Informatics

Hybrid Metaheuristics Based on Large Neighborhood Search and Mixed Integer Linear Programming for the Directed Feedback Vertex Set Problem

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Maria Bresich, BSc
Matrikelnummer 01506763

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Mitwirkung: Projektass. Dipl.-Ing. Johannes Varga, BSc BSc

Wien, 23. Jänner 2023

Maria Bresich

Günther Raidl

Informatics

Hybrid Metaheuristics Based on Large Neighborhood Search and Mixed Integer Linear Programming for the Directed Feedback Vertex Set Problem

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Software Engineering & Internet Computing

by

Maria Bresich, BSc
Registration Number 01506763

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Assistance: Projektass. Dipl.-Ing. Johannes Varga, BSc BSc

Vienna, 23rd January, 2023

Maria Bresich   Günther Raidl

Erklärung zur Verfassung der Arbeit

Maria Bresich, BSc

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, 23. Jänner 2023

Maria Bresich

Acknowledgements

At this point, I would like to thank my supervisor Günther Raidl and my co-supervisor Johannes Varga for their dedicated assistance, valuable advice, and constant feedback.

Special thanks also go to my boyfriend who consistently motivates me and is always there for me. Additionally, I sincerely thank my family for their constant support.

Danksagung

An dieser Stelle möchte ich mich bei meinen Betreuern Günther Raidl und Johannes Varga für ihre engagierte Unterstützung, wertvollen Ratschläge und stetiges Feedback bedanken.

Besonderer Dank gilt auch meinem Freund, der mich immer wieder motiviert hat und stets geduldig für mich da war. Ein herzliches Dankeschön auch an meine Familie für ihre andauernde Unterstützung.

Abstract

The directed feedback vertex set (DFVS) problem is an NP-complete combinatorial optimization problem defined on unweighted, directed graphs with the goal of finding a DFVS of minimum cardinality. A set of vertices is called a directed feedback vertex set if it contains at least one vertex of every directed cycle in the graph, such that the removal of the set from the graph leads to a directed acyclic graph (DAG). The general feedback vertex set problem is relevant for many different areas and real world scenarios including deadlock detection and recovery in operating systems, computer-aided design applications, and chemical engineering.

The aim of this work is to develop hybrid metaheuristics that provide solutions of high quality for challenging DFVS problem instances within a given time limit. Our solving approaches start with a graph reduction procedure using reduction rules from the literature to decrease the size of the graph and reduce the search space for possible solutions. An initial solution is generated with a greedy construction heuristic and improved with a classical local search procedure. For the metaheuristic framework, we employ large neighborhood search (LNS) with neighborhoods that are each defined by a destroy and a repair method. We introduce two neighborhood structures which are based on either moving vertices from the current solution to the corresponding DAG (enlarge-DAG) or the other way around (enlarge-(partial-)DFVS). A valid solution is restored by encoding the resulting smaller subproblem with a mixed integer linear programming (MILP) formulation and solving it with an existing solver. This combination of LNS and MILP leads to the hybridization of the metaheuristic. We propose two MILP formulations for the DFVS problem, which can be used within the hybrid LNS as well as in standalone solving procedures. One formulation called CEC is based on cycle elimination constraints and the other formulation MTZ is inspired by the subtour elimination constraints from Miller, Tucker, and Zemlin and derived from the literature.

We also investigate various strategies for selecting the degree of destruction for the LNS and a technique for dynamically choosing the MILP formulation for each instance. These enhancements lead to multiple variants of our hybrid metaheuristic, which are tested on benchmark instances and compared in terms of the achieved average solution quality.

The results of our computational study show that all hybrid LNS variants achieve higher average solution qualities than the standalone MILP solving procedures. Furthermore, the CEC-based formulation together with the enlarge-DAG neighborhood outperforms other approaches using the MTZ-based formulation as well as the combination of MTZ and enlarge-partial-DFVS neighborhood when having a time limit. The dynamic selection strategy for the degree of destruction that is based on the number of cycles of length two in the respective graph performs the best out of all tested strategies. Enhancing this approach with the dynamic selection of the MILP formulation leads to further improvements of the solution quality. Moreover, the main variants of our hybrid metaheuristic manage to outperform two state-of-the-art solving procedures from the literature, which are based on simulated annealing.

Kurzfassung

Das Directed Feedback Vertex Set (DFVS) Problem ist ein NP-vollständiges, kombinatorisches Optimierungsproblem, das für ungewichtete, gerichtete Graphen spezifiziert ist, und dessen Ziel es ist, ein DFVS minimaler Größe zu finden. Eine Knotenmenge wird als Directed Feedback Vertex Set (gerichtete kreiskritische Knotenmenge) bezeichnet, wenn sie zumindest einen Knoten eines jeden gerichteten Kreises im Graphen enthält, sodass das Entfernen der Knotenmenge aus dem Graphen zu einem gerichteten azyklischen Graphen (engl. directed acyclic graph, DAG) führt. Das Feedback Vertex Set Problem findet in vielen verschiedenen Bereichen Anwendung, wie zum Beispiel bei der Verklemmungserkennung und -beseitigung in Betriebssystemen, bei computerunterstützten Konstruktionsanwendungen und in der chemischen Verfahrenstechnik.

Das Ziel dieser Diplomarbeit ist die Entwicklung hybrider Metaheuristiken, die innerhalb eines Zeitlimits hochwertige Lösungen für das DFVS Problem erzeugen. Am Beginn unserer Lösungsansätze steht jeweils eine Prozedur zur Reduktion der Graphengröße, in der aus der Literatur bekannte Reduktionsregeln zur Anwendung kommen, und die den Suchraum für mögliche Lösungen verkleinert. Danach wird eine Greedy-Heuristik verwendet, um eine erste Lösung zu erzeugen, welche im Weiteren mit einer lokalen Suche verbessert wird. Unsere Metaheuristik besteht aus einer Large Neighborhood Search (LNS), deren Nachbarschaften jeweils von einer zerstörenden und einer reparierenden Operation bestimmt werden. Wir präsentieren zwei Nachbarschaften, in denen entweder Knoten von der momentanen Lösung zum entsprechenden DAG hinzugefügt werden (enlarge-DAG) oder umgekehrt (enlarge-(partial-)DFVS). Eine gültige Lösung wird dann wiederhergestellt, indem das resultierende kleinere Teilproblem in ein gemischt-ganzzahliges lineares Optimierungsmodell (engl. mixed integer linear program, MILP) überführt und mit entsprechender Software gelöst wird. Diese Kombination aus LNS und MILP führt zu einer Hybridisierung der Metaheuristik. Wir stellen zwei MILP Modelle für das DFVS Problem vor, welche sowohl als Bestandteil der hybriden LNS als auch in eigenständigen Lösungsverfahren verwendet werden können. Die CEC genannte Formulierung basiert auf Constraints zur Entfernung von Kreisen (engl. cycle elimination constraints). Die Formulierung namens MTZ wurde inspiriert von den Kurzzyklusungleichungen, auch Subtour-Eliminationsbedingungen genannt, nach Miller, Tucker und Zemlin und ist eine Abwandlung eines Modells aus der Literatur.

Weiters präsentieren wir verschiedene Strategien zur Wahl des Zerstörungsgrades in der

LNS und auch eine Vorgehensweise, wie die jeweils passende MILP Formulierung für jede Instanz gewählt werden kann. Das Hinzufügen dieser Erweiterungen ergibt mehrere Varianten unserer hybriden Metaheuristik, welche auf typischen Instanzen getestet und bezüglich ihrer durchschnittlich erreichten Lösungsqualität miteinander verglichen werden.

Unsere praktische Untersuchung zeigt, dass alle Varianten der hybriden LNS höhere Durchschnittswerte für die Lösungsqualität erzielen als die eigenständigen Lösungsverfahren basierend auf den MILP Formulierungen. Außerdem übertrifft die Leistung der CEC Formulierung zusammen mit der enlarge-DAG Nachbarschaft die der anderen Ansätze, welche MTZ oder auch die Kombination von MTZ und der enlarge-partial-DFVS Nachbarschaft verwenden, wenn eine zeitliche Beschränkung festgelegt wird. Das dynamische Auswahlverfahren für den Zerstörungsgrad, das auf der Anzahl an Kreisen mit Länge zwei im Graphen beruht, erzielt die besten Ergebnisse aller untersuchten Strategien. Die durchschnittliche Lösungsqualität wird weiter verbessert, wenn dieser Ansatz mit der dynamischen Auswahl der MILP Formulierung gepaart wird. Des Weiteren ist es uns gelungen, mit den Hauptvarianten unserer hybriden Metaheuristik die Leistungen zweier state-of-the-art Lösungsverfahren aus der Literatur, welche auf Simulated Annealing basieren, zu übertreffen.

Contents

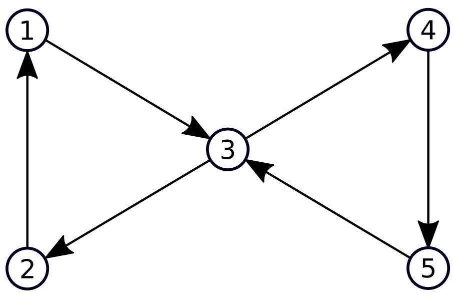
CHAPTER 1

Introduction

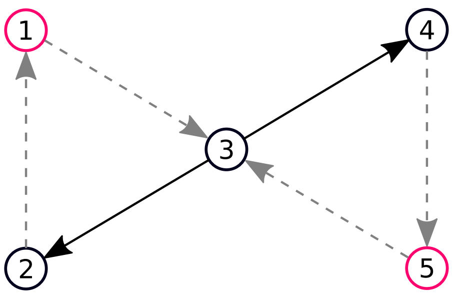
In graph theory, a *feedback set* is a set of vertices that contains at least one vertex of every cycle in a given graph, such that removing the feedback set from the graph results in an acyclic graph. The feedback vertex set (FVS) problem is a combinatorial optimization problem as its goal is to find a feedback set of minimum cardinality. There are multiple variants of the FVS problem, for example, depending on whether the graph is directed or undirected, weighted or unweighted. In this work, we focus on the unweighted and directed feedback vertex set (DFVS) problem as mentioned by Karp [Kar72] under the name feedback node set. The DFVS problem is defined on a directed graph and aims to find a minimum cardinality feedback set whose removal leads to a directed acyclic graph (DAG).

Figure 1.1a illustrates an instance of the unweighted DFVS problem with five vertices. The depicted graph contains two simple cycles involving three vertices each, namely $(1-3-2-1)$ and $(3-4-5-3)$. A possible solution is depicted in Figure 1.1b where the vertices 1 and 5 are removed from the graph and added to the feedback set $F = \{1, 5\}$, which then is of size, or cardinality, two. This is a valid solution as both cycles are broken and the remaining graph is acyclic. However, the solution is not optimal because there is another FS $F^* = \{3\}$ of smaller capacity as shown in Figure 1.1c. This FS only contains one vertex – vertex 3 – and its removal from the graph also removes both cycles. The feedback set F^* is even an optimal solution as at least one vertex has to be deleted to break the cycles and thus, you cannot find a feedback set of smaller cardinality.

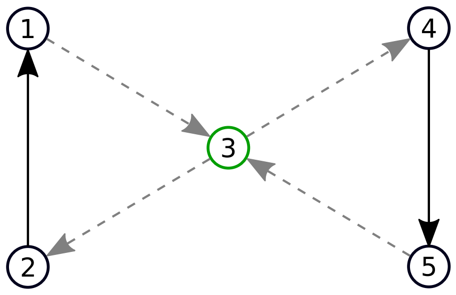
The FVS problem is applicable in many different areas and real world problems. These range from chemical engineering [BSNA21] and deadlock detection and recovery in operating systems over program verification and Bayesian inference up to VLSI chip design and computer-aided design applications. However, the FVS problem is NP-complete as shown by Karp [Kar72] and using exact algorithms to find optimal solutions becomes more and more computationally expensive or even infeasible with a growing instance size, which is based on the number of vertices and edges of a graph. It is due



(a) DFVS problem instance.



(b) Suboptimal solution $F = \{1, 5\}$.



(c) Optimal solution $F^* = \{3\}$.

Figure 1.1: A DFVS problem instance, a suboptimal solution, and an optimal solution. The first subfigure depicts an example of a DFVS problem instance with 5 vertices, 6 directed edges, and 2 simple cycles. The other two subfigures illustrate possible solutions and the resulting graphs without cycles. The red and green vertices mark the solution vertices and the dashed edges represent the correspondingly removed edges.

to this complexity that research on the variants of the FVS problem often focuses on approximation algorithms [BYGNR94], parameterized algorithms [CLL+08, FWY09] or graph classes for which efficient algorithms are possible [CCGP04, CCGP05].

However, mostly over the last decade, research on heuristic and especially metaheuristic procedures [COPS19, GLB13, MQR21, TFZ17] for FVS problems has developed as well. These approaches do not guarantee optimal solutions but they aim at providing good or near-optimal solutions in a shorter time and also for larger instances than exact algorithms. Compared to exact methods, this makes them more suitable and applicable to real-world scenarios where oftentimes a short runtime is more important than achieving optimal solutions. The performance and solution quality of metaheuristic approaches can be further increased by combining different ones or incorporating other techniques such as mathematical programming, constrained programming and machine learning, which leads to so-called *hybrid metaheuristics*.

1.1 Aim of the Work

The aim of this work is to design, implement and evaluate hybrid metaheuristics for the unweighted DFVS problem. Such a procedure consists of multiple components and we have to carefully decide for each of them which approach to use in order to achieve the best results.

The first step is to find suitable reduction rules that are applied in a preprocessing step. These rules define how the size of a given graph can be decreased, which is relevant for the performance of the overall procedure because a smaller graph is easier to handle and results in a smaller search space for possible solutions. However, as there is a vast amount of different graph topologies, we seek to find a set of rules that can achieve a reduction for most graphs. Besides, it is important that the applied reductions are not only effective but also efficient as the preprocessing should only take a fraction of the total runtime. Hence, we test and compare the results of various rules and their combinations. Such reductions could include removing all vertices with an in-degree or out-degree of zero or merging vertices with an in-degree or out-degree of one as proposed by Levy and Low [LL88].

For the general metaheuristic framework, we focus on a popular category of local search (LS) metaheuristics called *large neighborhood search* (LNS) which was proposed by Shaw [Sha98]. The key idea of LNS is to consider much larger neighborhoods than in normal LS by using more complex move operators or a sequence of them and to search these neighborhoods by more effective techniques than naive enumeration. This is often achieved by using a pair of a *destroy* and a *repair* method. A destroy method changes a part of the current solution, which can be done randomly or by following heuristic rules. The repair method then can fix and also modify the solution resulting from the destroy method in order to achieve again a valid solution. To this end, we design and test multiple move operators based on such destroy and repair methods and compare their performance regarding runtimes and resulting solutions. In our case, the destroy methods are used to move vertices either from the current solution to the resulting DAG or the other way around. The repair methods then employ a *mixed integer linear programming* (MILP) formulation to encode the subproblem created by the destroy method. This smaller DFVS problem is solved by an existing MI(L)P solver like the SCIP[1] [GAB$^+$20] or Gurobi[2] solver.

The hybridization of the metaheuristic is achieved by solving a restricted MILP model of the DFVS problem in the repair operator of the LNS procedure. One potential MILP model is inspired by the subtour elimination constraints from Miller, Tucker and Zemlin and is based on the formulation for the weighted and undirected FVS problem as introduced by Melo et al. [MQR21]. Other possible formulations model the problem of cycle elimination and the reachability problem in graphs. We seek to compare the

[1]https://www.scipopt.org
[2]https://www.gurobi.com

performance of the different MILP models not only within the metaheuristic framework but also as standalone solution approaches to the DFVS problem.

A construction heuristic is also needed for the metaheuristic procedure. This heuristic is used to generate an initial solution that serves as the starting point for the local search. For this, we use a greedy function proposed by Cai et al. [CHJ06] as provided by Tang et al. [TFZ17] as a base. This function computes for each vertex a value that is based on its in- and out-degree as well as the difference between these degrees. Then, vertices are greedily added to the DFVS in decreasing order of their value until the remaining graph becomes acyclic. We also consider variants of and extensions to this heuristic and compare the runtimes and the quality of resulting solutions to determine the most suitable construction method.

As the solution generated by the construction heuristic can be far from optimal, we then apply a classical local search to it. The intention behind this is to find or at least come closer to a local optimum because a smaller initial solution is beneficial for the following LNS. However, as this is only a preparatory step, we use a small neighborhood that can be easily searched. This is achieved by iteratively removing a single vertex from the DFVS and accepting the new solution if the remaining graph is still a DAG.

After choosing the best performing components, we test multiple variants of our proposed hybrid metaheuristic, which differ in the used parameters for the neighborhood, on a set of benchmark instances. Finally, the following three research questions will be answered:

RQ1: Which MILP model performs best outside of and also within the metaheuristic framework?

RQ2: Which neighborhood of the LNS achieves the best results with regards to runtime and solution quality?

RQ3: What is the performance of the variants of the proposed hybrid metaheuristic for the DFVS problem in terms of average solution quality and number of found optimal solutions and how do they compare to each other?

1.2 Methodological Approach

Below, the methodological approach is described:

1. The first step is to conduct a literature research. We will start with collecting knowledge about the FVS problem and its variants and then gather information about existing solving approaches and state-of-the-art methods. We will not only do research on exact and heuristic procedures but also on graph reduction rules that are used for preprocessing.

2. Various MILP formulations that are inspired by the literature will be designed and tested as standalone solvers on benchmark instances in a preparatory computational

study. Their performance will be compared with regards to average solution quality and number of found optimal solutions. Based on the results, the first part of RQ1 can be answered and the MILP model that is most suitable for us will be selected.

3. Then, we will use the knowledge acquired from the literature to pick promising reduction rules that should be fast and effective. We will test the rules separately as well as combinations of them in order to find the best setting which will then be incorporated in our solving procedure.

4. We will use the established LNS procedure as the metaheuristic framework of our approach and the greedy function by Cai et al. [CHJ06] as a base for the construction heuristic. Variants of and extensions to this heuristic will be tested on benchmark instances. We will compare the runtimes and the quality of resulting solutions to determine the construction method that has the best performance on average.

5. An LS with a small neighborhood will be applied in order to improve the initial solution generated by the construction heuristic and to provide a better starting point for the LNS.

6. For the LNS, we will apply and test various neighborhoods with different move operators. These options will be assessed based on their average solution quality and the number of found optimal solutions and this evaluation will yield answers to RQ2.

7. Next, we will enhance the LNS metaheuristic with the previously chosen MILP model to achieve a hybridization. The other formulations will be separately incorporated as well resulting in multiple variants of the hybrid metaheuristic. To evaluate our choice, these variants will be tested on benchmark instances and compared to each other in terms of average solution quality and number of found optimal solutions. The results will also be used to answer the second part of RQ1.

8. The last step is to do a comparative study of multiple variants of our hybrid metaheuristic, which are achieved by changing some parameters of the used neighborhood. Again, they will be tested on benchmark instances and compared by average solution quality and number of found optimal solutions. This evaluation will provide the information needed to answer RQ3.

1.3 Problem Formalization

The DFVS problem that is discussed in this work is defined on a directed graph $G = (V, E)$ with a set V of vertices and a set E of directed edges, also called arcs. We only consider unweighted and simple graphs, where simple means that there are no parallel edges or self-loops. A *topological ordering* of a directed graph denotes a total order, or linear order, of the vertex set such that for every arc, the start vertex comes before the end vertex in the topological ordering. Thus, all arcs have to go forward and point to the right when illustrating a topological ordering and any arc going to the left is called a *backward arc* and constitutes a violation of the property of the topological ordering. Therefore, a topological ordering is only possible for directed graphs that have no directed cycles as any cycle would introduce at least one backward arc into the ordering. A *directed acyclic graph* (DAG) is a directed graph without any directed cycles and a directed graph is a DAG if and only if it has a topological ordering. Every DAG can have multiple topological orderings but has at least one and such an ordering can be found in time $\mathcal{O}(|V| + |E|)$.

Let $V' \subseteq V$ be a subset of V, the induced subgraph is denoted by $G[V'] = (V', E')$ with the vertex set V' and the arc set E' of arcs from E whose endpoints are all contained in V', $E' = E \cap (V' \times V')$. A *directed feedback vertex set* (DFVS) is a subset F of vertices that contains at least one vertex of every simple cycle in G. Let $\bar{F} = V \setminus F$ be the complement of F, then the induced subgraph $G[\,\bar{F}\,]$ is a DAG if and only if F is a valid DFVS of G.

The goal of the DFVS problem is to find a minimum cardinality directed feedback vertex set F^* with $|F^*| \leq |F|$ for every DFVS $F \subseteq V$. By the definitions above, solving the DFVS problem is equivalent to finding a vertex set $\overline{F^*}$ of maximum cardinality such that the induced subgraph $G[\,\overline{F^*}\,]$ is acyclic.

1.4 Key Results

We conduct a computational study to test and evaluate our proposed solving approaches. The results show that our graph reduction procedure is successfully applicable to more than 75% of all tested instances and achieves reductions of up to 100%, i.e., sometimes even completely solving an instance, with an average runtime of less than a second. We believe that an extension with further reduction operations from the literature could increase the effectiveness while hardly impacting the efficiency, considering the performance of practical solvers that we learned about later.

The experiments with our standalone MILP solving procedures on the pace-public benchmark instances show that the formulation based on the Miller-Tucker-Zemlin (MTZ) constraints performs best in terms of average solution quality when no initial solution is provided, whereas the formulation based on cycle elimination constraints (CEC) is superior in the case of warm starts. Using the CEC-based formulation within our hybrid metaheuristics also leads to higher mean values for the solution quality than employing the MTZ-based formulation. Besides, the combination of CEC with the enlarge-DAG neighborhood structure outperforms all approaches based on the enlarge-partial-DFVS neighborhood. In general, all hybrid LNS variants achieve higher average solution qualities than the standalone MILP solving procedures.

Regarding the selection strategies for the degree of destruction, some dynamic selection strategies perform better than the fixed_degree(x) strategy, where x denotes the preselected value, and the random selection in terms of average solution quality for the pace-public data set. Enhancing the resulting variants of our hybrid LNS with dynamic selection of the MILP formulation leads to further improvements of the solution quality and to the overall best performing solving approach. These variants also show similar performances on other benchmark instances and manage to outperform two state-of-the-art solving procedures based on simulated annealing.

1.5 Structure of the Work

Chapter 2 discusses related work with focus on current state-of-the-art solving approaches for FVS problem variants. It also includes research on various reduction rules for graphs and a short overview of the best performing submissions to the PACE 2022 directed feedback vertex set challenge. Next, the used methodologies are introduced in Chapter 3 where we provide information on the fundamentals of local search, large neighborhood search, and mixed integer linear programming. Then, we present in detail our solving approaches in Chapter 4. We elaborate on our graph reduction procedure, the employed MILP formulations, and the different variants of our construction heuristic and local search. Moreover, we discuss the two neighborhoods and other components of our large neighborhood search as well as the hybridization. Chapter 5 deals with the computational study and the gathered results and insights. We end this thesis with the conclusion and an outlook on possible future extensions to this work in Chapter 6.

CHAPTER 2

Related Work

There are multiple variants of the FVS problem which generally differ in two dimensions: graph orientation and involvement of weights. Regarding the first, there are only two possibilities as a graph is either undirected or directed. For the latter, there are four variants. The most common terms are unweighted graphs, which have no weights at all, and weighted graphs which usually refers to edge-weighted graphs where a numerical value is associated with each edge. Another option are vertex-weighted graphs in which weights are assigned to the vertices as the name implies. It is also possible but rather uncommon to have edge weights as well as vertex weights in a graph. As FVS problems revolve around vertex sets, the weighted version is usually defined on vertex-weighted graphs. Hence, the four common FVS problems are unweighted and undirected, unweighted and directed, weighted and undirected, or weighted and directed.

Oftentimes, the main concepts of solving approaches that were designed for one variant can be adapted and applied to the others. This is also true for the closely related feedback arc set (FAS) problem that is defined on a directed graph and whose goal is to find a subset of arcs of minimum cardinality which contains at least one arc of every cycle in the graph. This problem can be reduced to the DFVS problem and vice versa while preserving feasible solutions. This is why we do not limit our literature review to only the unweighted DFVS problem but also consider other variants and related problems.

Another topic that is relevant to our solving approach for the DFVS problem is graph reduction. This problem is frequently discussed in literature about various graph problems as efficient solving techniques often go hand in hand with the need to the make the graph as small as possible. Thus, we list approaches to various feedback set problems and give a short description of each in the first section of this chapter, before also covering some works on graph reduction in the second section. The last section deals with the best performing solving approaches of the heuristic track of the PACE 2022[1,2] directed feedback vertex set challenge.

2.1 Feedback Set Problems

Galinier et al. [GLB13] introduce the first local search approach to the DFVS problem and also propose a new solution representation of the feedback sets. Instead of the direct representation as vertex sets, they make use of the property that every directed graph has a topological ordering if and only if it is a directed acyclic graph. Therefore, the DFVS problem is equivalent to finding a vertex set of maximum cardinality such that its induced subgraph is acyclic and this subgraph is represented by one of its topological orderings. This gives rise to an efficient neighborhood structure without the need for costly checks of candidate solutions because transforming a solution preserves its validity. Based on this, the authors then introduce a simulated annealing (SA) metaheuristic for the unweighted DFVS problem, which they call the Simulated Annealing for the Feedback Vertex Set Problem (SA-FVSP).

Tang et al. [TFZ17] extend the SA approach from Galinier et al. [GLB13] by applying nonuniform neighborhood sampling (NNS) in order to improve the algorithm which results in an algorithm named the Simulated Annealing for the Feedback Vertex Set Problem using Nonuniform Neighborhood Sampling (SA-FVSP-NNS). In the base algorithm, uniform probabilities are used in each iteration of the SA to select the next move which is effective but not efficient as the sampling process – the process of choosing a feasible move – is unguided. In this work, a new priority function and a sampling function to guide the search are proposed. The priority function estimates and scores the quality of neighbor solutions and utilizes the following greedy function by Cai et al. [CHJ06], which applies two heuristic rules:

$$h(v) = \deg^-(v) + \deg^+(v) - \lambda \times \left|\deg^-(v) - \deg^+(v)\right| . \tag{2.1}$$

This greedy function computes the heuristic value for a vertex v and applies two heuristic rules based on the indegree $\deg^-(v)$ and the outdegree $\deg^+(v)$ of the vertex, and the parameter λ determines the impact of the rules. The sampling function then translates the scores from the priority function into sampling probabilities for the moves. Therefore, these functions allow for prioritizing moves that probably lead closer to the global optimum by assigning them higher sampling probabilities which makes them more likely to be chosen in the sampling process. This results in reaching local optima faster and finding better solutions and thus, SA-FVSP-NNS outperforms SA-FVSP in terms of solution quality.

Melo et al. [MQR21] focus in their work on the minimum weighted FVS (MWFVS) problem that is defined on undirected and vertex-weighted graphs. Similar to Galinier

[1] https://pacechallenge.org/2022/
[2] https://pacechallenge.org/2022/tracks/#heuristic-track

et al. [GLB13] and Tang et al. [TFZ17], their approach is based on solving the complement of this problem, the maximum weighted induced forest (MWIF) problem. The goal of the MWIF problem is to obtain a set of vertices of maximum weight which induces an acyclic subgraph. The remaining vertices form a feedback set of minimum weight and therefore provide a solution for the MWFVS problem. The authors introduce the first two compact MIP formulations for the MWIF problem where compact refers to a polynomial number of variables and constraints, and keeping these numbers small is usually beneficial when using MIP solvers. One formulation is flow-based and the other is based on an adaptation of a lifted version of the Miller–Tucker–Zemlin (MTZ) constraints. As both formulations are designed for connected and directed graphs, they are also applicable to the DFVS problem. Based on these MIP models, Melo et al. propose two variants of a hybrid metaheuristic that combines a multi-start iterated local search (MILS) heuristic with a MIP-based local search (MIP-LS) procedure. These variants are called MILS$^+$-mtz and MILS$^+$-flow, depending on which formulation is used in MIP-LS.

First, the MILS heuristic runs multiple independent iterations of a sequence of a greedy randomized construction heuristic, a local search procedure, an iterated greedy heuristic, and a multi-step perturbation phase and it collects the solutions of all iterations. Afterwards, a subproblem of the original MWIF problem is generated and solved in the MIP-LS which leads to the hybridization of the metaheuristic. This subproblem is obtained by fixing certain decision variables at a value, which is achieved by adding according constraints to the respective MIP formulation, and these variables and their values are selected from the solutions of the MILS heuristic based on some criteria. The resulting MIP model should be easier to handle by a standard MIP solver than the original problem because there are fewer variables left that have to be optimized. Computational experiments by Melo et al. show that both variants of the hybrid metaheuristic improve the solutions by MILS and that MILS$^+$-mtz outperforms MILS$^+$-flow. They also report that on large instances, the flow-based formulation solves more instances optimally, while the MTZ-based formulation generates better solutions on average.

Baharev et al. [BSNA21] introduce three IP models for the minimum FAS (MFAS) problem on edge-weighted graphs. One formulation is based on the relationship of the MFAS problem and the linear ordering problem which can be reduced to the complement of the MFAS problem, the maximum acyclic subgraph problem. This approach uses triangle inequalities to encode a minimum cost ordering of the vertices and only requires a polynomial number of variables and constraints. Such an IP can be efficiently solved with custom cutting plane algorithms but it does not perform well on large or sparse graphs. The second model uses the minimum set cover formulation of the MFAS problem which requires the enumeration of all simple cycles in a graph as constraints have to be added for each one. This formulation can then be solved with a branch-and-bound solver. However, besides the problem of cycle enumeration, this formulation can also result in an exponential number of constraints and is therefore often not feasible. The third IP formulation is also based on the set cover approach but uses lazy constraint generation to overcome the intractability of enumerating all simple cycles at once. In this approach, the model is built sequentially as constraints are added iteratively when new cycles are detected which makes it fast and efficient and leads to a better performance on sparse graphs than the other two.

Another approach to the minimum FAS problem on unweighted graphs is given by Noughabi and Baghbani [NB14] who propose a genetic algorithm called SGA. They exploit the relationship of the FAS problem with the linear ordering problem in a similar way to Baharev et al. [BSNA21] as they represent solutions as vertex orderings where all backward arcs constitute a FAS.

Cutello et al. [COPS19] introduce HYBRID-IA, a hybrid metaheuristics for the undirected FVS problem on vertex-weighted graphs, which is specifically designed for large-scale instances. Their population-based metaheuristic is inspired by the immune system, which makes it an immune algorithm (IA). It starts with the random creation of candidate solutions and then, three immune operators called cloning, hypermutation, and aging are used for population generation, neighborhood exploration, and diversification respectively. After the following population selection, the hybridization is achieved by applying local search to the new population to refine and improve the chosen solutions. This procedure is iterated until a certain number of generations is reached. Computational experiments show that the local search has a notable positive impact on the solution quality and that HYBRID-IA outperforms other state-of-the-art metaheuristics for the considered problem on multiple graph classes.

2.2 Graph Reduction

Levy and Low [LL88] describe the following five contraction operations, or reduction rules, for directed graphs, which are simple but still efficient:

1. IN0(v): If vertex v has an indegree of zero, $\deg^{-}(v) = 0$, delete v and all incident edges.
2. OUT0(v): If vertex v has an outdegree of zero, $\deg^{+}(v) = 0$, delete v and all incident edges.
3. IN1(v): If vertex v has an indegree of one, $\deg^{-}(v) = 1$, and no self-loop, let vertex u be the unique predecessor of v. Merge v into u by adding edges from u to all successors of v and then deleting v and all incident edges.
4. OUT1(v): If vertex v has an outdegree of one, $\deg^{-}(v) = 1$, and no self-loop, let vertex u be the unique successor of v. Merge v into u by adding edges from all predecessors of v to u and then deleting v and all incident edges.
5. LOOP(v): If vertex v has a self-loop, add v to the FVS and then delete v and all incident edges.

These proposed operations can reduce the size of a graph without changing any properties that are relevant for the FVS problem. Another important aspect is that the order in which the reduction rules are executed has no impact on the resulting graph. If the rules are iteratively applied in arbitrary order until the graph cannot be reduced any further, then the remaining graph, the contracted graph, is unique as Levy and Low [LL88] prove.

Park and Akers [PA92] also suggest four reduction operations in their work on the MFAS problem on unweighted graphs. Two of their rules describe how arcs and vertices can be removed from a graph when dealing with self-loops and vertices with no incoming or outgoing arcs and they are also covered by the operations 1, 2, and 5 of Levy and Low [LL88]. The other two operations are based on graph partitioning and the idea is to first split the graph into strongly connected and then into bi-connected components in order to get smaller graphs that are easier to handle.

Lin and Jou [LJ00] employ a graph reduction procedure when dealing with the minimum DFVS problem on unweighted graphs. They not only use the five operations introduced by Levy and Low [LL88] but also propose the following three new reduction rules: the Π-edges (PIE) operation, the CORE operation, and the DOME operation.

- The PIE operation first finds the PIE of the graph which is the set of Π-edges. Π-edges are edges that are involved in 2-cycles – cycles between two vertices – and they are temporarily removed from the graph. Then, the component graph of the remaining graph is constructed. This is done by partitioning the graph into strongly connected components (SCCs) and creating a vertex in the component graph for each SCC. For each edge connecting one SCC to another SCC in the original graph, a corresponding edge is added to the component graph. The resulting component graph is by construction a DAG and this means that no edge is part of a cycle. Thus, all edges of the original graph that are associated with edges in the component graph can be safely removed.

- The CORE operation identifies a d-clique of the graph together with its cores and adds all vertices in the d-clique except for one core to the FVS. Then, all vertices in the d-clique and their incident edges are deleted from the graph. A d-clique is a subgraph that only contains edges that are in the PIE of the graph and every pair of vertices in the d-clique must be connected by Π-edges. A core of a d-clique is then a contained vertex all of whose incident edges are also part of the d-clique.

