Erasmus University Rotterdam

Erasmus School of Economics

Bachelor thesis

BSc2 Econometrics / Economics

Replicating and extending meta heuristics for the capacity separation problem in the CVRP

Author

Fernando Lasso Peña
406436

Supervisor

Naut Bulten

Second assessor

Dr. Shadi Sharif Azadeh

July 7, 2019

Abstract

An important step in approaching optimal solutions for the Capacitated Vehicle Routing Problem (CVRP) is finding a strong lower bound. The tabu search method provided by Augerat et al. (1998) was presented to be powerful heuristic when applied to solve the capacity separation problem in the cutting plane algorithm which iteratively solves a relaxed sub problem of the complete formulation of the CVRP. Our research verifies their computational findings can be replicated to a strong degree and confirms the methods presented in their paper. Furthermore, we propose another meta heuristics to tackle the capacity separation problem based on the Non-dominated Sorting Genetic Algorithm (NSGA). Although the algorithm does not turn out to outperform previous heuristics, we do find that when combined with the constructive heuristic, it matches the performance of the tabu heuristics requiring a smaller number of cuts. Finally, we change the cutting plane algorithm from an iterative solver to a dynamic variant in which the cut graphs of candidate solutions are passed to the separation phase through callback functions. This alternative approach leads improved lower bounds at a lower computation cost.

Keywords: Vehicle Routing Problem · Capacity Separation Problem · Tabu search · Non-dominated Sorting Genetic Algorithm · Cutting Plane Algorithm ·Callback · Linear Programming

Contents

1 Introduction

With around 750'000 results on a quick search of the Vehicle Routing Problem (VRP) on Google Scholar, this class of combinatorial optimization puzzles is likely one of the most researched topics in operations research. At its core, the objective is to find the optimal edges to be traversed by a fleet of vehicles to serve a given set of customer nodes at a certain distance from the central depot station from which every vehicle departs and returns. Countless extensions to this problem have been investigated by modifying the existing assumptions of the model or adding further constraints to account for. The real-life application only seems to have increased in relevance over the past years with the introduction of new vehicle types and rising demand for optimization in business operations. This thesis will focus on the VRP taking into consideration capacity constraints of vehicles with respect to the aggregate demand of the customers they serve. The aim of this research will be to take a step back and investigate one of the earlier works around this topic by Augerat et al. (1998), who introduced the first meta-heuristic to solve the identification problem that is at the heart of the Capacitated Vehicle Routing Problem (CVRP). First, by replication of the methods described in their paper, our numerical results will be contrasted to their findings. It will be of interest to observe how the much stronger processors of regular office computers today perform in comparison to the ones twenty years ago on the same problem instances and how their methods have aged in comparison to alternative solution methods. One such alternative will be investigated in our extension of their work. With increased interest in the possibility of artificial intelligence and its sub-fields, one of the meta heuristics that has seen leaps in development is the genetic algorithm which is part of the larger class of evolutionary algorithms. Its application is especially powerful in multi objective problem optimization, an area that has lon g caused experts headaches. Our research will attempt to tackle the capacity separation problem which is at the core of searching lower bounds for optimal objective values in the CVRP by applying this technique. We find that although the tabu heuristics introduced by Augerat et al. (1998) are very difficult to beat, the genetic algorithm is able to find deeper cuts into the solution space and achieves good lower bounds to the CVRP with a smaller number of constraints when combined with the constructive heuristic.

Another development in technology which can be taken advantage of by our research is the functionality of callbacks in the Cplex solver. This leads to our development of a second extension which modifies the traditional cutting plane algorithm. By not requiring the solver to find the optimal solution at each iteration for the given cuts, but searching for violated constraints in the cut graph of candidate solutions, we find that the solving times can be drastically decreased while finding even better lower bounds with all heuristics on most problem instances. The callback function proves that by not requiring to solve each sub problem to optimality at every iteration, we can save time in the Cplex solver whilst finding even more useful cuts in the solution space.

The thesis proposal will be structured as follows. After this introduction, the theoretical framework will be laid out to describe the landscape of vehicle routing problems since the writings of Augerat et al. (1998) and create context for the extension that we propose. Then, a discussion on the social relevance of the CVRP and our suggested extensions will be held in the social relevance

section. Afterwards, the data and Java Libraries section will describe the source of the data and present relevant tools to apply for this research. In the methodology section, the methods suggested by Augerat et al. (1998) are presented in addition to a detailed description of how our extensions are implemented. In the results sections, we first present our findings of the replication effort and afterwards the computational results of our extensions. To conclude, we summarize our findings and give recommendations to future research.

2 Theoretical Framework

The theoretical framework will start out by providing a more general overview of the state of research around the VRP to create context for our own findings. Then, existing extensions related to the VRP and the class of evolutionary algorithms we propose are described. Finally, we take a deeper dive into the theory surrounding the capacity separation problem.

2.1 The vehicle routing problem

The central focus of this thesis will be on the research by Augerat et al. (1998). However, to understand the theoretical framework and context of their paper, earlier work that they built on must be considered before continuing to examine how further research has built upon their findings. What is referred to today as the vehicle routing problem was first introduced to academic literature under the name 'The Truck Dispatching Problem' by Dantzig & Ramser (1959) with a fleet of gasoline delivery trucks dispatched from a bulk terminal needed to supply a large number of service stations at minimal cost. The goal is to design a set of routes for the fleet such that all customer demand is covered and sum of all customer demand on a route does not exceed the vehicle capacity at the smallest possible transportation cost. The problem can be sketched as an undirected graph with vertices representing the customer and depot nodes and edges connecting pairs of vertices. Each edge carries a cost when activated for a vehicle route which is usually determined to be a function of Euclidean distance between vertex coordinates.

In Laporte et al. (1985) , the authors define the variables in the CVRP as follows. For a set of edges E, each edge $e \in E$ can be activated through decision variable x_e at a cost c_e. An activated edge means that a vehicle passes between the vertices at the endpoints of this edge. For the set of vertices $S \in V$, the $\delta(S)$ defines the cut set between nodes v in S where S is a subset of V, meaning the set of edges with one endpoint in S and the other in $V \setminus S$. ($\delta(S) = \{e = (i,j) \in E : i \in S, \in V \backslash \mathrm{S}\}$).

Furthermore, we also define $\gamma(S)$ as the set of edges with both endpoints in S . Also, k and C denote the number of vehicles and their capacity respectively, $d(S)$ equals the sum of the demand of vertices in S and for any subset F of E, $x(F)$ gives the sum of x_e for all e in F. Finally, we define vertex 0 to be the depot node and V_0 to be the set $V \setminus \{0\}$. These definitions give a sufficient framework to formulate the CVRP in the following mathematical notation.

$$\text{minimize} \sum_{e \in E} c_e x_e \tag{1}$$

Subject to

$$x(\delta(\{0\}) = 2k, \tag{2}$$

$$x(\delta(\{i\}) = 2 \text{ for all } i \in V_0 \tag{3}$$

$$x(\delta(S) \geq 2\lceil \frac{d(S)}{C} \rceil \text{ for all } S \subseteq V_0, S \neq \emptyset \tag{4}$$

$$0 \leq x_e \leq 1 \text{ for all } e \in \gamma(V_0) \tag{5}$$

$$0 \leq x_e \leq 2 \text{ for all } e \in \delta(\{0\}) \tag{6}$$

$$x_e \text{integer for all } e \in E \tag{7}$$

Note that $\lceil \alpha \rceil$ is equal to the nearest integer greater or equal to α. The objective (1) minimizes the cost of transportation by summing the costs of activated edges. Constraint (2) sets the degree of the depot node equal to twice the number of vehicles in the system, given that each vehicle must leave and enter the depot. Similarly, the set of constraints (3) requires each customer node to have degree 2, since it must only be visited and left by a vehicle once. All constraints (4) have the double role of both ensuring a sufficient number of vehicles are present to serve a subset of customer nodes and the elimination of sub tours not including the depot. These constraints will also be referred to as the capacity constraints. Finally, constraints (5) to (7) define the range of our decision variables, since each edge logically should be integer and can only be traversed once with the exception of corner case (6) where a single customer is visited by a vehicle after which it returns to the depot through the same edge.

The culprit of the complexity of our model is the set of constraints (4), since it must hold for all possible subsets S of V_0. An exact formulation of the problem would require $2^{|V_0|} - 1$ distinct constraints to be accounted for, a task quickly exceeding the capacity of any computer and giving our problem's non-deterministic polynomial-time hardness (NP-Hard). Note that the VRP is a generalization of the well-known Traveling Salesman Problem (TSP) since the TSP is a specific instance of the VRP in which k is set to 1 and $C \geq d(V)$.

