidean instances with up to 442 nodes have been used for testing, see Table 3.1. For more details please refer to chapter "Computational results for GMVBCNP".

Table 3.1: TSPLib instances with geographical clustering [6]. Number of nodes vary for each cluster.

Instance Name	$\|V\|$	$\|E\|$	r
kroa150	150	11175	30
krob200	200	19900	40
ts225	225	25200	45
pr226	226	25425	46
gil262	262	34191	53
pr264	264	34716	54
pr299	299	44551	60
lin318	318	50403	64
rd400	400	79800	80
fl417	417	86736	84
gr431	432	92665	87

The preliminary tests showed that the best obtained results for all instances have average degree $3 \geq \overline{d} \geq 4$. Monma and Suri [24] proved that there always exist a MST with degree no more than five for trees in Euclidean plane. As the GMST is an extension of the MST the definition holds.

The set of tests have been performed on a grid engine system, that from the software point of view consists of one large machine with multiple cores/CPUs providing all the same technical infrastructure with respect to environmental settings. It is also permanently changing with respect to the number of available CPUs and hardware settings. For each setting of algorithm the tables show the objective values for the average values of the final solutions $\overline{C(T)}$ during 20 runs, their standard deviations and the average degree $\overline{d}$ of the nodes of the final solutions. The d-GMSTP has been tested with VNS and memetic algorithm (MA), however for MA we distinguish a-MA, involving adapting edges procedure for solution repair and k-MA involving the d-Kruskal algorithm creating always feasible solutions. Both k-MA and a-MA have been tested with different parameter settings as well as with and without solution archive proposed by [38] for avoiding duplicates.

The results given in Table 3.2 present the outcome for a-MA and k-MA and for different set up of d_{max}. While the results of the a-MA with archive (a-MA1) and without archive (a-MA0) do not obviously reveal which set up is better, for the k-MA the involvement of archive (k-MA1) certainly provides better results on all degrees for the same $tcgen$ =200 (denoting number of generations for termination according to convergence) and the values of $\overline{d}$ for d_{max} =10 are lower as well in comparison to $\overline{d}$ for the version without archive (k-MA0). This underlines the robustness of the involved archive for population management purposes when considering the degree constraint inside the Kruskal algorithm that always provides feasible solutions. One could ask why the results for d_{max} =3 and d_{max} =10 have the same $\overline{d}$ but different outcome.

Table 3.2: Results of a-MA and k-MA: with (MA1)/without archive (MA0), p_{mut}=0.6, p_{locimp}=0.05, different d_{max}.

		a-MA0			a-MA1			k-MA0			k-MA1		
		no arch			1 arch, $tcgen$=200			no arch, $tcgen$=200			1 arch, $tcgen$=200		
Instance	d_{max}	$\overline{C(T)}$	dev	$\overline{d}$	$\overline{C(T)}$	dev	$\overline{d}$	$\overline{C(T)}$	dev	$\overline{d}$	$\overline{C(T)}$	dev	$\overline{d}$
kroa150	2	10434.5	96.7	2	10493.9	79.5	2	10331.5	121.6	2	**10214.4**	37.6	2
krob200	2	12345.1	60.3	2	12339.7	59	2	12247.1	112.3	2	**12098.9**	50.2	2
ts225	2	**63948**	0	2	**63948**	0	2	64008.9	148.6	2	**63948**	0	2
gil262	2	1010.4	5.6	2	1012.2	8.5	2	1036.6	23	2	**1007.8**	21	2
pr264	2	24359	104	2	**24331.7**	89.8	2	24731	252.6	2	24450.7	246.8	2
pr299	2	21660.6	51.9	2	21682.9	26.3	2	22359.6	614.3	2	**21616.6**	308	2
lin318	2	19997.2	308.2	2	19955.3	289.8	2	20203.8	485	2	**19695.8**	66.7	2
rd400	2	6633.4	93.3	2	**6631.4**	112	2	7090.8	77.1	2	7023	123.1	2
fl417	2	9237	109.3	2	9164.5	130.8	2	9099.1	0.3	2	9099	0	2
pcb442	2	**21315.5**	260.8	2	21433.8	208.1	2	23092.7	248	2	22540.3	562.6	2
kroa150	3	9837.4	37.6	3	9819.1	15.3	3	9824.4	21.6	3	**9815.7**	2.9	3
krob200	3	11285.4	13.2	3	11282.4	18.4	3	11273.8	38.3	3	**11250**	6.4	3
ts225	3	62301	52.5	3	**62291.8**	31.8	3	62579.7	102.6	3	62418	103.2	3
gil262	3	**944.6**	3.4	3	944.8	3.3	3	965.6	15.5	3	947.1	5.2	3
pr264	3	21899.3	13.2	3	21895.6	6.5	3	21919.4	37.5	3	**21893.7**	16.7	3
pr299	3	20467.5	26.8	3	20473.9	24.1	3	20839.3	221.4	3	**20403.9**	117.4	3
lin318	3	18537.7	26	3	**18537**	57.8	3	18770.4	210.9	3	18557.6	28.4	3
rd400	3	**5978.4**	18.2	3	5984.5	23	3	6335.6	33	3	6249.5	150	3
fl417	3	**7982**	0	3	**7982**	0	3	7982.3	0.9	3	**7982**	0	3
pcb442	3	**19714.6**	84.3	3	19752.8	87.9	3	21016.3	70.4	3	20888.2	198.2	3
kroa150	10	9827.9	31.8	3	9819	14.4	3	9821.9	19.1	3	**9815.7**	2.9	3
krob200	10	11282.7	15.3	3	11282.2	18.4	3	11286.6	106.6	3	**11248**	10.5	3
ts225	10	**62296.8**	43.4	3	62298.5	36.6	3	62596	93.8	3	62472.5	114.5	3
gil262	10	946.9	5.7	3	945.2	3.8	3	962	14.7	3.2	**944.8**	1.8	3.1
pr264	10	21895.9	6.6	3	21898.8	10.9	3	21933.8	63.5	3	**21893.1**	10.8	3
pr299	10	20466.9	25.2	3	20477.4	39.9	3	20954.2	191.1	3	**20439.6**	142.3	3
lin318	10	**18533.2**	19.1	3.1	18539.4	22.5	3.1	18693.1	159.4	3.5	18541.3	14	3.1
rd400	10	5982.8	26.1	3	5982.1	22.6	3	6360.6	28.8	3.2	6263.6	162.5	3.1
fl417	10	**7982**	0	3	**7982**	0	3	**7982**	0	3	7982.1	0.2	3
pcb442	10	**19724.9**	75.1	3	19763.9	118.2	3	21032.1	81.5	3.5	20838.7	325.1	3.5