- The DOME operation is based on the fact that only minimal cycles, which are cycles that do not contain all vertices of any other cycle, have to be broken for the computation of a minimum FVS. Based on the notation of Π-edges, the authors define dominated (DOME) edges that have to fulfill certain conditions. They state that all dominated edges can be removed from a graph without self-loops because no dominated edge is part of a minimal cycle and so, only nondominated edges are necessary for computing a minimum FVS.

These reduction rules do not affect the optimality of the FVS. All eight contraction operations are repeated until no further reduction is possible. Afterwards, the remaining graph is partitioned into its SCCs, as also suggested by Park and Akers [PA92], and if this results in multiple subgraphs, then the reduction procedure is again executed for each one.

Fleischer et al. [FWY09] also propose multiple reduction rules for the minimum DFVS problem on unweighted graphs: Self-Loop, Edge Canonicalization, Dummy Nodes, Chaining Nodes, Shortcut, and Flower. The rules Self-Loop, Dummy Nodes, and Chaining Nodes are equivalent to the rules LOOP, IN0 and OUT0, and IN1 and OUT1 of Levy and Low [LL88], and the Edge Canonicalization operation is only relevant for multigraphs as it replaces parallel edges by a single edge. The rules Shortcut and Flower are based on the computation of the maximum flower size of a vertex u which is the maximum number of cycles that are pairwise disjoint except for the one shared vertex u. This can be efficiently done with standard min-cut computations.

The Shortcut rule removes all vertices with a maximum flower size of one together with their incident edges but adds so-called shortcuts – new edges connecting all predecessors of a removed vertex with all its successors. The rule Flower is meant to be used with fixed-parameter tractable (FPT) algorithms as it depends on the corresponding parameter. It adds all vertices with a maximum flower size greater than the parameter to the FVS, decreases the parameter by one, and then removes the vertices and their incident edges from the graph. These operations can be iteratively applied in arbitrary order until the graph cannot be reduced any further without changing relevant properties for the DFVS problem.

2.3 PACE 2022

Since 2016, the Parameterized Algorithms and Computational Experiments (PACE) challenge has been held yearly with the goal of advancing theoretical and practical developments in the field of algorithmics. Every year, the focus lies on another algorithmic problem which happened to be the DFVS problem in 2022. This challenge was one of the inspirations for the topic of this thesis but as it took place at the same time as the development of our solving approaches, we were not able to already learn from its results[3]. However, we want to include a short description of the main concepts of the top three submissions for the heuristic track, which are listed in Table 2.1 together with our submission called HyMeHeu, short for hybrid metaheuristic. The points stand for the geometric mean of the solution qualities over 200 DFVS problem instances, with the solution quality for each instance being measured as the achieved percentage of the best known solution value. Our submission was a preliminary version of the solving approach ultimately described in this thesis, similar to the variant based on the MILP formulation with cycle elimination constraints as described in Chapter 5.5.1 but with SCIP as the employed MILP solver. The three best performing solvers have a similar

[3]https://pacechallenge.org/2022/results/

Table 2.1: The points of the best three and our submission (HyMeHeu) to the heuristic track of the PACE 2022 challenge.

Rank	Solver	Points
1	DiVerSeS	99.912
2	DreyFVS	99.911
3	PACE22_FVSP_HUST_SCP	99.852
	⋮	
13	HyMeHeu	92.644

general structure with the main components being a graph reduction procedure, one or more construction heuristics, and one or more improvement heuristics. Besides, they are all implemented in C++, which also complicates a fair comparison with our submission and final versions of our solving procedure that have the disadvantage of the just-in-time compilation of Julia, the programming language of our choice.

DiVerSeS[4] **[Swa22].** The heuristic version of this solver by Sylwester Swat starts with a graph reduction procedure including not only the operations proposed by Levy and Low [LL88] and Lin and Jou [LJ00] but also generalizations of rules for the vertex cover problem, and various new methods. Then, an initial solution is created with a construction heuristic that is chosen from a set of such heuristics based on certain properties of the reduced graph. These heuristics involve, for example, solving the vertex cover problem on a derived graph or employing an agent-flow algorithm. The size of the initial solution is reduced by an algorithm that removes vertices according to topological ordering properties of the corresponding DAG. The resulting solution is further improved by one of various solving approaches, which is again picked depending of certain graph characteristics. One of them is an enhanced version of the SA-FVSP by Galinier et al. [GLB13], whereas another one is again based on topological orderings and the vertex cover problem.

DreyFVS[5] **[BBCO$^+$22].** This solver by Bathie et al. also employs the reduction rules from Levy and Low [LL88] and Lin and Jou [LJ00] with an additional extension of the CORE operation. However, this graph reduction procedure is not applied before but during the generation of an initial solution. The construction heuristic is based on disjoint cycle unions and the Sinkhorn-Knopp algorithm. In every iteration, first the heuristic is used to find a vertex to be added to the DFVS and then the remaining graph is reduced by applying the reduction rules to every strongly connected component. Afterwards, the initial solution is improved by two greedy local search heuristics, both of which are consecutively applied to each strongly connected component of the graph. The move of the first LS consists in vertex swapping where a vertex is removed from the current

[4]`https://github.com/swacisko/pace-2022`
[5]`https://github.com/Nanored4498/DreyFVS`

DFVS and in case that this vertex now introduces a cycle into the DAG, another vertex of the cycle is randomly chosen and added to the DFVS. The second LS heuristic is based on the perturbation of topological orderings.

PACE22_FVSP_HUST_SCP[6] [DZZ22]. Du et al. also incorporate the five reduction rules from Levy and Low [LL88] and the three from Lin and Jou [LJ00] into their solver. After the graph reduction procedure, an initial solution is found by transforming the DFVS problem instance into a vertex cover problem and applying a heuristic algorithm to this new problem. The initial solution is then improved with a descent heuristic. The main solving procedure consists of an extension of the SA-FVSP algorithm by Galinier et al. [GLB13], which is achieved by randomly deleting vertices from the topological ordering, increasing the temperature, and repeating the simulated annealing process when a local optimum is reached. Besides, a transformation to the set covering problem and an algorithm for solving the set covering problem are employed to optimize the solution towards the end of a SA iteration.

[6]https://github.com/yumi-di/PACE-2022

CHAPTER 3

Methodology

This chapter presents and explains the fundamental methods that are applied in our solution approaches for the DFVS problem. We introduce the general concepts of local search algorithms in the first section, before discussing large neighborhood search in more detail in the next section. Then, we describe the basics of mixed integer linear programming.

3.1 Local Search

Local search algorithms fall into the class of improvement heuristics and they can be applied to a great variety of problems including hard combinatorial optimization problems. The general procedure is shown in Algorithm 3.1. Such algorithms are built on top of a construction heuristic that provides an initial solution for the LS. The construction heuristic is usually a simple and fast procedure that finds a solution for the problem at hand. This is often achieved by a greedy approach which iteratively selects the locally best element until a feasible solution is reached. The LS then tries to improve this initial solution by repeatedly making small changes. The nature of these changes is defined by the respective *neighborhood structure* that is employed. A neighborhood structure is a function $N : S \rightarrow 2^S$ that maps each solution $x \in S$ to a set of *neighbors* $N(x) \subseteq S$, which is also called the *neighborhood* of x. Neighborhoods are often implicitly defined by *move operators* or *moves* which specify how a solution is changed in order to generate a neighbor. Some simple moves are adding or removing a single element to or from the solution or exchanging two elements.

A *step function* determines the order in which the neighborhood is searched and also which neighbor is selected next. Random neighbor, next improvement, and best improvement are three step functions that are commonly used. The random neighbor approach randomly generates a neighbor in $N(x)$ which is then selected. Next improvement searches the neighborhood in a predefined order and chooses the first neighbor that is better than

Algorithm 3.1: Local Search procedure LS with the neighborhood N for a minimization problem

```
1 begin
2     $x \leftarrow$ initial solution;
3     repeat
4         choose an $x' \in N(x)$;
5         if $f(x') \leq f(x)$ then
6             $x \leftarrow x'$;
7         end
8     until termination criterion satisfied;
9 end
```

x, whereas best improvement always explores the whole neighborhood and then selects the best solution. No strategy dominates the others in general, but the employed step function can have a great impact on the performance of the algorithm regarding runtime and solution quality.

The local search is applied until one or a combination of multiple termination criteria are satisfied, which can be reaching a certain time limit or a predefined number of iterations. Another possibility is to run the algorithm until a local optimum is reached. For example, for a minimization problem a solution x is locally optimal with regards to a neighborhood structure N if and only if $f(x) \leq f(x')$ holds for all $x' \in N(x)$. In other words, there exists no better solution in the neighborhood of the current solution and thus, the local search cannot improve the solution any further. However, a local optimum does not have to be a global optimum as there could be another solution $x^* \in S$ with $f(x^*) \leq f(x)$ which is not part of the neighborhood. This is a big drawback of LS algorithms but there are ways to diminish this limitation such as combining different local search methods or using iterated local search where the LS is repeatedly applied to different initial solutions. Another technique is to change the neighborhood structure or to employ larger neighborhoods but there is always a tradeoff between the neighborhood size and the complexity of the search that has to be kept in mind.

3.2 Large Neighborhood Search

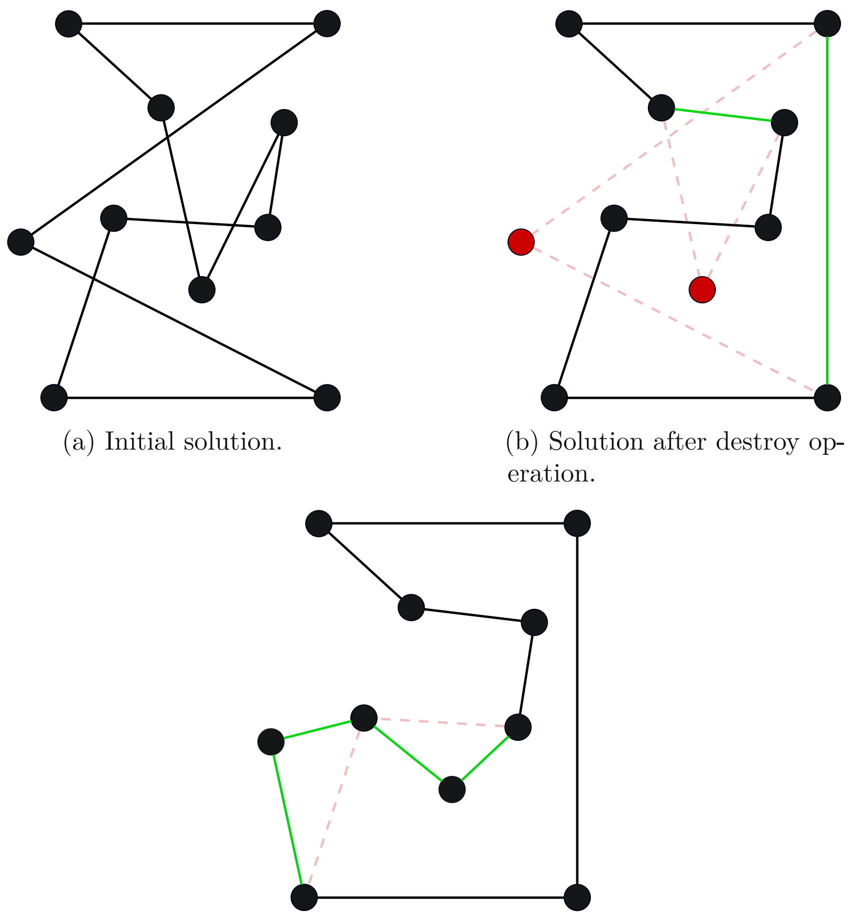
Large neighborhood search was proposed by Shaw [Sha98] to solve vehicle routing problems (VRPs). The following description is based on the work of Pisinger and Ropke [PR10] where the concepts of LNS are thoroughly explained.

Large neighborhood search is a prominent category of local search metaheuristics and more specifically, a type of *very large scale neighborhood search* (VLSN) algorithms. Any LS algorithm whose neighborhood size is either exponential in the size of the instance or just too big to be efficiently searched in its entirety by simply enumerating all candidate solutions qualifies as a VLSN algorithm. Therefore, the key idea of VLSN algorithms is to consider much larger neighborhoods than in normal LS but to investigate these neighborhoods by more effective techniques than naive enumeration. The intention behind this approach is to increase the quality of local optima and also of the final solution compared to LS with smaller neighborhood structures. For LNS, the neighborhood is only implicitly given by a pair of a *destroy* and a *repair* method which together constitute the move operator. A destroy method changes a part of the current solution, which usually renders it invalid. The repair method can then fix and also modify the resulting partial solution in order to achieve again a valid solution. Thus, the set of solutions that is generated by the sequential application of the destroy and repair methods to a current solution x represents the neighborhood $N(x)$.

In Figure 3.1, an example for such a destroy and repair operation for the *Traveling Salesman Problem* (TSP) is depicted. The TSP is a famous combinatorial optimization problem where a route through all given cities with the shortest travel distance has to be found. The tour has to be constructed in a way that each city is visited only once and that the starting point of the tour is also the end point. The problem is defined on an undirected graph with weights associated to each edge and it aims at finding a cyclic tour while minimizing the sum of edge weights. Figure 3.1a illustrates a feasible but not optimal solution of such a TSP instance. Starting with this solution, the destroy method randomly selects two cities, removes them from the tour and connects the respective preceding and succeeding cities with each other, which results in the state shown in Figure 3.1b. The repair operation restores a feasible solution by reinserting all now unvisited cities into the tour. This can be done using a greedy heuristic like the nearest insertion heuristic where a city with minimum distance to an already visited city is selected for the next insertion. The chosen city can then be inserted between any two cities in the tour such that the resulting route is the shortest possible. This selection and insertion process is repeated until all cities are part of the tour again and a valid solution is reached as depicted in Figure 3.1c.

In the example in Figure 3.1, the destroy operation picks two out of the ten cities for removal, which are 5% of the cities. If we use this degree of destruction of 5% for a TSP instance with 100 cities, then there are $\binom{100}{5} = 100!/(5! \times 95!) = 75287520$ possibilities to randomly pick the five cities to be removed. Different selections can result in the same solution after the application of the repair method, but this example still illustrates the huge dimensions of the neighborhood that such a neighborhood structure defines. Thus, this is also where the name *large neighborhood* search comes from.

The general LNS procedure is described by the pseudocode in Algorithm 3.2, which is taken from Pisinger and Ropke [PR10]. The three variables x, x^b, and x^t denote the current solution, the best solution found so far, and a temporary solution respectively. The destroy and repair operator are given by the functions d and r which take a solution as argument and return a changed copy of it. The algorithm takes a feasible solution x as input and returns the overall best solution x^b in the end. Like in local search, the initial solution x is often obtained by a construction heuristic and is used to initialized the best solution x^b in Line 3. The loop starting at Line 4 constitutes the main portion



(a) Initial solution.

(b) Solution after destroy operation.

(c) Solution after repair operation.

Figure 3.1: An example of a destroy and repair operation for a TSP instance. The red nodes represent the cities that are removed together with their incident edges from the tour during the destroy operation. The dashed red edges illustrate the removed edges and the green edges illustrate the edges that are added to the tour during the respective operation.

of the LNS where the stepwise improvements are made. In Line 5, the destroy method first returns a partly destructed solution and the repair method then modifies it to get a valid solution again. This new temporary solution x^t is then evaluated and either rejected or it becomes the new current solution depending on the *accept* function in Line 6. A simple acceptance criterion would be to only accept better solutions, whereas a more complex one could use the Metropolis criterion from simulated annealing (SA). In Line 9, the objective values of the temporary and the global best solution, denoted by the function c, are compared. If the temporary solution is better, it is stored as the new best solution in Line 10. Line 12 checks the stop criterion to decide whether or not another iteration has to be done. As for the acceptance criterion, there are various implementation possibilities for the termination criterion, some of which are mentioned in Chapter 3.1. If the termination criterion is satisfied, then the global best solution is returned in Line 13 and the algorithm stops afterwards.

Algorithm 3.2: Large Neighborhood Search procedure `LNS()` for a minimization problem

```
   Input: A feasible solution $x$
   Output: The best found solution $x^b$
1  begin
2      $x^b \leftarrow x$;
3      repeat
4          $x^t \leftarrow r(d(x))$;
5          if accept$(x^t, x)$ then
6              $x \leftarrow x^t$;
7          end
8          if $c(x^t) < c(x^b)$ then
9              $x^b \leftarrow x^t$;
10         end
11     until stop criterion is met;
12     return $x^b$;
13 end
```

The destroy and repair methods play an important role in the LNS metaheuristic as they define the neighborhood structure. For the destroy operator, the *degree of destruction* is essential and has to be chosen carefully. The degree of destruction determines the size of the destroyed part of the solution and therefore, it has a great impact on the exploration of the search space. If it is relatively small, then the search space might be too restricted and the benefit of a large neighborhood is mitigated. On the other hand, a very large degree of destruction might result in repeated re-optimization which can not only lead to long runtimes but also provide solutions of bad quality. One way to handle this difficult parameter is proposed by Ropke and Pisinger [RP06] who first determine a range of values based on the size of the instance and then randomly select the degree of destruction from this range in each iteration. A different approach is the progressive increment of the degree of destruction as seen in the work of Shaw [Sha98].

Another important aspect of the destroy operator is its capability of covering the whole search space, so that a global optimum can be found. Thus, it has to be avoided that only certain parts of the solution are destroyed over and over again. To do so, the destroy operator often contains some type of stochastic element to ensure diversification.

For the repair operator, there is the choice between optimal and heuristic methods. Here, optimal means that the destructed solution is repaired in a way such that the best possible feasible solution is achieved, whereas a heuristic only aims at constructing a feasible solution of good quality. A heuristic approach is usually faster but in return often needs more iterations to produce solutions comparable to optimal operations. An optimal method will always provide solutions of equal or better quality than the current solution. However, this approach might result in the LNS getting stuck at local optima which then must be countered with a high degree of destruction in the destroy method.

It is not only possible to use hand-coded methods for the repair operation but the metaheuristic framework also allows for the invocation of general purpose solvers. For example, a mixed integer programming or constraint programming solver could be used inside the repair method. In this case, the destroy and repair methods can be interpreted as *fix* and *optimize* operations. The destroy method serves as a fix operation where certain elements of the solution – namely the ones that are not destroyed – are fixed to their current state or value, whereas the destructed part is free for optimization. The optimization then happens in the repair method which leaves the fixed elements unchanged while searching a new feasible solution of better quality.

3.3 Mixed Integer Linear Programming

In this section, we discuss the basics of mixed integer linear programming. The following descriptions are mostly based on Chapters 1 and 10 of the book by Bertsimas and Tsitsiklis [BT97] and Chapter 1 of the work of Wolsey [Wol20] to which we would also like to refer interested readers for a more thorough explanation.

A mixed integer linear programming model, also called mixed integer linear program, can be used to mathematically model optimization problems. It is a type of linear programming (LP), the aim of which is to either minimize or maximize a linear cost function that is subject to linear equality and inequality constraints. The main components of an LP are a *cost vector* $\boldsymbol{c} = (c_1, \ldots, c_n)$, *decision variables* $(x_1, \ldots, x_n)$, a linear *cost function* $\boldsymbol{c}'\boldsymbol{x} = \sum_{i=1}^{n} c_i x_i$, and linear equality and inequality *constraints*. The cost function, or objective function, combines the cost vector and the decision variables, and the decision variables are subject to the constraints.

Thus, the general form of an LP for a minimization problem can be formulated as follows:

$$\text{minimize} \quad \boldsymbol{c}'\boldsymbol{x} \tag{3.1}$$

$$\text{subject to} \quad \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \tag{3.2}$$

$$\boldsymbol{x} \in \mathbb{R}^n \tag{3.3}$$

with $\boldsymbol{c} \in \mathbb{R}^n$, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, and $\boldsymbol{b} \in \mathbb{R}^m$ where m is the number of constraints. The objective function is given in Equation 3.1, the constraints are denoted by the inqualities in Equation 3.2, and Equation 3.3 defines the domain of the decision variables. In general, it is sufficient to use inequality constraints as every equality constraint can be expressed by two inequalities.

The constraints and the objective function can be seen as a description of the problem that is modeled in terms of which properties have to be satisfied and the goal that should be achieved. The decision variables represent the solution to this problem because finding a feasible solution to the linear program that satisfies all constraints provides a value assignment to all decision variables. These values can then be interpreted to get a solution to the problem. The goal is to find such a feasible assignment that minimizes or maximizes the objective function value, or in other terms to find an optimal solution. However, it is also possible that the constraints cannot be satisfied all at once and thus, no feasible solution exists. Besides, there are also LPs for which the objective value of an optimal solution is given by $-\infty$, in case of a minimization problem, because we can always find a feasible solution with an even smaller objective value than the one at hand. In this case, the objective value is called unbounded below and the LP is said to be unbounded.

One important characteristic of LPs is that even large instances can be solved efficiently in polynomial time. In practice, simplex algorithms are often used to solve linear programs but they have an exponential worst case time complexity. Interior point methods are oftentimes less efficient in practice but they are proven to have polynomial runtime, so they can be used instead for worst case instances of the simplex methods.

Mixed integer linear programs consist of the same components as LPs with the difference that the possible values for some but not all decision variables are limited to integers by introducing according constraints. A MILP with n integer variables and p real variables is then given by the following equations:

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}'\boldsymbol{x} + \boldsymbol{d}'\boldsymbol{y} && (3.4) \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} \geq \boldsymbol{b} && (3.5) \\ & \boldsymbol{x} \in \mathbb{Z}^n && (3.6) \\ & \boldsymbol{y} \in \mathbb{R}^p && (3.7) \end{aligned}$$

with $\boldsymbol{c} \in \mathbb{R}^n$, $\boldsymbol{d} \in \mathbb{R}^p$, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{B} \in \mathbb{R}^{m \times p}$, and $\boldsymbol{b} \in \mathbb{R}^m$ where m is the number of constraints.

Many relevant problems can be easily modeled by MILPs but unlike LPs, MILPs are in general NP-complete and therefore harder to solve. The combinatorial explosion of the number of possible solutions is also often a problem for MILPs where the decision variables can take arbitrary integer values. Therefore, heuristic methods such as local search heuristics are oftentimes applied for solving MILPs. Another solving approach are *LP-based Branch-and-Bound algorithms* which are employed in state-of-the-art solvers such as the Gurobi[1] and CPLEX[2] solvers as they often perform very well for MILP

[1] `https://www.gurobi.com`
[2] `https://www.ibm.com/analytics/cplex-optimizer`

models. These algorithms are based on the *linear programming relaxation* of the original MILP, which is an LP that is obtained by removing all integrality constraints from the MILP. This is done by replacing Constraint 3.6 with

$$\boldsymbol{x} \in \mathbb{R}^n. \tag{3.8}$$

This LP then has a solution space that also contains all solutions of the MILP, and as mentioned above, such an LP can be solved efficiently. If the optimal solution to the LP also satisfies the original integrality restrictions, it is also an optimal solution for the MILP. Otherwise, the LP solution is used to set a *lower bound*, also called *dual bound*, for the optimal solution of the MILP. Moreover, this allows for selecting one of the integer decision variables that are assigned fractional values in the LP solution and then, two constraints can be added to exclude this fractional value from possible solutions. Such a variable is referred to as a *branching variable* and its corresponding constraints are added separately to the original MILP, which results in two, more restricted MILPs. These two MILPs are handled in the same way by solving their LP relaxations and choosing branching variables. This process is continued until a solution to the original MILP is found or it can be determined that there is no such solution.

For MILP models, it is often preferable to only have a small number of constraints but sometimes, it is beneficial to use a formulation for a problem with a large amount of constraints. To handle the huge number of constraints, *delayed* or *lazy constraint generation* can be employed by MILP solvers, which means that the solver starts with an incomplete model and dynamically adds constraints during the solving procedure. These additional constraints are then called *lazy constraints* or *cuts* and their generation is often supported by MILP solvers by means of callback functions. These callback functions are invoked when the currently best solution, the *incumbent solution*, is improved by some heuristic and they can check whether the new solution would violate one or more constraints that are not part of the model yet. If such a violation occurs, then at least one violated constraint is added in order to render the solution invalid. An optimal solution can usually be found without having to add all possible constraints and thus, applying lazy constraint generation can lead to a substantial performance increase when solving MILPs with many constraints.

CHAPTER 4

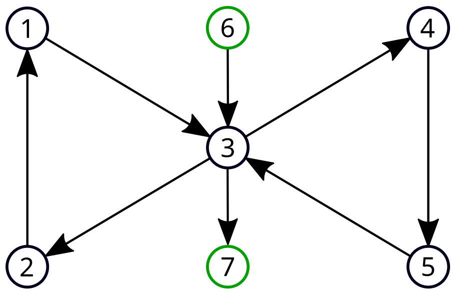
Solving Approaches

In this chapter, we present and discuss our solving approaches to the directed feedback vertex set problem. First, we explain the graph reduction that is done in a preprocessing step of our solving procedure. The second section introduces different MILP formulations of the problem, which can be used as standalone solvers as well as within an LNS procedure. Then, we discuss various construction heuristics before presenting some local search heuristics. The application of an LNS metaheuristic to the DFVS problem is described in depth in the next section, including the introduction of two neighborhood structures. The last section deals with the hybridized LNS that is achieved by using one of the MILP formulations and a MILP solver to respectively encode and solve the subproblem in the repair operation of the LNS.

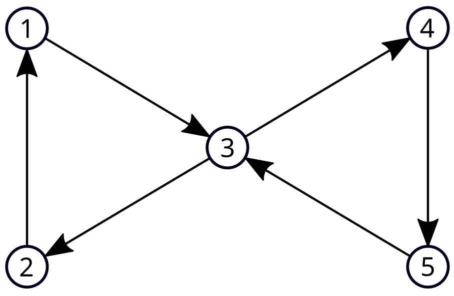
4.1 Graph Reduction

Our solving procedure starts with the preprocessing of the input graph where various measures for graph reduction are taken. The main step is the application of five reduction rules, which are used to reduce the size of the graph and make it easier to handle. Four of the rules come from Levy and Low [LL88] and the fifth reduction rule is inspired by Park and Akers [PA92]. We actually use all five contraction rules by Levy and Low [LL88] as described in Chapter 2.2, but we combine rules IN0 and OUT0 into a single step called IN/OUT0 which is illustrated in Figure 4.1. In this example, vertex 6 has an indegree of zero, $\deg^{-}(6) = 0$, and vertex 7 has an outdegree of zero, $\deg^{+}(7) = 0$, so both vertices and their incident edges are removed.

Rules IN1 and OUT1 are depicted in Figure 4.2 and Figure 4.3 respectively. In Figure 4.2a, vertex 1 has a single incoming edge from vertex 2, which means that $\deg^{-}(1) = 1$ and rule IN1 is applicable. The unique predecessor of vertex 1 is vertex 2, and the direct successors are vertices 3 and 4. Thus, we merge vertex 1 into vertex 2, add edges from vertex 2 to vertices 3 and 4, and remove vertex 1 and all its incident edges from the

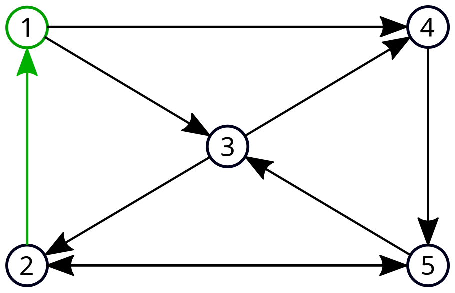


(a) Initial graph.

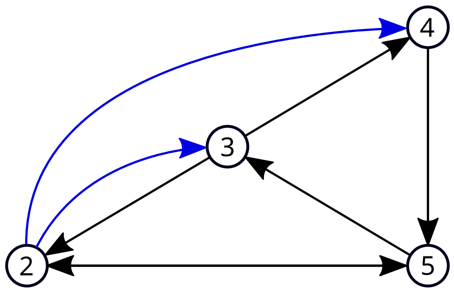


(b) Reduced graph.

Figure 4.1: An example of the application of reduction rule IN/OUT0. The green vertices in the initial graph mark the vertices for which the rule is applicable.

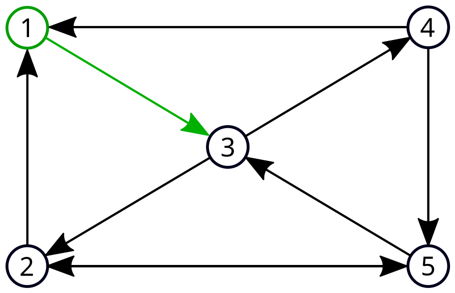


(a) Initial graph.

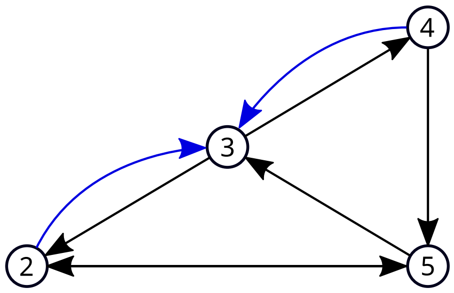


(b) Reduced graph.

Figure 4.2: An example of the application of reduction rule IN1. The relevant vertex and edge are marked green in the initial graph and the blue edges in the reduced graph are the newly added edges.



(a) Initial graph.

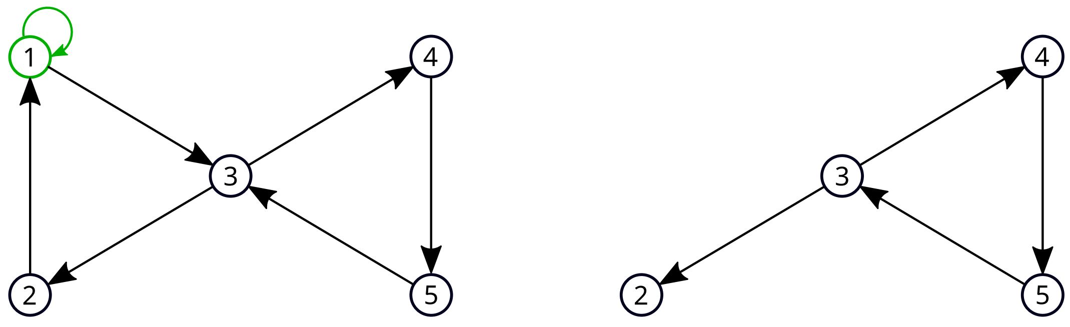


(b) Reduced graph.

Figure 4.3: An example of the application of reduction rule OUT1. The relevant vertex and edge are marked green in the initial graph and the blue edges in the reduced graph are the newly added edges.

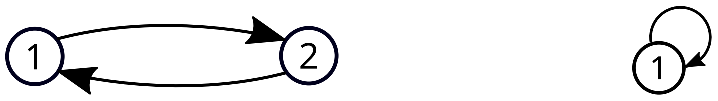
graph. Rule OUT1 works in a similar manner as shown in Figure 4.3. Again, it is vertex 1 that can be merged as it has an outdegree of one, $\deg^+(1) = 1$. Its unique successor is vertex 3 and the direct predecessors are vertices 2 and 4. Vertex 1 is then merged into vertex 3 by adding edges from vertices 2 and 4 to vertex 3 and deleting vertex 1 and its incident edges.

Rule LOOP deals with vertices with self-loops and states that all such vertices have to be added to the directed feedback vertex set and then they can be safely removed from the graph together with their incident edges as depicted in Figure 4.4. As our input graphs are exclusively simple graphs, this rule will never be directly applicable. However, it is possible to have self-loops in the graph after the application of the merging rules IN1 and OUT1, which is illustrated by the simplified example in Figure 4.5. Therefore, it is important to include this reduction in our procedure.



(a) Initial graph. (b) Reduced graph.

Figure 4.4: An example of the application of reduction rule LOOP. The relevant vertex and edge are marked green in the initial graph. The rule adds vertex 1 to the directed feedback vertex set, $F = \{1\}$, and then removes the vertex and its incident edges from the graph.



(a) Initial graph. (b) Reduced graph.

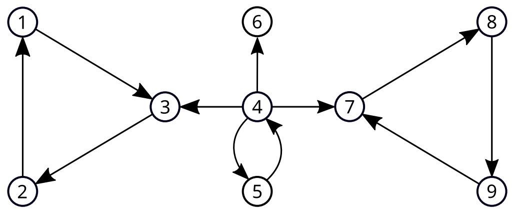
Figure 4.5: An example of how self-loops can be introduced by the reduction rules. When applying, for example, rule IN1 to vertex 2 in the initial graph, then vertex 1 ends up with a self-loop in the reduced graph.

Our fifth reduction rule is called SCC-reduction and is similar to one by Park and Akers [PA92] as mentioned in Chapter 2.2 because it is based on the partitioning of the graph into its strongly connected components. Additionally, we also include some more steps to achieve a stronger reduction, which leads to the following definition of the rule:

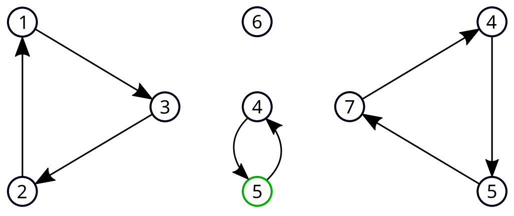
1. Partition the graph into its SCCs.
2. If an SCC contains a single vertex v, remove v and its incident edges from the graph.

3. If an SCC consists of two vertices, randomly pick one of them and add it to the DFVS. Then remove both vertices and their incident edges from the graph.
4. All other SCCs remain unchanged.