2.2 Extensions to the VRP

Uncountable extensions to the model have been developed and researched since the emergence of the Truck Dispatching Problem. Some of its more popular versions and solving methods are presented in Kumar & Panneerselvam (2012) where multiple flavors are described. There is the Vehicle Routing Problem with Back hauls (VRPB) in which customers can return some goods. The vehicles must take into account that after delivery, a pre-determined amount of inventory must be picked up from customer nodes. Then, the VRP can be modified to take into account time windows (VRPTW)

associated with each customer node which define an interval of time in which the node must be served. Furthermore, there is the case of multiple depots as investigated by Cordeau et al. (1998) in which vehicles can depart and return from the same depot but with multiple depots to assign vehicles and customers to. All of these extensions can be mixed to create further complications and attempt to solve real life problems to optimality. One such example is the vehicle routing with back hauls and time windows problem (VRPBTW) researched by Thangiah et al. (1996). In terms of solution techniques, a large pallet of possibilities has already been explored. There are a few classes of solution techniques to be considered. The first are the exact approaches, in which the full solution space is explored through methods such as branch and bound, in which a state space search systematically enumerates candidate solutions. These types of methods are considered impractical since the computation time becomes excessive for larger problem instances and their performance can be beat by simple heuristics in terms of computational speed for solutions of sufficient quality. This conclusion was already made in the early days of VRP research by authors such as Christofides & Eilon (1969) The alternative solution method described in their work is categorized as a heuristic solving technique, which explores a relatively small solution space but generally delivers strong results with much less effort. The savings Algorithm presented by Clarke & Wright (1964) is an example of a Constructive method, which builds up a solution gradually from a starting point. Then, Beasley (1983) explores a clustering and routing approach in which the problem is decomposed into a component of node clustering and one of route construction, also known as a two-phase algorithm. Finally, in recent years we have seen a large increase in the use of clever meta heuristics. Whereas Augerat et al. (1998) build on earlier tabu search methods for route modifications as presented in Gendreau et al. (1994), other methods which appear to obtain their inspiration from nature have emerged. One example of such solution techniques is present by Bullnheimer et al. (1999) who investigate Ant Colony Optimization which is a subclass of Swarm intelligence to analyse the collective behavior of decentralized systems. In the CVRP, this meta heuristic constructs solutions by combining an adaptive memory with a local heuristic function by having multiple artificial ants perform local searches by moving in many possible directions in the solution space. Another branch of nature that has been taken advantage of is biological evolution. Genetic algorithms such as the Non-dominated Sorting Genetic Algorithm (NSGA) are applied to solve multi-objective VRP problems in which additional objectives to the minimization of route length are explored. The NSGA is designed to find a strong set of solutions by evolving a population of candidate solutions. In an iterative process, candidates along the Pareto Front are crosses and manipulated into next generations while poor performers are eliminated. One example of an application of NSGA can be found in the research by Jozefowiez et al. (2005) who also attempt to quantified and optimize a balance between routes. The algorithm of interest for this research is the third version of the NSGA framework as presented in Kalyanmoy & Jain (2014). The framework introduces a reference-point-based many-objective evolutionary algorithm which builds on previous NSGA algorithms and focuses on population members that are non-dominated, yet close to a set of supplied reference points.

2.3 The capacity separation problem

The point of the capacity separation problem is to identify sets S for which the capacity constraint is violated in order to add cuts to the model which reduce the solution space and the lower bound given by the relaxed problem as much as possible. Since we do not have extensive knowledge of what the solution space, also called the polytope, looks like, our best guess to make large cuts is to find those sets that cause the largest violation in the current optimal solution to the relaxed sub problem. However, selecting sets for which the capacity constraint is violated is not an easy task. In our research, we will attempt to explore further alternatives to the meta heuristics Augerat et al. (1998) have presented to find relevant capacity constraints to improve the lower bound of the optimal object value. Since the publication of their paper, Denis Naddef & Rinaldi (2002) have proven that the separation problem is NP-Hard. This means that it is virtually impossible to guarantee a polynomial-time algorithm which identifies a constraint with certainty if it exists. At best, we can use heuristics which have proven to produce strong lower bounds with minimal gaps within reasonable computation time. The separation problem is at the core of the research by Augerat et al. (1998). In some way, the capacity separation problem is an instance of the Bin Packaging Problem (BPP), in which packages with weight equal to the demand of customers in a number of sets equal to the number of vehicles must be assigned to bins equal to the size of the vehicle capacity. By relaxing the vehicle capacity constraint to hold infinite capacity with a single vehicle, we generalize to the TSP while relaxing edge costs to zero returns the (BPP) decision problem. With these two traditionally challenging models at heart, it is no surprise the CVRP is difficult to solve. Toth & Vigo (2015) provide data sets for the VRP proven to be solved to optimality with the number of nodes just approaching a thousand while meta-heuristics by Arnold (2019) finds good solutions for real-life data sets with up to 30,000 customers. In comparison, Applegate et al. (2009) find an optimal tour for the TSP containing close to one hundred thousand cities.

As pointed out by Ralphs (2003), the BPP and TSP are often at odds because the cost structure depends on routing choices and cannot simultaneously take the packaging structure into consideration. They tackle this interplay by an approach to separate the TSP from the BPP in order to benefit from the highly optimized techniques to search the TSP solution space. This paper will attempt to approach the separation problem from a different angle by extending existing literature with the use of an evolutionary algorithm. Genetic algorithms have been successful in solving the decision problem for the BPP. (Junkermeier (2015))

To our knowledge no evolutionary algorithms have been applied to the capacity separation problem. There are multiple reasons why applying evolutionary heuristics would be an interesting addition to literature. As pointed out by Augerat et al. (1998), the tabu technique they introduced is prone to identify a large set of similar constraints which may reduce the cut of a newly identified constraint on the solution space and increase the load on the solver which has to consider more constraints in each iteration. The genetic algorithm proposed in this thesis belongs to a class of heuristics which can identify creative solutions. By creative, we mean a set of solutions which are as

dissimilar as possible to avoid the problem of an oversupply of similar constraints. In the capacity separation problem, we expect the genetic algorithm to identify sets that the heuristics provided by Augerat et al. (1998) fail to identify. By providing our findings, we will provide future efforts to solve the capacity separation problem with a reference on the performance of this class of algorithms.

In addition to extending the literature with a new meta heuristic for the capacity separation problem, we also want to present an alternative method for the cutting plane algorithm. Instead of iteratively solving the relaxed problem with the identified type 4 constraints, we propose making use of callback functions which allow us to generate constraints on the fly without requiring the solver to return an optimal solution at each step. In the methodology section, we explain why this will result in more efficient and equally valid outcomes.

3 Social Relevance

From its early emergence as a Truck Dispatching Problem to the more recent applications of Drone Routing and about everything in between, it is easy to recognize the economical relevance of the VRP. The optimization of this class of combinatorial problems is beneficial for business, society and even the environment. By finding the shortest and least costly routes to perform a certain group of tasks, a company can save on transportation costs in its supply chain, serve customers more efficiently, avoid unnecessary loss of time and maximize resource utilization. According to Hasle et al. (2007), given that an estimated 10% of manufacturers operational costs arise from transportation, we can even estimate the transportation cost savings arising from computer optimization of VRP instances to be around 5% of the all operations. Given that already only in Europe about 10% of GDP is made up by the transportation sector, it is clear that route optimization programs in this almost two trillion euro industry have massive impacts on every day life.

From a social perspective, the average person also benefits from route optimization. Think about the sharing economy in which the flow of resources can be optimally utilized through ride sharing platforms (Aiko et al. (2017)) and bike sharing systems (Schuijbroek et al. (2017)). Furthermore, the VRP can be extended to provide frameworks for vehicle routing in home health care applications (Lia et al. (2013)), medical emergency situations (Creput et al. (2011)) or even disaster response operations (Gharib et al. (2018)).

Finally, one should consider the environmental advantages of route optimization. Not only do companies save on their transportation costs, also less fuel is required to support their operations, which would help to reduce the company's CO2 footprint. Some models even aim at minimizing criteria based on quantified environmental factors, as seen in the paper by Ubeda et al. (2011). A recent paper by Soysal et al. (2015) explores how multi-echelon distribution strategies addresses energy usage, congestion, traffic-related air pollution, accidents and noise by using intermediate depots in urban logistics. Also, with rising interest in alternative fuel-powered vehicles, research in Green Vehicle Routing Problems (G-VRP) aids to overcome the constraints that are raised by their typically limited travel range (Erdogan & Miller-Hooks (2012)).

In conclusion, the VRP carries aside from its mathematical challenge an important contribution to society through its usability in a broad range of applications. The aim of this thesis will be to contribute to the field by replicating numerical results from one of the earlier contributors in this field Augerat et al. (1998) and build upon the masses of research written since then by contributing our extensions. Replicating results may not bring new ideas to the table but is an essential part of academic literature aiming to ensure high quality research in the field. By continuing to peer review and verifying proposed methods and results, past research becomes more valuable and trustworthy. Then, the extension provides an interesting addition to the meta heuristics applied by Augerat et al. (1998) by implementing an approach to the VRP from a very different angle. To our knowledge, no evolutionary algorithms have been applied to the capacity separation problem to this point and their implementation could increase the efficiency of such algorithms and provide a framework for searching lower bounds in larger problem instances. Furthermore, our extension to the cutting plane algorithm solving technique may lead existing heuristics to be applied more efficiently and turn out to have further applications in similar fields.