The reason is that some low cost edges could be discarded during the search because their insertion could violate degree constraint when using d-Kruskal for k-MA. The same can be observed for VNS using d-Kruskal for solution creation. Results given in Table 3.7 presents better outcome for VNS with d_{max} =10 than for VNS with d_{max} =3 however $\overline{d}$=3 for all test instances. However comparing the versions of a-MA0 and k-MA0 without solution archive, the proposed method for solutions repair using the adapting edges procedure provides significantly better results on all degrees and instances with number of nodes over 200. Considering the degree constraint directly during the solution creation procedure can limit the search space and cause premature convergence of MA. While running a-MA the number of solutions violating the degree constraint has been measured. Some solutions created by initialization, crossover and local improvement have $deg > d_{max}$ and need to be repaired. The average values of $\overline{adapt}$ given in Table 3.3 for a-MA determines the average number of solutions violating degree constraint. By the results in Table 3.3 we can observe that for lower value of $d_{max} = 2$ the number of solutions requiring repair procedure increases significantly in comparison to $\overline{adapt}$ for $d_{max} = 3$, however those numbers depend on the test instance. The number of adapts for a-MA1 is higher as well in comparison to a-MA0, which is caused by the conversion method used in solution archive. Before the offspring is introduced to the population its recurrence is examined, and in case the duplicate is discovered it is converted until it does not represent the non-duplicate solution [38]. Each conversion of solution can produce an infeasible individual violating degree constraint so it is necessary to call the adapting edges procedure and therefore for a-MA1 the $\overline{adapt}$ grows up respectively. The Table 3.3 does not include $\overline{adapt}$ column for VNS and k-MA03 (memetic algorithm using d-Kruskal for solution creation with local improvement probability set to p_{locim} =0.5) because both approaches are based on d-Kruskal creating always feasible solutions, so no repair method is required. Both a-MA and k-MA were tested with different p_{locimp} settings. The results in Table 3.6 present the outcome for k-MA and the Table 3.5 presents the outcome for a-MA. The increase of p_{locimp} value for a-MA does not provide significantly better results. The premature convergence produces worse results for p_{locimp}=0.2 in comparison with p_{locimp}=0.05 for all degrees. The differences in relative results of $\overline{time}$ presented on Figure 3.7 for a-MA and different p_{locimp} setting are significant for all degrees, however the results are not significantly improved. The opposite behavior can be observed for k-MA results in Table 3.6. The increase of p_{locimp} setting provides solutions with lower connection cost, what can be seen on Figure 3.6 presenting the relative results of $\overline{C(T)}$ for k-MA, where $p_{locimp} = 0.5$ equals 100%.

The MA has been tested with different p_{mut} settings and the results are given in Table 3.4. The best $\overline{C(T)}$ were obtained with p_{mut} =0.6 and this value was used for the other tests' sets. The results given in Table 3.7 present the comparison of the outcomes obtained by different approaches. The results for VNS with stopping condition $tcgen$ =50 are the best for all instances and d_{max} settings. The used value of p_{locimp} =0.5 for k-MA provided better results than VNS only for a couple of instances, mostly for d_{max} =2 and d_{max} =3. However the running time for k-MA is almost two times higher than for VNS, what is presented by the $\overline{time}$ results in Table 3.7. From the performed tests and obtained results I can conclude that VNS is the most robust approach used in this thesis for d-GMSTP.