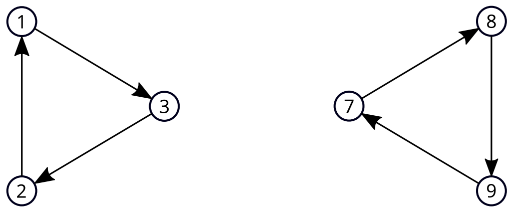
Figure 4.6 shows an example of how this rule works and to simplify matters, we ignore other reduction rules, which would be applicable. The original graph before the application of the rule is depicted in Figure 4.6a and Figure 4.6b illustrates the partitioning into the four SCCs. One SCC consists only of vertex 6 and according to step 2, this vertex can be removed. Vertices 4 and 5 constitute another SCC for which step 3 is applicable: Vertex 5 is added to the DFVS and both vertices and all their incident edges are removed from the graph. The other two SCCs are not further affected by this rule which leaves us with the graph shown in Figure 4.6c and the current DFVS $F = \{5\}$.



(a) Initial graph.



(b) Partitioned graph.



(c) Reduced graph.

Figure 4.6: An example of the application of reduction rule SCC. The green vertex in the partitioned graph marks the vertex that is added to the DFVS according to the rule.

As mentioned in Chapter 2.2, these reduction rules do not change graph properties relevant for the DFVS problem and their application order can be arbitrary. During testing, all these operations proved to be fast enough to be used in combination and also for repeated application. Therefore, the five reduction rules are repeatedly applied until none of them leads to a reduction of the graph anymore. They are executed in the order in which they are described here because, for example, it is beneficial to apply rules IN1 and OUT1 before LOOP as these rules might introduce new self-loops in the graph. The preliminary DFVS, which results from rules LOOP and SCC, is stored and will be added in the end to the solution from the remaining solving procedure.

After the exhaustive application of the reduction rules, the remaining graph is partitioned into its SCCs, which are then sorted by increasing size. If their number exceeds 1000, we merge smaller ones into subgraphs of up to 100 vertices to reduce memory consumption.

In general, all resulting subproblems are processed separately and sequentially, each starting with another repeated application of the five reduction rules. At this point, it can be possible to shrink the subgraphs even further because the graph partitioning removes edges, which can give the reduction rules new points of application. Where necessary, the DFVS is again extended with vertices selected by the reduction rules. Afterwards, components with exactly three vertices are treated as special cases that are dealt with by randomly selecting two vertices that are also added to the preliminary DFVS. In this case, you always have to pick two vertices due to the previous graph reduction which ensures that each pair of vertices in such a component is connected by edges in both directions, or in other words, these subgraphs of size three are always cliques. Subgraphs or SCCs with up to 100 vertices are also handled differently from bigger ones as they are directly encoded with one of the mixed integer programming models that are introduced in Chapter 4.2. This is also the reason why we chose the size of the merged subgraphs in case of many SCCs to be at most 100 as the small SCCs would have fallen into this category as well. The remaining bigger SCCs are passed to the large neighborhood search algorithm which is discussed in Chapter 4.5.

4.2 MILP Formulations

The DFVS problem can be formulated as a mixed integer linear programming model and there are various possibilities to do so. In this section, we introduce two main formulations, which are based on different approaches at tackling the DFVS problem. The first model is inspired by the subtour elimination constraints from Miller, Tucker, and Zemlin and is therefore called the *MTZ-based formulation*. The other formulation is based on the problem of cycle elimination and uses so-called *cycle elimination constraints* (CEC).

4.2.1 MTZ-Based Formulation

The MTZ-based formulation is derived from the MILP model by Melo et al. [MQR21]. Although their model was developed for solving the vertex-weighted and undirected FVS problem, it is designed for connected and directed graphs, which makes it suitable for us to use as a base. However, we make some minor changes in order to adapt the model to the DFVS problem on unweighted graphs and we also slightly modify the input graphs of the DFVS problem to match the resulting MILP formulation. The modification consists in adding a *source vertex* s to the graph and inserting directed edges from s to all other vertices. Thus, a graph $G = (V, E)$ results in a new graph $G_s = (V_s, E_s)$ with $V_s = V \cup \{s\}$ and $E_s = E \cup \{(s, v) \ : \ v \in V\}$. Let F be a directed feedback vertex set for G_s, $\bar{F}$ its complement, and the subgraph $G[\bar{F}]$ the induced DAG. The MTZ-based formulation models the complement of the DFVS problem and therefore, the aim is to maximize the cardinality of $\bar{F}$ and an optimal solution represents the largest acyclic subgraph. For the MILP model, we use three sets x_{uv}, y_v, and Φ_v of decision variables with

$$x_{uv} = \begin{cases} 1 & \text{if arc } (u,v) \in E_s \text{ is contained in } G[\,\bar{F}\,]; \\ 0 & \text{otherwise;} \end{cases} \tag{4.1}$$

$$y_v = \begin{cases} 1 & \text{if vertex } v \in V_s \text{ is contained in } G[\,\bar{F}\,]; \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

A variable Φ_v is introduced for each vertex $v \in V_s$ and is called a *potential variable*. In a valid solution, the values of potential variables have to increase with the distance of v to s, as stated by the MTZ constraints. The MTZ-based formulation of the DFVS problem is then given by the following model:

$$\max \sum_{v \in V} y_v \tag{4.3}$$

$$\text{s.t. } \Phi_u - \Phi_v + |V_s| x_{uv} \leq |V_s| - 1 \qquad \forall (u,v) \in E_s \tag{4.4}$$

$$y_v \leq \Phi_v \qquad \forall v \in V \tag{4.5}$$

$$\Phi_v \leq |V_s| - 1 \qquad \forall v \in V \tag{4.6}$$

$$x_{uv} \leq y_u \qquad \forall (u,v) \in E_s \tag{4.7}$$

$$x_{uv} \leq y_v \qquad \forall (u,v) \in E_s \tag{4.8}$$

$$x_{uv} \geq y_u + y_v - 1 \qquad \forall (u,v) \in E_s \tag{4.9}$$

$$y_s = 1 \tag{4.10}$$

$$\Phi_s = 0 \tag{4.11}$$

$$x_{uv} \in \{0,1\} \qquad \forall (u,v) \in E_s \tag{4.12}$$

$$y_v \in \{0,1\} \qquad \forall v \in V_s \tag{4.13}$$

$$\Phi_v \in \mathbb{R}_+^{|V_s|} \qquad \forall v \in V_s \tag{4.14}$$

The objective function in Equation 4.3 maximizes the number of vertices $v \in V$ which constitute $\bar{F}$ and belong to the DAG, or in other words, the vertices for which $y_v = 1$. Constraint 4.4 models the requirement for potential variables that the potential value Φ_v of a vertex v has to be larger than the potential value Φ_u of a vertex u if the corresponding edge $(u,v) \in E_s$ is chosen to be contained in $G[\,\bar{F}\,]$ by setting $x_{uv} = 1$. This constraint ensures that a valid solution contains no cycles by requiring that in the path from the source vertex s to a given vertex v, the potential value of v is bigger than the potential of its predecessors. Constraints 4.5 and 4.6 define the bounds for the potential values of vertices. For a vertex $v \in \bar{F}$, given by $v \in V$ with $y_v = 1$, Constraint 4.5 states that the potential value must be bigger than zero, whereas Constraint 4.6 ensures that the potential value does not exceed the length of the longest possible simple path starting at the source vertex s, which is denoted by $|V_s| - 1$. Constraints 4.7, 4.8, and 4.9 state that the solution is the induced subgraph $G[\,\bar{F}\,]$. First, Constraints 4.7 and 4.8 guarantee that if an edge (u,v) is in the solution, then also both endpoints u and v are selected. On the other hand, Constraint 4.9 requires that the edge (u, v) between two selected vertices u und v also has to be in the solution. Constraint 4.10 always forces the source vertex s to be selected. Constraint 4.11 fixes the potential value Φ_s of the source vertex s at zero, which is possible as Φ_s is not affected by Constraints 4.5 and 4.6 who do not include s. Lastly, Constraints 4.12, 4.13, and 4.14 define the domains of the decision variables and guarantee their integrality and nonnegativity respectively.

This model is a so-called *compact* formulation because it uses only a polynomial number of variables and constraints. To be precise, both the number of variables and constraints of the MTZ-based formulation are in $\mathcal{O}(|V|+|E|)$. As mentioned before, this formulation is used to solve the complement of the DFVS problem and this leads to the following observation, which corresponds to the observation made by Melo et al. [MQR21] for the MWFVS problem:

Observation 1. *Let the vector $\boldsymbol{y}$ be a solution for the complement of the DFVS problem, then the corresponding solution for the DFVS problem is denoted by $1 - \boldsymbol{y}$. Thus, the objective value of the DFVS problem is given by $\sum_{v \in V} (1 - y_v)$.*

We noticed that it is possible to reduce the size of the model even further as for the DFVS problem, there is no need to explicitly model edges. Therefore, we removed all decision variables x_{uv} and their associated constraints, which results in the following formulation:

$$
\begin{aligned}
\max \quad & \sum_{v \in V} y_v && && (4.15)\\
\text{s.t.} \quad & \Phi_u - \Phi_v + |V_s| y_v \leq |V_s| - 1 && \forall (u,v) \in E_s && (4.16)\\
& y_v \leq \Phi_v && \forall v \in V && (4.17)\\
& \Phi_v \leq |V_s| - 1 && \forall v \in V && (4.18)\\
& y_s = 1 && && (4.19)\\
& \Phi_s = 0 && && (4.20)\\
& y_v \in \{0,1\} && \forall v \in V_s && (4.21)\\
& \Phi_v \in \mathbb{R}_+^{|V_s|} && \forall v \in V_s && (4.22)
\end{aligned}
$$

Aside from the removed variables and constraints, this model differs from the original one above in only one aspect: Constraint 4.16 replaces Constraint 4.4. In Constraint 4.16, the decision variable y_v is used instead of x_{uv} as before. However, the meaning of the constraint remains the same as it still prevents cycles in the solution by enforcing increasing potential values along paths originating in the source vertex s. The objective function 4.15 and Constraints 4.17 through 4.22 are the same as before and their meaning is as previously defined. This reduced model has $\mathcal{O}(|V|)$ decision variables but the number of constraints is still in $\mathcal{O}(|V|+|E|)$ because of Constraint 4.16. Observation 1 also applies to this formulation. For the remainder of this work, the name MTZ-based formulation refers to this smaller model and MTZ-orig denotes the larger original formulation.

4.2.2 CEC-Based Formulation

Another possibility to model to DFVS problem as a mixed integer linear program is the CEC-based formulation. As the name implies, the CEC-based formulation is based on cycle elimination constraints and its goal is to add at least one vertex of each simple cycle to the directed feedback vertex set. To further speed up the solving process, we also define so-called *2-cycle constraints* which are separate constraints for all cycles of length two – 2-cycles. Additionally, we strengthen them by identifying cliques in the graph G_{2cyc} that has an undirected edge for each cycle of length two. In more detail, $G_{2cyc} = (V_{2cyc}, E_{2cyc})$ with $E_{2cyc} = \{(u, v) \ : \ u < v, (u, v) \in E, (v, u) \in E\}$ and $V_{2cyc} = \{u, v \ : \ u, v \in V, (u, v) \in E_{2cyc}\}$. Cliques in G_{2cyc} correspond to cliques in G where all pairs of vertices are connected by 2-cycles, hence the name 2-cycle constraints.

To find the largest clique for each vertex $v \in V_{2cyc}$, we start with an edge $(v, u) \in E_{2cyc}$, which already is a clique of size two. Then, we apply a greedy approach to extend this clique: First, we get all neighbors of v and all neighbors of u in G_{2cyc} and determine their intersection. Thus, the intersection is the set of all vertices that are part of cliques involving v and u. If the intersection is empty, then the clique cannot be extended and we already have a maximal clique. Otherwise, we have to identify the largest possible clique, which is done by greedily picking a vertex from the intersection and adding it to the clique. This process is repeated until a maximal clique is found. In the end, a 2-cycle constraint is added for each such clique. However, the model would already be complete without these 2-cycle constraints and they are only added for runtime improvements. Therefore, we install a time limit for computing and adding such constraints, which might result in not adding all possible ones in case of a large amount of 2-cycles in the graph.

For the CEC-based formulation, let F be a directed feedback vertex set for the input graph and $\bar{F}$ its complement. We introduce decision variables y_v for all vertices $v \in V$ with

$$y_v = \begin{cases} 1 & \text{if for vertex } v \in V \text{ it holds that } v \in \bar{F}; \\ 0 & \text{otherwise.} \end{cases} \tag{4.23}$$

Furthermore, let C denote a simple cycle and $\mathcal{C}$ the set of all simple cycles in the original graph. Similarly, we use K to denote a clique in G_{2cyc} and $\mathcal{K}$ for the set of such cliques that we identified by using the procedure described above. The CEC-based formulation is then expressed as follows:

$$\min \sum_{v \in V} (1 - y_v) \tag{4.24}$$

$$\text{s.t.} \sum_{v \in C} (1 - y_v) \geq 1 \qquad \forall C \in \mathcal{C} \tag{4.25}$$

$$\sum_{v \in K} y_v \leq 1 \qquad \forall K \in \mathcal{K} \tag{4.26}$$

$$y_v \in \{0, 1\} \qquad \forall v \in V \tag{4.27}$$

The objective function 4.24 minimizes the number of vertices $v \in F$ by assigning $y_v = 0$ to as many vertices $v \in V$ as possible while satisfying the constraints. The cycle elimination constraints are given by Constraint 4.25. Constraint 4.26 ensures that at most one vertex of each clique K remains in the DAG because in order to break all cycles involved in a clique we have to add all vertices but one to the directed feedback vertex set. The domain of the decision variables y_v is given by Constraint 4.27, which restricts them to the binary values zero and one. The number of variables is in $\mathcal{O}(|V|)$ but the number of constraints can grow exponentially with the size of the graph. Therefore, we employ lazy constraint generation for dynamically adding the cycle elimination constraints during the solving process in order to improve the performance.

4.2.3 Applications of MILP Formulations

We use the MTZ- and CEC-based formulations in two ways when tackling the DFVS problem: as standalone solution approaches and within the metaheuristic framework. In the former case, the MILPs can be used on their own to solve instances of the DFVS problem as they are designed to completely model the problem. First, we apply the graph reduction measures as described in Chapter 4.1 to reduce the size of the input graph and potentially split it into smaller subproblems. Each (sub-)problem is then encoded with one of the formulations and handed over to a MI(L)P solver like the SCIP[1] [GAB+20] or Gurobi[2] solver for optimization. If the solver terminates without an error, the resulting variable assignments are used to determine the vertices in the DFVS.

The other approach is to use a MILP model and solver within the repair method of the LNS metaheuristic. In this case, one of the formulations is used to encode the smaller DFVS subproblem that is posed by the partial solution as received from the destroy operator. Then, a MI(L)P solver can solve the subproblem and the resulting solution is used to restore a valid solution.

[1]https://www.scipopt.org
[2]https://www.gurobi.com

4.3 Construction Heuristics

A construction heuristic is required by the LNS metaheuristic for providing an initial solution, which has to be a valid DFVS. Although it is not necessary, it is beneficial to already generate a solution of high quality because it serves as the starting point for the following LNS. However, it might be even more important to have a fast CH as the LNS should be able to achieve major improvements anyway and the goal is to develop an overall efficient solving procedure. Below, we propose two different approaches to building an initial solution and a third one that is based on a combination of the ideas from the previous two, and then we compare them to each other.

4.3.1 Degree-Based Construction Heuristic

The degree-based construction heuristic utilizes the greedy function by Cai et al. [CHJ06] as previously given in Equation 2.1: $h(v) = \deg^-(v) + \deg^+(v) - \lambda \times |\deg^-(v) - \deg^+(v)|$. In this function, two heuristic rules based on the indegree $\deg^-(v)$ and the outdegree $\deg^+(v)$ of a vertex v are applied for the computation of its heuristic value $h(v)$. The first heuristic corresponds to the first part of the equation, $\deg^-(v) + \deg^+(v)$, and states that a vertex whose in- and outdegree are large is likely to be contained in an DFVS of minimum cardinality. The second part of the equation, $|\deg^-(v) - \deg^+(v)|$, reflects the second heuristic which indicates that a small difference between the in- and outdegree makes a vertex more likely to go into a minimum DFVS. Thus, the higher the heuristic value of a vertex, the higher the chance that it belongs to the DFVS according to this heuristic. Moreover, the parameter λ determines the impact of these rules on the heuristic value. If the value of λ is zero or very small, then the first heuristic overpowers the second one, whereas a large value strengthens the influence of the second heuristic. We use $\lambda = 0.3$ as recommended by Cai et al. [CHJ06].

In the degree-based CH, we first compute the heuristic values of all vertices in the graph and sort them in decreasing order of this value. This means that the vertex with the highest value, who should also be the one with the highest probability of belonging to the solution, comes first in the sorted list of vertices. Then, we iterate over the sorted list and execute two steps for each vertex v:

1. Add v to the directed feedback vertex set F.
2. Check whether $G' = (V \setminus \{v\}, E)$ is acyclic.

If G' is acyclic, then F is a valid solution and we stop, otherwise the iteration continues.

4.3.2 Construction Heuristic Based on Topological Ordering

The idea behind the construction heuristic based on topological ordering is, as the name implies, to exploit the procedure of finding a topological ordering of the input graph. To determine a topological ordering of a graph, we employ an algorithm based on depth first search (DFS). As mentioned before, a graph has topological ordering if and only if it is acyclic. Therefore, an algorithm that can find a topological ordering often involves some type of cyclicity check and when a cycle is detected, it terminates without solution as no topological order is possible in this case. We make use of this feature in our procedure but instead of terminating when encountering a cycle, we add the current vertex under examination to a DFVS F. This allows the algorithm to continue the construction of a topological ordering until all vertices in the graph have been examined. In the end, this gives us not only a valid DFVS but also a topological ordering of the corresponding DAG.

The DFVS produced by the DFS-based topological ordering algorithm can already be used as an initial solution but it is often quite large. To achieve a better solution quality, we further enhance the CH with additional checks for the vertices in F. First, we mark all vertices in the previously found topological ordering as visited because they do not have to be checked again, and we create a new, empty DFVS F'. For each vertex $v \in F$, we then

1. collect all direct successor vertices,
2. compute the intersection S between the successors and visited vertices,
3. add v to F' if $S \not\subseteq F'$,
4. mark v as visited.

The underlying idea is that we iteratively try to append the vertices in F to the end of the topological ordering. If a vertex has a successor that already has been checked and is part of the topological ordering, appending the vertex would introduce a backward arc and destroy the topological sort. Thus, such vertices cannot be added to the topological ordering and have to be in the DFVS.

This enhancement proved beneficial as in general it decreases the solution size while barely impacting the runtime of the construction heuristic. We also tried sorting the vertices based on their heuristic value, as described for the degree-based CH, before applying the DFS but overall this did not lead to an improved solution quality.

4.3.3 Construction Heuristic Based on Degree and Topological Ordering

This construction heuristic combines the main concepts of the previous two approaches: the greedy function and topological orderings. The idea behind this CH is to sort the vertices based on their heuristic value and to use this order to greedily build a topological ordering of maximal cardinality. In the end, the vertices that are not contained in the topological ordering provide a valid DFVS and an initial solution for the DFVS problem.

First, we employ the greedy function by Cai et al. [CHJ06] with the recommended balancing factor $\lambda = 0.3$ to compute the heuristic value of all vertices. In contrast to the degree-based CH, the vertices are then sorted in increasing order of their heuristic value, so that the vertex with the lowest value comes first. According to the heuristic, this vertex is the most unlikely to belong to a minimum DFVS and therefore, it is also the vertex with the highest chances to be contained in a maximum topological ordering. Then, we use the same iterative procedure as in the CH based on topological ordering to greedily construct a topological ordering and the according DFVS. The only difference is that at the start, the topological ordering and thus also the set of visited vertices are empty.

4.3.4 Comparison

A desirable construction heuristic is often characterized by a high solution quality and a short runtime. Regarding the latter, we consider runtimes of a couple seconds as acceptable for large DFVS problem instances. The degree-based CH does not fulfill these runtime requirements because, for example, it exceeds even 60 seconds for big instances, as we found out during testing. This bad performance is due the cyclicity check that has to be repeated after every vertex, which tends to be expensive for large graphs as the check itself takes longer and also usually has to be done quite often until a directed feedback vertex set is found. For instance, DFS can be used to determine whether a graph is cyclic or not, which can be done in time $\mathcal{O}(|V| + |E|)$, but this has to be repeated for every vertex in the worst case.

In contrast, this is not a problem with the CH based on topological ordering where no such repeated check has to be done as it only takes one run of the DFS to find a valid DFVS and topological ordering. As mentioned before, the additional enhancement does not substantially worsen the runtime of this construction heuristic and it still satisfies our runtime requirements. However, one potential drawback is that the resulting DFVS mostly depends on the topological ordering found by the DFS. This topological ordering might not be the most suitable for the applied heuristic, which then can also lead to a bad initial solution. Hence, even though the CH based on topological order provides better initial solutions than the degree-based CH on average, there is still room for improvement.

This is where the CH based on degree and topological ordering comes in because it removes the dependence on a specific topological ordering and only exploits the general concept of topological orderings. Besides, it uses the greedy function from the degree-based CH to guide the construction of the topological ordering, but it does not involve any expensive cyclicity checks. Thus, it combines the strengths of the other two construction heuristics while avoiding their weaknesses. This results in a construction heuristic that is not only comparably fast as the CH based on topological ordering but also provides better initial solutions than the other two. Therefore, we use the CH based on degree and topological ordering in our solving procedure for the DFVS problem.

4.4 Local Search

The solution generated by the construction heuristic can be far from optimal, so we try to improve the solution quality by applying a classical local search. The intention behind this is to find or at least come closer to a local optimum because a smaller initial solution is beneficial for the following LNS. However, as the LS is only a preparatory step, we define two small neighborhoods that can be easily searched: the one-flip neighborhood and the topological ordering neighborhood. This leads to two LS procedures and we also propose an enhanced variant of the LS based on the topological ordering neighborhood.

Before describing the neighborhood structures in detail and comparing them to each other, we present the configuration of the LS components that remain the same in all our LS algorithms: During the search, only better solutions are accepted and for the step function, we always use *next improvement* to search the neighborhood in a predefined order and determine the first improving neighbor. In general, the local search is applied until a local optimum is reached but this can be very time-consuming for large DFVS problem instances. Therefore, we introduce a time limit of 60 seconds as additional termination criterion.

4.4.1 One-Flip Neighborhood

The one-flip neighborhood structure is defined by a move that changes the state of one vertex in the DFVS. To be precise, the move operator tentatively removes a single vertex from the current DFVS, which is equivalent to adding it to the corresponding DAG, and accepts the resulting solution if it is still a valid DFVS whose removal leads to an acyclic graph. To guide the search, we determine an order for the vertices of the DFVS, which is again based on the heuristic and greedy function by Cai et al. [CHJ06] (cf. Equation 2.1). The vertices are sorted in increasing order of their heuristic value, so that the vertex for which it is most likely that it can be added to the DAG is selected first. Then, we have to determine whether the resulting candidate solution is a valid DFVS by checking the remaining graph for cycles. If no cycle is detected, this solution is returned, otherwise the vertex is put back into the DFVS and the next vertex is picked for removal.

In the case that a valid solution is found, we additionally also store the position of the selected vertex in the sequence of vertices belonging to the DFVS. Due to this, the following iterations of the local search can simply continue with the next vertex in the order instead of having to repeatedly check vertices that have already been deemed unsuitable. This is possible because the heuristic value of the vertices and therefore also their position in the sequence does not change together with the solution as the heuristic value is solely based on the degree of the vertices in the original graph. This enhancement does not only strengthen the guidance of the local search but also speeds up the search process which is important as only a limited amount of time is available.

4.4.2 Topological Ordering Neighborhood

This neighborhood structure is based on the move proposed by Galinier et al. [GLB13] as part of their simulated annealing algorithm for the DFVS problem. Let F be the current directed feedback vertex set and S a topological ordering of $G[\bar{F}]$, then the move operator consists of the following steps:

1. Select a vertex $v \in F$.
2. Insert v at position i into S, such that $S[i] = v$.
3. Remove from S all inneighbors u of v with $u = S[j]$ and $j \geq i$.
4. Remove from S all outneighbors u of v with $u = S[j]$ and $j < i$.

The intuition behind this move is to only generate valid solutions by simultaneously inserting a vertex into the topological ordering and removing from the topological ordering all direct neighbors that are now involved in backward arcs. As removing such vertices from the topological ordering is equivalent to adding them to the DFVS, this move might result in a worse solution than the one it started with. However, we only accept improving solutions, so we skip vertices whose insertion into the topological ordering would lead to the same or an increased size of the DFVS.

In general, there are $|S| + 1$ positions where the vertex can be inserted, which results in a neighborhood of size $\mathcal{O}(|V|^2)$. To reduce this size, Galinier et al. [GLB13] propose a candidate list strategy to determine a subset of moves of high quality. This strategy limits the number of possible insertion positions of a vertex $v \in F$ to the following two: position $i^+(v)$, which is directly before its first outneighbor in S, and position $i^-(v)$, which is directly after its last inneighbor in S. Let $N^-(v)$ denote the set of incoming neighbors of a vertex v, $N^+(v)$ the set of outgoing neighbors, $I^-(v) = \{i : S[i] \in N^-(v)\}$ the positions of all inneighbors in the topological ordering, and $I^+(v) = \{i : S[i] \in N^+(v)\}$ the positions of all outneighbors in the topological ordering. Then, the following holds for the two positions:

$$i^-(v) = \begin{cases} \max(I^-(v)) + 1 & \text{if } I^-(v) \neq \emptyset; \\ 1 & \text{otherwise;} \end{cases} \tag{4.28}$$

$$i^+(v) = \begin{cases} \min(I^+(v)) & \text{if } I^+(v) \neq \emptyset; \\ |S| + 1 & \text{otherwise.} \end{cases} \tag{4.29}$$

These positions lead to moves of high quality because they will create only a limited number of backward arcs in the topological ordering. Using position $i^-(v)$ for inserting vertex v avoids any conflicts with its incoming neighbors. Furthermore, position $i^-(v)$ is defined as the position directly after the last inneighbor, so at the same time it minimizes the number of backward arcs to outneighbors. When switching in- and outneighbors, these properties also hold for position $i^+(v)$. Restricting the move operation to these two positions for each vertex decreases the size of the neighborhood to $\mathcal{O}(|V|)$. We reduce the search space even further by only selecting the better position of the two, which is the position that introduces fewer backward arcs. However, if this position still introduces any backward arcs, then we reject the current vertex and its corresponding candidate solution as we only accept improving solutions.

To guide the search and improve the performance, we use the same approaches as for the one-flip neighborhood. On the one hand, we iterate over the vertices in the current DFVS in increasing order of their heuristic value, which is computed with the function by Cai et al. [CHJ06] (cf. Equation 2.1). On the other hand, we again store the position of the chosen vertex in the sorted list of solution vertices and continue the next LS iteration at this position while skipping all already rejected vertices.

4.4.3 Enhanced Topological Ordering-Based Local Search

We also propose an enhancement of the local search based on the topological ordering neighborhood. In case that no improvement is achieved when reaching a local optimum, the LS algorithm immediately terminates. However, if the previously best solution is improved, we use a DFS procedure to determine another topological ordering of the DAG corresponding to this new best solution. It is not guaranteed that a different topological ordering is found but it is possible as a DAG can have multiple such orderings. If no new topological ordering is found, the LS algorithm terminates and otherwise, it restarts with the new ordering. This procedure is repeated until no improvements or no new topological orderings are found anymore.

4.4.4 Comparison

The biggest disadvantage of the one-flip neighborhood are the repeated cyclicity checks during the moves, which causes similar runtime problems as for the degree-based construction heuristic. For the cyclicity checks, we employ a breadth first search (BFS) procedure to determine the existence of paths in the current DAG from any outneighbor of the selected vertex to any inneighbor. If such a path exists, then the removal of the vertex from the DFVS would lead to a cycle and is therefore infeasible. Despite this rather efficient check, the LS based on the one-flip neighborhood rarely reaches a local optimum within the given time frame, which we observed during a limited number of preliminary experiments.

In contrast, some more preliminary tests showed that using the topological ordering neighborhood almost always leads to a substantial speedup of the local search as no cyclicity checks have to be done. However, this neighborhood structure might struggle with the same drawback as the CH based on topological ordering: the dependence on a specific topological ordering. This might be the reason why the resulting solutions are on average worse than the ones produced with the one-flip neighborhood. Hence, we introduced the enhanced topological ordering-based LS in order to diminish this shortcoming. The enhancement does not subvert the advantage of the topological ordering neighborhood as this variant of the LS algorithm is still faster than the LS with the one-flip neighborhood. On the other hand, it also does not improve the solutions enough to reach the solution quality provided by the one-flip neighborhood. In fact, the additional LS runs with new topological orderings rarely lead to further improvements of the solutions. As we prioritize the solution quality of the initial solution over the runtime, we employ the LS based on the one-flip neighborhood in our solving procedure.

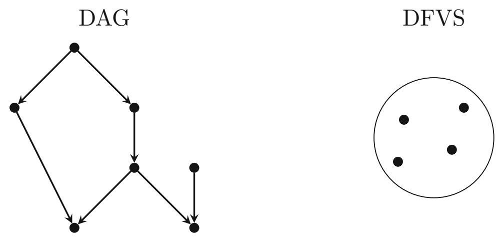
4.5 Large Neighborhood Search

This section discusses large neighborhood search as the metaheuristic framework of our solving procedure for the DFVS problem. First, two different neighborhood structures are proposed, which are based on either moving vertices from the current solution to the corresponding DAG or moving vertices from the current DAG to the corresponding solution. In the next sections, we discuss different approaches for choosing the degree of destruction as well as various options for element selection in the destroy method. The employed acceptance criterion and termination criteria are presented last.

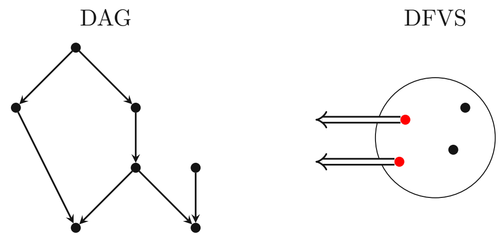
4.5.1 Enlarge-DAG Neighborhood

The underlying idea of this neighborhood is to take vertices out of the current solution, add them and their original incident edges back to the corresponding DAG, and then solve the DFVS subproblem posed by the resulting graph. This move does not only increase the size of the DAG, hence the name *enlarge-DAG neighborhood*, but it also potentially introduces cycles, which have to be broken again. Thus, the enlarged DAG represents a new DFVS problem instance that is smaller than the original one and therefore should be easier to solve.

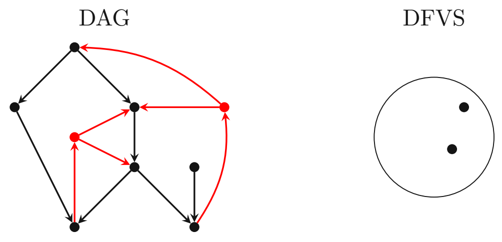
Before we describe the destroy and repair method in more detail, an example of such a move is illustrated in Figure 4.7. The initial state as determined by the current solution is given in Figure 4.7a. We start with a valid DFVS, so the corresponding DAG is actually acyclic. Figure 4.7b shows the selection two vertices marked in red from the DFVS. Then, these vertices are added to the DAG together with all their incident edges to their original neighbors that are also contained in the DAG. The resulting graph as well as the remaining DFVS are depicted in Figure 4.7c. This graph is not acyclic anymore because of the newly introduced cycles such as the one pointed out in Figure 4.7d. In the repair operator, we solve the DFVS problem on this graph which gives us the red vertex in Figure 4.7e as a minimum cardinality DFVS of this subproblem. This vertex is then removed from the enlarged DAG and added to the global DFVS. Lastly, Figure 4.7f illustrates the final state with the new and smaller DFVS and its corresponding DAG.



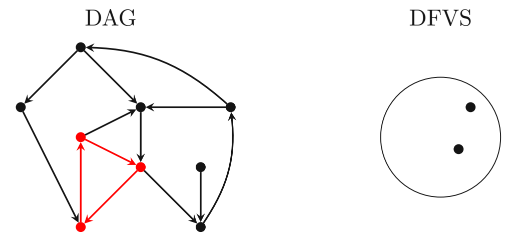
(a) Initial state with current solution.



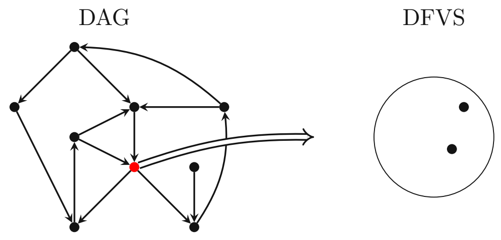
(b) Element selection.



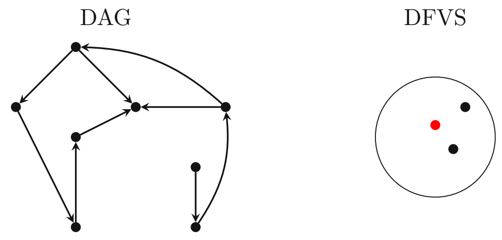
(c) Enlarged DAG.



(d) Enlarged DAG with exemplary cycle marked in red.



(e) Solving the subproblem.



(f) Final state with new solution.

Figure 4.7: An example of a move in the enlarge-DAG neighborhood. The left-hand side of each subfigure illustrates the respective state of the DAG over different stages of the move and the corresponding DFVS is shown on the right-hand side.

Destroy and Repair Operators

Let F be a valid DFVS and the vertex set of the corresponding DAG be denoted by $D = V \setminus F$. In the destroy method, a k-element subset $A \subseteq F$ of the DFVS is selected. The destroyed solution is then obtained by removing these k elements from the DFVS and adding them to the DAG: $D' := D \cup A$, $F' := F \setminus A$. This can introduce cycles in the subgraph $G' = G[\,D'\,]$, so the repair operator has to remove a set F'' of vertices from D' to obtain a DAG once again. By definition, this is exactly the DFVS problem but for the smaller graph G'. Reducing the instance size like this allows for using different solving approaches here than for the larger original instance. Besides, we also apply the graph reduction rules as described in Chapter 4.1 to further reduce the size of G'. In case that during the graph reduction, vertices are already identified to be contained in the DFVS of G', these vertices are added to F''. Then, we employ a MILP solver to find a valid DFVS for the remaining graph, which is described in Chapter 4.6 in more detail.