4 Data & Java Libraries

In their paper, Augerat et al. (1998) make use of three sets of problem instances. All data is conveniently stored online on a website which was created by Ivan Xavier and by the authors of Uchoa et al. (2016) and is still maintained by Daniel Oliveira. From their database, we retrieve the LITLIB library which contains all the problem instances from literature referred to by Augerat et al. (1998) in addition to the customized A and B libraries created for their research. For all instances, the node coordinates and demands are given in a standardized TSPLib format and optimal solutions are available. The availability of these optimal solutions provides an interesting new insight since the publication of Augerat et al. (1998), because they have only been proven in a publication by Baldacci et al. (2004). This knowledge will add to the discussion and interpretation of the numerical results since it provides us with a more precise estimation of the distance to optimality of our solutions.

Furthermore, open source frameworks by Anderson (2012) on Object Oriented Cplex implementations for TSP routing optimization using User Cut Callbacks and Lazy Constraint Callbacks will be consulted. For the undirected graph architecture, we made use of the open source *Java Universal Network/Graph Framework* (n.d.). The flow network classes used for identifying minimum cuts in the constructive heuristic is based on the open source code published by Wayne & Sedgewick (2011) released under the GNU General Public License, version 3 (GPLv3). For the implementation of NSGA, this paper will make use of the open source Java Framework for Multi-objective programming. (*Java Framework for Multiobjective Optimization* (2012)) The problem instances will be loaded and solved on a Mac book Air running a 1,7 GHz Intel Core i7 processor with 8 GB 1600 MHz DDR3 memory using Java in an Eclipse IDE.

5 Methodology

In our methodology section, we first touch upon the methods proposed by Augerat et al. (1998) and how these techniques are incorporated into our own research. Then, we elaborate on the workings of the genetic algorithm proposed for our extension. Finally, we describe the details and relevance of the callback functionality for the cutting plane algorithm.

5.1 Methods by Augerat et Al.

The first methods applied in this research will be based on the separation of capacity constraints in the VRP as demonstrated in Augerat et al. (1998). Although all techniques will be addressed in this paper, detailed explanations of the parameter configurations to apply for an exact replication of their research can be found in their paper. We attempt to find the relevant sets for which the corresponding capacity constraints can be added to the relaxed model formulation. In what is known as a cutting plane algorithm, the LP containing the objective and constraints (2) and (3) of the full formulation of the CVRP is iteratively solved and fed with identified sets of valid inequalities. Although the optimal outcome of this algorithm would be a feasible integer solution in which all capacity constraints are satisfied, it is more likely to end when no more valid inequalities can be found. Based on the format of constraints (4), we can formulate $f(S) = x(\delta(S) - 2\lceil \frac{d(S)}{C} \rceil$ which for any $f(S) < 0$ gives a set S corresponding to the capacity constraint which is violated.

At each iteration of the cutting plane algorithm, an undirected support graph can be constructed with the nodes of the problem instance and edges drawn for all edges with corresponding decision variable $x_e > 0$. A cut set $d(S)$ is now defined as the edge cut set in this support graph.

A few simple heuristics are given to be applied for any of the set identification techniques used by Augerat et al. (1998) to easily find further relevant sets. The first is for any set S to check the complement set V_0 minus S for which $f(S)$ can easily be calculated. The second is to check for sets including neighboring nodes for sets which contain demand close to the maximum capacity of the minimum number of vehicles that could service such demand. Furthermore, by means of a shrinking procedure, routes that exceed the maximum capacity for a vehicle can be identified. Finally, they consider the nodes in connected routes that do not contain the depot node.

Augerat et al. (1998) benchmark their tabu search meta heuristic with the constructive and the greedy heuristic. The constructive heuristic adds a sink node to the support graph and finds the minimum cut between the source (depot) node and that sink after setting custom weights to the edges which ensure that the set of nodes on the sink side of the cut give the maximum violation of the capacity constraints. The greedy randomized algorithm starts out with an initial set of nodes in V_0 and iteratively check for a violated constraint and adds a node for which the weight of the cut set between the candidate nodes and the current set is maximum. Multiple strategies for the initial set of nodes to consider are investigated.

The central technique that Augerat et al. (1998) want to present in their work is the tabu search algorithm. It consists of an expansion and an interchange phase which is executed with multiple

starting points and parameters. For each number of vehicles $p \leq k$, the expansion phase iteratively constructs a set of nodes for which to check the capacity constraints. It does this similarly to the neighbor node search heuristic by selecting a node adjacent to the current set. It desires the expansion to just exceed the capacity capability of p vehicles. In the interchange phase, nodes neighboring the node set in the support graph, or those nodes in the node set neighboring the complement set are respectively added or removed from the node set, maximizing the weight of the cut set between these nodes and the node set and complement respectively. At each of iteration of the interchange phase, the last modified node is added to the tabu list and remains unchanged for a given number of iterations. Augerat et al. (1998) have considered a range of parameter inputs to tune the algorithm. Since some of the identification techniques produce a large number of similar violated constraints, some more filtering steps are undertaken to limit the burden on the solver. Only constraints violated by more than 0.01 ($f(S) <$ -0.01) and a number of valid inequalities equal to the minimum of 125 and twice the number of nodes are added at each iteration. If the number of identified sets exceeds that limit, only the constraints with the largest violations are included.

When implementing the heuristics presented by Augerat et al. (1998), there are multiple ways to structure the methodology to identify valid constraints. We classify the five basic functions proposed to be applied for every meta heuristic as simple heuristics. These are the methods that identify node sets not connected to the depot, shrink edges and examine the super node capacities, explore the neighbors and complement of a possible solution and check the components of the cut graph for violations. Our algorithm first checks the components and shrinks the graph after which one of the three meta heuristics is applied. Also, each time a valid constraint is identified, the simple heuristics also examine the neighbors and complements for making valid constraints. Also, we take into account that a set may be dominated by a set including its neighbor in case the larger set requires an extra vehicle to cover its demand, thus only requiring to consider the constraint related to the larger set. As such, the simple heuristics are considered in every iteration and for every identified constraint.

5.2 A genetic meta heuristic for the separation problem

The first extension we propose is an alternative to the meta-heuristics proposed by Augerat et al. (1998) to the separation problem. The tabu search method proposed by them gains its edge from constructing promising sets by expanding the set and interchanging nodes from multiple starting points and with multiple numbers of vehicles as the capacity limit. While effective, this approach is only able to find particular types of sets, namely those clustered together in the current optimal LP solution.

The algorithm used by our extension is a popular version of the genetic algorithm, namely the NSGA This algorithm starts off with an initial population of candidate solutions after which it iteratively kills of some of the members that are dominated by alternatives and evolve members that are close to the Pareto frontier. The frontier represents a population for which none of the objective can be improved without diminishing any other. In the figure below, this process is laid out in the

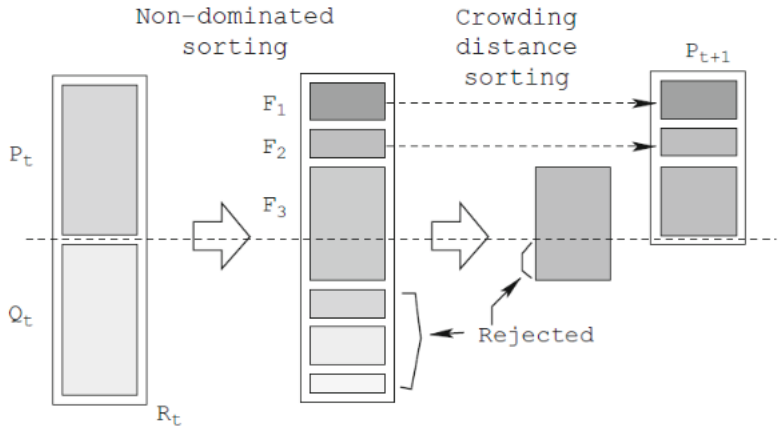

Figure 1: Visualization of the NSGA algorithm Kalyanmoy (2001)

following steps. For a given population P_t, create a total population R_t by creating offspring Q_t from P_t through selection, genetic crossover, and genetic mutation. The total population is sorted on an ascending level of non-domination. A candidate solution is dominated by another if it scores worse on all of its objectives. The ranked population is grouped into front rankings after which the top half will be considered in the next iteration whereas the bottom half will be rejected. Front ranking that are partially rejected will select the nodes to evolve by crowding distance sorting, meaning that solutions which are similar to others are rejected.

In our implementation, the decision variable will be which nodes to include in a set to check the corresponding capacity constraints for.