Table 3.3: Results of $\overline{time}$ and adoptions numbers for a-MA: with/without archive, p_{mut}=0.6, p_{locimp}=0.05,different d_{max}, results of $\overline{time}$ for VNS and results of $\overline{time}$ for k-MA03 without archive and p_{locimp}=0.5.

		a-MA0		a-MA1		VNS	k-MA03
		without archive		1 arch, $tcgen$=200		$tcgen$=50	$tcgen$=200
Instance	d_{max}	$\overline{time}$	$\overline{adopts}$	$\overline{time}$	$\overline{adopts}$	$\overline{time}$	$\overline{time}$
kroa150	2	17.1	15267	19.9	16733.6	17.7	54.2
krob200	2	25.4	12521.2	35.6	15722.9	82.6	230.0
ts225	2	7.9	2925.5	8.9	3408.2	25.1	92.0
gil262	2	120.4	33299.4	117	32500.9	127.7	515.4
pr264	2	70.7	15003.8	72.9	15673	298.9	376.8
pr299	2	75.9	14413.1	90.9	16280.2	419.5	589.1
lin318	2	151.4	37295.4	176.5	41337.1	191.3	572.1
rd400	2	158	18055.4	183.5	21971.8	344.2	760.2
fl417	2	58.7	5752.7	51	4989.3	37.5	206.2
pcb442	2	336	29117	209.4	18663.1	453.3	908.5
kroa150	3	5.4	125.4	6.3	129.8	21.4	50.0
krob200	3	9.1	217.9	11.6	263.8	62.8	158.4
ts225	3	15	69.3	17.3	72.6	199.7	363.7
gil262	3	30.7	757	33.3	758.7	204.7	347.1
pr264	3	48.6	438.4	51.4	410.9	229.4	308.6
pr299	3	58.8	3905.8	54.3	3324.3	272.7	601.8
lin318	3	46.2	1525.4	42	1504.8	306.4	634.3
rd400	3	192.3	2438.7	162.2	2237.3	469.7	770.8
fl417	3	37.4	112.7	37.7	106.4	28	251.4
pcb442	3	218.4	1229.6	165.4	1341.9	459.4	724.8

Table 3.4: Results of a-MA with different p_{mut} settings, p_{locimp}=0.05, $tcgen$ =200, **without** archive.

Instance		p_{mut} =0.8			p_{mut} =0.6			p_{mut} =0.4		
Name	d_{max}	$\overline{C(T)}$	std dev	$\overline{d}$	$\overline{C(T)}$	std dev	$\overline{d}$	$\overline{C(T)}$	std dev	$\overline{d}$
kroa150	2	10484.3	89.5	2	**10434.5**	96.7	2	10465.5	79.4	2
krob200	2	**12328.6**	50.8	2	12345.1	60.3	2	12354.5	65.2	2
ts225	2	63967.8	88.5	2	**63948.0**	0	2	**63948.0**	0.0	2
gil262	2	1013.8	7.5	2	**1010.4**	5.6	2	1013.6	7.4	2
pr264	2	**24346.8**	88.0	2	24359	104	2	24410.6	101.7	2
pr299	2	21682.9	66.6	2	**21660.6**	51.9	2	21692.2	25.8	2
lin318	2	20035.4	333.0	2	19997.2	308.2	2	**19866.2**	148.8	2
rd400	2	6676.9	109.7	2	**6633.4**	93.3	2	6666.4	94.2	2
fl417	2	9237.5	116.5	2	**9237**	109.3	2	9252.1	107.6	2
pcb442	2	21401.1	253.9	2	**21315.5**	260.8	2	21375.6	162.3	2
kroa150	3	9839.0	38.5	3	9837.4	37.6	3	**9822.5**	23.0	3
krob200	3	11294.3	31.2	3	11285.4	13.2	3	**11279.4**	12.8	3
ts225	3	**62296.0**	48.5	3	62301	52.5	3	62298.8	48.7	3
gil262	3	947.3	6.4	3	**944.6**	3.4	3	945.8	3.6	3
pr264	3	21899.5	6.7	3	**21899.3**	13.2	3	21901.4	13.2	3
pr299	3	20485.4	47.7	3	**20467.5**	26.8	3	20474.4	45.5	3
lin318	3	**18534.6**	24.6	3	18537.7	26	3	18537.8	21.8	3
rd400	3	**5976.**6	13.6	3	5978.4	18.2	3	5979.4	16.8	3
fl417	3	**7982.0**	0.0	3	**7982.0**	0	3	**7982.0**	0.0	3
pcb442	3	19717.0	58.5	3	**19714.6**	84.3	3	19754.9	116.0	3
kroa150	10	9833.4	45.4	3	9827.9	31.8	3	**9821.5**	22.0	3
krob200	10	11285.7	12.3	3	**11282.7**	15.3	3	11284.0	12.7	3
ts225	10	62325.5	62.0	3	**62296.8**	43.4	3	62303.1	53.1	3
gil262	10	**944.8**	4.3	3.1	946.9	5.7	3	945.5	5.8	3
pr264	10	**21895.9**	10.3	3	**21895.9**	6.6	3	21896.8	7.5	3
pr299	10	20471.3	34.8	3	**20466.9**	25.2	3	20477.2	34.2	3
lin318	10	18534.5	23.6	3	18533.2	19.1	3.1	**18522.8**	16.8	3
rd400	10	5992.9	32.5	3	**5982.8**	26.1	3	5984.8	24.4	3
fl417	10	**7982.0**	0.0	3	**7982.0**	0	3	**7982.0**	0.0	3
pcb442	10	19745.0	91.1	3.1	**19724.9**	75.1	3	19725.0	85.9	3

Table 3.5: Results of a-MA0 **with** different p_{locim} settings, p_{mut}=0.6, $tcgen$ =200, without archive.