After solving the subproblem, the vertex set $F = F' \cup F''$ provides a new DFVS for the original graph.

4.5.2 Enlarge-DFVS Neighborhood

This neighborhood structure is based on moving vertices from the current DAG to the corresponding solution, so it basically works the opposite way of the enlarge-DAG neighborhood. Thus, this move enlarges the DFVS which might seem counterintuitive at first as our goal is to decrease the size of the direct feedback vertex set in order to improve the solution. However, we use the subgraph induced by the enlarged DFVS as a base and construct a new graph from it in a way that enables us to solve the DFVS problem on this new smaller graph and find a DFVS for it that is also a DFVS for the original graph. Below, we first describe the how the destroy and repair operators of this move work, before illustrating the neighborhood structure with an example.

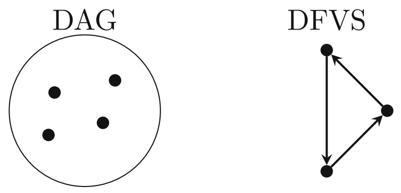
Destroy and Repair Operators

Let F be a valid DFVS and the vertex set of the corresponding DAG be denoted by $D = V \setminus F$. The destroy operator chooses a k-element subset $A \subseteq D$ of the DAG and destructs the solution by adding these k elements to the DFVS: $F' := F \cup A$, $D' := D \setminus A$. The resulting solution F' is still a valid DFVS but of bigger cardinality than before. Thus, the aim of the repair method is to add vertices from F' to D' to obtain a larger DAG and reduce the size of the DFVS. In other words, our goal is to find the maximum set of vertices $B \subseteq F'$ that can be added to D' without introducing any cycles in the DAG. Like in the enlarge-DAG neighborhood, such a set can be found by solving a DFVS problem, this time on the graph $G' = (F', E')$ with

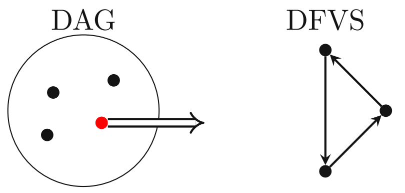
$$E' = (E \cap F'^2) \cup \{(v, w) : v, w \in F', \text{an inneighbor of } w \text{ is reachable} \\ \text{from an outneighbor of } v \text{ in } G[D']\}. \quad \text{(Edge Condition)}$$

The new graph G' is based on the vertices in F' and their original incident edges but the set of edges is extended in a way such that G' also reflects the reachability of its vertices that they would have in the original graph G. If two vertices $v, w \in F'$ both have neighbors in the DAG $G[D']$ and there is a directed path in $G[D']$ from any outneighbor of v to any inneighbor of w, then w would be reachable from v in the original graph. Hence, an edge (v, w) is inserted into G' according to the (Edge Condition). Moreover, this leads to every simple cycle C in the original graph having a corresponding cycle $C' \subseteq C$ in G'. This is due to the edge condition and the fact that for every cycle C of G at least one involved vertex is contained in G' as F' is still a valid DFVS of G. Overall, the definition of the edge condition then ensures that a vertex set $F'' \subseteq F'$ is a DFVS of G' if and only if F'' is also a DFVS of G. Again, a MILP solver can be used to find this DFVS and solve the smaller subproblem.

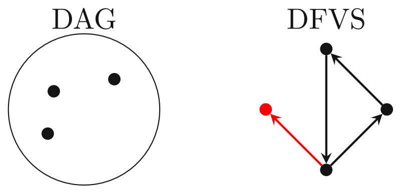
Figure 4.8 illustrates the workflow of such a move in the enlarge-DFVS neighborhood, starting with the initial solution and its corresponding DAG. The element selection from



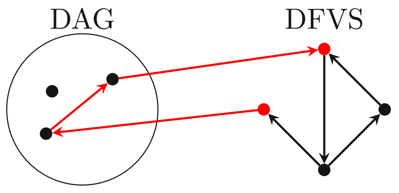
(a) Initial state with current solution.



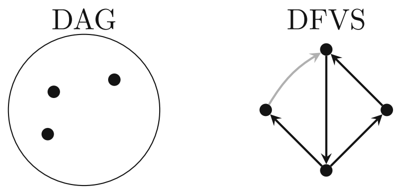
(b) Element selection.



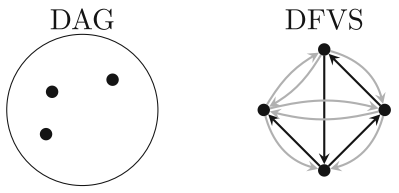
(c) Enlarged DFVS.



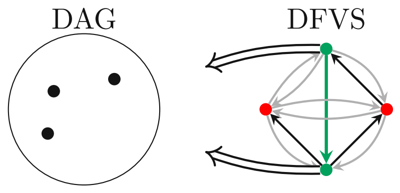
(d) Reachability check.



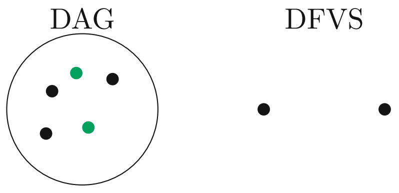
(e) Adding edges according to the (Edge Condition).



(f) Fully enlarged DFVS.



(g) Solving the subproblem.



(h) Final state with new solution.

Figure 4.8: An example of a move in the enlarge-DFVS neighborhood. The right-hand side of each subfigure illustrates the respective state of the DFVS over different stages of the move and the corresponding DAG is shown on the left-hand side.

the DAG done in the destroy operator is shown in Figure 4.8b where the red vertex is chosen to be added to the DFVS. The vertex is removed from the DAG and inserted together with its incident edges into the graph induced by the DFVS, which is depicted in Figure 4.8c. The next step is to satisfy the edge condition, for which all vertices in the DFVS have to be checked for reachability. Figure 4.8d illustrates such a reachability check for the two red vertices. As a path between a corresponding outneighbor and inneighbor in the DAG is found, the vertices are proven to be connected in the original graph. Therefore, an according directed edge between the vertices is added to G' as shown in Figure 4.8e. This process is repeated for every pair of vertices which results in the graph given in Figure 4.8f. Then, the DFVS problem is solved on G'. The red vertices in Figure 4.8g are determined to be a DFVS of G', whereas the green vertices and edges denote the corresponding DAG. Thus, the green vertices constitute the set B that can be added to D' without introducing any cycles. This is ensured by the (Edge Condition) which also asserts that the DFVS of G' represents a valid solution for G. The resulting DFVS and DAG can be seen in Figure 4.8h, which also illustrates that the DFVS is now smaller than before the move.

Enlarge-Partial-DFVS Neighborhood

The enlarge-DFVS neighborhood structure is quite powerful, but for large problem instances constructing the graph G' is often too expensive for the neighborhood to be used in practice. Thus, we propose a variant called the *enlarge-partial-DFVS neighborhood*. This neighborhood structure is based on the same concepts as the original one but for the creation of G', we only consider a subset $F_1 \subseteq F$ of the DFVS, hence the name *partial-DFVS*. Let $F = F_1 \cup F_2$ be the initial DFVS and $A \subseteq D$ the selected k-element subset. Then, we have $F' = F_1 \cup A$ and the definition of G' and the (Edge Condition) remain the same. As before, the DFVS problem is solved for G', for example with a MILP solver, but the resulting DFVS F'' cannot directly be used as the DFVS of G. We also have to consider the other part F_2 of the initial DFVS, which was not used in the subproblem. Therefore, the new solution is given by the set $F'' \cup F_2$.

Using only a subset of the DFVS when constructing G' reduces the effort imposed by the reachability checks that have to be done to satisfy the (Edge Condition). However, the performance difference between the original neighborhood and this variant in terms of runtime as well as solution quality heavily depends on the size of the partial DFVS. After a limited number of preliminary experiments, we decided to use a size of 3000 also having regard to the condition of keeping the resulting subproblem small enough to be solved by a MILP solver. When applying this variant of the neighborhood structure, we often achieve a substantial speedup of the LNS compared to using the original enlarge-DFVS neighborhood.

4.5.3 Degree of Destruction

In the definition of our neighborhood structures, the degree of destruction is denoted by the parameter k that specifies the size of the subset of elements chosen in the destroy operator. Selecting a suitable degree of destruction is crucial for an LNS procedure because this parameter determines how much of the current solution is destroyed and thus also the size of the neighborhood and the search space. Besides, it is of special importance for us as it influences the size of the DFVS subproblem that is handed to a MILP solver in the repair method and solving larger MILP models can take exponentially longer. We considered using a certain percentage of the DFVS size but as larger instances tend to have DFVSs of bigger cardinality, this approach would make the degree of destruction at least slightly dependent on the instance size. This might then lead to large subproblems in the repair operator, which is exactly what we try to avoid. Therefore, we propose three other techniques for choosing the degree of destruction:

1. selecting k randomly in each iteration by choosing it from a specific range,
2. using a fixed value for k,
3. dynamically choosing k based on certain (graph) properties.

The first approach is similar to one suggested by Ropke and Pisinger [RP06] but in contrast to them, we do not determine the value range based on the instance size but simply select a fixed value that is used as an upper bound for k for all instances. Again, this is done to restrict the size of the subproblem given to the MILP solver. Randomly choosing k in each iteration provides the possibility of intensification as well as diversification of the search, depending on whether a smaller or larger value is picked.

For the second technique, we also choose a fixed value but this time, no stochastic element is involved and the value of k never changes. Thus, the same degree of destruction is applied in all LNS iterations and for all DFVS problem instances and we call this strategy fixed_degree(x) where x stands for the selected value. Out of all approaches, this one provides the most control over the size of the subproblem in the repair method. However, the solution quality is often strongly related to the degree of destruction, so choosing a value that is unsuitable will usually lead to finding few to no improvements during the LNS. Unlike in the first approach, there is also no element of randomness that could compensate for a poorly chosen value of k. This is the reason why we do not use this technique in our final solving procedure but testing it with different values for k, which is discussed in depth in Chapter 5.6.1, provided valuable information for the third approach.

The third technique is a combination of the other two with additional enhancements that link the degree of destruction to graph properties or the MILP formulation that is applied in the repair method or both. The employed MILP formulation is either preselected for all instances or dynamically chosen for each DFVS problem, which is described in more detail in Chapter 4.6.1. Regarding the enhancements, we noticed during testing of the fixed_degree approach that the performance of the various degrees of destruction seemed to be influenced not only by certain characteristics of the graphs such as the number of 2-cycles and the number of vertices but also by the MILP formulation used to encode the subproblems. Therefore, we analyzed the results for possible correlations and strategies for choosing the degree of destruction (cf. Chapter 5.6.2). Based on the analysis, we designed multiple strategies, some of which use stochastic elements like in the first approach, and all of them involve fixed values for k:

#2-cycles_sections. For this strategy, we predefine sections based on mutual exclusive ranges for the number of 2-cycles in a graph and then use the insights from the tests with the fixed_degree approach to pick the on average best performing degree of destruction for each section. If multiple degrees achieve the same average, we choose the lowest one. We also differentiate between the employed MILP formulations, such that for each section two values for k are selected: one for the MTZ-based formulation and one for the CEC-based formulation. These ranges and values are fixed and used for all DFVS problem instances. At the start of the LNS procedure, we first determine the used MILP formulation and the number of 2-cycles that are present in the graph at hand. Then, we look up the corresponding section and its assigned degree of destruction, which will be applied in all LNS iterations.

MILP_triple. In this strategy, we pick for each MILP formulation three values for k before the LNS procedure starts. The selected values are again derived from the tests with the fixed_degree approach and are named *best-mean*, *mode*, and *most-best*. Best-mean denotes the degree of destruction that achieves the best mean in terms of solution quality. Mode stands for the lowest tested degree that results most often in a solution of smallest known size, whereas most-best is the degree that found the most solutions of smallest size, hence the best solutions. It can be observed that these values are independent from any properties of the input graph. In each iteration, one of the three values is randomly selected for the respective MILP formulation.

#2-cycles_sections_triple. This strategy is a combination of the concepts of the #2-cycles_sections and MILP_triple techniques. We reuse the sections from #2-cycles_sections and preselect for each section and each MILP formulation three degrees of destruction called *best-2cycle-mean*, *most-2cycle-best*, and *best-mean*. The best-2cycle-mean corresponds to the values selected in the #2-cycles_sections strategy and the best-mean is the same as in the MILP_triple strategy. Most-2cycle-best is similar to most-best from MILP_triple as it denotes the degree of destruction that produces the most best solutions but with regard to each 2-cycle section and with a threshold of 99% for the solution quality. Thus, we have a mixture of dependent and independent values regarding graph properties. Before starting the LNS procedure, we compute the number of 2-cycles in the graph and determine the corresponding triple of k-values for each MILP formulation. During the LNS, we randomly select one of the three values in each iteration.

corr_log-#2-cycles. This strategy is based on the linear correlation between the base-10 logarithmic value of the number of 2-cycles in a given graph and the base-10 logarithmic value of the lowest degree of destruction that achieved the best solution for this graph, which we found once again based on the tests with the fixed_degree approach. Again, we differentiate between the various MILP formulations and for each formulation, we use the corresponding regression line to define a function that computes the value of k based on the number of 2-cycles in the input graph. Thus, the degree of destruction is determined with this function before the LNS and then used in every iteration.

corr_log-#vertices. This strategy works exactly like the corr_log-#2-cycles technique and the only difference is that the base-10 logarithmic value of the number of vertices of the graph is used instead of the number of 2-cycles. We again determine a function for each MILP formulation based on the corresponding regression line and apply these functions to find the degree of destruction for each instance. This value is then fixed for all LNS iterations.

4.5.4 Element Selection

Aside from the degree of destruction, it is also important which part of the solution is destroyed as this also guides the diversification and intensification of the search as well as influences the likelihood of cycling. This part of the solution is denoted by the subset A of vertices in the definition of our neighborhood structures and we propose three general ideas for selecting the elements of A:

1. random selection,
2. heuristic selection,
3. tournament selection.

Random Selection. As the name implies, random selection means to randomly choose the elements. The selection is free from any external influences and the vertices in the solution are uniformly distributed. This approach allows for more diversification in the search process but lacks any guidance towards promising solutions.

Heuristic Selection. This strategy is based on the heuristic values of the vertices as computed with Function 2.1 by Cai et al. [CHJ06]. The vertices are then sorted according to their heuristic value and the first k elements in the sorted list are chosen. The order in which the vertices are sorted depends on the employed neighborhood structure. For the enlarge-DAG neighborhood, the vertices are sorted in increasing order as the vertices in the current DFVS with the smallest heuristic value are more likely to be part of the DAG. The opposite is done for the enlarge-DFVS neighborhood where the vertices in the DAG with the highest heuristic value are selected because they are more likely to belong to the DFVS. This approach is intended for an intensification of the search and works quite well at the start of the LNS but its effectiveness decreases after the first few iterations. The reason for this is that the selection based on the heuristic value is deterministic because the heuristic value of the vertices never changes. This strategy will always pick the same vertices and after the first unsuccessful iteration no improvement is possible anymore unless the degree of destruction changes.

Tournament Selection. This is a popular technique for selecting individuals from a population in genetic algorithms and the underlying idea is to first randomly select a given number of individuals and then to choose the best one(s) from this subset of the population. This procedure, called a *tournament*, is repeated until the desired number of chosen individuals has been reached. Our approach tries to exploit the strengths of the other two selection techniques while diminishing their weaknesses as it combines the element of randomness from the random selection with the guidance of the heuristic selection. We start with randomly choosing individuals, here vertices, from the population, which is either the current DFVS or the DAG, the former for the enlarge-DAG and the latter for the enlarge-DFVS neighborhood. Then, we use the heuristic values of the vertices to determine the best one. Like in the heuristic selection, we always pick the vertex with the smallest heuristic value for the enlarge-DAG neighborhood and the vertex with the highest heuristic value for the enlarge-DFVS neighborhood. This results in a deterministic tournament selection as we select the best vertex in each tournament. One important parameter is the *tournament size*, which is the number of vertices that are randomly chosen for each tournament, as it controls the *selection pressure*. A large tournament size leads to strong selection pressure whereas a small tournament size results in weak selection pressure. For example when considering the enlarge-DAG neighborhood, a vertex with a high heuristic value is more unlikely to be chosen in a tournament if the tournament size is large because the probability that a better vertex is also present in the tournament is higher than in case of a small tournament size. If the tournament size is set to one, then the tournament selection turns into random selection.

After a limited number of preliminary tests, we decided on using tournament selection in our solving procedures. Regarding the tournament size, we started out with a fixed percentage of the solution size but this resulted in a too strong selection pressure. Thus, we switched to using fixed small values and based on some more preliminary experiments, we then chose a tournament size of three.

Element Selection in the Repair Operator of the Enlarge-Partial-DFVS Neighborhood

In the repair operator of the enlarge-partial-DFVS neighborhood of the LNS, we only consider a subset $F_1 \subseteq F$ of the DFVS and thus, the vertices for this partial DFVS have to be selected somehow. In general, the same three techniques as described above can be applied here as well and we again decided on tournament selection. The tournament size is also set to three and the vertices are still evaluated based on their heuristic value. Since we select a subset of the DFVS and the goal is to reduce its size, which is similar to what is done in the destroy method of the enlarge-DAG neighborhood structure, we reuse the approach of selecting the vertex with the smallest heuristic value in each tournament.

4.5.5 Acceptance Criterion and Termination Criteria

During the large neighborhood search, the acceptance criterion defines when a candidate solution from the neighborhood is accepted as the new incumbent solution. We use the straightforward approach of only accepting better solutions, which are solutions with a strictly lower objective value than the currently best solution. Originally, we intended to employ the Metropolis criterion from simulated annealing similar to Galinier et al. [GLB13] and Tang et al. [TFZ17]. This criterion differs from ours in cases where the candidate solution is worse than the incumbent solution as it enables accepting such solutions with a certain probability. For us however, these cases are irrelevant and will never occur due to the design of our hybridized LNS procedure, which is described in more detail in Chapter 4.6. In short, we combine the LNS with a MILP solver in a way that only generates candidate solutions of equal or higher quality compared to the incumbent solution. Hence, the Metropolis criterion would never be applied and our simple acceptance criterion is sufficient.

The termination criterion determines when the LNS has to stop unless a local optimum is found before the condition is satisfied. Which criterion is suitable heavily depends on the goal that should be achieved. For example, when testing the different neighborhood structures only in terms of their capability to improve the solution, we only use a limit on the number of iterations. This allows for showcasing the full potential of the neighborhoods and comparing them based on the solution quality that they can provide. However, such an approach might favor complex neighborhoods that enable big improvements but are also very time-consuming, such as the enlarge-DFVS neighborhood. For our final solving procedure, we aim for an overall good performance which includes a high solution quality as well as low runtimes. Thus, we use a combination of two termination criteria: a runtime limit and a limit on the number of unsuccessful iterations. A runtime limit often results in expensive neighborhoods performing worse than more efficient ones, even if the latter are not as powerful. Regarding the second condition, an iteration is considered unsuccessful if the incumbent solution could not be improved in this iteration. The idea behind limiting the number of such iterations is that it is rather unlikely to still find a better solution in the neighborhood after already searching it many times. This criterion is installed to further the efficiency of the solving procedure as it enables termination before the time limit is reached.

4.6 Hybrid Large Neighborhood Search

Our proposed LNS procedures for the DFVS problem can be used together with heuristics to solve the subproblems in the repair operators, for example the heuristics introduced in Chapter 4.3 can be applied. However, we hope to obtain larger improvements and better results by enhancing the LNS metaheuristic with another technique, which leads to a so-called hybrid metaheuristic. Our hybridization is achieved by first encoding the subproblem in the repair operator of the respective neighborhood structure with one of our proposed MILP formulations and then solving it with a MILP solver. Both formulations can generally be used in both neighborhoods but for the enlarge-(partial-)DFVS neighborhood, we use only the MTZ-based formulation. The reason is that the graph G' of the subproblem can be expected to be quite dense because of the (Edge Condition), which might substantially increase the effort needed to build and solve the CEC-based model.

As mentioned in Chapter 3.2, a repair method can be either optimal or heuristic and our approach falls into the category of optimal repair operators. In general, the MILP solver provides an optimal solution for the subproblem which results in generating only equally good or better solutions than the incumbent solution. However as one of our termination criteria is a time limit, we also impose a time limit on the optimization procedure of the MILP solver. This is not a problem per se as state-of-the-art solvers are fast enough to produce at least feasible solutions for our DFVS subproblems but it could lead to suboptimal solutions that might be even worse than the incumbent. This also means that in practice, our repair operator is not optimal anymore. Even though we cannot guarantee an optimal solution for the subproblem, there is a way to ensure that the repaired solution is not worse than the current solution: providing the MILP solver with an initial solution. This is also called a warm start where the MILP solver starts the optimization with a feasible initial solution and the final solution cannot be worse than at the start. If we use the current solution as this initial solution, we will always achieve equal or improving solutions.

4.6.1 Dynamic Selection of the MILP Formulation

This section describes a variant of the hybrid LNS using the enlarge-DAG neighborhood, on which we focused during testing. Instead of preselecting the MILP formulation to be used for all instances, we want to dynamically select the formulation for each instance. This approach is similar to the #2-cycles_sections strategy for the dynamic selection of the degree of destruction as the selection criteria are also based on certain graph properties and sectioning is used as well. When testing the LNS variants with the dynamic selection of the degree of destruction, as described in Chapter 5.6.2, we noticed differences in the performance of the MILP formulations for various instances which seem to be linked to the product of the number of 2-cycles and the density of the graphs. Thus, we divided these values into multiple sections and determined for each section the best performing MILP formulation, which is explained in detail in Chapter 5.6.3. This allows us to dynamically select the most suitable MILP formulation for every DFVS problem instance. Besides, we also provide such a classification for the use of the MILP models as standalone solution approaches as the sections and assigned formulations for this case differ from the ones for the hybrid metaheuristic.

CHAPTER 5

Computational Study

In this chapter, we evaluate our proposed solving procedures for the DFVS problem on unweighted graphs. First, we specify the setup for our computational study and the basics of our implementation and we also describe the problem instances we use in the study. Next, we evaluate the performance of our graph reduction procedure, before discussing the application of the different MILP formulations as standalone solution approaches. In the following sections, we evaluate the performance of our hybrid metaheuristics, starting with a comparison of the various neighborhood structures. We then present and analyze the performance of different configurations of the hybrid metaheuristic with focus on the strategies for selecting the degree of destruction and the employed MILP formulation. Lastly, we compare our best performing procedures to two state-of-the-art solving approaches for the DFVS problem from the literature.

5.1 Computing Environment and Implementation

We conducted this computational study on the computational grid, or compute cluster, provided by the Algorithms and Complexity Group of TU Wien. All experiments were performed on a single core of an Intel Xeon E5540 processor with a memory limit of 23 GB. Our solving procedures were all written in Julia 1.7.1[1] [BEKS17] and we used the JuMP package [DHL17] in version 0.22.2 for implementing our MILP models and as interface to the MILP solvers. For the MILP solvers, we chose Gurobi 9.5.1[2] [Gur22] and SCIP 7.0[3] [GAB+20]. Both are state-of-the-art MILP solvers with the difference that Gurobi is a leading commercial product whereas SCIP is open source and one of the fastest non-commercial solvers.

[1]`https://julialang.org`
[2]`https://www.gurobi.com`
[3]`https://www.scipopt.org`

Table 5.1: Number of instances and graph sizes in the data sets used for the computational study.

Data Set	Size	Number of Vertices		Number of Edges	
		$n_{\min}$	$n_{\max}$	$m_{\min}$	$m_{\max}$
pace-public	100	843	875713	2103	5105039
pace-private	100	1024	2394385	3473	5021410
fsp-data	40	50	1000	100	30000
fsp-data_50	10	50	50	100	900
fsp-data_100	10	100	100	200	1400
fsp-data_500	10	500	500	1000	7000
fsp-data_1000	10	1000	1000	3000	30000

5.2 Benchmark Instances

For testing our solving approaches, we use three existing data sets of benchmark instances. Two of the data sets are taken from the heuristic track of the PACE 2022[4,5] challenge and one set[6] comes from Pardalos et al. [PQR98], which was created for testing their greedy randomized adaptive search procedure (GRASP) for the DFVS problem. We will refer to the two sets from PACE 2022 as *pace-public* and *pace-private* and to the third as *fsp-data*. These three sets provide a great variety of graphs with different topologies but they all consist only of simple, directed graphs without any self-loops or multi-edges. Table 5.1 shows the ranges of the number of vertices n and edges m of the graphs in the different sets together with the respective number of instances.

The sets pace-public and pace-private consist of 100 instances each and the sizes of the contained graphs range from small to rather large. These instances were mostly generated using KaGen[7] [FLS+18, SS16], a set of generators for network models, and five graph models:

- Erdős-Rényi Graphs,
- Random Geometric Graphs,
- Random Hyperbolic Graphs,
- Random Delaunay Graphs,
- Barabási-Albert Graphs.

[4]https://pacechallenge.org/2022/
[5]https://pacechallenge.org/2022/tracks/#heuristic-track
[6]Feedback set problem at http://mauricio.resende.info/data/index.html
[7]https://github.com/sebalamm/KaGen

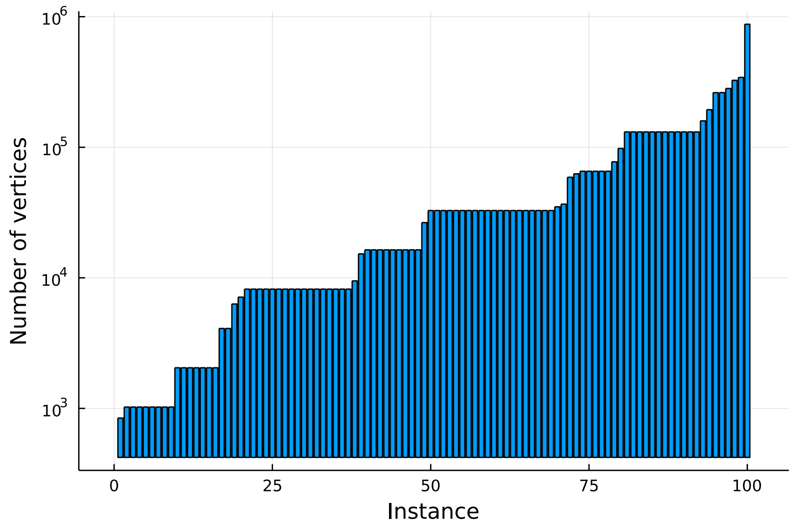


Figure 5.1: The number of vertices of each instance in the pace-public data set.

In general, a random amount of edges was removed from bidirected graphs and for the last type of graphs, additional backward edges were randomly inserted into the topological ordering. Besides, the sets also contain real-world instances from the SNAP repository and the overall goal was to provide hard DFVS instances where random walks will fail. An illustration of the number of vertices and edges of each instance of the pace-public set is shown in Figure 5.1 and Figure 5.2 respectively. As the existence of 2-cycles in a graph is also an important feature for our solving procedures, we depict the ratio of edges involved in 2-cycles in Figure 5.3. This shows that in more than half the instances at least 50% of their edges are part of 2-cycles.

The fsp-data set consists of 40 smaller graphs which are divided into four subsets based on their number of vertices as given in Table 5.1: fsp-data_50, fsp-data_100, fsp-data_500, and fsp-data_1000. Each subset contains 10 instances, which were randomly generated with varying densities to generate sparse as well as dense graphs.

5.3 Graph Reduction

In this section, we evaluate the performance of our graph reduction procedure as described in Chapter 4.1 with regard to various metrics. To do this, we assess the state of the input graph and the current DFVS at two points in the procedure. The first checkpoint is after the first exhaustive application of the five reduction rules and the second is after the graph partitioning and merging and repeated application of the reduction rules to the SCCs. However, the latter only applies to instances where the graph can be partitioned into multiple SCCs. If a graph consists of only one strongly connected component after the first checkpoint, the reduction procedure terminates. The results of the first assessment are given in Table 5.2 and Table 5.3.

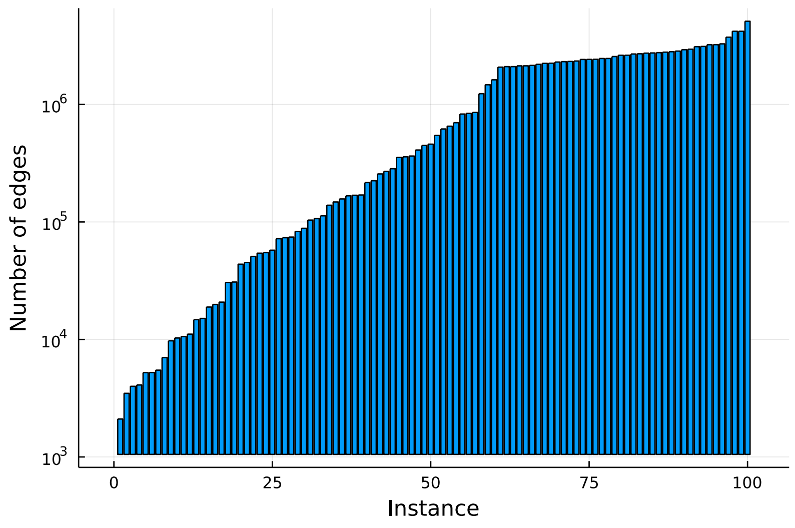


Figure 5.2: The number of edges of each instance in the pace-public data set.

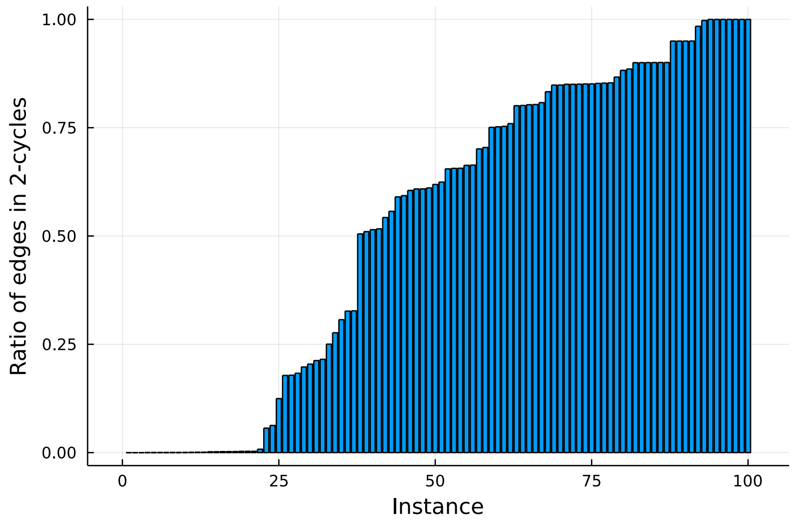


Figure 5.3: Ratio of edges involved in 2-cycles for each instance in the pace-public data set.

Table 5.2 shows for how many instances in each data set we were able to achieve certain properties. As specified there, our graph reduction procedure can reduce the number of vertices of more than 75% of the instances in pace-public and pace-private. For fsp-data, at least 50% of the graphs are decreased in size, which is also true for each of the four subsets. Furthermore, we observe for all instances in all data sets that whenever the vertex number can be reduced, the same is true for the edge number. One instance in pace-public is even fully reduced, which means that the reduction rules are able to

Table 5.2: Number of instances in each data set with certain properties after the graph reduction procedure. The number of vertices is denoted by n, and n_r is the reduced number after the first exhaustive application of the five reduction rules. If $n_r = 0$, then the graph could be fully reduced and is empty. The number of strongly connected components of the reduced graph is given by $|SCC|$.

Data Set	Size	Number of Instances with:		
		$n_r < n$	$n_r = 0$	$\|SCC\| > 1$
pace-public	100	77	1	22
pace-private	100	81	0	20
fsp-data	40	23	0	0
fsp-data_50	10	5	0	0
fsp-data_100	10	6	0	0
fsp-data_500	10	7	0	0
fsp-data_1000	10	5	0	0

remove all vertices and edges from the graph. In contrast, there are also instances in each data set which cannot be reduced at all in terms of vertices as well as edges. Concerning the partitioning of the graphs into multiple SCCs after the first reduction step, this is achieved for 22 and 20 instances in pace-public and pace-private respectively, but it is not possible for any instances in fsp-data. For example, the 22 instances in pace-public could be split into a total of 21709 SCCs, which are then reduced to 2923 by merging smaller ones. Of these 2923, 854 subgraphs from ten different instances, which are about 29.22%, are then further reduced by the second application of the reduction rules. In the end, 2688 subproblems have at most 100 vertices each and can be directly solved with a MILP solver. Thus, only 235 of the original 21709 SCCs are handled by our following hybrid LNS, which is slightly over 1% and reduces the effort of our solving procedure substantially.

Table 5.3 lists some statistical values regarding the achieved reductions as well as the runtimes as measured at the first checkpoint. The reductions are given as a percentage of the original value in the input graph, so for example, the number of vertices of instances in pace-public can be decreased by about 21.08% on average whereas the number of edges has an average reduction of about 17.26%. The mean reductions for instances in pace-private are slightly lower with approximately 17.75% for the vertices and 14.47% for the edges. The average vertex reduction in the whole fsp-data set is about 13.87% and we observe for most subsets that the reduction decreases the more vertices a graph contains. The only exception is the subset of graphs with 50 vertices, for which lower vertex and edge reductions are achieved on average than for graphs with 100 vertices. The data sets pace-public and pace-private contain at least one instance each that is fully or nearly fully reduced. This cannot be achieved for any instance in fsp-data where the highest reduction is 78% for vertices and 64% for edges. Besides, the values of the standard deviation σ indicate a quite high dispersion of the achieved reductions for all

Table 5.3: The average, the standard deviation σ, and the maximum reduction of vertices and edges as percentage of the original number as well as the average, standard deviation σ, and maximum runtime in seconds for each data set. The reductions were achieved by the exhaustive application of our five reduction rules to the input graphs.