Although it may seem intuitive to only give the genetic algorithm the objective of finding the largest valid inequality, our proposed heuristic also attempts to consider more objectives. In particular, aside from maximizing the violation, the demand is minimized and cut edge value is maximized. These different objectives all relate to the capacity constraint and appear to create a beneficial interplay in the population development. Single objective settings appear to lead to population convergence in which similar cuts are found. By posing counter acting objectives, the population turns out to remain more diverse and evolves on more complex patterns. Both the algorithm with only a violation objective and one with multiple goals are implemented.

Furthermore, we aid the population development by applying a simple heuristic to each population member. In this simple heuristic, all neighbors of the population member node set in the current cut graph are checked and if the union results in a larger violation of the capacity constraint, it is added to the node set. Since the genetic algorithm by itself is blind to what the cut graph actually looks like, this heuristic provides a simple way to slightly improve the population at each step while still maintaining diversity.

A few parameters are of interest when implementing this algorithm. First of all, the user sets the population size which is maintained for each evaluation. Then, the number of objectives to include

can be tuned between only the violation or all three of them in order they were presented. Furthermore, a parameter is given for the maximum number of evaluation in case the terminating condition is not triggered. In practise, our algorithm is given a large number of maximum evaluations, but is halted when a given number of strongly violated constraints were identified. Finally, our framework offers two versions of the NSGA, namely NSGAII and NSGAIII. The NSGAIII method adds to its predecessor the ability to stick to reference points to maintain population member distance. As such, we will use the NSGAII when only the violation is maximized while for a multi objective problem we turn to NSGAIII.

Based on some variations of the parameters, we recommend to set the population to half the number of nodes of the problem instance, with between 250 and 1000 evaluations. When about ten constraints are identified with a violation larger than half the amount of vehicles in the problem, the algorithm are halted. Furthermore, if no valid sets could be identified, the algorithm is repeated with ten times more evaluations.

5.3 Callback enabled cutting plane algorithm

For the second extension, we consider the cutting plane algorithm. In the version proposed by Augerat et al. (1998), it intends to iteratively solve the linearly relaxed sub problem for the degree and range constraints and all the identified capacity constraints and search for other violated constraints after each full solve. However, since the publication of their paper, the Cplex solver which they also used for their research has provided functionality for callback classes. A callback class returns potential candidate nodes to the user while solving the problem. This provides us with the opportunity to generate constraints on the fly based on partial solutions. In particular, our algorithm will make use of the lazy constraint callback class which can be extended to dynamically generate and add lazy constraints during the branch and cut search. This type of constraint is called lazy because it is only added to the formulation if the constraint is violated by a candidate solution. As such, a lazy constraint can restrict the feasible region of a problem and will always be satisfied in the final solution of the solve call.

This alternative cutting plane algorithm process has multiple consequences.First of all, the time spent in the Cplex solver is obviously reduced since it does not require to fully solve the problem at each iteration. Another outcome is that particular candidate solutions may not be optimal for the given constraints. Obviously, valid constraints can still be identified from the candidate solutions and may even be especially helpful given that more varied cut graphs are explored by our heuristics. By more varied, we mean that when only optimal solutions are traversed by our heuristics, it is more likely that these are similar to previous optimal solutions and thus produce similar cuts. Finally, some of the candidate solutions passed to the callback may have objective values above the upper bound of a problem instance. Imagine the case in which such a candidate solution is passed to the callback and the heuristic would not identify a valid constraint. Instead of halting the branch and cut at this point, which would lead to wrong lower bound solutions, the standard procedure for our solver will be to continue searching for more optimal solutions. Since none of the cuts could cut off

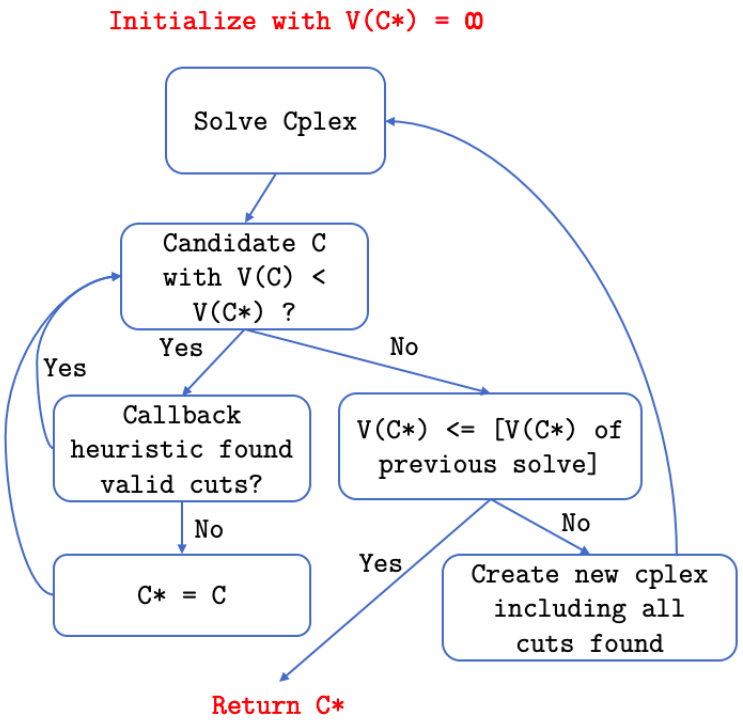

Figure 2: Callback algorithm

the optimal solution, candidates with objective values lower than the previous solution should be identified and the search continues until a candidate with a solution less than or equal to the lower bound is found and no lazy constraints are identified. In fact, the algorithm will go as follows. At each iteration, identify violated constraints and add them as lazy constraints to the model. If at a given iteration no valid cuts are identified, the solver will first pass some further candidate solutions with lower objective values to the callback in case they exist. If no better candidate solution is found, the latest candidate solution with the lowest objective value for which no lazy constraints were added will be accepted as the final answer. However, it is possible that after this candidate solution was found, in new candidate solutions constraints were identified that violate the first. Since at the moment a node is found for which currently no violated constraints can be identified, Cplex will treat it as a feasible solution regardless of what happens in the future. Although this scenario does not necessarily occur and may not make our algorithm halt prematurely in most cases, there could still be merit into circumventing this behavior in future research. This is a matter of trade-off between time spent in the heuristic and the quality of the final lower bound found. For our methodology, we introduce an additional step in which we choose to restart the solver with all constraints found to this point every time it finds a larger lower bound. The entire process is visualized in figure 2.

6 Results

The results section will have the following structure. Before presenting the results, a discussion will be held on why replication of the presented methods can vary from the original and which factors appear to have the most severe impact on diverging outcomes. Then, because one of the main efforts of our research is the replication and verification of the research by Augerat et al. (1998), the success of our replication will be evaluated first. Since our reproduced outcomes are similar enough to the ones reported by Augerat et al. (1998), further results drawn from the extensions will be bench marked against the replication. This is because it is impossible that the algorithms by Augerat et al. (1998) were copied exactly and a comparison between two heuristics based on the same underlying code base and solver is more informative. After the replication is evaluated, the extensions are presented. The results from our genetic meta heuristic using the iterative solving algorithm as proposed by Augerat et al. (1998) are presented first. It will be compared with the different heuristics for all three libraries LITLIB, ALIB and BLIB. Afterwards, we examine the performance of the callback method vis-à-vis the original cutting plane algorithm. For each heuristic, we discuss the effect of the alternative solving method. In the appendix, we also refer to and briefly describe the source code used to generate the results which is available in a separate zip file.

6.1 Explaining variance in replicated findings

Before discussing our own results, we would like to clarify why the algorithms in Augerat et al. (1998) are not deterministic and can lead to varying outcomes. The first factor to consider is that the lower bound found by a cutting plane algorithm is dependent on which sub problem of the full formulation is defined. A sub problem is composed of all the cuts identified at the time. As discussed, the heuristics are not exact methods and rely on intuitive search algorithms on the cut graph. However, on two slightly similar cut graphs, it is entirely possible that a heuristic identifies a constraint in one but not the other. On top of this, there is a regular pattern in which a heuristic has a lot of trouble in multiple consecutive cut graphs to identify valid constraints but afterwards effortlessly makes a lot of progress in later iterations in which it does identify a lot of deep cuts in the polytope. This phenomenon will be discussed more extensively in the discussion on the greedy heuristic, which especially suffers from it. Even though all methods are well documented, slight changes in the order of applying simple heuristics or settings of parameters can lead to different solving paths.