Instance		p_{locim} =0.05				p_{locim} =0.2			
Name	d_{max}	$\overline{C(T)}$	std dev	$\overline{d}$	$\overline{time}$	$\overline{C(T)}$	std dev	$\overline{d}$	$\overline{time}$
kroa150	2	**10434.5**	96.7	2	17.1	10506.5	80	2	24.4
krob200	2	12345.1	60.3	2	25.4	**12325.3**	39.2	2	84.8
ts225	2	**63948**	0	2	7.9	**63948**	0	2	27.7
gil262	2	**1010.4**	5.6	2	120.4	1016.9	9.5	2	236.5
pr264	2	**24359**	104	2	70.7	24423.4	117.6	2	142.6
pr299	2	**21660.6**	51.9	2	75.9	21701.8	41	2	145
lin318	2	**19997.2**	308.2	2	151.4	20006.7	203.6	2	387.9
rd400	2	**6633.4**	93.3	2	158	6660	72	2	277.7
fl417	2	9237	109.3	2	58.7	**9151.9**	163.1	2	149.4
pcb442	2	**21315.5**	260.8	2	336	21435.6	228.8	2	429.9
kroa150	3	**9837.4**	37.6	3	5.4	9853.1	32.6	3	11.9
krob200	3	11285.4	13.2	3	9.1	**11285.2**	18.6	3	19.8
ts225	3	62301	52.5	3	15	**62294.6**	51	3	25.6
gil262	3	**944.6**	3.4	3	30.7	944.9	3.6	3	73.5
pr264	3	**21899.3**	13.2	3	48.6	21905.5	16.5	3	119.9
pr299	3	**20467.5**	26.8	3	58.8	20497.7	43.8	3	97.7
lin318	3	18537.7	26	3	46.2	**18516.2**	10.6	3	97.8
rd400	3	**5978.4**	18.2	3	192.3	5980.9	17.5	3	407.9
fl417	3	**7982**	0	3	37.4	**7982**	0	3	95.5
pcb442	3	**19714.6**	84.3	3	218.4	19720	74.6	3	451
kroa150	10	9827.9	31.8	3	8	**9822.9**	20.8	3	44.5
krob200	10	**11282.7**	15.3	3	8.5	11285.7	12.3	3	18.5
ts225	10	62296.8	43.4	3	10.4	6**2280.1**	24.9	3	27.3
gil262	10	946.9	5.7	3	21	**943.4**	2.1	3	65
pr264	10	**21895.9**	6.6	3	26.6	21898.2	8.6	3	114
pr299	10	20466.9	25.2	3	32.1	**20464.9**	25.5	3	107.7
lin318	10	18533.2	19.1	3.1	32.6	**18522.6**	12.1	3.1	89.6
rd400	10	**5982.8**	26.1	3	95.5	5983.3	20.6	3	436
fl417	10	**7982**	0	3	21	**7982**	0	3	89.2
pcb442	10	**19724.9**	75.1	3	128.2	21867.3	34.2	3.1	330.2

Table 3.6: Results of k-MA with different settings of p_{locim}, p_{mut} =0.6, $tcgen$ =200, without archive.

Instance		k-MA01, p_{locim} =0.05			k-MA02, p_{locim} =0.2			k-MA03, p_{locim} =0.5		
Name	d_{max}	$\overline{C(T)}$	std dev	$\overline{d}$	$\overline{C(T)}$	std dev	$\overline{d}$	$\overline{C(T)}$	std dev	$\overline{d}$
kroa150	2	10331.5	121.6	2	10209.2	9.8	2	**10206**	0	2
krob200	2	12247.1	112.3	2	12149.6	95.3	2	**12076.9**	2.7	2
ts225	2	64008.9	148.6	2	63967.8	88.5	2	**63948**	0	2
gil262	2	1036.6	23	2	1014.5	22.5	2	**993.4**	5.4	2
pr264	2	24731	252.6	2	24361.5	153.2	2	**24243.1**	106.3	2
pr299	2	22359.6	614.3	2	21680.7	355.8	2	**21434.9**	58.9	2
lin318	2	20203.8	485	2	19703.8	94.5	2	**19657.1**	43.8	2
rd400	2	7090.8	77.1	2	6947.4	175.3	2	**6786.9**	254.3	2
fl417	2	9099.1	0.3	2	**9099**	0	2	**9099**	0	2
pcb442	2	23092.7	248	2	22689.4	400.7	2	**21999.1**	691	2
kroa150	3	9824.4	21.6	3	**9815**	0	3	9815.7	2.9	3
krob200	3	11273.8	38.3	3	11252	10.3	3	**11244.6**	2.5	3
ts225	3	62579.7	102.6	3	62413.4	59.6	3	**62325.3**	59.5	3
gil262	3	965.6	15.5	3	946.9	5.4	3	**943.3**	1.8	3
pr264	3	21919.4	37.5	3	21897.6	18.1	3	**21886.7**	2	3
pr299	3	20839.3	221.4	3	20367.9	46.1	3	**20334.4**	19.9	3
lin318	3	18770.4	210.9	3	18571.5	84.6	3	**18528.8**	13.8	3
rd400	3	6335.6	33	3	6294.6	116.7	3	**6121.**8	172.9	3
fl417	3	7982.3	0.9	3	7982	0	3	7982	0	3
pcb442	3	21016.3	70.4	3	20961.2	125.7	3	**20613.7**	567.6	3
kroa150	10	9821.9	19.1	3	9819.3	19	3	**9815**	0	3
krob200	10	11286.6	106.6	3	11246.8	6.2	3	**11246.2**	4.5	3
ts225	10	62596	93.8	3	62449.8	85.6	3	**62341.8**	80.5	3
gil262	10	962	14.7	3.2	946.2	7	3	**943.5**	2.1	3
pr264	10	21933.8	63.5	3	21892.2	14.5	3	**21886.7**	2.9	3
pr299	10	20954.2	191.1	3	20451.1	187.3	3	**20346.6**	46.3	3
lin318	10	18693.1	159.4	3.5	**18552.1**	42.6	3.2	18533.4	16.9	3.1
rd400	10	6360.6	28.8	3.2	6301.5	120	3.1	**6090.5**	176.9	3.1
fl417	10	**7982**	0	3	**7982**	0	3	**7982**	0	3
pcb442	10	21032.1	81.5	3.5	20924.4	311.1	3.3	**20518.9**	593.4	3.3