Data Set	Reduction of Vertices [%]			Reduction of Edges [%]			Runtime [s]		
	Average	σ	Max	Average	σ	Max	Average	σ	Max
pace-public	21.08	31.01	100.00	17.26	27.14	100.00	0.77	2.99	27.62
pace-private	17.75	28.09	99.99	14.47	25.66	99.99	0.48	0.87	7.10
fsp-data	13.87	21.69	78.00	8.63	16.59	64.00	0.08	0.07	0.16
fsp-data_50	15.00	25.25	78.00	9.98	20.26	64.00	0.07	0.07	0.16
fsp-data_100	15.10	24.05	74.00	10.25	19.07	60.50	0.09	0.07	0.16
fsp-data_500	14.54	24.75	75.60	8.88	18.65	59.30	0.09	0.06	0.15
fsp-data_1000	10.83	13.98	38.70	5.41	7.34	21.03	0.08	0.08	0.15

data sets, which is probably due to the diverse instances. However, the graph reduction process barely impacts the overall performance of our solving procedures as the average runtime is less than 0.8 seconds for pace-public, less than 0.5 seconds for pace-private, and less than 0.1 seconds for fps-data. For pace-public and pace-private there are some outliers with runtimes of 27.62 seconds and 7.1 seconds, whereas such extremes are not observed for fsp-data.

The impact of our graph reduction procedure on the directed feedback vertex set is shown in Table 5.4 where we measure the size of the DFVS at the first checkpoint, denoted by F, as well as at the second checkpoint, denoted by F'. The results illustrate that the reduction rules are also capable of finding vertices that belong to a minimum DFVS as they selected vertices for F for 70 respectively 58 instances in pace-public and pace-private. At the second checkpoint, which is only applicable for graphs that can be partitioned into multiple SCCs, we observe that $|F'| > |F|$ holds for nine instances in pace-public and ten instances in pace-private, which means that the DFVS has been extended compared to the first checkpoint. However, the success rate is lower for fsp-data where vertices belonging to F were only found in nine cases and no further vertices were picked for F'. For instances in pace-public and pace-private where $|F| > 0$, between 2% and 3% of the vertices were added on average to the DFVS over the whole graph reduction procedure. A lower mean value of 0.92% is observed for instances in fsp-data.

5.4 Comparison of MILP Formulations

Before our MILP formulations from Chapter 4.2 are employed within the metaheuristic framework, we utilize them in standalone solving procedures. The goal is to test and evaluate their capabilities to find optimal or at least good heuristic solutions for instances of various sizes within a given time frame. Our focus is on comparing the solution qualities that the different formulations achieve within the same time, so we set a time limit of 550 seconds for each instance, which is derived from the time limit of the heuristic

Table 5.4: The average, minimum, and maximum size of the DFVS as percentage of the original number of vertices as well as the number of instances in each data set for which such a DFVS is not empty. The DFVS after the first application of the reduction rules to the input graph is denoted by F and F' represents the DFVS after the graph partitioning and merging and repeated application of the reduction rules. For the metrics, we only consider instances where $|F| > 0$.

Data Set	Size	Size of DFVS F				Size of DFVS F'			
			[%]				[%]		
		> 0	Average	Min	Max	$> \|F\|$	Average	Min	Max
pace-public	100	70	2.89	0.002	28.54	9	2.91	0.002	28.54
pace-private	100	58	2.00	0.003	16.44	10	2.01	0.003	16.48
fsp-data	40	9	0.92	0.100	3.00	0	0.92	0.100	3.00

track of the PACE 2022 challenge. Each formulation is tested twice on every instance in the pace-public data set, once with Gurobi as the employed solver and once with SCIP. Moreover, we set a memory limit of 20 GB for both solvers as some of the MILP models of larger instances, especially the ones produced with the original MTZ-based formulation, tend to consume a lot of memory. Besides, each solver can only use a single thread.

We first apply our graph reduction procedure to each input graph and then encode the resulting (sub-)problems with one of the MILP formulations and solve it with the respective MILP solver. In case of multiple subproblems, we dynamically adjust the time limit such that the MILP solver has the same amount of time for solving each subproblem. If a subproblem is solved to optimality before its assigned time is used up, the spare time is evenly split and distributed to the remaining subproblems. When using the CEC-based formulation, we also have to set a time limit for the construction of the 2-cycle constraints as described in Chapter 4.2.2. After a limited number of preliminary tests, we decided to restrict the construction of the 2-cycle constraints to half of the time provided for the MILP optimization. This might seem like a lot but these constraints proved to be very beneficial for the CEC-based formulation. Besides, the constraints are completed in a fraction of the given time for most instances, such that the available runtime for the MILP solver is usually only slightly affected.

The results of the experiments are illustrated by the box plots in Figure 5.4, which present the distribution of the achieved solution qualities for each MILP formulation and solver. The solution qualities are measured as percentages of the best known solution for each instance, where the best known solution is the solution with the smallest DFVS over all solutions recorded by the PACE 2022 challenge and our own solving procedures. For simplicity, we will refer to these best known solutions as optimal solutions for the remainder of this work. The blue box plots represent the results achieved with Gurobi as MILP solver and the red ones show the solutions from SCIP. We can observe that the commercial product Gurobi outperforms the open source project SCIP for all MILP

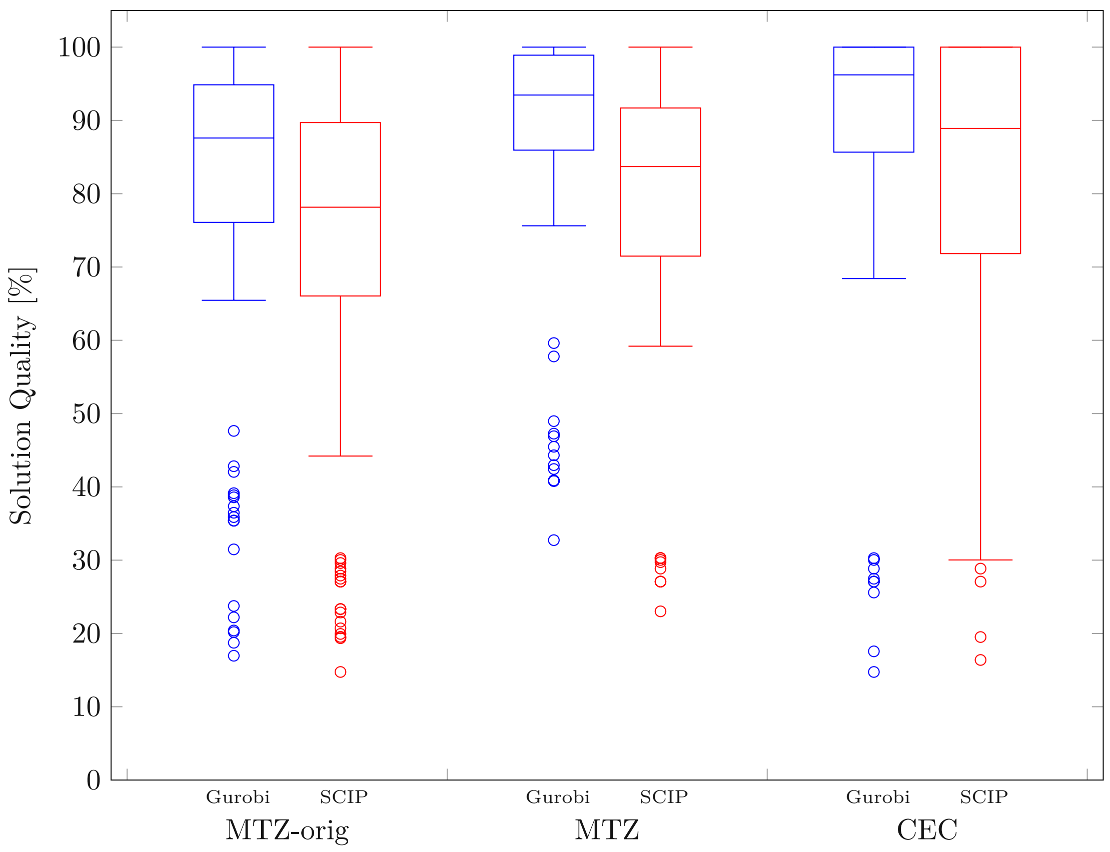


Figure 5.4: Box plots showing the solution quality of various standalone MILP solving procedures as percentages of the best known solution values. Each MILP formulation is tested with Gurobi as well as SCIP as the employed MILP solver.

formulations as, for example, Gurobi generally produces a smaller variance of the results. The box plots also illustrate that the improved MTZ-based formulation performs better than the original one. However, when compared to the CEC-based formulation, the results are more ambiguous. On the one hand, CEC achieves a higher median than MTZ, which means that CEC finds solutions of high quality for more instances. On the other hand, all of the outliers produced by the combination of CEC and Gurobi have a worse solution quality than the worst solution found by Gurobi for the MTZ-based formulation.

These observations are confirmed by the statistical values given in Table 5.5, where the arithmetic and geometric means of the solution qualities and also the number of found optimal solutions are listed. We consider the arithmetic mean as well as the geometric mean of the solution qualities to get more insights as the geometric mean is less affected by outliers, of which there are quite a few as depicted in the box plots. However, the MTZ-orig formulation is inferior to the other two formulations in all recorded metrics, which is why we do not consider this formulation in the following experiments. When

Table 5.5: The arithmetic and geometric means of the solution quality of standalone MILP solving procedures as percentage of the best known solution values as well as the number of found optimal solutions. For each MILP formulation, we differentiate between the MILP solvers Gurobi and SCIP.

Formulation	Average Solution Quality [%]				Optimal Solutions	
	Arithmetic Mean		Geometric Mean			
	Gurobi	SCIP	Gurobi	SCIP	Gurobi	SCIP
MTZ-orig	78.83	71.77	73.28	65.24	14	4
MTZ	87.35	78.78	84.93	75.61	18	7
CEC	87.34	81.59	82.59	77.23	38	26

employing Gurobi as the MILP solver, the formulations MTZ and CEC show similar performances with MTZ being about 2.34% better in terms of the geometric mean. In contrast, CEC outperforms the MTZ-based formulation when SCIP is used but in general, the average solution qualities achieved with SCIP are worse than with Gurobi. Based on these results, we decided to focus on Gurobi as the employed MILP solver in our solving procedures.

Both solvers are able to match the quality of the best known solution for some instances, independent of the used formulation. However, Gurobi finds between ten and twelve more optimal solutions for the different MILP formulations than SCIP. There is also a substantial difference between the formulations as CEC is able to produce 38 optimal solutions, whereas MTZ results in only 18 optimal solutions. Furthermore, this means that the CEC formulation performs considerably worse than MTZ for other instances as otherwise, it would also have a higher average solution quality than MTZ but this is not the case. This conforms to the observations made based on the box plots. We assume that the on average worse performance of the CEC-based formulation has something to do with the time required for the preprocessing of the CEC models by the MILP solver and for the computation of an initial feasible solution.

5.4.1 Extension with Construction Heuristic and Local Search

As we intend to provide an initial solution to the MILP solver employed within the LNS framework, we also evaluate the impact of such a warm start on the standalone MILP solving procedures. The procedures remain the same for the most part and the only difference is that for each (sub-)problem, an initial solution is computed before it is handed over to the MILP solver. We test two approaches to come up with such an initial solution: The first is to solely apply the construction heuristic based on degree and topological ordering, denoted by CH, and the second is to follow this CH up with a local search using the one-flip neighborhood, denoted by CH+LS. In this way, we can also observe whether, and if so, how much the quality of the initial solution influences the results of the MILP formulations as the combination of the CH and LS provides solutions of higher quality. We conduct these experiments with the MTZ- and CEC-based formulations using Gurobi with a single thread as the MILP solver. The time limit is again set to 550 seconds for each instance and dynamically adjusted in case of multiple SCCs. However, we also have to set a time limit for the local search procedure when using the combination of the CH and LS, as the one-flip neighborhood turned out to be time-consuming for some instances during preliminary tests. In general, we set a time limit of 60 seconds for the LS but this also has to be adjusted if there are multiple subproblems. In this case, the LS time limit is based on the available time for the respective subproblem and it is set to either 60 seconds or to one third of the available time, depending on which one is shorter. If the LS terminates before reaching its time limit, the remaining time is given to the respective MILP optimization.

The results of these tests are illustrated by the box plots in Figure 5.5, which show the distribution of the achieved solution qualities for each MILP formulation and warm start approach. The approaches using only the CH are represented by the blue box plots whereas the CH and LS combination is given by the red box plots. First of all, we observe a notable difference to the results without warm start. For example, there is less variance in the results produced with the MILP warm start, no solution achieves less than 25% of the optimal solution, and the solutions exhibit an overall higher quality. This shows that in our case, an initial solution is beneficial for the MILP solver. Moreover, a better initial solution also improves the performance of the MILP solver as the warm start provided by the CH+LS approach results in higher solution qualities. Regarding the different MILP formulations, the box plots do not give a lot of information about which one performs better.

To gain more insights on the performance of the MILP formulations, we again measure the arithmetic and geometric means of the solution qualities and list them in Table 5.6 together with the number of optimal solutions. These metrics illustrate that the differences between the performances of the CEC- and MTZ-based formulations decrease to about 1% when using warm starts. We also want to point out that CEC now outperforms MTZ, which is in contrast to the results without warm starts. In general, the recorded values confirm the observations from the box plots regarding the benefits of providing an initial solution to the MILP solver. This is especially visible in the change of the geometric mean, which, for example for the CEC-based formulation, increases from 82.59% to 89.55% and 93.43% for the CH and CH+LS based approaches, respectively. This substantial improvement of the performance of CEC might be linked to the fact that this formulation does not do well on certain instances when having a time limit. Providing an initial solution ensures that the solver terminates with at least this solution even in the worst case where it was not able to do any optimization. However, the different warm start approaches do not impact the number of found optimal solutions, which remain at 38 for the CEC-based formulation and 18 for MTZ.

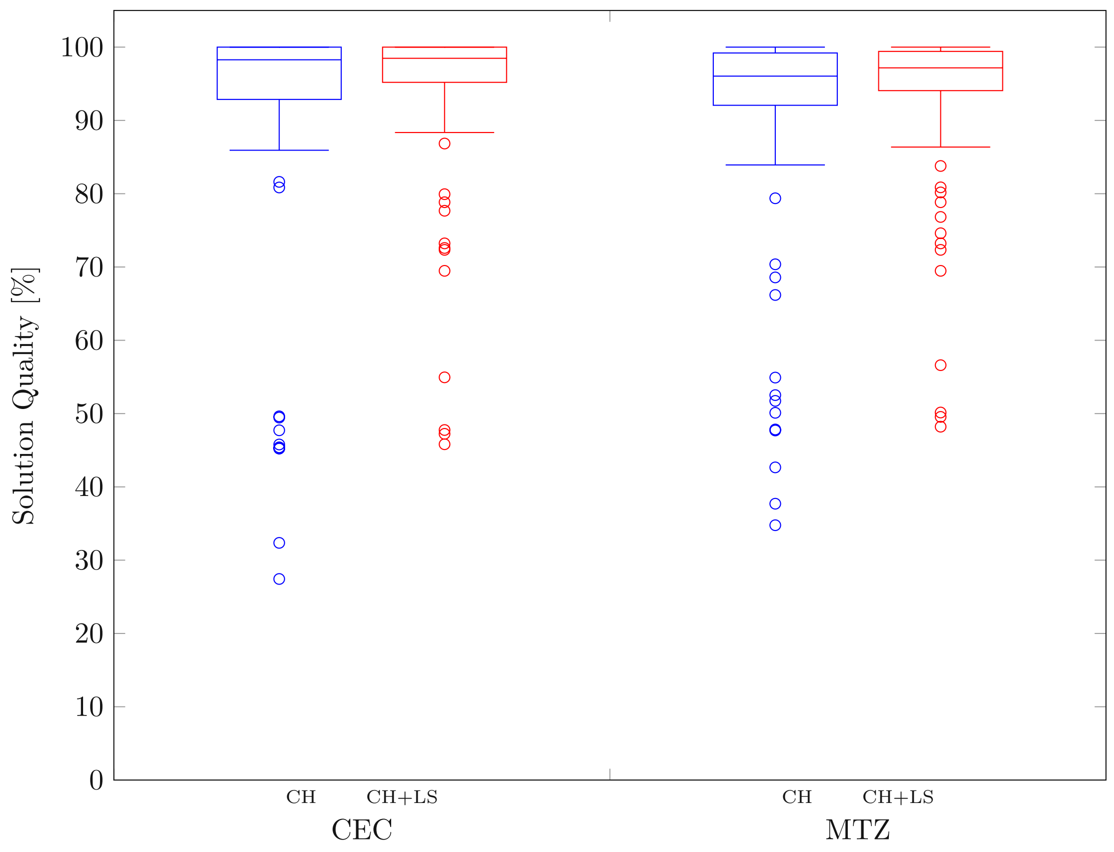


Figure 5.5: Box plots showing the solution quality of standalone MILP solving procedures with warm starts in Gurobi as percentages of the best known solution values. Warm start means that an initial solution is provided to the MILP solver.

5.5 Comparison of Neighborhood Structures Within the Hybrid Large Neighborhood Search

In this section, we test our hybrid LNS as described in Chapter 4.6 with our proposed neighborhood structures (cf. Chapters 4.5.1 and 4.5.2). First, we evaluate and compare the practicability of the different neighborhoods by conducting experiments with a time and memory limit for the solving procedures. Then, we further analyze the capabilities of the best variant in depth by providing more time and memory.

5.5.1 Testing the Neighborhoods' Practicability

As we want to find out which neighborhood structure performs best and is the most suitable for real-world scenarios, we test multiple variants of the hybrid large neighborhood search with limited resources regarding the available time and memory. We extend these tests by using different MILP formulations within the repair operator of the respective

Table 5.6: The arithmetic and geometric means of the solution quality of standalone MILP solving procedures using Gurobi with warm starts as percentage of the best known solution values as well as the number of found optimal solutions. For each MILP formulation, we differentiate between initial solutions provided solely by the construction heuristic (CH) and initial solutions provided by the CH followed by a local search (CH+LS).

Formulation	Warm Start	Average Solution Quality [%]		Optimal Solutions
		Arithmetic Mean	Geometric Mean	
CEC	CH	91.73	89.55	38
	CH+LS	94.36	93.43	38
MTZ	CH	90.75	88.91	18
	CH+LS	93.41	92.58	18

neighborhood structure. In general, we restrict the runtime of the whole solving procedure to 550 seconds and the available memory to eight gigabytes. These limitations are both derived from the requirements of the heuristic track of the PACE 2022 challenge. We use Gurobi as the MILP solver in all approaches and set a thread limit of one and a memory limit of four gigabytes for the solver. Furthermore, we set two time limits of 60 and 30 seconds for the optimization process of Gurobi, where the second time frame is only relevant if Gurobi fails to find an optimal solution for the encoded DFVS subproblem within the first 60 seconds. If that happens, we restart the optimization from the last feasible solution found by Gurobi and not only provide another 30 seconds but also set an optimality gap of 0.5%. This setting allows Gurobi to terminate with an optimal solution when the relative difference between the feasible solution and the best proven bound is equal or lower to the gap value. Such an approach can be beneficial in terms of runtime as proving the optimality of a feasible solution can be quite hard and time consuming.

All tested solving approaches apply our graph reduction procedure first, before using the construction heuristic based on degree and topological ordering to find an initial solution for each subproblem. The initial solution is then improved by running a local search with the one-flip neighborhood for at most 60 seconds. Aside from the different neighborhoods, the configuration of the LNS is the same for all variants. During the search, only improving solutions are accepted and the procedure terminates when either the time limit is reached or 20 consecutive unsuccessful iterations have been performed. As mentioned in Chapter 4.5.4, we employ tournament selection in each destroy operator with a tournament size of three and selection based on the heuristic values of the vertices. Moreover, the degree of destruction is randomly chosen in each iteration with possible values ranging from five to 1000. In later experiments, this simple random selection is replaced by the more advanced selection techniques described in Chapter 4.5.3, which are analyzed in detail in Chapter 5.6.

Regarding the neighborhood structures, we only evaluate the enlarge-DAG and enlarge-partial-DFVS neighborhood because preliminary tests already showed that the original enlarge-DFVS neighborhood is often too expensive to be used in practice. This is why we use the enlarge-partial-DFVS neighborhood instead where we limit the size of the partial DFVS to 3000 and employ tournament selection based on the heuristic values of vertices with a tournament size of three. For this neighborhood, we focus on the MTZ-based MILP formulation, which we test with and without the provisioning of an initial solution. For the enlarge-DAG neighborhood, we utilize the MTZ-based as well as the CEC-based formulation and solve both with warm starts.

The box plots in Figure 5.6 show the results by depicting the achieved solution qualities of the various approaches. For the LNS variants that employ the MTZ-based formulation, barely any differences are noticeable independent of the used neighborhood structure. The usage of warm starts also does not seem to have any substantial impact on the solution qualities obtained by the enlarge-partial-DFVS neighborhood. It can only be observed that the enlarge-partial-DFVS neighborhood has some stronger outliers than the enlarge-DAG neighborhood. However, when comparing the two MILP formulations within the enlarge-DAG neighborhood, it becomes apparent that the CEC-based models perform on average better than the MTZ-based ones. Overall, the enlarge-DAG neighborhood with the CEC-based formulation seems to beat the other approaches in terms of solution quality. In general, these results also illustrate the advantages of the hybrid LNS approaches over the standalone MILP solving procedures as the variance in the results decreases and mostly higher quality solutions are obtained. For example, no solution produced with the enlarge-DAG neighborhood achieves less than 50% of the optimal solution, whereas the worst solutions obtained by the extended MILP procedures range between 27% and 48%.

To confirm our observations, we also measure the arithmetic and geometric means of the solution qualities as well as the number of found optimal solutions, which are given in Table 5.7. These metrics show not only the superiority of the hybrid LNS approaches over the standalone MILP solving procedures but also that the enlarge-DAG neighborhood structure together with the CEC-based formulation achieves the best solution quality on average. However considering MTZ-based models, the LNS variants with the enlarge-partial-DFVS neighborhood perform better than the one with the enlarge-DAG neighborhood but they still do not reach the quality provided by the CEC-based formulation. This conforms to the results from the extended standalone MILP solving procedures where CEC outperforms MTZ. Surprisingly, the MTZ-based approach without MILP warm starts results in a slightly better mean solution quality than when initial solutions are provided to Gurobi for the MTZ-based models. This contradicts our observations for the standalone MILP solving procedures where warm starts on average improve the final solutions. We assume that this difference is linked to the smaller size of the DFVS subproblems that are now solved by the MILP solver, for which it is usually easier to find an initial solution than for bigger instances. Regarding the number of found optimal solutions, the hybrid LNS variants perform almost equally, with three of them finding six optimal solutions and one finding seven. However, these numbers show a drastic decrease when compared to the extended standalone MILP solving procedures where the CEC- and MTZ-based approaches obtain 38 and 18 optimal

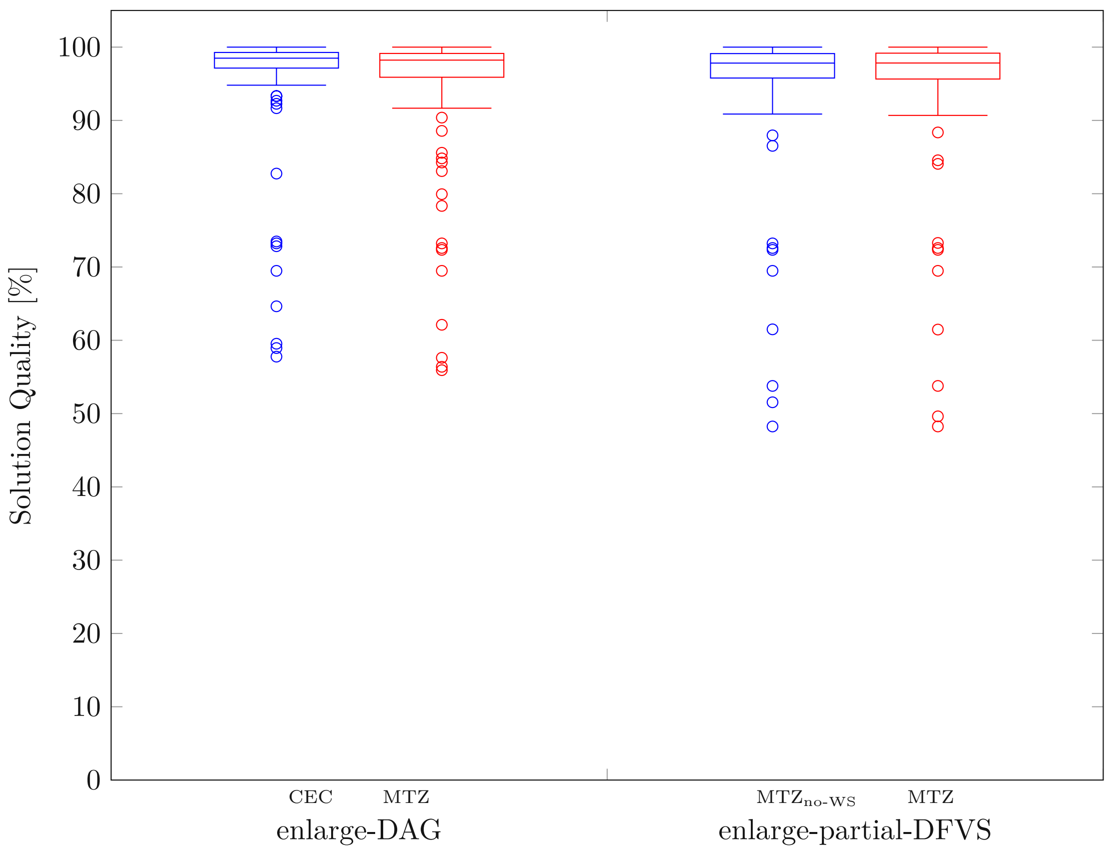


Figure 5.6: Box plots showing the solution quality of various LNS neighborhoods and MILP formulations using Gurobi as percentages of the best known solution values. All approaches employ warm starts for the MILP solving, except for $\text{MTZ}_{\text{no-WS}}$ where no initial solution is supplied.

solutions respectively. The fact that the hybrid LNS procedures still perform better overall is due to the substantial increase in the average solution quality and thus, that solutions of low quality are produced for less instances.

5.5.2 Testing Enlarge-DAG Neighborhood's Capabilities

After determining that the variant of the hybrid LNS with the enlarge-DAG neighborhood and the CEC-based MILP formulation performs best under time and memory restrictions, we wanted to gain more insights on the general capabilities of this approach. In order to find out more about this procedure's potential to improve the objective value of a solution, we conduct further experiments with some changes in the configuration. Most settings remain the same as before but the most important differences are the removal of the limited number of unsuccessful iterations as termination condition and the increase of the time limit to 100 minutes. We also increase the available memory to 16 gigabytes

Table 5.7: The arithmetic and geometric means of the solution quality as well as the number of found optimal solutions of various LNS neighborhoods and MILP formulations. The mean values are given as percentage of the best known solution values. For each neighborhood, we differentiate between the MILP formulations that are employed in the repair operator. The MILP models are all solved with Gurobi with warm starts except for $\text{MTZ}_{\text{no-WS}}$, which is also solved by Gurobi but without an initial solution.

Neighborhood	Formulation	Average Solution Quality [%]		Optimal Solutions
		Arithmetic Mean	Geometric Mean	
enlarge-DAG	CEC	95.42	94.87	6
	MTZ	94.36	93.72	6
enlarge-partial-DFVS	$\text{MTZ}_{\text{no-WS}}$	94.70	93.99	6
	MTZ	94.55	93.81	7

Table 5.8: Various properties of the test instances with distinction between the original and reduced graphs. We list the vertex number n, the edge number m, and the density of the respective graph. For the reduced graph, the number of contained 2-cycles is also given.

Instance	Original Graph			Reduced Graph			
	n	m	Density	n_r	m_r	Density	2-cycles
h_055	15268	74339	3.189×10^{-4}	14802	73681	3.363×10^{-4}	8004
h_071	8192	168714	2.514×10^{-3}	8185	168612	2.517×10^{-3}	68134
h_087	58960	269439	7.751×10^{-5}	54119	261749	8.937×10^{-5}	24002
h_101	65536	618237	1.440×10^{-4}	64737	613839	1.465×10^{-4}	276850
h_119	342573	1619304	1.380×10^{-5}	323730	1590029	1.517×10^{-5}	160001
h_165	131072	2622100	1.526×10^{-4}	130969	2620941	1.528×10^{-4}	1050264
h_193	32768	3734348	3.478×10^{-3}	32768	3734348	3.478×10^{-3}	1867174

although the previous configuration was not a limiting factor for the performance of any of the tested approaches. Besides, we extend the second time limit for Gurobi from 30 seconds to 90 seconds. Moreover to keep the testing efforts within reason, we select seven exemplary instances from the pace-public data set, on which we run these new tests. Some properties of these instances are given in Table 5.8, which lists the vertex number n, the edge number m, and the density of the original and reduced graphs and also the number of contained 2-cycles for the reduced graphs. These values illustrate that our graph reduction procedure is not so effective for these instances as all reductions lie below the average as given in Table 5.3. Therefore, we consider these instances to be among the harder ones for our solving procedures as the previous tests showed that our approaches tend to struggle with achieving large improvements of the solution quality for such instances.

To a certain degree, these concerns are proven true by the results as given in Table 5.9 and illustrated by Figure 5.7. In five out of the seven cases, the achieved improvement of

Table 5.9: Results of the tests with relaxed termination conditions for the seven test instances. The size of the initial solution provided by the CH followed by the LS is denoted by $|F_{\text{initial}}|$ and $|F_{\text{restricted}}|$ stands for the size of the solution achieved with the best approach from Chapter 5.5.1 where only restricted resources are available. The size of the solution obtained with the relaxed termination conditions and extended resources is given by $|F_{\text{relaxed}}|$. In the total improvement columns, the relative improvements from $|F_{\text{initial}}|$ to $|F_{\text{relaxed}}|$ and from $|F_{\text{restricted}}|$ to $|F_{\text{relaxed}}|$ are shown. The total iteration values are the number of executed LNS iterations and the gap denotes the longest sequence of unsuccessful iterations before another improvement is found.

Instance	$\|F_{\text{initial}}\|$	$\|F_{\text{restricted}}\|$	$\|F_{\text{relaxed}}\|$	Total Improvement [%]		Iterations	
				initial → relaxed	restricted → relaxed	Total	Gap
h_055	6196	6099	5639	8.99	7.54	326	24
h_071	6495	6298	6262	3.59	0.57	46093	24035
h_087	30995	24612	23209	25.12	5.70	54	24
h_101	49520	47653	46944	5.20	1.49	7178	335
h_119	195535	158442	144123	26.29	9.04	136	1
h_165	112521	108005	105879	5.90	1.97	3282	56
h_193	31778	31698	31640	0.43	0.18	9387	1475

the solution quality is below ten percent. In contrast, the improvement for the remaining two instances, h_087 and h_119, is above 25%, which indicates that our solving approach is indeed capable of substantial improvements. The performance difference for these two instances might be linked to the fact that they constitute the two sparsest graphs out of the seven with a relatively low number of 2-cycles. In return, such graphs usually result in a smaller and maybe easier to solve MILP model when using the CEC-based formulation as we do here. These two instances together with instance h_055 are also the only ones where the improvement of the solution quality compared to the solutions obtained by the variant with restricted resources is above five percent. However, these are also the three instances for which the resulting solutions are still the furthest away from the best known solutions in terms of solution quality. For the other four instances, we record improvements of less than two percent in comparison to the restricted approach. This shows that increasing the runtime more than tenfold does not lead to drastic further improvements.

These observations make sense considering that the applied variant of the hybrid LNS is the one that performs best with limited resources and that the employed enlarge-DAG neighborhood is the more efficient one. On the other hand, the enlarge-DFVS neighborhood structure might have a bigger potential for large improvements over such a long time but does not perform well within short time frames, which are more common in real-world scenarios. However, our tested approach is still able to find at least small improvements up to the end of the given time as shown by the plots in Figure 5.7. Another interesting detail is that better solutions are still found even after many unsuccessful iterations as, for example for instance h_071, the objective value was improved after more than 24000 iterations without effect. This indicates that the search gets stuck at a certain

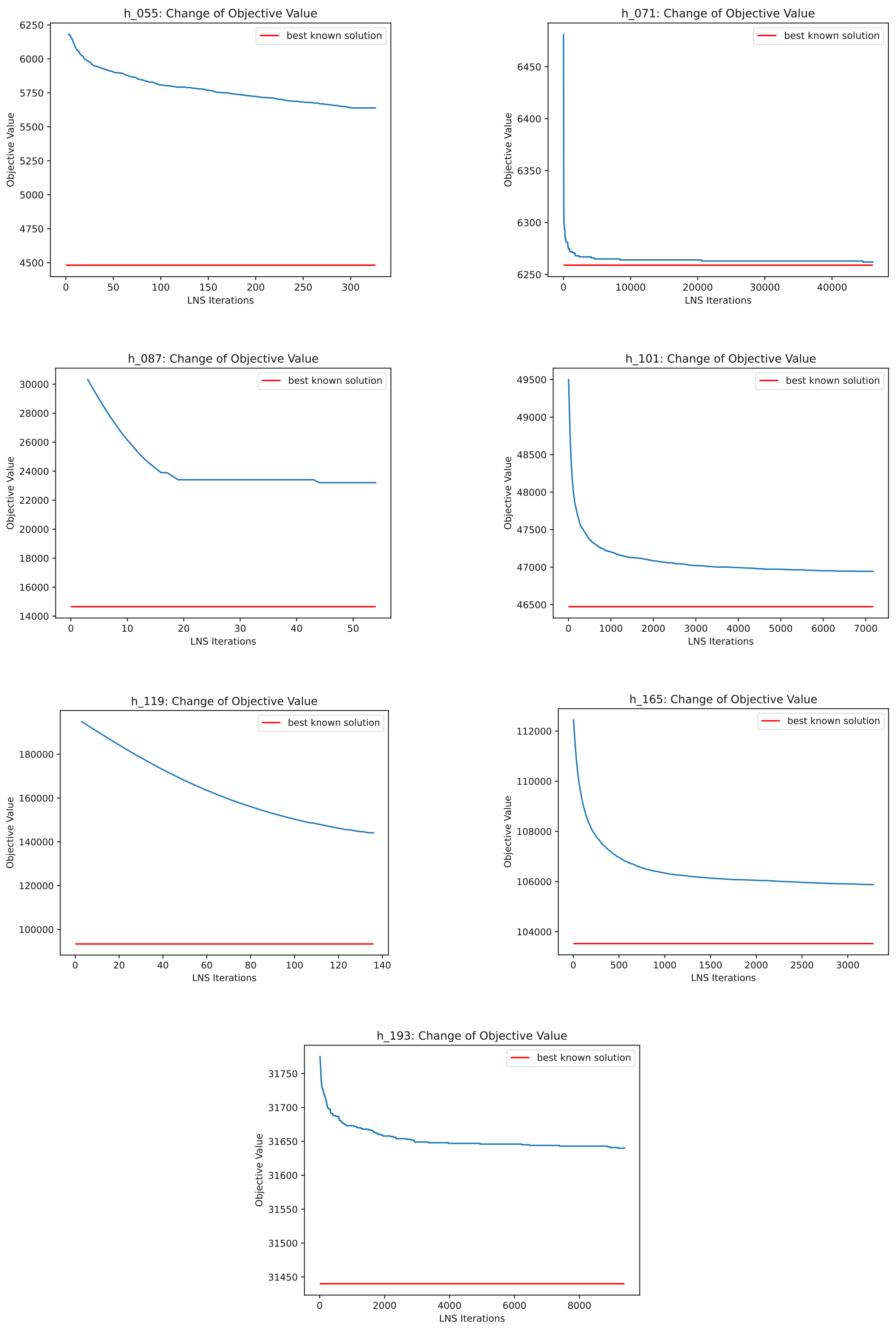


Figure 5.7: Change of the objective value of the test instances over the LNS iterations.