The second factor to consider is how the algorithm manages its cuts. In our results, we observe that not only the identification of cuts is important, but also the process that follows it. First of all, the algorithm must decide how many of the simple heuristics to apply. If for each cut identified by a meta heuristic, all neighbors of the corresponding node set are checked for violations in addition to the node set complement, in each iteration a lot of cuts could be found. As discussed by Augerat et al. (1998) for their tabu method, one can choose to limit the number of cuts to pass to the solver

in order to reduce the load. They chose to set this limit at the minimum of 125 and twice the number of nodes in the graph. However, they do not specify whether this is the number of nodes in the full graph or in the shrunken cut graph. Finally, when setting these limits, one can choose in what order to consider the cuts. In case a number of constraints an order of magnitude above the constraint limit per iteration is found, one can either loop over them in order they were found or sort them by violation from largest to smallest. However, do consider that when sorting the cuts by violation, one may only pass those constraints corresponding to large sets of nodes because they accumulate their violations, while smaller cuts compromised of subsets of this larger constraint could make deeper cuts in the polytope.

Finally, a third factor is randomness. Especially the greedy and tabu heuristic contain many random decisions in which a promising node set is grown in different directions based on a set of potential candidates to add to it. Therefore, it is impossible to exactly reproduce results unless the original code and random seeds are given. Even the constructive method can return different solution in multiple implementations. This is because multiple minimum cuts could be possible, but also because the shrinking algorithm could identify different constraints if its shrinking order changes or when the simple heuristic examines components before or after the shrinking. One different cut could lead to a different cut graph in the next iteration and go on to completely change the chain of events.

All three factors can have large impact on the stopping point of our algorithm, but also on the number of iterations and cuts that are made. It is quite difficult to set an ideal balance, because each factor is problem specific. Where incorporating more cuts could be useful for some instances and lead to a smaller feasible region rapidly, in other instances half the number of cuts would have virtually the same impact while not slowing down the solver as much. Finally, one should consider that the reported solving times by a paper published twenty years ago have little meaning when comparing absolute values. This is because not only have computing speeds gone up drastically, also the Cplex solver used in both our algorithms has been updated over the years and may show different performances.

6.2 Replication results

The replication results will be presented as follows. First, the performance of the constructive, greedy and tabu 1 heuristic on the different LITLIB instances will be compared. Then, the summary results for the five original heuristics by Augerat et al. (1998) are bench marked against our results for the three libraries. We present the relative percentage difference with our replication in color scaled tables. The colors show the largest under performance in red and over performance in green across all tables for each column respectively. The column names correspond with the tables by Augerat et al. (1998). They call the lower bound found by each heuristic, the number of cuts added to the model, the number of performed iterations, the GAP as upper minus lower divided by upper bound and the times spent in separation and Cplex L.B., GAP, Cuts, Iter, SEP and CPLEX respectively. In the appendix another column OGAP Diff was added to the absolute to show the

adjustment in the GAP if we do not take the same upper bounds as Augerat et al. (1998) but the actual optimal solutions which were found more recently. Since these differences are negligible and do not bear further relevance for this research, we continue the rest of our results with the old upper bounds. For the summary results, we apply the same comparison in addition to showing the absolute difference between how many times a heuristic was the best performer for a problem instance. While the most important results are discussed in this section, detailed tables for all heuristics and libraries can be found in the appendix.

The first comparison will be drawn between the performance of our constructive, greedy and tabu heuristic on finding lower bounds for the problem instances from literature. The findings by Augerat et al. (1998) are presented in tables 2 to 4 of their paper and we show the relative differences with our findings in tables 1, 2 and 3. Detailed output for all heuristics can be found in the appendix in tables 12 to 16. Even though we have held an extensive discussion on the source of diverging results, we observe relatively consistent outcomes in our replication. For the constructive heuristic, the difference in lower bound found is off by an average of less than a tenth of a percent with individual differences at most half a percent off. For the tabu heuristic, there is even less variance in the relative difference with the largest difference being the large fisher135 problem instance in which our heuristic finds a lower bound merely 0.34 percent smaller than that by Augerat et al. (1998). Overall, our tabu1 replication has a relative under performance of a small 0.12 percent. These differences are almost negligible and give a strong indication that the findings by Augerat et al. (1998) were generated with the same inputs and a very similar algorithm.

Table 1: Relative differences of replication using CONS

Name	L.B.	GAP	Cuts	Iter	SEP	CPLEX
cmt101	0.03%	-11.16%	49%	-9%	-82%	-79%
eil30	-	-0.10%	26%	64%	-98%	-92%
eil33	-0.04%	8.53%	32%	27%	-98%	-90%
eil51	-0.25%	12.62%	27%	75%	-95%	-89%
eilA101	0.47%	-13.69%	312%	43%	-71%	-45%
eailA76	0.58%	-7.36%	290%	70%	-75%	-65%
eilC76	0.46%	-10.10%	54%	-14%	-91%	-90%
eilD76	-0.07%	1.52%	130%	33%	-89%	-75%
fisher135	0.05%	-5.11%	116%	-5%	-79%	-80%
fisher45	-0.46%	513.87%	55%	19%	-96%	-79%
fisher72	-	-0.07%	29%	120%	-90%	-82%

Unfortunately, the greedy algorithm did not meet these standards in the replication. This is likely due to a slightly different implementation which could have larger consequences on the final findings. The method was only briefly touched upon in the paper and Augerat et al. (1998) give little insight into their final set up. Although they mention that they mix their strategy for the greedy initial set S between only one node and a set not including the depot, there is no exact explanation on how this is done. In practice, it appeared that giving priority to strategy one especially in the

Table 2: Relative differences of replication using GREEDY

Name	L.B.	GAP	Cuts	Iter	SEP	CPLEX
cmt101	-0.33%	408.51%	62%	35%	11%	-50%
eil30	-	-0.10%	492%	10%	-93%	-84%
eil33	-0.47%	262.91%	230%	57%	-49%	-70%
eil51	-1.64%	130.99%	107%	73%	61%	-59%
eilA101	-3.14%	117.81%	6%	-3%	-32%	-78%
eailA76	-2.25%	38.48%	-1%	-2%	-23%	-88%
eilC76	-4.24%	118.65%	-9%	46%	25%	-89%
eilD76	-1.88%	47.33%	-15%	-49%	-77%	-94%
fisher135	-5.91%	791.88%	-10%	14%	-52%	-91%
fisher45	-0.01%	10.50%	340%	11%	-75%	-71%
fisher72	-	-0.07%	322%	25%	-80%	-59%

Table 3: Relative differences of replication using TABU1

Name	L.B.	GAP	Cuts	Iter	SEP	CPLEX
cmt101	0.02%	-32.25%	128%	120%	-10%	-40%
eil30	-	-0.10%	240%	56%	-80%	-86%
eil33	-	-0.20%	139%	-8%	-87%	-90%
eil51	-0.07%	5.91%	60%	29%	-84%	-70%
eilA101	-0.23%	8.79%	14%	47%	-89%	-74%
eailA76	-0.29%	5.23%	-14%	15%	-92%	-86%
eilC76	-0.22%	6.43%	-18%	-10%	-93%	-85%
eilD76	-0.18%	4.61%	-22%	-28%	-95%	-86%
fisher135	-0.34%	56.54%	41%	173%	-51%	-86%
fisher45	-	-	196%	10%	-86%	-85%
fisher72	-	-0.07%	129%	23%	-70%	-84%

early stage of the algorithm improved performance, after which increasing the number of trials and limiting the number of iterations by a given maximum number of cuts also helped. In a deeper dive into the algorithm, it appeared that the greedy algorithm often got stuck in a stage at which it had effort identifying even only one constraint for multiple iterations, after which it made more rapid progress in the cutting algorithm later on. This phenomenon is illustrated in figure 3.

For our replication, this means that the lower bound we found was on average almost two percent lower than the one presented in their paper, with the fisher135 instance under performing by as much as six percent. Nonetheless, there are a number of instances for which the lower bound found is very close to the one by Augerat et al. (1998). We have strong reason to believe this is because for the under performing instances, our methods fail to find the appropriate cuts at certain iterations causing the solver to prematurely terminate.