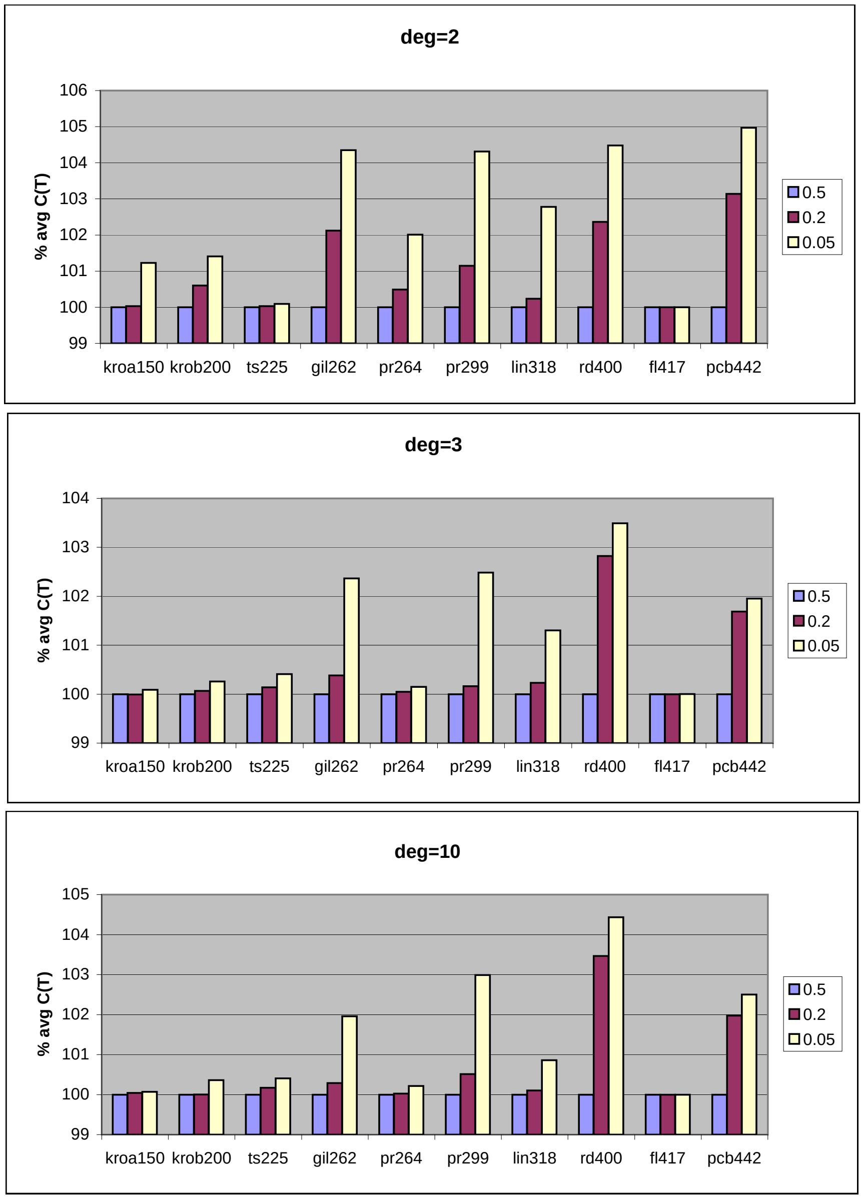


Figure 3.6: Relative results of $\overline{C(T)}$ on TSPlib instances for k-MA with different p_{locimp} and different degree deg, where p_{locimp}=0.5 equals 100%

Table 3.7: Results of a-MA and k-MA with settings p_{locim} =0.05, p_{mut} =0.6, $tcgen$ =200, without archive and VNS with $tcgen$ =50.