Table 5.10: Various metrics for the results of the tests with relaxed termination conditions for the seven test instances. First, the number of iterations and runtime needed to reach 50% of the total improvement are listed. Then, the number of executed iterations, the improvement as percentage, and the percentage of the total improvement that are achieved within 550 seconds are shown.

Instance	50% of Total Improvement		550 s		
	Iterations	Time [s]	Iterations	Improvement [%]	% of Total Improvement
h_055	45	3200.189	6	0.71	7.90
h_071	15	24.269	4038	3.51	97.85
h_087	9	122.931	11	17.06	67.90
h_101	83	134.330	574	4.43	85.13
h_119	47	233.889	73	18.97	72.15
h_165	99	263.551	253	4.21	71.24
h_193	207	156.870	810	0.32	75.18

point, which we assume is linked to an unsuitable degree of destruction. This supports us in our decision to put more effort into finding an appropriate degree of destruction and developing a strategy to do so, which we discuss in depth in Chapter 4.5.3 and come back to in Chapters 5.6.1 and 5.6.2.

The efficiency of the combination of the enlarge-DAG neighborhood structure and the CEC-based formulation is also confirmed by the values recorded in Table 5.10 and depicted in Figure 5.8. These results show that for six of the seven test instances, at least 50% of the total improvement, which is obtained over the full 100 minutes, are already achieved within the first five minutes and a relatively small amount of LNS iterations. The only exception is instance h_055, for which the 50% mark is reached after more than 53 minutes. However, this explains the big improvement we mentioned before that is now made for this instance compared to the tests with the shorter time limit. When limiting the examined time frame to the 550 seconds that are used in the previous tests, we observe that again for all instances except h_055 more than 67% of the total improvement are made within this time. In more detail, three of those instances achieve more than 70% of the total improvement, one over 85%, and one even more than 97%.

Another interesting observation regarding the 550 seconds time frame is made for instance h_119. For this instance, 73 of the in total 136 LNS iterations are completed within this time span, which is more than 53% of the iterations in less than 10% of the time. As this behavior differs from the other test instances, we further analyze the results and are able to link the discrepancy to the time needed for the MILP optimization within the repair operator of the LNS. To illustrate our findings, Gurobi's solve time per LNS iteration for instance h_119 is depicted in Figure 5.9. It shows that up to about 60 iterations, the solve time is mostly less than one second and afterwards, it starts to increase substantially. The following iterations need on average about 42 seconds for the MILP optimization and in 21 iterations, Gurobi even makes use of the full 150 seconds. Although we do not know the reason for the uncommonly drastic increase of the MILP

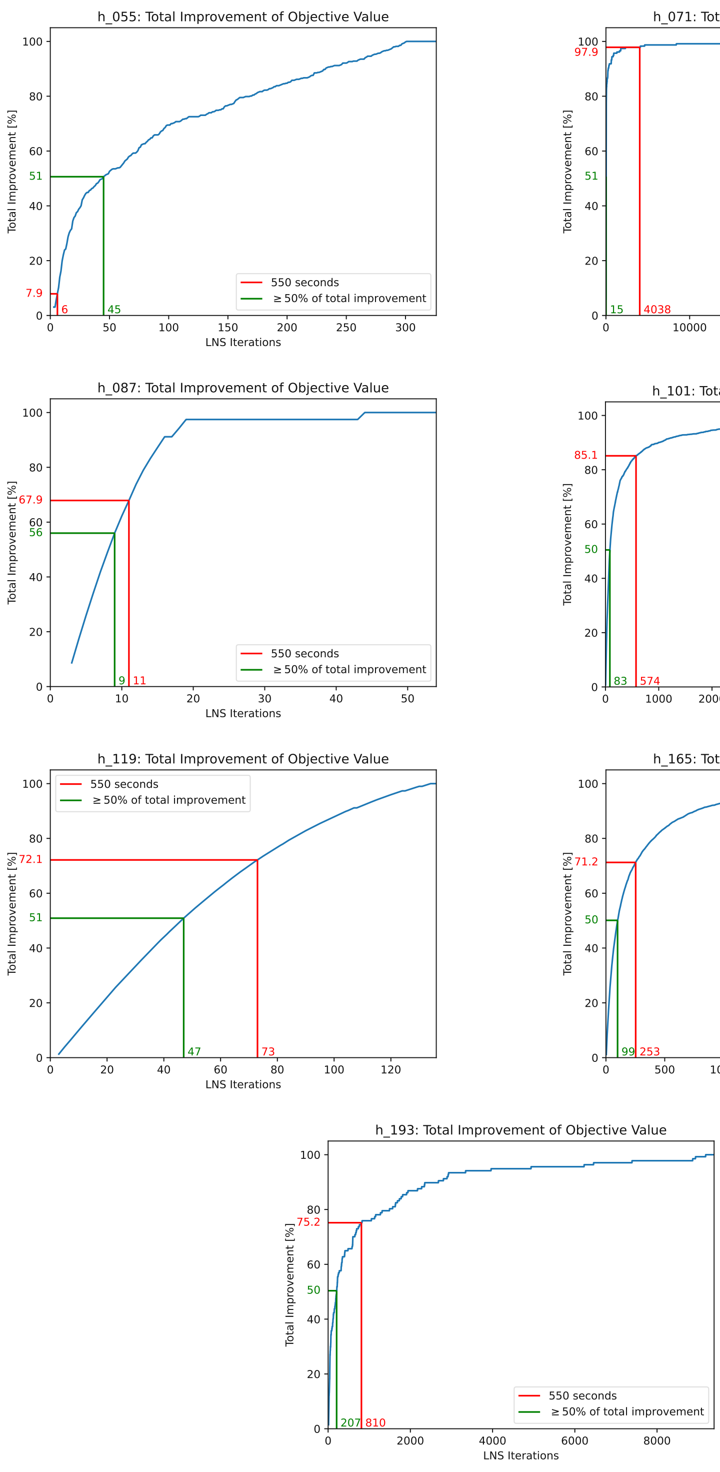


Figure 5.8: Development of the total improvement of the objective value of the test instances over the LNS iterations.

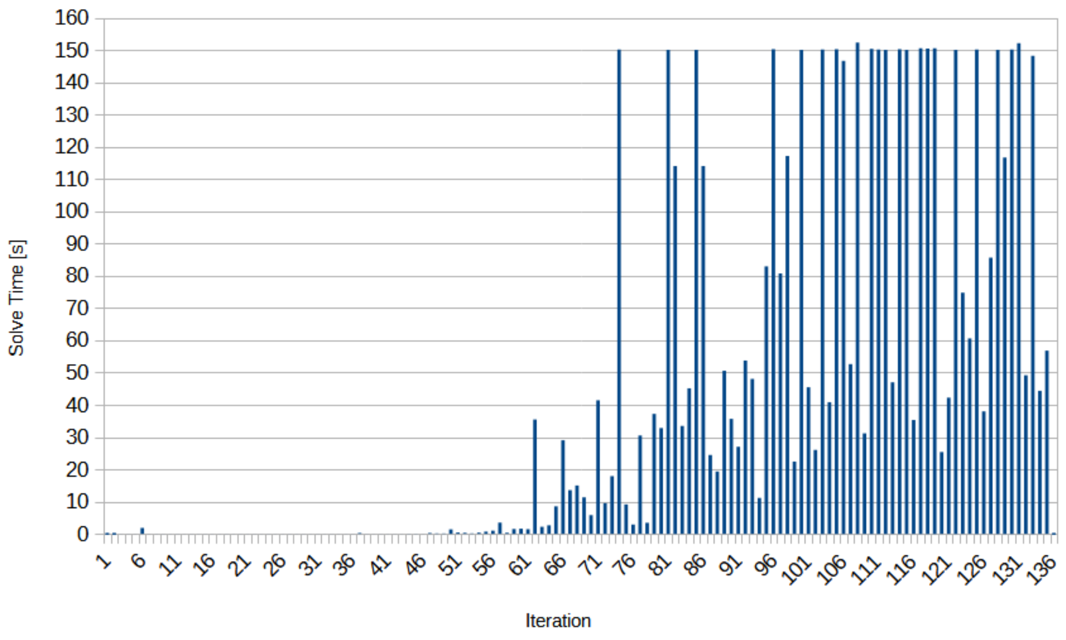


Figure 5.9: MILP solve time in seconds per LNS iteration for pace-public test instance h_119.

solve time for this instance, it explains why a lot less LNS iterations are completed after the first 550 seconds. Nevertheless, there are only four unsuccessful iterations where no improvement of the solution is achieved.

5.6 Parameter Tuning of the Hybrid Large Neighborhood Search

In this section, we evaluate the performance of various configurations of the hybrid metaheuristic based on the enlarge-DAG neighborhood, which are obtained by changing certain parameters. First, we test and analyze the usage of different fixed values for the degree of destruction in the destroy operator of the LNS, before utilizing the gained insights for dynamically choosing the value for each instance. Then, we use the previous results to additionally develop a strategy for the dynamic selection of the MILP formulation that is employed in the repair operator instead of simply preselecting the formulation for all instances. All these experiments are run on the pace-public data set and in order to test the generalizability of our selection strategies, we apply the resulting solving approaches to the instances in the pace-private data set, which is discussed in the last section.

Some settings stay the same for all conducted experiments. For the termination criteria, we use a time limit of 550 seconds and restrict the number of consecutive unsuccessful LNS iterations to 50. The overall available memory is set to 23 gigabytes whereof at most 20 gigabytes are granted to the employed MILP solver. The MILP solver of choice is Gurobi for all solving approaches and we always make use of its warm start capabilities by providing an initial solution. Aside from that, the remaining procedure is mostly the same as in Chapter 5.5.1: We again limit Gurobi to a single thread and its optimization time to 60 seconds, which is extended with another 30 seconds if no optimal solution has been found so far. In this case, the optimality gap is also relaxed to 0.5%. All tested variants start with our graph reduction procedure and compute initial solutions with the construction heuristic based on degree and topological ordering. Then, we apply a local search based on the one-flip neighborhood with a time limit of 60 seconds to improve the solution quality. Regarding the LNS, we use the enlarge-DAG neighborhood and only improving solutions are accepted during the search. In the destroy operator, we employ tournament selection with a tournament size of three and selection of the vertex with the smallest heuristic value.

5.6.1 Fixed Degree of Destruction

As we mentioned before, choosing a suitable degree of destruction k is important for the performance of the large neighborhood search. Thus, our previous approach of randomly selecting a value from a predefined range probably does not leverage the full potential of the LNS. However, what is considered suitable strongly depends not only on the problem at hand but also on the respective problem instance. For now, we leave the latter out of consideration and focus on finding generally applicable values for the DFVS problem, but we will discuss dynamic selection of k for each instance in Chapter 5.6.2.

Here, we analyze the fixed_degree(x) strategy as described in Chapter 4.5.3. The value x stands for the chosen degree of destruction which is then used for all tested instances and in all LNS iterations. In order to provide a wide variety and to gain more insights, we selected 15 values ranging from as small as 25 up to 100000. As the performance of our hybrid metaheuristics also depends on the employed MILP formulation and the degree of destruction directly impacts the size of the model to be solved, we run all tests once with the CEC-based and once with the MTZ-based formulation. The results are illustrated by the box plots in Figure 5.10 and Figure 5.11 for CEC and MTZ respectively. When using the CEC-based formulation, the solution quality seems to improve for an increasing value of k up to 3000, as the median gets higher and the variance decreases. After this supposed threshold, the solution quality seems to decrease the higher the degree of destruction. We do not observe the same behavior for the results achieved with the MTZ-based formulation, where the solution quality does not steadily improve with an increasing k. However, there still seems to be a threshold of about 3000 after which the solution quality continuously decreases.

These observations are only partially confirmed by the data recorded in Table 5.11, which again lists the arithmetic and geometric means of the solution qualities as well as the number of found optimal solutions for all tested values for the degree of destruction and both MILP formulations. These metrics verify the decrease of the average solution quality for increasing values of k over 3000 for both the CEC- and the MTZ-based

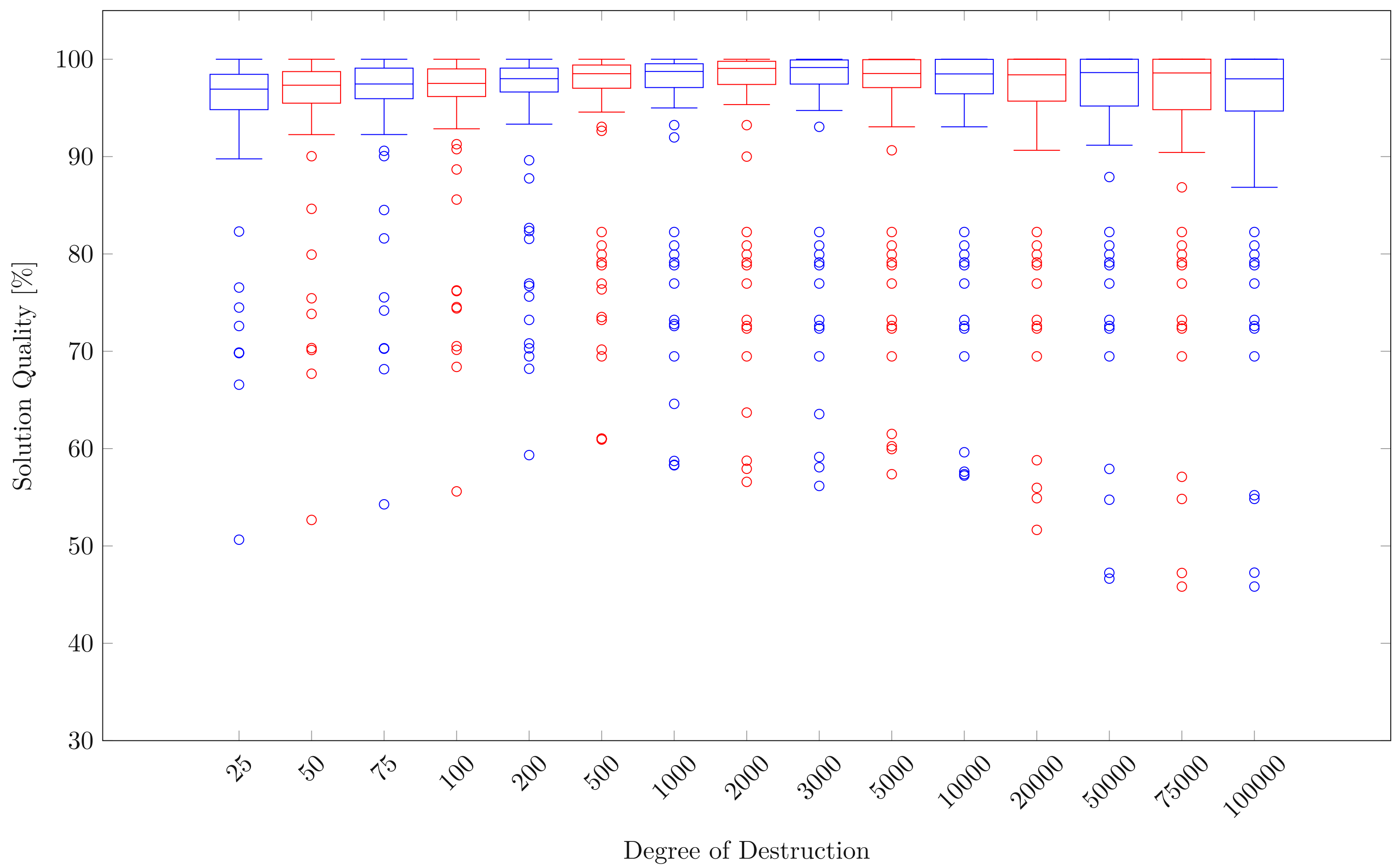


Figure 5.10: Box plots showing the solution qualities as percentages of the best known solution. The solutions were produced with the enlarge-DAG neighborhood with various degrees of destruction while employing the CEC-based formulation and Gurobi in the repair operator.

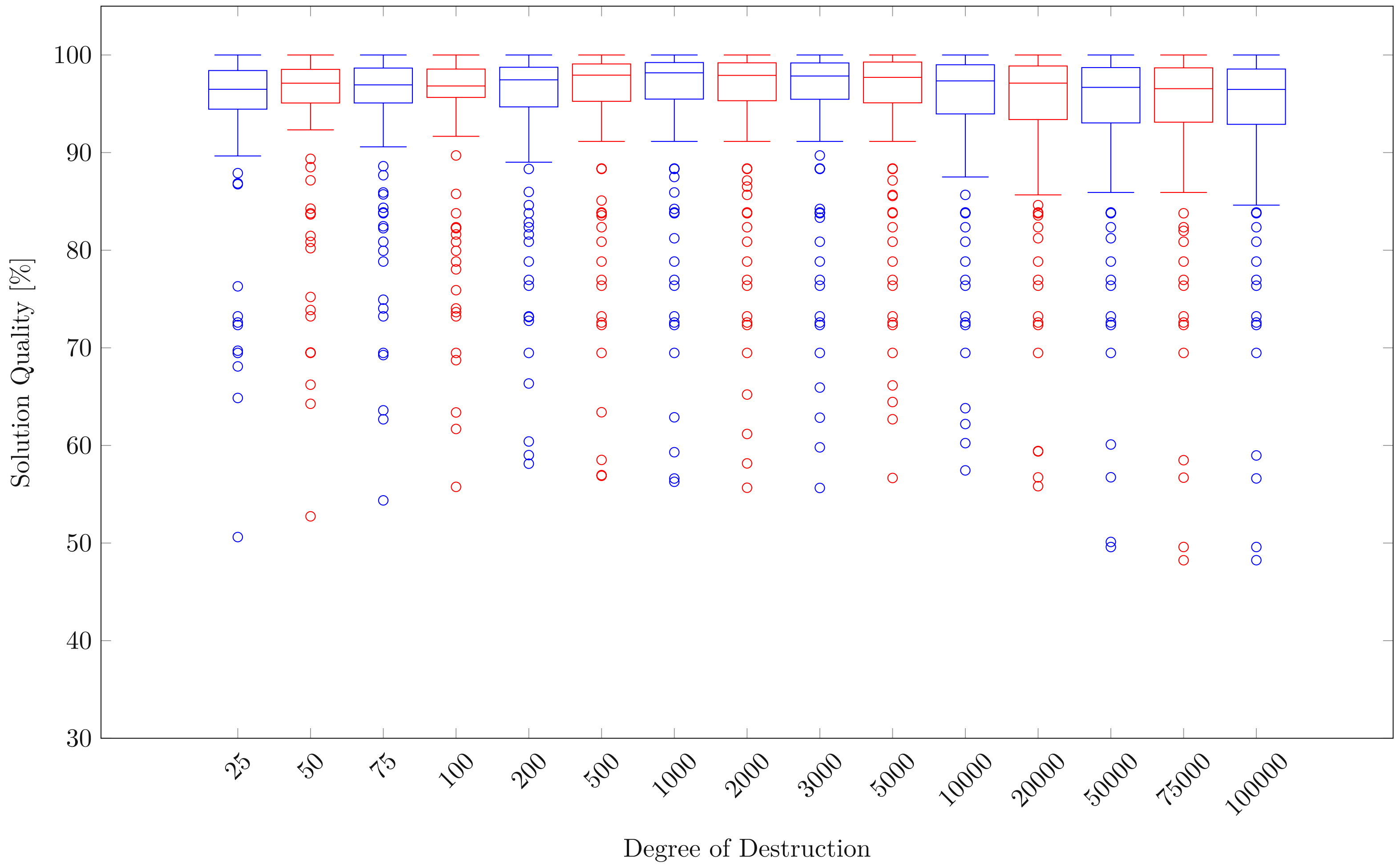


Figure 5.11: Box plots showing the solution qualities as percentages of the best known solution. The solutions were produced with the enlarge-DAG neighborhood with various degrees of destruction while employing the MTZ-based formulation and Gurobi in the repair operator.

formulation. On the other hand, a steady rise of the solution quality up to a degree of destruction of 3000 when using CEC models is not reflected by the mean values where the highest average solution quality of 95.47% is measured for $k = 75$. This discrepancy is probably due to the fact that in the box plots, more outliers are recorded for higher degrees of destruction than for lower ones and the bad solution quality for these instances negatively impacts the mean values. This is also part of the reason why for the CEC-based formulation, the solution approach with $k = 100000$ achieves the lowest average solution quality of 93.79% although it obtains the most optimal solutions, namely 35, whereas only 11 are found with $k = 75$. Another factor is that for many instances, the LNS almost degenerates into pure MILP optimization when using such high degrees of destruction and as the MILP solver is only granted a short amount of time, often only few improvements are made. The same is true for the solving approaches employing the MTZ-based formulation.

When using MTZ models, the best average solution quality, 93.91%, is measured for $k = 25$, which is a lower degree of destruction than for CEC. This might be due to the fact that the CEC-based formulation is impacted by the existence of 2-cycles, which are more likely in the enlarge-DAG neighborhood the more vertices are added to the current DAG, whereas the MTZ-based formulation is independent of 2-cycles. Besides, 25 is the lowest tested degree of destruction, which leaves room for speculation that an even lower value might achieve even better results when using the MTZ-based formulation within our solving procedure. However, as of now the MTZ-based formulation seems to be inferior to the CEC-based one no matter which degree of destruction is applied in our hybrid metaheuristic.

Moreover, the performance of the fixed_degree(x) strategy in comparison to the results that are achieved with the random selection of the degree of destruction (cf. Chapter 5.5.1) depends on the employed MILP formulation. On the one hand, the best average solution quality achieved with fixed_degree(75) and CEC models (95.47%) is slightly better than with random selection (95.42%). On the other hand, the highest mean of 93.91% obtained by fixed_degree(25) for MTZ models is considerably lower than the random selection's average of 94.36%. This shows that fixing the value of k for all DFVS problem instances is not always advantageous as different instances benefit from different degrees of destruction due to their varying graph sizes and topologies. Therefore, we refined our selection technique and moved on to dynamically choosing the degree of destruction for each instance, which is discussed in the next section.

5.6.2 Dynamic Selection of Degree of Destruction

In this section, we first present some details of our five strategies for the dynamic selection of the degree of destruction k as described in Chapter 4.5.3 and then evaluate and compare their performances within our hybrid LNS. All these strategies allow for choosing a suitable value for k based on certain graph properties of the DFVS problem instance at hand or the employed MILP formulation or both. For these experiments, we always preselect the MILP formulation that is applied in the repair operator. Besides,

Table 5.11: The arithmetic and geometric means of the solution quality of various degrees of destruction k and MILP formulations for the enlarge-DAG neighborhood as percentage of the best known solution values as well as the number of found optimal solutions. For each degree of destruction, we differentiate between the MILP formulations that are employed in the repair operator. The MILP models are all solved with Gurobi with warm starts. The best values are written in bold, using red for CEC and blue for MTZ.

k	Formulation	Average Solution Quality [%]		Optimal Solutions
		Arithmetic Mean	Geometric Mean	
25	CEC	94.62	94.20	7
	MTZ	**93.91**	**93.42**	8
50	CEC	95.29	94.91	10
	MTZ	93.77	93.26	7
75	CEC	**95.47**	**95.10**	11
	MTZ	93.46	92.92	7
100	CEC	95.31	94.92	11
	MTZ	93.31	92.73	7
200	CEC	95.03	94.60	14
	MTZ	93.38	92.79	8
500	CEC	94.89	94.34	16
	MTZ	93.71	93.06	8
1000	CEC	94.84	94.21	15
	MTZ	93.87	93.22	9
2000	CEC	94.95	94.29	18
	MTZ	93.80	93.16	10
3000	CEC	95.03	94.37	21
	MTZ	93.89	93.28	**12**
5000	CEC	94.83	94.19	25
	MTZ	93.77	93.19	**12**
10000	CEC	94.60	93.93	25
	MTZ	93.47	92.88	**12**
20000	CEC	94.37	93.62	28
	MTZ	93.08	92.43	**12**
50000	CEC	94.20	93.32	32
	MTZ	92.78	92.03	**12**
75000	CEC	94.06	93.18	34
	MTZ	92.61	91.83	**12**
100000	CEC	93.79	92.90	**35**
	MTZ	92.54	91.77	**12**

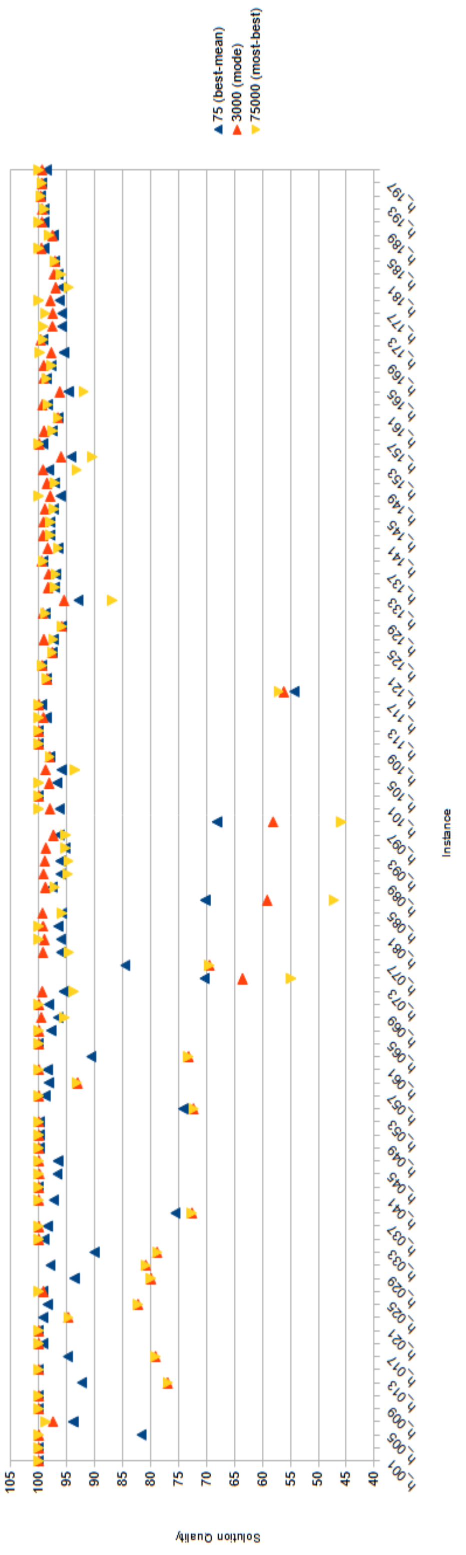
the values that are used in the five approaches are all derived from the analysis of the results of our previous tests with the fixed_degree(x) strategy, which are discussed in Chapter 5.6.1.

#2-cycles_sections. This selection strategy is based on the number of 2-cycles in the problem graph and the used sections are shown in Table 5.12 together with their assigned degrees of destruction for both MILP formulations. The number of instances in the pace-public data set that belong to each section are also given as they are the reason why the ranges of the sections vary quite a lot. At first, we defined a greater number of sections with more consistent ranges but this partitioning resulted in multiple sections containing only between one to five instances each. As the corresponding degrees of destruction are chosen based on their average solution quality, we aimed for bigger sample sizes and merged some consecutive sections with few instances to obtain at least ten instances in each section.

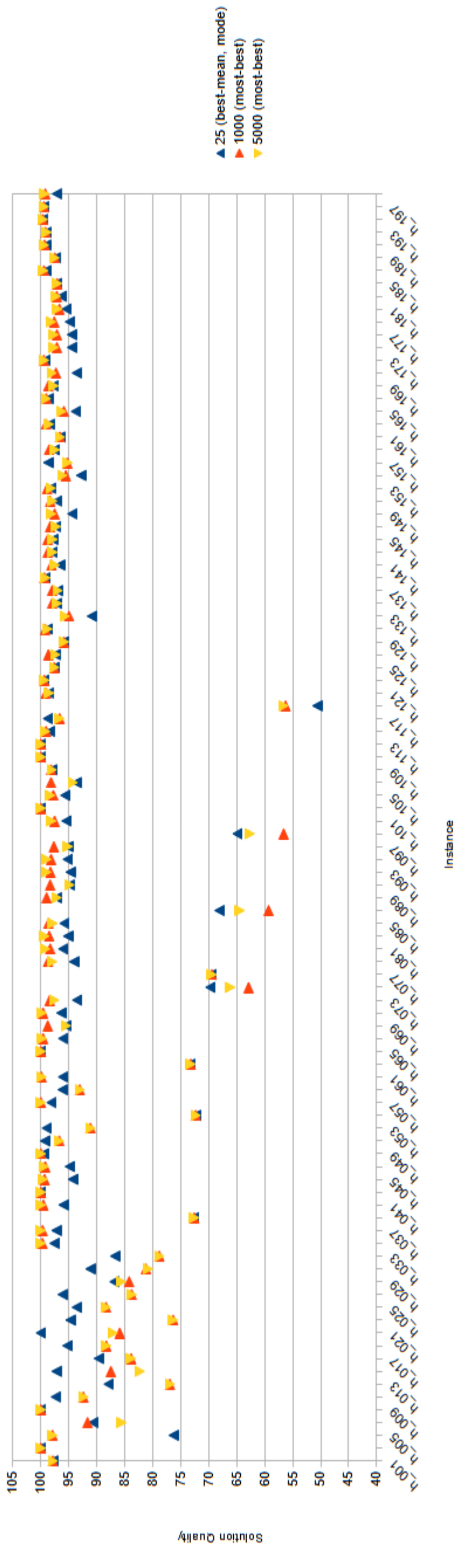
Table 5.12: The ranges of the 2-cycles sections in terms of the number of 2-cycles together with the number of pace-public instances contained in each section and the corresponding value of the degree of destruction k for the two MILP formulations.

2-cycles Section		Instances	k	
From	To		CEC	MTZ
0	100	22	50	25
101	10000	11	200	75000
10001	100000	16	200	5000
100001	200000	14	2000	1000
200001	1000000	15	2000	500
1000001	1200000	12	3000	1000
1200001	∞	10	50000	3000

MILP_triple. For this strategy, we preselect three values for k for each MILP formulation and the applied degree of destruction is then chosen randomly from these values in each LNS iteration depending on which MILP formulation is used. Table 5.13 lists these values, namely the best-mean, mode, and most-best. It shows that the best-mean and mode have the same value of 25 for the MTZ-based formulation. In order to still obtain three different values to choose from, we replace the mode with a second value for the most-best, 5000, which achieves the same number of solutions of smallest size as $k = 1000$ during the fixed_degree(x) tests. A comparison of the results from these tests for the selected three degrees of destruction of both MILP formulations is given in Figure 5.12, which illustrates the found solution quality for each instance. Furthermore, it can be observed that the values of the three metrics are always higher for the CEC-based formulation. This corresponds to the results achieved with the fixed_degree(x) strategy where the best performing degree of destruction for MTZ is also lower than for CEC.



(a) Using the CEC-based formulation.



(b) Using the MTZ-based formulation.

Figure 5.12: Solution quality of each pace-public instance achieved with the fixed_degree(x) selection strategy. The depicted degrees of destruction correspond to the MILP_triple values of each MILP formulation.

Table 5.13: The best-mean, mode, and most-best values for the degree of destruction k for the two MILP formulations. The MTZ-based formulation has two values that find the same number of solutions of smallest size, whereas all other values are unambiguous.

Formulation	Degree of Destruction k		
	best-mean	mode	most-best
CEC	75	3000	75000
MTZ	25	25	1000 5000

#2-cycles_sections_triple. As this selection strategy is a combination of the previous two, it reuses the 2-cycles sections and some of the values for k. In each LNS iteration, the degree of destruction is randomly picked from the best-2cycle-mean, most-2cycle-best, and best-mean whose values for both MILP formulations are given in Table 5.14. The best-mean never changes because it is the same as in the MILP_triple approach and therefore independent of the 2-cycles sections. Besides, the values of the best-2cycle-mean are taken from the #2-cycles_sections strategy. For the most-2cycle-best, we observe again that the values for the MTZ-based formulation are always smaller or equal to the CEC values.