The second column of our comparison is the relative difference in GAP. This metric is strongly correlated to the relative difference in lower bound but does give us some interesting information on the severity of underperformance. This is because the closer a lower bound gets to optimality, the more difficult it becomes for our heuristics to identify new cuts due to the more fractional edge weights of the cut graph representing possible edge activations. If the lower bound found by Augerat et al. (1998) already contained a large GAP, like for eilA76, a small relative difference in lower bound would not mean so much since it does not close the relative GAP as much. On the other hand, for instances already close to optimality, it is difficult to find improvements and even small underperformance would mean that our heuristic terminated much too early. One such instance is fisher fisher45, which has GAP less than a tenth of a percent for all heuristics by Augerat et al. (1998). Looking at differences in GAP for our constructive heuristic, it is worth to mention that the overperformances in the some of the Eilon problem instances are not that surprising and unexpected, since the relative change in GAP is still tiny. The only problematic case is fisher45,

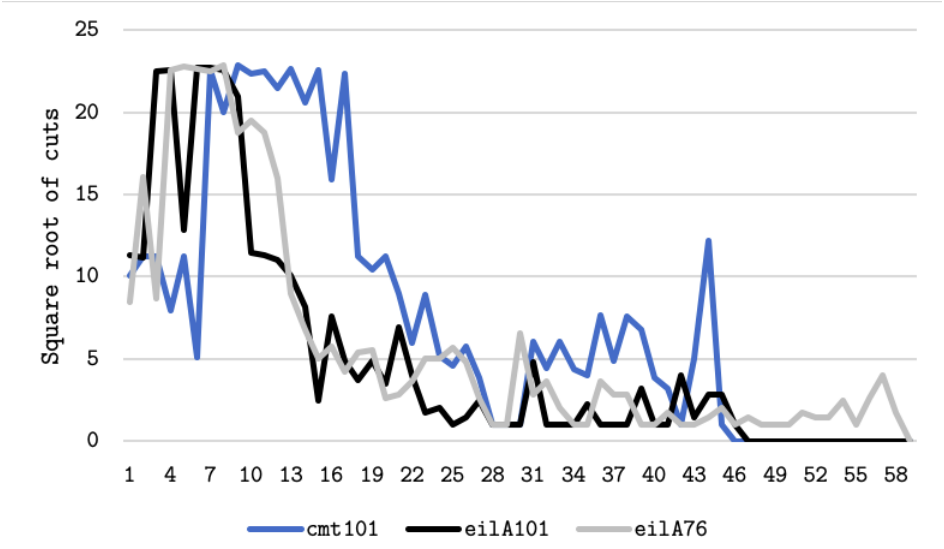

Figure 3: Cuts per iteration

for which not only the relative lower bound is the further off, also the GAP relatively increased by a factor of five, which is quite large considering all other are less than 15 percent off. For the greedy algorithm, as expected given the performance in lower bounds, almost all GAP statistics increased by a large percentage. Fortunately, for the tabu heuristic, this looks much better with all differences below 10 percent with two relatively small outliers with out and underperformance of 32 and 57 percent in cmt101 and fisher135 respectively. For both the CUTS and ITER statistics, our findings are not in line with those of Augerat et al. (1998). On average, for the given heuristics, we find about twice the number of cuts and perform about 40 percent more iterations. Since this holds for the three heuristics in table 2 to 4 of Augerat et al. (1998), we assume this is not because of a different implementation of any heuristic but the way we deal with the simple heuristics and cut managements as discussed in the previous subsection. Looking at the time spent in separating constraints and in Cplex, we see the expected decrease in CPU time, enabled through the faster processing units of today's computers. With a few exceptions, the reduction in time spent in both separation and Cplex is decreased by about 80 percent for the problem instances from literature. Again, the replicated greedy heuristic is an outlier with a few extremely slow separation processes. This is likely related to our previous description about the implementation of the greedy heuristic, where it must spend a long time to find single cuts in certain cut graphs.

Moving on, we examine the replication findings of the summary results for the three different problem instance libraries as presented in Augerat et al. (1998) tables 5 to 7 which are again contrasted in tables 4, 5 and 6. The full aggregated summary results can also be found in the appendix in tables 20, 21 and 22. For the literature problem instances, there is little news for the constructive, greedy and tabu 1 heuristic since their results were analyzed in depth in the previous paragraphs. However, the same conclusions apply to tabu 2 and tabu 3, where the average GAP is only 2.67 and 7.93 percent different from the original results. Also, the maximum GAP statistics are very close, with the constructive outperforming slightly by 7.36 percent, the greedy under performing by 40 percent, tabu 2 virtually identical with only 1 percent difference and the remaining tabu heuristics off by about 5 percent. Another new piece of information is the count of instances for which each heuristic performs best. Since the greedy heuristic now performs worse than originally, its count decreases and passes one win to the constructive and one to the tabu heuristics. Within the tabu heuristics, tabu 3 gave up five wins to the other tabu heuristics. It is difficult to say why tabu 3 lags behind a bit now for the literature instances, but looking at the number of cuts and iterations, it seems to have had less success than in the original results where it used to have the most, and now the least out of the tabu heuristics.

For the A and B libraries, the results look a bit different yet again. In both libraries, the constructive method outperforms between 6 and 26 percent with a much smaller maximum GAP. It is the best heuristic one time more often than it was before, possible taken from the greedy heuristic which again did not perform very well. Also, the constructive method performed really well on the time spent in separation, reducing the time spent by 93 to 94.5 percent. Interestingly, each of the three tabu heuristics have slightly differing results on these two libraries. For the A library, all tabu

Table 4: Relative differences with original on LITLIB

Name	Avg GAP	Max GAP	Best	Cuts	Iter	SEP	CPLEX
CONS	-2.31%	-7.36%	50%	123%	13%	-81%	-78%
GREEDY	79.11%	38.48%	-50%	25%	14%	-38%	-87%
TABU1	5.42%	5.23%	-	23%	33%	-85%	-81%
TABU2	2.67%	1.11%	67%	54%	50%	-69%	-77%
TABU3	7.93%	5.38%	-50%	12%	1%	-93%	-83%

Table 5: Relative differences with original on ALIB

Name	Avg GAP	Max GAP	Best	Cuts	Iter	SEP	CPLEX
CONS	-6.49%	-12.98%	-	94%	-11%	-94%	-87%
GREEDY	109.00%	135.91%	-100%	40%	-6%	-50%	-91%
TABU1	19.25%	10.95%	-54%	43%	21%	-91%	-84%
TABU2	23.17%	2.40%	-67%	55%	28%	-82%	-84%
TABU3	6.28%	-4.03%	-38%	43%	6%	-96%	-84%

Table 6: Relative differences with original on BLIB

Name	Avg GAP	Max GAP	Best	Cuts	Iter	SEP	CPLEX
CONS	-26.28%	-58.00%	-	122%	-5%	-93%	-77%
GREEDY	39.77%	-27.22%	-75%	91%	23%	-20%	-72%
TABU1	-19.22%	-64.06%	-29%	76%	41%	-82%	-70%
TABU2	4.97%	43.28%	-57%	76%	26%	-67%	-79%
TABU3	-29.02%	-63.96%	-55%	83%	28%	-82%	-76%

methods have a slight underperformance between 6 and 23 percent and get the win a bit less often than before. This is likely because Augerat et al. (1998) found the same lower bound using different heuristics more often which led to shared wins. In our implementation, these draws appeared to happen less often with heuristic winning a problem instance with small decimal numbers at times. Where tabu 3 underperformed in the literature instances, in the B library it has a 30 percent much lower average GAP, likely due to a large reduction of 64 percent in the maximum GAP. Since it still did not get more wins over the other heuristics, this is likely due to a much better performance on one or a few problem instances which terminated prematurely in the algorithms of Augerat et al. (1998). This is also what appears to have happened for our implementation of tabu 2 in the B library, which sees a large increase in its maximum GAP, probably deteriorating the average GAP with it. The story for the number of cuts and iterations made in the two libraries is similar as before, finding between 50 and 80 percent more cuts with about 30 percent more iterations for the tabu heuristics, and even a decrease in iterations for the constructive methods which appears to have found faster routes to strong lower bounds.

Finally, we will contrast the comparative tables in Augerat et al. (1998) tables 8 to 10. For tables 7 and 8, we show not the relative but the absolute differences while for table 9 we show the relative again. Detailed output is in the appendix in tables 23 to 25. Given that we already concluded a larger number of cuts were found by our algorithm, it is no surprise that when a heuristic is followed up by another, the latter finds a larger number of cuts compared to the ones reported by Augerat et al. (1998). However, we find this number to be even larger than expected, with on average about one hundred cuts more for every follow up heuristic than originally. Especially the heuristics following up the greedy method find many cuts, which is to be expected given its poor performance. On the flip side, the follow up greedy heuristic barely finds any new cuts relative to the others. Aside from the differences, there are also some similar patterns. The follow up constructive heuristic barely finds and cuts after the tabu heuristics but does do so after the greedy. Tabu 2 finds relatively few cuts compared to tabu 1 and 3 and each of the tabu heuristics do really well when following up the constructive method.