Instance		a-MA0, p_{locim} =0.05				k-MA03, p_{locim} =0.5				VNS			
Name	d_{max}	$\overline{C(T)}$	std dev	$\overline{d}$	$\overline{time}$	$\overline{C(T)}$	std dev	$\overline{d}$	$\overline{time}$	$\overline{C(T)}$	std dev	$\overline{d}$	$\overline{time}$
kroa150	2	10434.5	96.7	2	37.4	**10206**	0	2	54.2	**10206**	0	2	17.7
krob200	2	12345.1	60.3	2	58.7	**12076.9**	2.7	2	230.0	12081	49.2	2	82.6
ts225	2	**63948**	0	2	31.6	**63948**	0	2	92.0	**63948**	0	2	25.1
gil262	2	1010.4	5.6	2	30.7	993.4	5.4	2	515.4	**989**	0	2	127.7
pr264	2	24359	104	2	120.4	24243.1	106.3	2	376.8	**24178.3**	118.1	2	298.9
pr299	2	21660.6	51.9	2	23.4	21434.9	58.9	2	589.1	**21387.8**	58.4	2	419.5
lin318	2	19997.2	308.2	2	5.4	19657.1	43.8	2	572.1	**19640**	40.1	2	191.3
rd400	2	6633.4	93.3	2	17.1	6786.9	254.3	2	760.2	**6510.3**	50	2	344.2
fl417	2	9237	109.3	2	15.9	**9099**	0	2	206.2	**9099**	0	2	37.5
pcb442	2	21315.5	260.8	2	9.1	21999.1	691	2	908.5	**21151.2**	70	2	453.3
kroa150	3	9837.4	37.6	3	25.4	**9815.7**	2.9	3	50.0	9817.6	5.3	3	21.4
krob200	3	11285.4	13.2	3	10.3	11244.6	2.5	3	158.4	**11244**	0	3	62.8
ts225	3	62301	52.5	3	46.2	62325.3	59.5	3	363.7	**62275.6**	23	3	199.7
gil262	3	944.6	3.4	3	151.4	943.3	1.8	3	347.1	**942.7**	1.4	3	204.7
pr264	3	21899.3	13.2	3	46.3	**21886.7**	2	3	308.6	21891.3	6.6	3	229.4
pr299	3	20467.5	26.8	3	218.4	20334.4	19.9	3	601.8	**20323.1**	12.7	3	272.7
lin318	3	18537.7	26	3	223.7	18528.8	13.8	3	634.3	**18506.2**	10	3	306.4
rd400	3	5978.4	18.2	3	312.5	6121.8	172.9	3	770.8	**5962.1**	21	3	469.7
fl417	3	**7982**	0	3	48.6	**7982**	0	3	251.4	**7982**	0	3	28
pcb442	3	**19714.6**	84.3	3	70.7	20613.7	567.6	3	724.8	19810.9	58.9	3	459.4
kroa150	10	9827.9	31.8	3	50.7	**9815**	0	3	47.7	**9815**	0	3	14.6
krob200	10	11282.7	15.3	3	58.8	11246.2	4.5	3	154.0	**11244**	0	3	4.8
ts225	10	62296.8	43.4	3	75.9	62341.8	80.5	3	284.3	**62270.1**	5.7	3	75.4
gil262	10	946.9	5.7	3	59.6	943.5	2.1	3	344.3	**942**	0	3	169.5
pr264	10	21895.9	6.6	3	192.3	21886.7	2.9	3	288.6	**21886.4**	1.6	3	230.2
pr299	10	20466.9	25.2	3	158.0	20346.6	46.3	3	545.9	**20318.6**	9.3	3	261
lin318	10	18533.2	19.1	3.1	169.1	18533.4	16.9	3.1	530.2	**18509.6**	10.8	3	282.1
rd400	10	5982.8	26.1	3	15.0	6090.5	176.9	3.1	887.1	**5957.8**	14.8	3	426.3
fl417	10	**7982**	0	3	7.9	**7982**	0	3	247.9	**7982**	0	3	21
pcb442	10	**19724.9**	75.1	3	14.4	20518.9	593.4	3.3	851.1	19749.3	64.6	3	491.4