Table 5.14: The ranges of the 2-cycles sections in terms of the number of 2-cycles together with the best-2cycle-mean, most-2cycle-best, and best-mean values of the degree of destruction k for the two MILP formulations.

2-cycles Section		Degree of Destruction k					
From	To	best-2cycle-mean		most-2cycle-best		best-mean	
		CEC	MTZ	CEC	MTZ	CEC	MTZ
0	100	50	25	75	25	75	25
101	10000	200	75000	500	500	75	25
10001	100000	200	5000	2000	1000	75	25
100001	200000	2000	1000	5000	5000	75	25
200001	1000000	2000	500	1000	500	75	25
1000001	1200000	3000	1000	2000	500	75	25
1200001	∞	50000	3000	2000	1000	75	25

corr_log-#2-cycles. In this strategy, we use the linear correlation between the base-10 logarithmic value of the number of 2-cycles in the instance graph and the base-10 logarithmic value of the lowest degree of destruction that has the best performance on this graph during the fixed_degree tests. Out of all the properties that we analyzed for possible correlations with the degree of destruction, this one results in the highest values. The Pearson's correlation coefficient is about 0.72 for the CEC-based formulation and 0.60

for the MTZ-based formulation, which both indicate a strong correlation. Based on these correlations, we derive a function to compute k from the corresponding regression lines for each MILP formulation. The scatter plots in Figure 5.13 visualize the correlations as well as the regression lines for better illustration.

Let x be the base-10 logarithmic value of the number of 2-cycles and y the base-10 logarithmic value of the degree of destruction, which is denoted by y_{CEC} and y_{MTZ} for the respective MILP formulation. Thus, the degree of destruction is given by $k = 10^y$, and the functions are defined as follows:

$$y_{CEC} = 0.433\,x + 1.304 \tag{5.1}$$

$$y_{MTZ} = 0.365\,x + 1.200 \tag{5.2}$$

corr_log-#vertices. This strategy applies the same approach as the corr_log-#2-cycles technique but uses the base-10 logarithmic value of the vertex number instead of the number of 2-cycles. For the number of vertices, we observe the second highest correlation values with the degree of destruction with Pearson's correlation coefficients of about 0.72 for CEC and 0.58 for MTZ. This again implies a strong correlation which is why we decided to include this approach in our tests as well.

Like before, we use the correlations and corresponding regression lines as depicted in Figure 5.14 to derive the following functions for computing the degree of destruction:

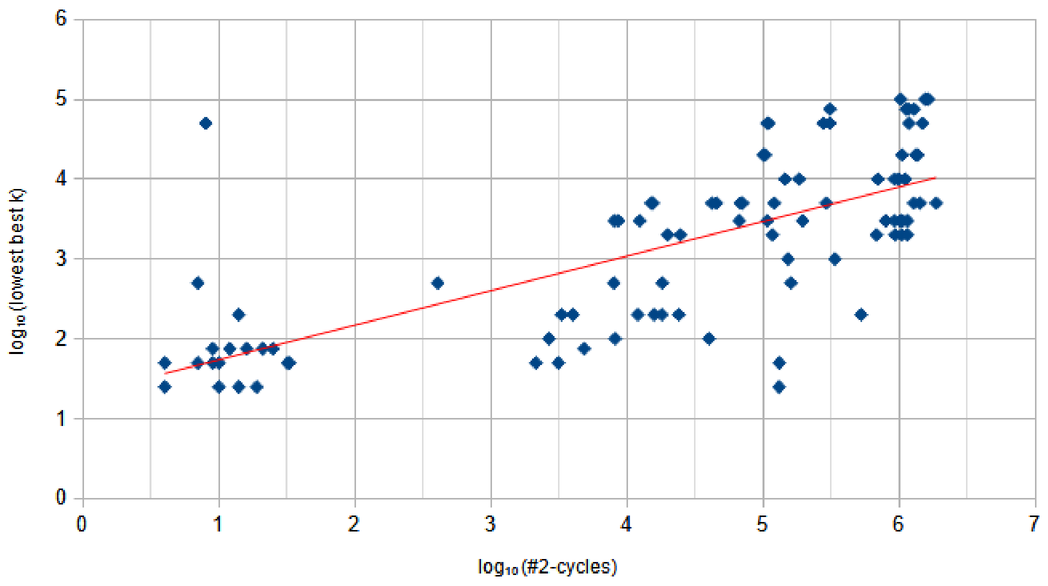
$$y_{\text{CEC}} = 1.004\,x - 0.936 \tag{5.3}$$

$$y_{\text{MTZ}} = 0.808\,x - 0.535 \tag{5.4}$$

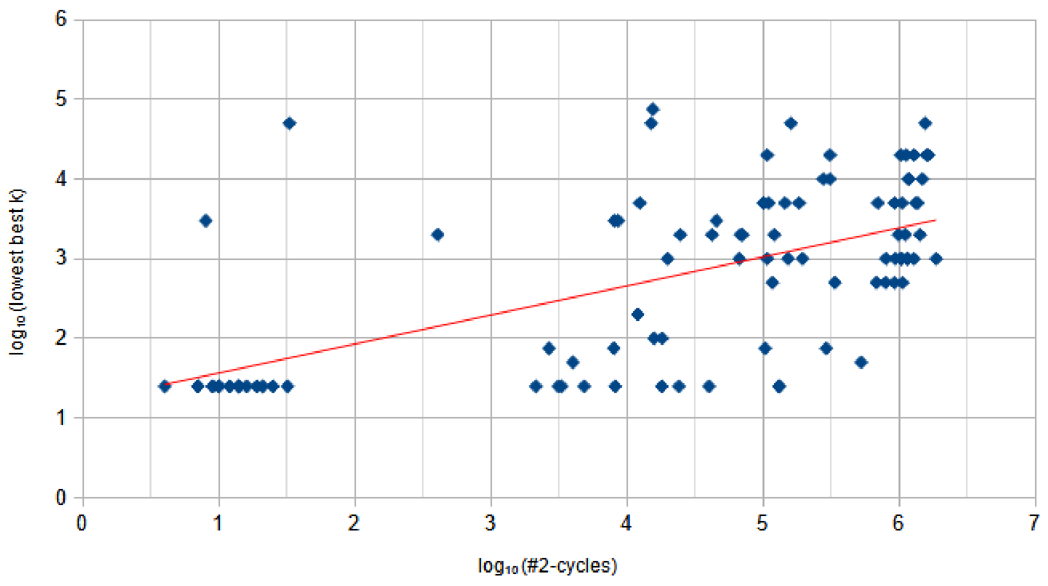
where x denotes the base-10 logarithmic value of the vertex number and y_{CEC} and y_{MTZ} again stand for the base-10 logarithmic value of the degree of destruction of the respective MILP formulation. The degree of destruction to be applied to the instance at hand can then be calculated as $k = 10^y$.

Comparison

The solution qualities obtained by the five selection strategies are illustrated in Figure 5.15 and Figure 5.16, the first depicting the results for the CEC-based formulation and the second for the MTZ-based formulation. In both cases, the median increases and the variance decreases for all five strategies in comparison to the fixed_degree(x) selection of the degree of destruction. When the CEC-based formulation is employed, there are barely any differences in the box plots noticeable, only the MILP_triple strategy seems to perform a little worse. Similar observations are made for the MTZ-based formulation, with the addition that in general the results seem to be worse than when using CEC models, considering the greater variances and slightly lower medians. However, independent of the employed MILP formulation, all approaches achieve a solution quality of at least 55% for all pace-public instances.

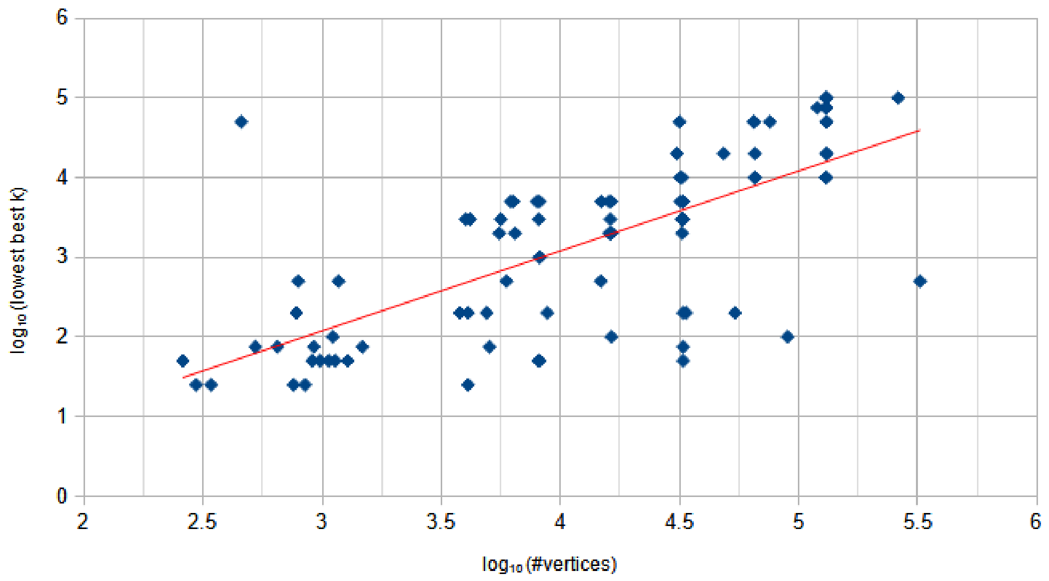


(a) Using the CEC-based formulation.

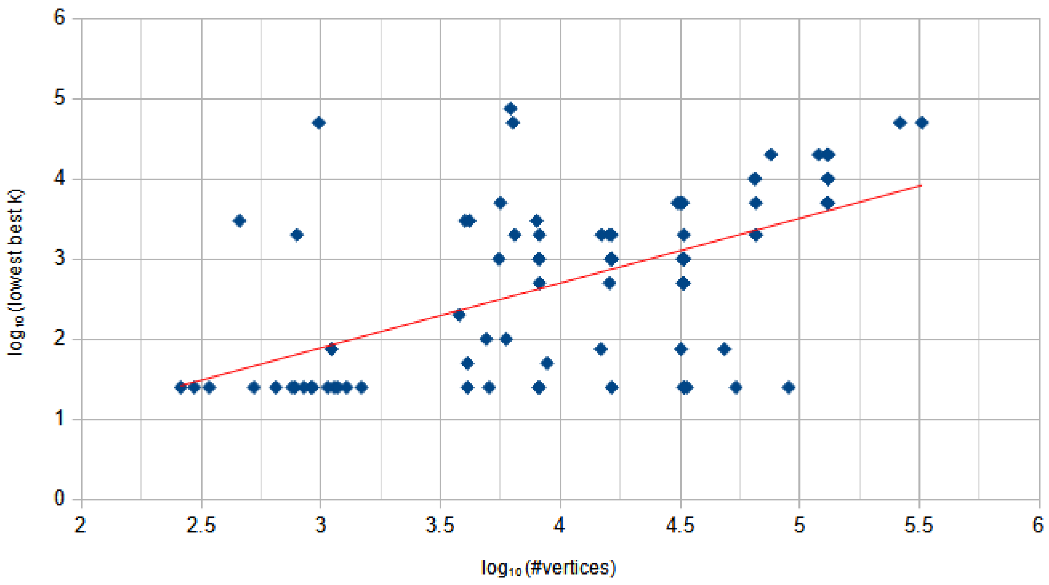


(b) Using the MTZ-based formulation.

Figure 5.13: Scatter plots of the base-10 logarithmic values of the number of 2-cycles in the graph and the lowest best degree of destruction k of each pace-public instance for both MILP formulations resulting from the tests with the fixed_degree selection strategy. The regression line is depicted in red.



(a) Using the CEC-based formulation.



(b) Using the MTZ-based formulation.

Figure 5.14: Scatter plots of the base-10 logarithmic values of the number of vertices in the graph and the lowest best degree of destruction k of each pace-public instance for both MILP formulations resulting from the tests with the fixed_degree selection strategy. The regression line is depicted in red.

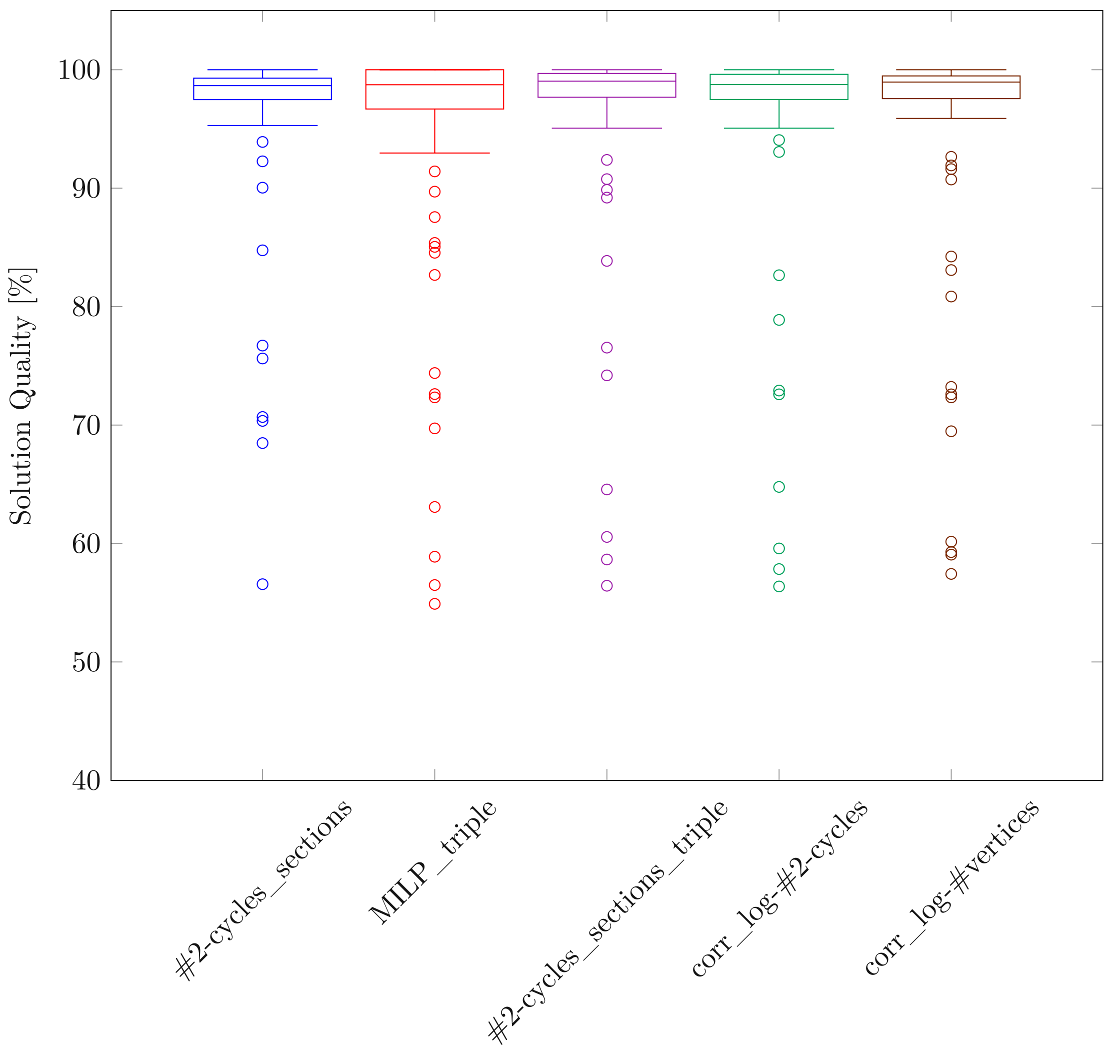


Figure 5.15: Box plots of solution qualities in % of pace-public instances produced by various selection strategies for the degree of destruction in the enlarge-DAG neighborhood. All procedures use the CEC-based formulation and Gurobi in the repair operator.

To gain more insights on the performance of the different selection strategies, we again examine the arithmetic and geometric means of the solution qualities and the number of found optimal solutions. This data is given in Table 5.15 and confirms most of the observations made above. It shows that the approaches employing the CEC-based formulation always achieve better average results than the corresponding ones with the MTZ-based formulation. Besides, it is also true that the MILP_triple selection strategy results in the lowest mean value of 95.16% for the CEC-based formulation. For the MTZ-based formulation however, strategy corr_log-#vertices with a mean of 94.22% performs slightly worse than MILP_triple with 94.24%. Interestingly, all variants using

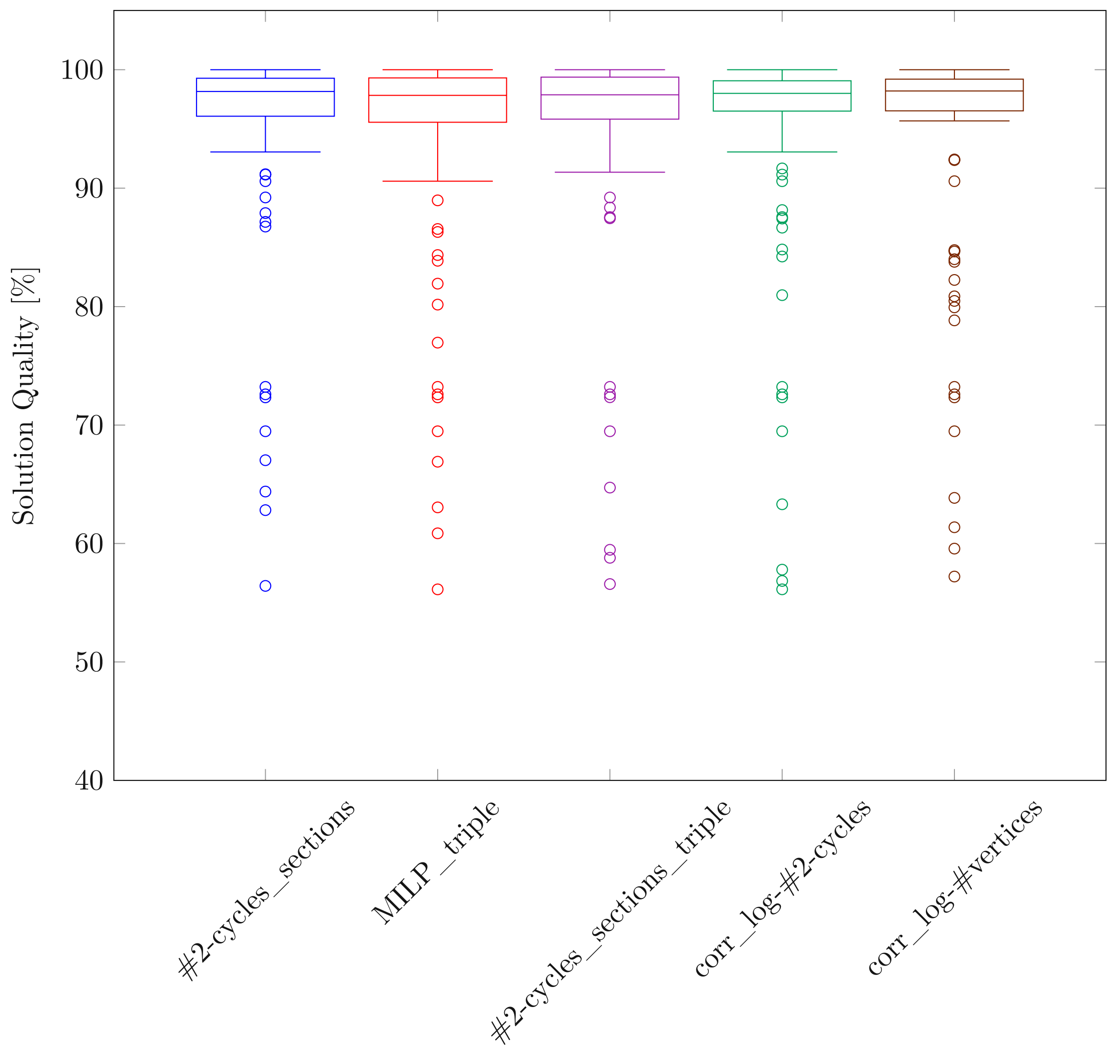


Figure 5.16: Box plots of solution qualities in % of pace-public instances produced by various selection strategies for the degree of destruction in the enlarge-DAG neighborhood. All procedures use the MTZ-based formulation and Gurobi in the repair operator.

MTZ models outperform on average the best fixed_degree approach (93.91%) with the same formulation, whereas for the CEC-based formulation, fixed_degree(75) with a mean of 95.47% is better than MILP_triple (95.16%) and corr_log-#vertices (95.42%). Regarding the MILP_triple strategy, this might be due to the value of most-best being 75000, which is quite a high value for the degree of destruction for many instances. Thus, if this value is often picked during the random selection in the LNS iterations, probably only few improvements are made. This might also be the reason why this variant performs on average worse than when random selection of the degree of destruction is used (95.42% for CEC, cf. Chapter 5.5.1). Considering the MTZ-based formulation, MILP_triple as

Table 5.15: The arithmetic and geometric means of the solution quality of the pace-public instances as well as the number of found optimal solutions of various selection strategies for the degree of destruction in the enlarge-DAG neighborhood for both MILP formulations. The mean values are given as percentage of the best known solution values. For each selection strategy, we differentiate between the MILP formulations that are employed in the repair operator. The MILP models are all solved with Gurobi with warm starts. The best values are written in bold, using red for CEC and blue for MTZ.

Selection Strategy	Formulation	Average Solution Quality [%]		Optimal Solutions
		Arithmetic Mean	Geometric Mean	
#2-cycles_sections	CEC	**96.49**	**96.15**	12
	MTZ	**95.07**	**94.57**	**13**
MILP_triple	CEC	95.16	94.53	**32**
	MTZ	94.24	93.68	**13**
#2-cycles_sections_triple	CEC	96.32	95.84	17
	MTZ	94.89	94.34	**13**
corr_log-#2-cycles	CEC	96.13	95.62	15
	MTZ	94.50	93.89	8
corr_log-#vertices	CEC	95.42	94.81	9
	MTZ	94.22	93.62	6

well as corr_log-#vertices achieve worse mean solution qualities in comparison with the random selection approach (94.36%).

The best average solution quality is found by the #2-cycles_sections strategy for both MILP formulations, with a mean of 96.49% for CEC and 95.07% for MTZ. This selection strategy also obtains the most optimal solutions for the MTZ-based formulation, together with MILP_triple and #2-cycles_sections_triple who all find 13. Though, this is considerably less than the 32 optimal solutions that are achieve by the MILP_triple strategy when employing the CEC-based formulation. We assume that this might again be linked to the possibility of $k = 75000$ as this value resulted in the most best solutions during the fixed_degree tests.

5.6.3 Dynamic Selection of MILP Formulation

During the previous experiments, we noticed performance differences between the various approaches for the tested instances, which seem to be linked to the employed MILP formulation. This inspired the development of a strategy for the dynamic selection of the MILP formulation for each instance, which is described in Chapter 4.6.1. Here, we present the application details and results of this dynamic selection of the MILP formulation.

In our hybrid LNS, there are two application points of the MILP formulation, for which the selection strategies slightly differ: the direct MILP optimization for (sub-)problems with up to 100 vertices and the MILP formulation that is used within the repair operator of the LNS for bigger (sub-)problems. In both cases, the selection is based on the product

Table 5.16: The ranges of the sections for the product of the number of 2-cycles and the density together with the respective selected MILP formulation to be used for the direct MILP optimization. The start values of the sections are exclusive (except for 0) and the end values are inclusive (except for ∞).

2-cycles × Density Section		MILP Formulation
From	To	
0.00	0.04	MTZ
0.04	0.40	CEC
0.40	20.00	MTZ
20.00	50.00	CEC
50.00	150.00	MTZ
150.00	500.00	CEC
500.00	∞	MTZ

Table 5.17: The ranges of the sections for the product of the number of 2-cycles and the density together with the respective selected MILP formulation to be used within the LNS. The start values of the sections are exclusive (except for 0) and the end values are inclusive (except for ∞).

2-cycles × Density Section		MILP Formulation
From	To	
0	5	CEC
5	20	MTZ
20	700	CEC
700	1000	MTZ
1000	3000	CEC
3000	7000	MTZ
7000	∞	CEC

of the number of contained 2-cycles and the density of the graph at hand. This criterion seems to be the most promising for the selection strategy out of all analyzed properties. Similar to the #2-cycles_sections strategy for the dynamic selection of the degree of destruction, we partition the product values into mutually exclusive sections, which are each assigned a MILP formulation. On the one hand, the definition of the sections and applied formulations for graphs of at most 100 vertices is derived from the results of the standalone MILP solving procedures without extensions, which are discussed in Chapter 5.4. On the other hand, we utilize the results from the tests with the various dynamic selection strategies for the degree of destruction as presented in Chapter 5.6.2 to determine the sections and their assigned formulations for the usage within the LNS. The resulting selection strategies are given in Table 5.16 and Table 5.17.

We test the dynamic selection of the MILP formulation together with the following three

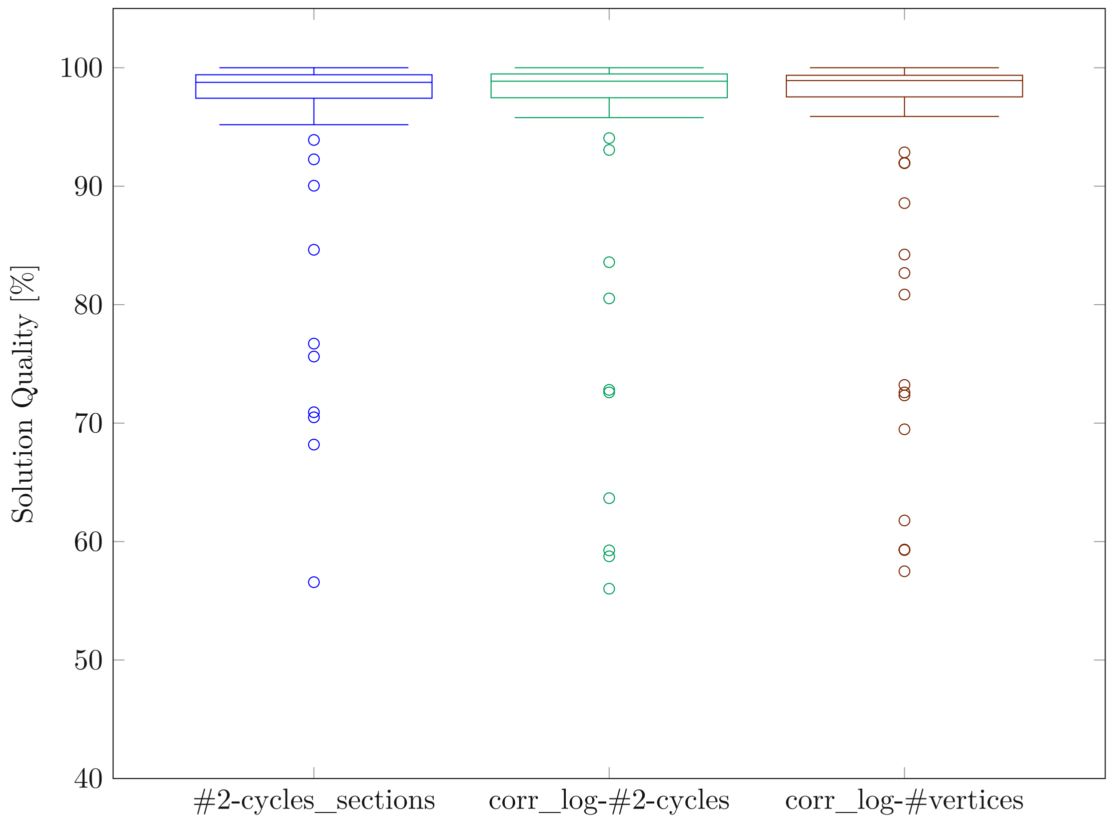


Figure 5.17: Box plots of solution qualities in % of pace-public instances produced by various selection strategies for the degree of destruction in the enlarge-DAG neighborhood with dynamic selection of the MILP formulation. All procedures use Gurobi as MILP solver in the repair operator.

strategies for the dynamic selection of the degree of destruction: #2-cycles_sections, corr_log-#2-cycles, and corr_log-#vertices. The results are illustrated by the box plots in Figure 5.17, which show that again all approaches achieve solution qualities above 55% for all pace-public instances. In general, the plots indicate a similar performance of all three approaches. In comparison to the corresponding variants that always use the CEC-based formulation, there are also barely any differences noticeable. However, there seem to be improvements regarding the variance when compared with the MTZ-based approaches.

Again, we examine further metrics such as the arithmetic and geometric means of the solution qualities and the number of optimal solutions that are found. Table 5.18 lists the corresponding values for the three variants under test. This data confirms that all three variants with the dynamic selection of the MILP formulation outperform all approaches with the preselected MTZ-based formulation, where the best achieved mean value is 95.07%. Similar to the previous tests with a fixed MILP formulation, the combination

Table 5.18: The arithmetic and geometric means of the solution quality of the pace-public instances as well as the number of found optimal solutions of the enlarge-DAG neighborhood with dynamic selection of the MILP formulation and various selection strategies for the degree of destruction. The mean values are given as percentage of the best known solution values. The MILP models are all solved with Gurobi with warm starts.

Selection Strategy	Average Solution Quality [%]		Optimal Solutions
	Arithmetic Mean	Geometric Mean	
#2-cycles_sections	96.55	96.21	15
corr_log-#2-cycles	96.15	95.63	14
corr_log-#vertices	95.39	94.80	9

with the #2-cycles_sections strategy shows the best performance out of the three with a mean solution quality of 96.55% and 15 optimal solutions. Regarding the average solution quality, this is even the best result achieved so far, which validates the application of the dynamic selection of the MILP formulation. The dynamic MILP selection together with the corr_log-#2-cycles strategy also performs slightly better than the CEC-based variant (96.15% vs. 96.13%), whereas the combination with the corr_log-#vertices strategy is slightly worse (95.39% vs. 95.42%). The latter is even outperformed by the best approach with a fixed degree of destruction and a preselected MILP formulation, which is the CEC-based fixed_degree(75) strategy with a mean value of 95.47%.

As these tests conclude the parameter tuning of our solving approach, we shortly take a look at the overall results. Figure 5.18 illustrates the performance of the main variants of our hybrid LNS, which differ in the selection strategy for the degree of destruction and the employed MILP formulation. The empirical cumulative distribution function plot depicts the cumulative solution quality gap over all pace-public instances for each approach. The more to the left and the higher a line is drawn, the better is the corresponding approach as this means that it has an overall small solution quality gap. For example, the variant using MILP_triple and CEC, shown in yellow, finds the most optimal solutions as illustrated by the 0% gap at the beginning, but it has the biggest cumulative gap in total as it ends the farthest to the right with a value of about 45%. In contrast, the approaches based on the #2-cycles_sections and corr_log-#2-cycles strategies and employing CEC or the dynamic MILP selection do not start as well but still have a cumulative gap of less than 5% for more than 90 instances.

The top five variants in terms of average solution quality are also summarized in Table 5.19. This overview illustrates that all top performing approaches benefit from the application of a dynamic selection strategy for the degree of destruction as no variant using completely random selection or a fixed degree of destruction shows up. Besides, approaches with the additional dynamic selection of the MILP formulation are represented twice, with one of them having the best performance. All other variants use the CEC-based formulation, which confirms once again that this formulation is superior to the MTZ-based one

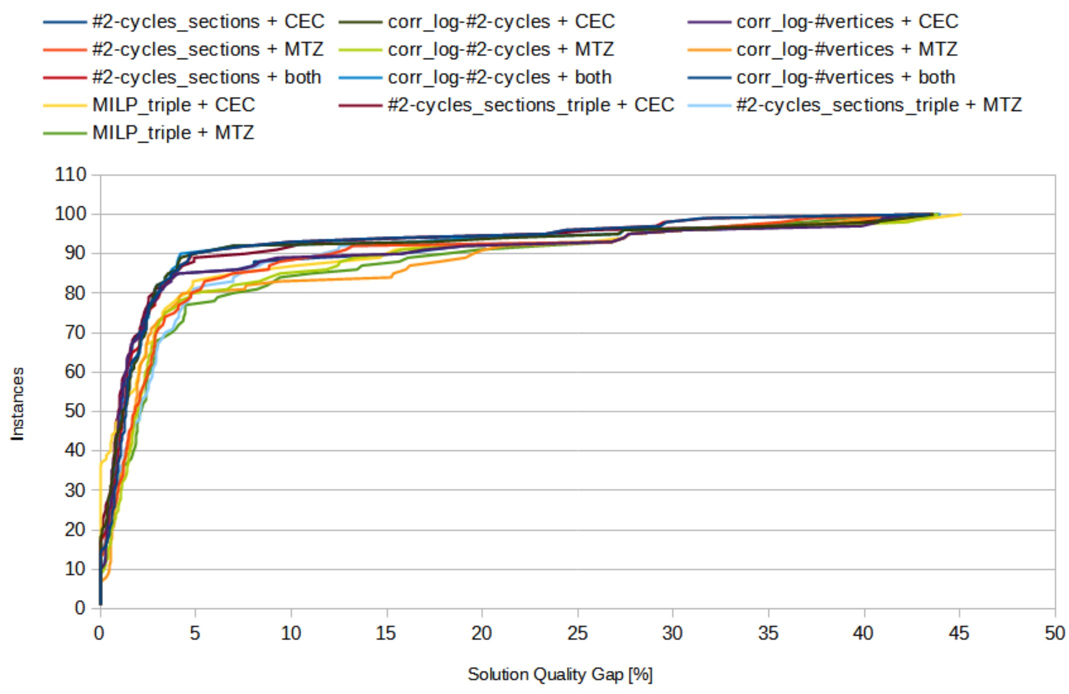


Figure 5.18: Empirical cumulative distribution function plot comparing the solution quality gap as percentage over all instances in the pace-public set for all main variants of the hybrid LNS. The variants differ in the selection strategy for the degree of destruction and the employed MILP formulation, where *both* denotes the dynamic selection of the formulation. The solution quality gap describes the difference between 100%, which represents the best known solution, and the achieved solution quality.

considering the application within our DFVS problem solving procedure.