Table 7: Difference in average cuts with original

	CONS	GREEDY	TABU1	TABU2	TABU3
CONS	-	84.66	194.53	269.58	244.46
GREEDY	330.18	-	411.98	343.21	501.70
TABU1	27.39	31.66	-	65.49	78.33
TABU2	85.20	36.29	123.05	-	157.05
TABU3	22.93	13.78	36.27	27.50	-

The average maximum slack found by the different follow up heuristics is much closer to the findings by Augerat et al. (1998). As before, the greedy heuristic underperforms strongly but for all other heuristics, the results look much steadier aside from consistently being between 0,2 and 0,5 higher. These are quite small difference given that slack can be up to twice the number of

Table 8: Difference in maximum slack of cuts with original

	CONS	GREEDY	TABU1	TABU2	TABU3
CONS	-	0.24	0.20	0.28	0.31
GREEDY	2.18	-	1.69	1.81	1.77
TABU1	0.65	0.44	-	0.43	0.52
TABU2	0.76	0.30	0.39	-	0.42
TABU3	0.40	0.28	0.27	0.32	-

Table 9: Difference in CPU time with original

	constructive	GREEDY	TABU1	TABU2	TABU3
CONS	-	143%	15%	315%	24%
GREEDY	585%	-	103%	465%	114%
TABU1	-4%	-7%	-	106%	-73%
TABU2	175%	37%	18%	-	40%
TABU3	10%	-22%	-53%	54%	-

vehicles of the problem instances. Also, the relative differences between the performances of the tabu heuristics are again similar to the original findings. Finally, we examine the differences in CPU times. However, the results here are drastically different from the original findings with no clear patterns. This is due to the fact that the time spent in separation and Cplex is already heavily reduced on average, thus causing these values to be largely determined by outliers where one heuristic terminated prematurely or a follow up needed a lot of time to find a few extra cuts. The interpretation of these results can thus be given less importance. Overall, we summarize our findings as strongly reflecting and confirming the results presented by Augerat et al. (1998) with exception of the greedy heuristic. For all others, the most relevant statistics like the lower bound and the GAP are especially well replicated thus validating the original methods.

6.3 Genetic meta heuristic findings

For our first extension, we analyze the quality of our genetic algorithms. We present the results in the same fashion as tables 5 to 7 in Augerat et al. (1998), with summary statistics for each heuristic over all problem instances of the three libraries. The summary statistics for each library apart can be found in the appendix in tables 20 to 22. Here, we will examine the results of the aggregate summary in table 10. In the appendix, detailed results for the extension heuristics on the LITLIB problem instances can also be found in tables 17 to 19.

The original genetic algorithm which only searches for the set of nodes making up the largest violation in the cut graph is denoted by the name 'Genetic'. It is quite clear that in terms of lower bounds found, this heuristic does not perform as well as one would hope. With an average GAP of 3.7 percent, it doubles the GAP of tabu 3 in addition to having one of the worst maximum GAPs. It does perform best twice, but this only because on B-n35-k5 and Fisher72 all heuristics found the

Table 10: Summary of replication of all problem instances

Name	Avg GAP	Max GAP	Best	Cuts	Iter	SEP	CPLEX
Constructive	2.50	6.75	6	758.30	57.89	0.84	4.59
Greedy	3.80	11.77	4	1,399.54	40.79	7.01	4.51
Taboo 1	2.06	5.63	21	1,331.39	38.21	3.05	5.18
Taboo 2	2.31	10.30	10	1,278.59	36.67	2.26	4.48
Taboo 3	1.86	5.63	30	1,345.50	40.08	3.31	5.25
Genetic	3.72	9.35	2	236.18	68.47	3.30	2.40
ConGen1	2.16	6.43	11	812.08	50.83	5.69	5.05
ConGen2	2.02	6.36	15	878.70	58.05	6.89	5.78

same lower bounds. A few other details jump to eye. Obviously, since the algorithm terminates early given the smaller lower bounds found, there are less cuts and less time spent in separation and Cplex than a well performing heuristic would have. Still, it is not especially fast in separation, which is to be expected given that the genetic algorithm evolves a population over hundreds of generations at each iteration. It seems that the algorithm performs a large number of iterations in addition to only using small number of cuts. In general, we would be happy to have a method that achieves strong lower bounds with fewer cuts, since a large number of cuts makes the LP solver slow. We notice that the genetic algorithm still outperforms the greedy algorithm with only a fraction of the number cuts. This observation led to the idea to use the genetic algorithm to create an improved heuristic aimed at finding a good lower bound with as few cuts as possible. Another existing heuristic that shows similar qualities is the constructive heuristic. The combined upgraded heuristic we came up with is one where the constructive heuristic finds the minimum cuts as it did originally. The resulting list of valid sets on the sink side are now passed to the genetic heuristic, which will search for further violated constraints within these sets, in addition to examining the different components of the original cut graph at each iteration. We implemented two versions of this heuristic, one in which we optimize only the violation, and another in which also demand is minimized, and the edge cut weight is maximized. These two heuristics are called ConGen1 and ConGen2 respectively. Given that this new upgraded method builds upon the constructive heuristic, it must perform at least as well as the latter. This is confirmed in our findings which show a strong decrease in average GAP from 2.5 percent in the constructive to 2.16 and 2.02 in the upgraded heuristics ConGen1 and ConGen2. These methods thereby match the performance of tabu heuristics but retain the property of needing a much smaller number of cuts. The tabu 3 heuristic, which with an average GAP of 1.86 percent still dominates the other methods in absolute performance on the lower bound found, requires over 50 percent more cuts than ConGen2 which is now the second-best performer with only 879 cuts on average. The new heuristic methods do come at a cost of spending about twice as much time as the tabu heuristics in separation. We consider this shortcoming acceptable given the linear computation complexity of the heuristics, improved processing speeds and the possibility to run many parallel threads to run the genetic algorithm.

6.4 Results from the callback functionality

The second extension alters the cutting plane algorithm and goes into the separation phase for each candidate solution the solver finds. As such, it will enter many more separation phases because it does not wait for the solver to find an optimal solution at every iteration. Another aspect of the callback function is that even when an intermediate solution was found, we can restart the solver with all the constraints found so far because it will then return different candidate solutions for which our heuristics may find more cuts which improve our lower bounds. For the callback algorithm results for all heuristics on the LITLIB problem instances and the relative differences with our findings with the iterative cutting plane algorithm, please refer to tables 26 to 41. The results discussed here are based on the same summary statistics used for extension 1. Again, all individual library summary statistics and differences with the iterative solver are in the appendix in tables 42 to 48. In this subsection, we will compare the relative percentage change of our callback algorithm findings compared to our iterative solving results in table 11.

Table 11: Relative differences with iterative on all problem instances

Name	Avg GAP	Max GAP	Best	Cuts	Iter	SEP	CPLEX
CONS	-7.52%	-4.77%	-17%	296%	8%	11%	-77%
GREEDY	-29.25%	-35.56%	-	586%	22%	86%	347%
TABU1	-13.86%	-1.86%	24%	344%	3%	-4%	-66%
TABU2	-23.64%	-46.55%	190%	353%	11%	46%	-69%
TABU3	-0.64%	2.10%	-67%	539%	15%	-20%	-67%
GEN	-10.67%	21.34%	50%	450%	47%	85%	-68%
CONGEN1	-5.73%	-2.38%	-27%	363%	19%	35%	2%
CONGEN2	-4.32%	-4.38%	7%	304%	17%	39%	98%

The first thing to be noticed is that the average GAP has decreased massively across all heuristics. Especially the tabu 2 heuristic sees strongly improved results, with a 20 percent lower average GAP. This makes sense, because the edge that tabu 1 and 3 had over tabu 2 is varied parameters to give the heuristic a second chance to find cuts. However, by restarting the solver in the callback algorithm, the tabu 2 heuristic also receives a second chance and thus has less change of terminating early. This probably contributed to a large decrease in its maximum GAP. These decreases in maximum GAP are to be expected with the callback algorithm, since the problem illustrated in figure 3 is less of a problem here. When the solver terminates too early, it can be restarted and the next time around will search cut graphs in which it does find new constraints. Also, tabu 1 is a big winner in this extension, increasing its win count from 21 instances to 26. The greedy genetic heuristic still underperforms but did see a 30 and 10 percent decrease in average GAP with the genetic still only needing a very small number of cuts relative to all other methods. Finally, the ConGen heuristics did see decreases in average GAP but not of the same impact as the others. Overall, for all heuristics, the number of cuts drastically increased by a factor from 3 to 6, with a few more iterations as well. Interestingly, it also finds on average much more cuts per iteration.

Where the iterative solve returned a full average of 20 cuts per iteration, the callback solver finds close to one hundred. This was expected to some extent because, as discussed in the methodology section, the separation phase inspects a much more varied set of cut graphs. Finally, the time spent in separation only increased slightly for most methods. The greedy and genetic algorithms again jump out with slower solving times, which can be attributed to premature termination in the iterative solve. The tabu heuristics perform about equally well in time spent but with the tabu 2 heuristic needing slightly more time to facilitate the 20 percent decrease in average GAP. The ConGen methods are about 40 percent slower in separation. Finally, for the time spent by the Cplex solver, we observe our alternative solving extension really pays off in efficiency, decreasing the solving time by about 70 percent for all heuristics with exception of the greedy heuristic. This can be attributed to not requiring the solver to find the global optimal solution at every iteration anymore, strongly reducing the computation time. The exception is for the combined constructive and genetic heuristics, which now require a much larger amount of time in the Cplex solver. Even though these methods still uses a relatively small number of cuts to find its best lower bound, it is possible that because the number of cuts has tripled, the effect on the Cplex solving time turned out to weigh more heavily than expected with the Cplex solving time for ConGen2 doubling. In summary, the callback functionality proves to be a very promising alternative for the traditional solve and repeat cutting plane algorithm which provides better lower bounds in a shorter time. All summary results and the performance on each literature problem instance for the heuristics using the callback algorithm can be found in the appendix where a column 'Follow Ups' indicates how often the solver was restarted.