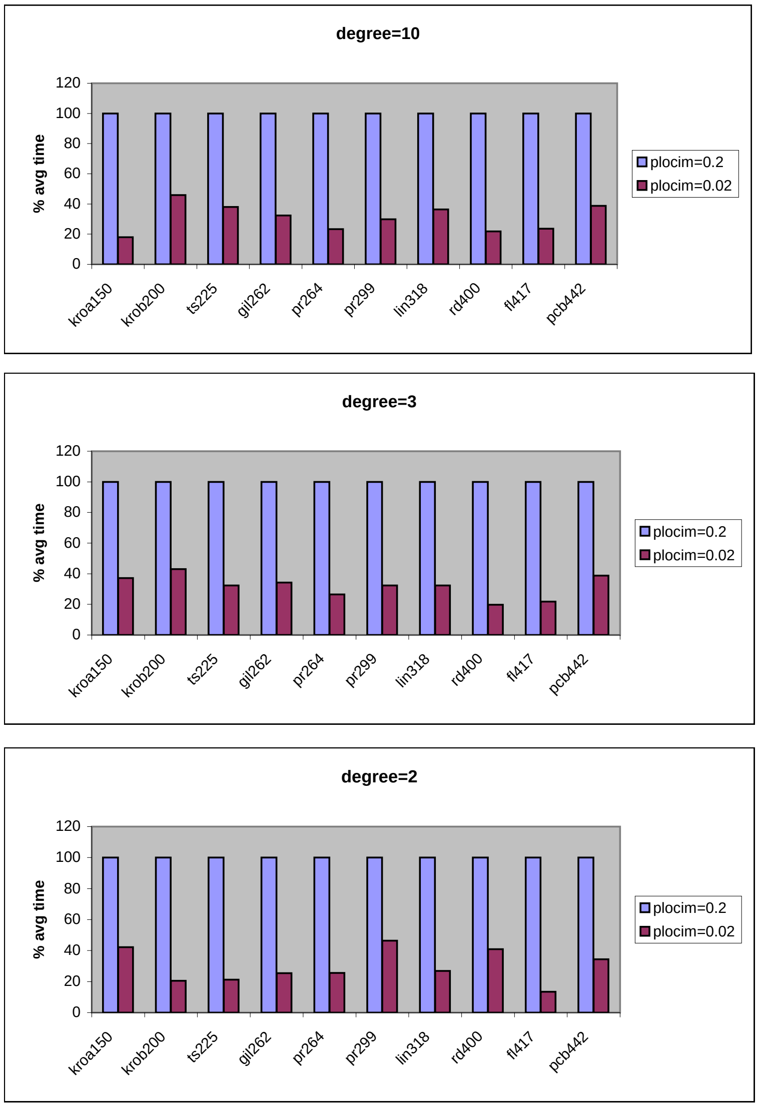


Figure 3.7: Relative results of $\overline{time}$ on TSPlib instances for a-MA with different p_{locimp} and different degree deg, where p_{locimp}=0.2 equals 100%

4 Summary and Outlook

In thesis I considered two NP-hard problems not addressed in the literature so far: the Generalized Degree Constrained Minimum Spanning Tree Problem (d-GMSTP) and the Generalized Minimum Vertex Bi-connected Network Problem (GMVBCNP). The fundamental strategy was to design the metaheuristics allowing to find the best solutions in an efficient way. For the Generalized Minimum Vertex Bi-connected Network Problem (GMVBCNP) the Memetic Algorithm (MA) using two neighborhood structures as local improvement strategy has been proposed. They are exponentially large, however the used graph reduction technique significantly limited the search for the best neighbor. The GMVBCNP was tested on Euclidean instances with up to 442 nodes with different parameters settings. The best results were obtained for local improvement probability equalled to 1, however concerning the computation time it was recommended to use probability equalled to 0.2 for local improvement. Two different population management strategies, which can be either used together or separately has been introduced as well. By the performed tests it has been proved that edge population management provides better results however it requires as well much more time than delta population management. As no reference values are available it was not possible to compare the obtained and presented in this thesis results with different ones, however Memetic Algorithm combined with delta population management and with local improvement probability set to 0.2 generates high quality solutions in acceptable time.

For the Generalized Degree Constrained Minimum Spanning Tree Problem (d-GMSTP) Variable Neighborhood Search (VNS) with VND as a local improvement and Memetic Algorithm has been proposed. The VNS alternates between three neighborhood structures, which always generate feasible solutions with respect to degree constraint. One of them is based on the approach proposed by Pop and two of them are based on the approach proposed by Ghosh. The Memetic Algorithm uses two neighborhood structures as local improvement strategy and alternates between two approaches for consideration of degree constraint. One of them repairs the solutions violating degree constraint, whereas the second generates always the feasible individuals. The MA has been tested with different parameter settings as well as in conjunction with solution archive for avoiding duplicates. All tests for the d-GMSTP were performed on Euclidean instances with up to 442 nodes. The best results were obtained for VNS which seems to be more robust than MA proposed in this thesis for solving the d-GMSTP.

Further work can be done for both problems, and other heuristic methods can be applied, e.g. VNS for the GMVBCNP. The additional neighborhoods and the other population management strategies can be considered. The both problems were tested only on Euclidean instances, so it might be interesting to test them on grouped and random Euclidean instances.

Bibliography

[1] J. C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA JOURNAL ON COMPUTING*, 6:154–160, 1994.

[2] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35:268–308, 2003.

[3] T. H. Cormen, C. E.Leiserson, R. L.Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2001.

[4] K. P. Eswaran and R. E. Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5(4):653–665, 1976.

[5] S. P. Feketea, S. Khullerb, M. Klemmstein, B. Raghavacharic, and N. Young. A network-flow technique for finding low-weight bounded-degree spanning trees. *Journal of Algorithms*, 24:310–324, 1997.

[6] C. Feremans. *Generalized Spanning Trees and Extensions*. PhD thesis, Universite Libre de Bruxelles, Brussels, Belgium, 2001.