5.6.4 Testing the Generalizability of Tuned Parameters

All previously described experiments and the parameter tuning concerning the degree of destruction and MILP formulation are done on the instances of the pace-public data set. In order to test the generalizability of our selection strategies to other DFVS problem instances, we apply all resulting variants of the hybrid metaheuristic to the pace-private data set. In contrast to the pace-public instances, we do not have access to the solutions from the PACE 2022 challenge for this data set, so our analysis is only based on the results produced by our own solving procedures.

An overview of the performance of all tested approaches is given by the box plots in Figure 5.19. The plots illustrate that all obtained solutions have a quality of at least 70%, with most of them achieving even more than 75%. However, this increased quality compared to the pace-public instances is probably due to the fact that many of the optimal

Table 5.19: The top five solving approaches for the pace-public instances together with their arithmetic and geometric means of solution qualities as well as the number of found optimal solutions. The mean values are given as percentage of the best known solution values. Each solving approach is defined by the applied selection strategy for the degree of destruction and the employed MILP formulation, where *both* denotes the dynamic selection of the formulation. The MILP models are all solved with Gurobi with warm starts.

Selection Strategy	Formulation	Average Solution Quality [%]		Optimal Solutions
		Arithmetic Mean	Geometric Mean	
#2-cycles_sections	both	96.55	96.21	15
#2-cycles_sections	CEC	96.49	96.15	12
#2-cycles_sections_triple	CEC	96.32	95.84	17
corr_log-#2-cycles	both	96.15	95.63	14
corr_log-#2-cycles	CEC	96.13	95.62	15

solutions for pace-public are derived from the challenge and not our own solutions, which is not the case here. Independent thereof, we observe that the MTZ-based approaches mostly exhibit a notably worse performance than the others in terms of higher variances and lower medians. The only exception is the combination with the MILP_triple selection strategy, for which the difference compared to the CEC-based formulation is not as strong. The reason for this might be that this selection strategy generally seems to perform worse than the others, irrespective of the employed MILP formulation. Regarding the variants with the dynamic selection of the MILP formulations, their achieved solution qualities appear to be similar to the results obtained solely with CEC models. This conforms to our observations for the pace-public instances.

Another illustration of all the variants' performances is given by the empirical cumulative distribution function plot in Figure 5.20, which depicts the cumulative solution quality gaps over all pace-private instances. As observed in the box plots, the MTZ-based approaches show in general a worse overall performance as they result in higher cumulative gaps. For example, the combination of MTZ with the corr_log-#vertices strategy, drawn in orange, results in the highest value of about 28%. Similar to the results for the pace-public data set, the variants employing the #2-cycles_sections strategy together with the CEC-based formulation and the dynamic MILP selection seem to perform well as they solve more than 95 instances with a cumulative gap of less than 5%.

For further insights, we again examine the arithmetic and geometric means of the solution qualities and the number of found optimal solutions, which are shown in Table 5.20. These metrics confirm our observations and also match the results obtained for the pace-public data set for the most part. The mean values verify that the approaches using the MTZ-based formulation have a worse average performance than the others, with the exception of the MILP_triple combination where MTZ slightly outperforms CEC with 96.62% versus 96.51%. However, it is also true that the MILP_triple strategy achieves the lowest average solution quality of all CEC-based approaches, although it finds the

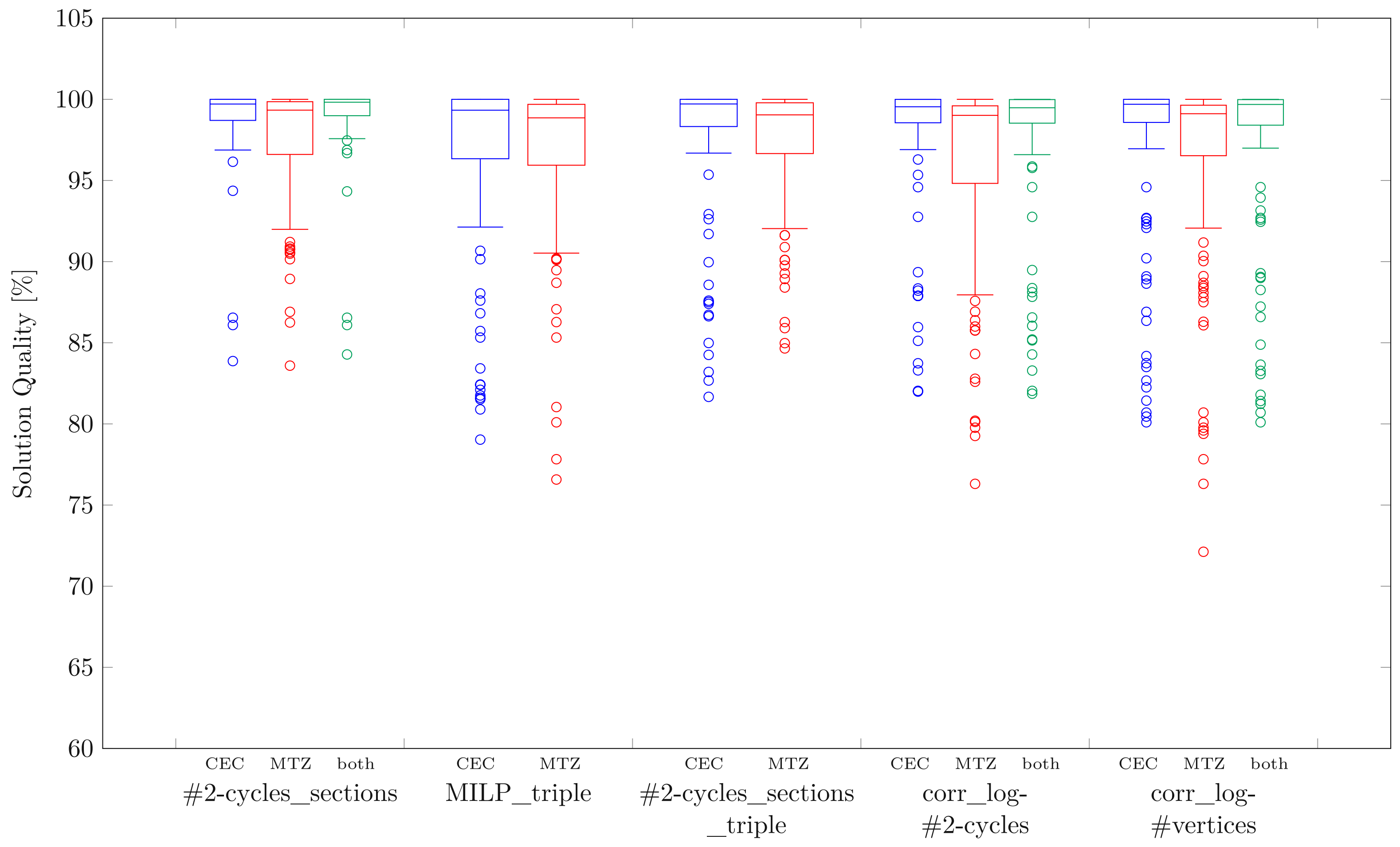


Figure 5.19: Box plots of solution qualities in % of pace-private instances produced by various selection strategies for the degree of destruction in the enlarge-DAG neighborhood. The employed MILP formulation is either CEC, MTZ, or dynamically chosen (both). All procedures use Gurobi with warm starts in the repair operator.

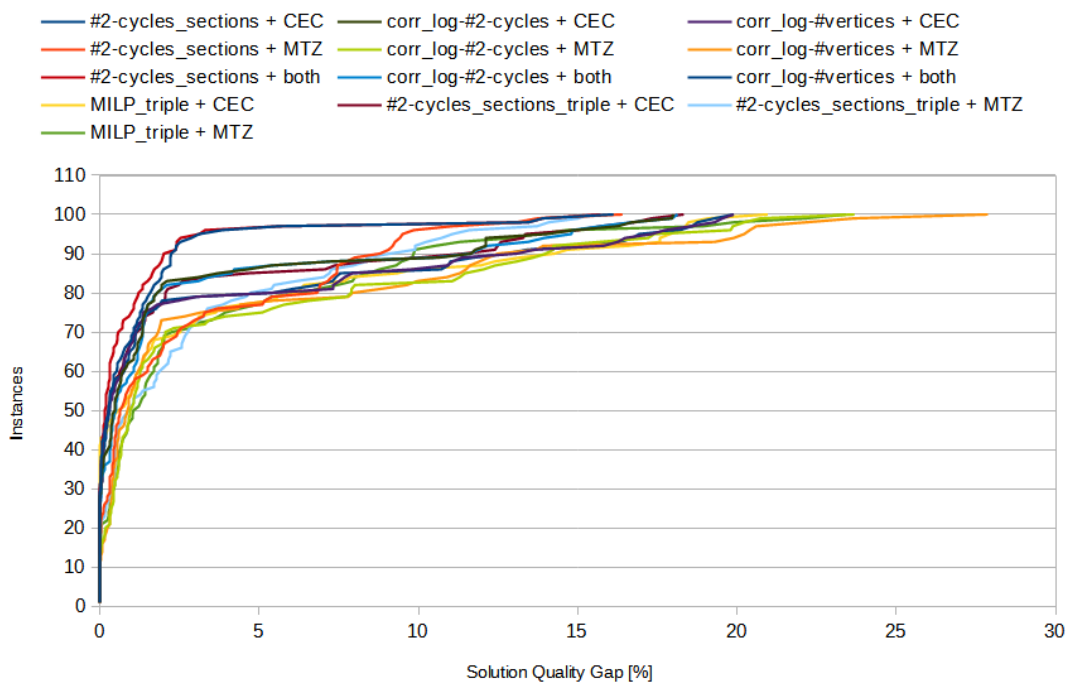


Figure 5.20: Empirical cumulative distribution function plot comparing the solution quality gap as percentage over all instances in the pace-private set for all main variants of the hybrid LNS. The variants differ in the selection strategy for the degree of destruction and the employed MILP formulation, where *both* denotes the dynamic selection of the formulation. The solution quality gap describes the difference between 100%, which represents the best known solution, and the achieved solution quality.

most optimal solutions. In terms of mean values, the performance of the three variants employing the dynamic selection of the MILP formulation is similar to the CEC-based ones, with two being slightly better (#2-cycles_sections, corr_log-#vertices) and one slightly worse (corr_log-#2-cycles). The dynamic MILP selection together with the #2-cycles_sections strategy even obtains the highest value of 99.00%. This selection strategy for the degree of destruction also shows the best average performance for the CEC-based approaches and achieves the overall second highest mean with 98.86%. These results also correspond to the ranking observed for the pace-public instances. Besides, like before, the strategies corr_log-#2-cycles and #2-cycles_sections_triple also perform well when using the CEC-based formulation or dynamic MILP selection. Overall, we see similar results for the pace-public and pace-private data set, which confirms the generalizability of our selection strategies.

Table 5.20: The arithmetic and geometric means of the solution quality of the pace-private instances as well as the number of found optimal solutions of the enlarge-DAG neighborhood with fixed and also dynamic selection of the MILP formulation and various selection strategies for the degree of destruction. The mean values are given as percentage of the best known solution values. For each selection strategy, we differentiate between the MILP formulations that are employed in the repair operator, which are either CEC, MTZ, or the dynamic selection of the formulation (both). The MILP models are all solved with Gurobi with warm starts.

Selection Strategy	Formulation	Average Solution Quality [%]		Optimal Solutions
		Arithmetic Mean	Geometric Mean	
#2-cycles_sections	CEC	98.86	98.83	26
	MTZ	97.33	97.25	17
	both	**99.00**	**98.96**	26
MILP_triple	CEC	96.51	96.33	**39**
	MTZ	96.62	96.48	15
#2-cycles_sections_triple	CEC	97.60	97.48	27
	MTZ	97.18	97.10	17
corr_log-#2-cycles	CEC	97.65	97.53	25
	MTZ	96.03	95.83	12
	both	97.56	97.44	21
corr_log-#vertices	CEC	96.94	96.77	27
	MTZ	96.05	95.81	10
	both	96.96	96.79	20

5.7 Comparison With State-Of-The-Art Solving Approaches

To conclude our computational study, we do a comparison of our proposed hybrid metaheuristics with two state-of-the-art solving approaches for the DFVS problem, which are SA-FVSP by Galinier et al. [GLB13] and SA-FVSP-NNS by Tang et al. [TFZ17] (cf. Chapter 2.1). Similar to our procedures, both approaches are based on local search and apply a metaheuristic but they use simulated annealing instead of large neighborhood search. The SA-FVSP algorithm approaches the DFVS problem by solving the complement problem of finding a vertex set of maximum cardinality whose induced subgraph is acyclic. Thus, a solution can be represented by one of its topological orderings. The employed neighborhood structure is based on the move of inserting a vertex that is not part of the current DAG at a specific position into the topological ordering and then removing all adjacent vertices that are now involved in backward arcs. The SA-FVSP-NNS extends this solving approach with nonuniform neighborhood sampling for the selection of the next move.

As we do not have access to the source codes of these metaheuristics and re-implementing both of them is out of the scope of this work, we rely on the test results presented in the work of Tang et al. [TFZ17] for our comparison. The authors re-implemented the SA-FVSP algorithm in C++ and enhanced it with NNS to develop the SA-FVSP-NNS. Both metaheuristics were tested on a ThinkPad T420i with Intel Core i3-2350M CPU, 4GB RAM, and Windows 7 as operating system, and all experiments were conducted on the fsp-data benchmark set. Tang et al. used three different settings and each approach was run 20 times on each instance. For the comparison, we always consider the smallest solution recorded for the configuration with no time restriction but a limit of ten consecutive stages without improvement, where each stage consists of the repeated application of the move operator until a certain number of successful moves is reached. However, we are aware that the different computing environments, programming languages, and settings probably have an impact on the performance of the compared solving approaches.

Regarding the testing of our hybrid metaheuristics, we use the same computing environment and settings as in Chapter 5.6. We analyze the performance of the same eight variants considered in Chapter 5.6.4 with the different selection strategies for the degree of destruction and MILP formulation by applying all of them to each fsp-data instance once. For the evaluation, we again measure the solution qualities as percentages of the optimal solution values. To do so, we identify the best solutions for the fsp-data benchmark set by comparing the results reported by Pardalos et al. [PQR98], Galinier et al. [GLB13], and Tang et al. [TFZ17] and also our own results.

The box plots in Figure 5.21 illustrate the solution qualities obtained by SA-FVSP, SA-FVSP-NNS, and our solving procedures. On the one hand, SA-FVSP-NNS shows a better performance than SA-FVSP. On the other hand, both SA approaches perform notably worse than our hybrid LNS variants. Considering only our procedures, their results in comparison to each other appear to be similar to the ones achieved for the pace-public and pace-private instances. In general, the MTZ-based approaches perform slightly worse than the others in terms of medians and variances but the difference is not as strong as for the other data sets. We again observe similar solution qualities for the approaches using the CEC-based formulation and the ones with dynamic MILP selection but this time, the pure CEC employment has the higher medians. However, all variants except for one achieve solution qualities of at least 80% for all instances, the exception being the combination of MTZ and MILP_triple. This selection strategy for the degree of destruction again shows the worst performance overall, just like it did for the other data sets.

For a better comparison, we also evaluate the arithmetic and geometric means of the solution qualities as well as the number of found optimal solutions, which are presented in Table 5.21. This data confirms that our solving procedures outperform the SA-based ones. The algorithms SA-FVSP and SA-FVSP-NNS achieve average solution qualities of 65.59% and 71.89%, whereas all variants of our hybrid LNS obtain mean values of at least 93%. Independent of the employed strategy for the dynamic selection of the degree of destruction, CEC-based approaches always achieve higher means than MTZ-based ones. The dynamic MILP selection is also superior to the MTZ-based formulation but interestingly, it performs worse than solely using CEC models. This is in contrast to the

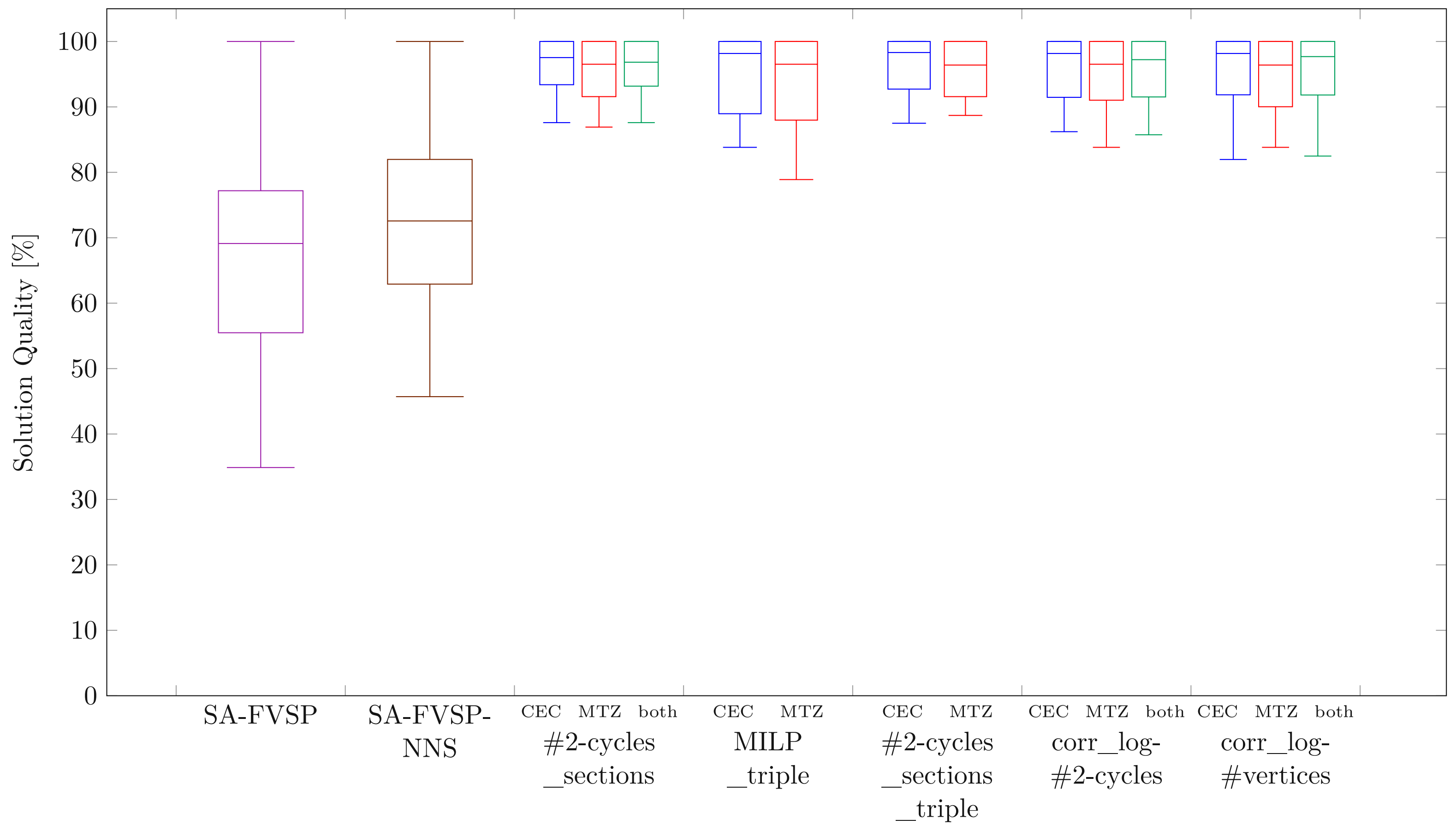


Figure 5.21: Box plots of solution qualities in % of fsp-data instances produced by SA-FVSP, SA-FVSP-NNS, and our approaches with various selection strategies for the degree of destruction in the enlarge-DAG neighborhood. The employed MILP formulation is either CEC, MTZ, or dynamically chosen (both). All of our procedures use Gurobi with warm starts in the repair operator.

Table 5.21: The arithmetic and geometric means of the solution quality of the fsp-data instances as well as the number of found optimal solutions of SA-FVSP, SA-FVSP-NNS and our approaches using the enlarge-DAG neighborhood with fixed and also dynamic selection of the MILP formulation and various selection strategies for the degree of destruction. The mean values are given as percentage of the best known solution values. For each selection strategy, we differentiate between the MILP formulations that are employed in the repair operator, which are either CEC, MTZ, or the dynamic selection of the formulation (both). The MILP models are all solved with Gurobi with warm starts.

Hybrid LNS Algorithms		Average Solution Quality [%]		Optimal Solutions
Selection Strategy	Formulation	Arithmetic Mean	Geometric Mean	
#2-cycles_sections	CEC	**96.44**	**96.37**	18
	MTZ	95.58	95.49	15
	both	96.09	96.01	15
MILP_triple	CEC	94.93	94.77	**19**
	MTZ	93.65	93.38	15
#2-cycles_sections_triple	CEC	96.32	96.23	**19**
	MTZ	95.62	95.54	15
corr_log-#2-cycles	CEC	95.82	95.71	18
	MTZ	94.94	94.80	15
	both	95.68	95.57	15
corr_log-#vertices	CEC	95.71	95.58	18
	MTZ	94.68	94.53	15
	both	95.55	95.42	15
SA Algorithms				
SA-FVSP		65.59	63.24	1
SA-FVSP-NNS		71.89	70.40	1

results that we obtained for the two pace data sets where the dynamic MILP selection improves the average solution quality for two of the three enhanced variants. These results indicate that our strategy for the dynamic selection of the MILP formulation does not work as well on the fsp-data instances and that there is still room for improvement. This might be due to the smaller graph sizes in fsp-data compared to the pace-public and pace-private instances.

However, our selection strategies for the degree of destruction seem to transfer quite well to this data set as the #2-cycles_sections strategy again achieves the highest average solution quality with a value of 96.44% in combination with the CEC-based formulation. The strategy #2-cycles_sections_triple together with CEC performs only slightly worse, with a mean value of 96.32%, and corr_log-#2-cycles is still the third best selection strategy. As expected, MILP_triple obtains the lowest average solution qualities, although it again finds the most optimal solutions when combined with CEC models. For this data set though, the number of 19 optimal solutions is matched by the combination of #2-cycles_sections_triple and CEC.

CHAPTER 6

Conclusion and Future Work

In this work, we designed and implemented hybrid metaheuristics for solving the unweighted DFVS problem. We used large neighborhood search as the metaheuristic framework and enhanced it by solving a mixed integer linear programming model of the DFVS subproblem in the repair operator, which led to the hybridization. We examined different approaches to the various components including graph reduction rules, a construction heuristic, a local search, a neighborhood structure, and a MILP formulation. Furthermore, multiple variants of the base metaheuristic procedure resulted from different selection strategies for the degree of destruction used in the destroy method and for the MILP formulation that is employed in the repair method.

For the graph reduction procedure, we make use of the five operations suggested by Levy and Low [LL88] and an additional rule based on strongly connected components and inspired by Park and Akers [PA92]. After the reductions, the remaining graph is partitioned into its SCCs and the resulting DFVS subproblems are handled separately. To generate an initial solution, we propose a construction heuristic based on vertex degrees and topological orderings. This solution is then improved by running a local search procedure with a one-flip neighborhood structure where one vertex from the current DFVS is moved to the corresponding DAG. Besides, we employ two MILP formulations to encode the DFVS problem: CEC, which involves cycle elimination constraints, and MTZ, which is based on the subtour elimination constraints from Miller, Tucker, and Zemlin.

For the LNS, we introduce two neighborhood structures called enlarge-DAG and enlarge-(partial-)DFVS, which are based on either moving vertices from the current solution to the corresponding DAG or moving vertices from the current DAG to the corresponding solution. We propose multiple strategies for selecting the degree of destruction including random selection, the usage of a fixed value, and dynamic selection based on various graph properties or the MILP formulation. We employ tournament selection for choosing elements in the destroy operator and only accept improving solutions. The LNS terminates when either a time limit or a specified number of consecutive unsuccessful iterations is reached.

Our graph reduction procedure proved to be successfully applicable to more than 75% of all tested instances and achieved reductions of up to 100%. Even though some graphs could not be reduced at all and there is still room for improvement, the employed reduction rules are generally efficient enough to be exhaustively applied also to the largest graphs we considered as the average runtime is less than a second. In about 57% of the cases, the reduction procedure was also able to already identify vertices that belong to a directed feedback vertex set of minimum cardinality.

Regarding our first research question (RQ1), the MTZ-based formulation performed best in terms of average solution quality when using pure MILP optimization and Gurobi as the employed solver, whereas the CEC-based formulation found more optimal solutions within our runtime limits. Considering the standalone MILP solving procedure with warm starts, CEC was superior to MTZ regarding the average solution quality as well as the number of obtained optimal solutions. Similarly, CEC models outperform MTZ-based ones within our metaheuristic framework.

The answer to RQ2 is that the enlarge-DAG neighborhood combined with the CEC-based formulation achieves better mean solution qualities than the enlarge-partial-DFVS neighborhood when having a time limit. Independent of the employed neighborhood structure, the hybrid LNS variants result on average in higher solution qualities than the standalone MILP solving procedures. In terms of found optimal solutions, the approach using enlarge-partial-DFVS together with MTZ models slightly outperformed the other neighborhood structure.

For RQ3, we focus on the experiments conducted on the pace-public set of benchmark instances. When considering the average solution quality and regarding the selection strategies for the degree of destruction, our dynamic selection strategies outperformed the fixed_degree(x) strategy, which in turn achieved better results than the random selection. To be precise, the #2-cycles_sections strategy resulted in an arithmetic mean of 96.49%, whereas fixed_degree(75) achieved 95.47% and random selection led to 95.42%, all of them in combination with the enlarge-DAG neighborhood and the CEC-based formulation. In terms of the number of found optimal solutions, fixed_degree(100000) together with CEC is the best approach with 35 optimal solutions, just ahead of MILP_triple combined with CEC with 32 and by far better than random selection with seven or six optimal solutions depending on the neighborhood. Enhancing the variants based on dynamic selection of the degree of destruction with the additional dynamic selection of the MILP formulation led to further improvements. Overall, the combination of the #2-cycles_sections strategy and the dynamic MILP selection achieved the highest average solution quality of 96.55%, although it finds only 15 optimal solutions.

We also tested the generalizability of our proposed selection strategies by applying the resulting variants of the hybrid LNS to the pace-private data set. These experiments led to similar results as for the pace-public instances, which confirms that our observations generalize. Furthermore, we compared our hybrid metaheuristics to two state-of-the-art solving procedures for the DFVS problem by Galinier et al. [GLB13] and Tang et al. [TFZ17], which are both based on simulated annealing. Like in the literature, we used the fsp-data benchmark set and observed that our solving approaches outperformed both of the other procedures.

Our dynamic selection strategy for the MILP formulation did not work as well on the fsp-data instances as it did for the pace-public and pace-private data sets, which indicates that there is still room for improvement. This might be due to the smaller graph sizes in fsp-data compared to the pace instances, for which the selection criterion might have to be adjusted. Such an adapted strategy could then also be employed within our (extended) standalone MILP solving procedures to further evaluate its applicability and performance. Regarding the selection strategies for the degree of destruction, it would be interesting to also examine values below 25 when using the MTZ-based formulation since fixed_degree(25) achieved the best average solution quality for this selection approach and MILP formulation. Furthermore, it might be beneficial to then consider potential new findings for the dynamic selection strategies and adjust them accordingly.

Future work may also investigate machine learning approaches for the decision making within the hybrid metaheuristic. This could be used, for example, for the element selection or the selection of the degree of destruction in the destroy operator. The applied learning technique and machine learning model would be crucial aspects to be determined.

Another interesting approach could be to incorporate a transformation of the DFVS problem into another combinatorial optimization and graph problem into the metaheuristic framework, as seen for the best performing solvers of the PACE 2022 challenge. Among other things, they used transformations to the vertex cover and set cover problems. It might be possible to utilize such an approach for the construction heuristic or within the repair operator of our LNS procedures.

Furthermore, the results of the PACE 2022 challenge seem to confirm our idea that extending our graph reduction procedure would be beneficial for the overall performance of our solving approach. An enhancement with the three operations PIE, CORE, and DOME as introduced by Lin and Jou [LJ00] as well as the Shortcut rule by Fleischer et al. [FWY09] seems reasonable.

List of Figures

List of Tables

List of Algorithms

Bibliography

[BBCO$^+$22] Gabriel Bathie, Gaétan Berthe, Yoann Coudert-Osmont, David Desobry, Amadeus Reinald, and Mathis Rocton. DreyFVS. `https://doi.org/10.5281/zenodo.6638217`, 2022. Version 1.0.0.

[BEKS17] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

[BSNA21] Ali Baharev, Hermann Schichl, Arnold Neumaier, and Tobias Achterberg. An exact method for the minimum feedback arc set problem. *ACM Journal of Experimental Algorithmics*, 26:1–28, 2021.

[BT97] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1st edition, 1997.

[BYGNR94] Reuven Bar-Yehuda, Dan Geiger, Joseph Naor, and Ron M. Roth. Approximation algorithms for the vertex feedback set problem with applications to constraint satisfaction and bayesian inference. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '94)*, pages 344–354, 1994.

[CCGP04] Francesco Carrabs, Raffaele Cerulli, Monica Gentili, and Gennaro Parlato. Minimum weighted feedback vertex set on diamonds. *Electronic Notes in Discrete Mathematics*, 17:87–91, 2004.

[CCGP05] Francesco Carrabs, Raffaele Cerulli, Monica Gentili, and Gennaro Parlato. A linear time algorithm for the minimum weighted feedback vertex set on diamonds. *Information Processing Letters*, 94(1):29–35, 2005.

[CHJ06] Xuan Cai, Jingwei Huang, and Guoqiang Jian. Search algorithm for computing minimum feedback vertex set of a directed graph. *Jisuanji Gongcheng/Computer Engineering*, 32(4):67–69, 2006.

[CLL$^+$08] Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *Journal of the ACM*, 55(5):1–19, 2008.

[COPS19] Vincenzo Cutello, Maria Oliva, Mario Pavone, and Rocco A. Scollo. An immune metaheuristics for large instances of the weighted feedback vertex set problem. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1928–1936. IEEE, 2019.

[DHL17] Iain Dunning, Joey Huchette, and Miles Lubin. JuMP: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[DZZ22] YuMing Du, QingYun Zhang, and ShunGen Zhang. pace-2022. `https://doi.org/10.5281/zenodo.6644409`, 2022. Version 1.0.0.

[FLS+18] Daniel Funke, Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Moritz von Looz. Communication-free massively distributed graph generation. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 336–347. IEEE, 2018.

[FWY09] Rudolf Fleischer, Xi Wu, and Liwei Yuan. Experimental study of FPT algorithms for the directed feedback vertex set problem. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009*, volume 5757, pages 611–622. Springer, 2009.

[GAB+20] Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, Gregor Hendel, Christopher Hojny, Thorsten Koch, Pierre Le Bodic, Stephen J. Maher, Frederic Matter, Matthias Miltenberger, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Christine Tawfik, Stefan Vigerske, Fabian Wegscheider, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 7.0. ZIB-Report 20-10, Zuse Institute Berlin, 2020.

[GLB13] Philippe Galinier, Eunice Lemamou, and Mohamed Wassim Bouzidi. Applying local search to the feedback vertex set problem. *Journal of Heuristics*, 19(5):797–818, 2013.

[Gur22] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, pages 85–103. Springer US, 1972.

[LJ00] Hen-Ming Lin and Jing-Yang Jou. On computing the minimum feedback vertex set of a directed graph by contraction operations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(3):295–307, 2000.

[LL88] Hanoch Levy and David W. Low. A contraction algorithm for finding small cycle cutsets. *Journal of Algorithms*, 9(4):470–493, 1988.

[MQR21] Rafael A. Melo, Michell F. Queiroz, and Celso C. Ribeiro. Compact formulations and an iterated local search-based matheuristic for the minimum weighted feedback vertex set problem. *European Journal of Operational Research*, 289(1):75–92, 2021.

[NB14] Havva A. Noughabi and Farzaneh G. Baghbani. An efficient genetic algorithm for the feedback set problems. In *2014 Iranian Conference on Intelligent Systems (ICIS)*, pages 1–4. IEEE, 2014.

[PA92] Sungju Park and Sheldon B. Akers. An efficient method for finding a minimal feedback arc set in directed graphs. In *[Proceedings] 1992 IEEE International Symposium on Circuits and Systems*, volume 4, pages 1863–1866. IEEE, 1992.

[PQR98] Panos M. Pardalos, Tianbing Qian, and Mauricio G. C. Resende. A greedy randomized adaptive search procedure for the feedback vertex set problem. *Journal of Combinatorial Optimization*, 2:399–412, 1998.

[PR10] David Pisinger and Stefan Ropke. Large neighborhood search. In Michel Gendreau, editor, *Handbook of Metaheuristics*, pages 399–419. Springer, 2010.

[RP06] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

[Sha98] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In Michael Maher and Jean-Francois Puget, editors, *Principles and Practice of Constraint Programming — CP98*, pages 417–431. Springer, 1998.

[SS16] Peter Sanders and Christian Schulz. Scalable generation of scale-free graphs. *Information Processing Letters*, 116(7):489 – 491, 2016.

[Swa22] Sylwester Swat. swacisko/pace-2022: First release of DiVerSeS, a solver for the Directed Feedback Vertex Set problem. `https://doi.org/10.5281/zenodo.6759809`, 2022. Version 1.0.2.

[TFZ17] Zhipeng Tang, Qilong Feng, and Ping Zhong. Nonuniform neighborhood sampling based simulated annealing for the directed feedback vertex set problem. *IEEE Access*, 5:12353–12363, 2017.

[Wol20] Laurence A. Wolsey. *Integer Programming, Second Edition.* John Wiley & Sons, Inc., 2020.