[7] C. Feremans, M. Labbe, and G. Laporte. Generalized network design problems. *European Journal of Operational Research*, 148(1):1–13, 2003.

[8] M. Fischetti, J. J. Salazar, and P. Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45:378–394, 1997.

[9] D. Ghosh. Solving medium to large sized Euclidean generalized minimum spanning tree problems. Technical Report NEP-CMP-2003-09-28, Indian Institute of Management, Research and Publication Department, Ahmedabad, India, 2003.

[10] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

[11] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Learning*. Addison-Wesley, Reading, Massachusetts, 1989.

[12] B. Golden, S. Raghavan, and D. Stanojevic. Heuristic search for the generalized minimum spanning tree problem. *INFORMS Journal on Computing*, 17(3):290–304, 2005.

[13] J. Holland. *Adaptation In Natural and Artificial Systems*. University of Michigan Press, 1975.

[14] B. Hu, M. Leitner, and G. R. Raidl. The generalized minimum edge biconnected network problem: Efficient neighborhood structures for variable neighborhood search. accepted for Networks.

[15] B. Hu, M. Leitner, and G. R. Raidl. Combining variable neighborhood search with integer linear programming for the generalized minimum spanning tree problem. *Journal of Heuristics*, 14(5):473–499, 2008.

[16] B. Hu and G. R. Raidl. Effective neighborhood structures for the generalized traveling salesman problem. In J. van Hemert and C. Cotta, editors, *Evolutionary Computation in Combinatorial Optimisation – EvoCOP 2008*, volume 4972 of *LNCS*, pages 36–47, Naples, Italy, 2008. Springer.

[17] B. Hu and G. R. Raidl. A memetic algorithm for the generalized minimum vertex-biconnected network problem. *9th International Conference on Hybrid Intelligent Systems - HIS 2009*, pages 63–68, 2009.

[18] H.-J. Kim. www.ibluemojo.com.

[19] J. Knowles and D. Corne. A new evolutionary approach to the degree-constrained minimumspanning tree problem. *Evolutionary Computation, IEEE Transactions on*, 4:125–134, 2000.

[20] M. Krishnamoorthy, A. T. Ernst, and Y. M. Sharaiha. Comparison of algorithms for the degree constrained minimum spanning tree. *Journal of Heuristics*, 7:587–611, 2001.

[21] M. Leitner. Solving two generalized network design problems with exact and heuristic methods. Master's thesis, Vienna University of Technology, Vienna, Austria, 2006.

[22] I. Ljubic. *Exact and Memetic Algorithms for Two Network Design Problems*. PhD thesis, Technische Universitat Wien, Wien, Austria, 2004.

[23] I. Ljubic and G. R. Raidl. A memetic algorithm for minimum-cost vertex-biconnectivity augmentation of graphs. *Journal of Heuristics*, 9(5):401–428, 2003.

[24] C. Monma and S. Suri. Transitions in geometric minimum spanning trees (extended abstract). In *SCG '91: Proceedings of the seventh annual symposium on Computational geometry*, pages 239–249, New York, NY, USA, 1991. ACM.

[25] P. Moscato. Memetic algorithms: A short introduction. In D. Corne et al., editors, *New Ideas in Optimization*, pages 219–234. McGraw Hill, 1999.

[26] Y. S. Myung, C. H. Lee, and D. W. Tcha. On the generalized minimum spanning tree problem. *Networks*, 26:231–241, 1995.

[27] S. C. Narula and C. A. Ho. Degree constrained minimum spanning tree. *Computers and Operations Research*, 7:239–249, 1980.

[28] P. Pop, W. Kern, and G. Still. An approximation algorithm for the generalized minimum spanning tree problem with bounded cluster size, 2001. Internal Report.

[29] P. C. Pop. *The Generalized Minimum Spanning Tree Problem*. PhD thesis, University of Twente, The Netherlands, 2002.

[30] G. R. Raidl. Efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem. *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, 1:104–111, 2000.

[31] G. R. Raidl and B. A. Julstrom. Edge sets: an effective evolutionary coding of spanning trees. *Evolutionary Computation, IEEE Transactions on*, 7:225–239, 2003.

[32] C. C. Ribeiro and M. C. Souza. Variable neighborhood search for the degree-constrained minimum spanning tree problem. *Discrete Applied Mathematics*, 118:43–54, 2002.

[33] M. Savelsbergh and T. Volgenant. Edge exchanges in the degree-constrained minimum spanning tree problem. *Computers and Operations Research*, 12:341–348, 1985.

[34] R. Sedgewick. *Algorithms*. Addison-Wesley, 1984.

[35] K. Srensena and M. Sevauxb. Ma—pm: memetic algorithms with population management. *Computers and Operations Research*, 33, 1214-1225 2006.

[36] A. Volgenant. A lagrangean approach to the degree-constrained minimum spanning tree problem. *European Journal of Operational Research*, 39:325–331, 1989.

[37] T. Watanabe and A. Nakamura. Edge-connectivity augmentation problems. *Journal of Computer and System Sciences*, 35:96–144, 1987.

[38] M. Wolf. Ein lsungsarchiv-untersttzter evolutionrer algorithmus fr das generalized minimum spanning tree-problem. Master's thesis, Vienna University of Technology, 2009.