7 Conclusion

Our research attempted to replicate and verify the computational results of Augerat et al. (1998) and to provide two extensions to the capacity separation problem for the CVRP. In this section, we will summarize our takeaways and conclusions regarding these objectives. To finish off, we give recommendations for further research.

Although our replicated results showed diverging outcomes especially on statistics like the number of cuts, iterations or time spent in separation and in the Cplex solver, the essential lower bound targets could be replicated to a very precise degree. We contribute the aforementioned divergence to a slightly different implementation of our algorithms which generate more cuts. That conclusion also holds for the relatively poor performance of our replication of the greedy heuristic which was the only method of which our lower bounds did not match the research by Augerat et al. (1998). Furthermore, our faster computational speed allows for a reduction in time spent in separation and Cplex by about 80

Our second contribution to literature consists of a genetic algorithm in the capacity separation problem. By evolving a population deciding which nodes to incorporate in a set corresponding to the capacity constraints, we hoped to present a meta heuristic which tackled the separation

problem from a different angle to outperform the existing meta heuristics. Unfortunately, our implementation of the genetic algorithm using the NSGA algorithm showed premature termination in too many problem instances making it too unreliable to find strong lower bounds. However, our findings showed that the genetic algorithm did find different types of cuts than the tabu heuristics, requiring a much smaller number of constraints to achieve certain lower bounds. By combining the genetic algorithm with the more consistent constructive heuristic, we found a meta heuristic which matched the average performance of the tabu heuristics across all problem instances with about 40 percent less cuts. Although this did come at a cost of twice the time spent in separation, this should be acceptable given the polynomial-time complexity of these algorithms and the advantage of having less cuts slowing down the solver for larger problem instances.

Finally, we replicated the methods by Augerat et al. (1998) and our genetic algorithms using an alternative cutting plane algorithm. Instead of solving the sub problem with identified capacity constraints and searching for violated cuts iteratively, we return candidate solutions from the solver through a callback function. Intuitively, we find decreased time spent in the Cplex solver for the heuristics by Augerat et al. (1998) since the solver does not need to find the optimal solution at each iteration. The change in time spent in separation is more ambiguous, mainly due to the more unexpected over performance in the quality of lower bounds. The strong decrease in average gap of on average 10 percent is likely to be attributed to the larger variety of cut graphs of candidate solutions returned by the callback function. More varied cut graphs imply more varied cuts, which is reflected in the 3 to 6-fold increase in cuts found. Especially the tabu 2 method performs really strongly using the callback functionality because it benefits the most from restarting the solver after it terminates.

To conclude, we want to pass some takeaways and recommendations for further research. First of all, we have verified that the tabu heuristic developed by Augerat et al. (1998) is a powerful method for finding lower bounds for the CVRP. Although our genetic algorithm has not outperformed the tabu heuristic consistently, we do see further potential for an NSGA algorithm to do so. Especially the mutation settings which crosses the non-dominated population over for new generations can be modified to accommodate smarter evaluations. Also, there is room for further simple heuristics which improve population members, similar to our neighbor check at each candidate evaluation of the genetic algorithm. Finally, we strongly recommend incorporating the callback functionality into the cutting plane algorithm. It has proven to be a faster method which finds more varied cuts leading to stronger lower bounds. Given the larger number of cuts generated, more research could be directed at how to manage large number of cuts found instead of simply passing everything to the solver. All of these recommendations may be valuable for future implementations of the cutting plane algorithm using any of the methods presented in this paper.

8 Appendix

8.1 Java Code

The following bullet list describes the different classes in each package of our Java project. The full source code is available upon request in a separate zip file.

- _Front_
  - *Main_Iterative*
    Main class for the iterative solver. All parameters can be modified either in this class or in the HeuristicParameters csv.
  - *Main_Callback*
    Main class for the callback solver. Set parameters similar to the iterative solver.
- cvrpHeuristics
  - *ConGenHeuristic*
    Extending the CvrpHeuristic with the upgraded constructive plus genetic heuristic.
  - *ConstructiveHeuristic*
    Extending the CvrpHeuristic with the constructive heuristic.
  - *CvrpHeuristic*
    Abstract class which can be solved to return identified valid sets corresponding to the capacity constraints. Provides simple heuristic functionality to extensions.
  - *GeneticHeuristic*
    Extending the CvrpHeuristic with the genetic heuristic.
  - *GreedyHeuristic*
    Extending the CvrpHeuristic with the greedy heuristic.
  - *TabuHeuristic*
    Extending the CvrpHeuristic with the tabu heuristics.
- cvrpObjects
  - *CvrpParameters*
    Class which is passed around different classes to access parameters. Reads parameters from the ParameterParser in the parsers package.
  - *GeneticHelper*
    Class to pass candidate solutions of the genetic algorithm to in order to improve the variables by a neighbor check.
  - *ValidSet*
    Class storing the properties of a set corresponding to the capacity constraints.

  - *ValidSetBuilder*
    Class used to constructs sets to check the capacity constraints for.

- graph
  - *Tour*
    A tour constitutes of nodes connected by integer edges and properties like demand and distance.
  - *VrpContractionGraph*
    A contractionGraph stores an input graph and shrinks its edges. Through this class, retrieve demands, labels, nodes and edges from both the input and the shrunken graph.
  - *VrpEdge*
    An edge in a VRP problem instance.
  - *VrpNode*
    A node in a VRP problem instance.

- oocplexExtended
  - *AbstractVrpFormulation*
    Abstract class providing functionality for solving a simple VRP problem with a minimal edge distance objective and degree constraints.
  - *CvrpCutsGenerator*
    A class for generating cuts based on the cut graph returned by the iterative solver.
  - *CvrpDynamicFormulation*
    Extends the AbstractVrpFormulation by implementing one of the cut generators and overwriting the solving procedure depending on whether an iterative or callback solver is chosen.
  - *LazyVrpCallbackCuts*
    A class for generating cuts based on the cut graphs returned by the callback functionality.
  - *VrpDegreeConstraints*
    Create and store the degree constraints for the VRP.
  - *VrpEdgeVariables*
    Create an immutable bi map between the edge objects and decision variables in the VRP.
  - *VrpMinDistanceObjective*
    Create and store the minimum distance objective of the activated edges in the VRP.

- parsers
  - *CvrpLibInstance*
    Full instance of a CVRP problem providing all necessary properties.

- *CvrpParser*
  Parse a problem instance in TSPLib format into a CvrpLibInstance.
- *OutputWriter*
  Use a CSVPrinter to write output to a csv file.
- *ParameterParser*
  Use a CSVParser to read parameters from a csv file.
- *ProblemInstanceParser*
  Use a CSVParser to read which problem instances to load from a csv file.

- princetonMinCut
  - *FlowNetwork*
    Convert an undirected graph into a flow network to solve the minimum cut.
  - *FordFulkerson*
    Apply the Ford Fulkerson algorithm to find the minimum cut between two nodes.

- transformers
  - *VrpDemandTransformer*
    Map a VRPNode object to its corresponding demand.
  - *VrpEdgeTransformer*
    Map a VRPEdge object to its corresponding distance cost.
  - *VrpNodeLabels*
    Map a VRPNode object to its corresponding label.

- util
  - *CplexVrpUtil*
    Static class to provide a number of methods to create or handle cplex variables.
  - *GraphUtil*
    Static class to provide methods to be applied to cut graph objects.
  - *MapUtil*
    Static class to facilitate some general methods on map objects.
  - *Timer*
    Custom timer class in which times between events are stored through keys.
  - *VisualizeGraph*
    Debugging class which can generate a panel for visualizing intermediate cut graphs, display or highlight nodes and edges and log messages.
  - *VrpLoopExtractor*
    Static class used for extracting loops or connected components from a cut graph.

8.2 More computational results

Table 12: Constructive heuristic on LITLIB

Name	L.B.	GAP	Cuts	Iter	SEP	CPLEX	OGAP Diff
cmt101	818.40	0.20	805	77	2.36	9.94	-
eil30	508.50	4.78	86	18	0.02	0.04	(0.18)
eil33	831.38	0.43	226	33	0.05	0.15	-
eil51	509.50	2.21	127	28	0.07	0.15	-
eilA101	793.52	2.87	1,733	80	3.28	14.55	(0.24)
eilA76	777.68	6.75	1,495	85	2.06	7.83	(0.45)
eilC76	706.26	3.91	995	66	1.26	3.35	-
eilD76	658.19	4.19	1,062	76	1.27	4.57	(0.70)
fisher135	1,153.28	1.01	3,348	243	13.68	89.02	(0.26)
fisher45	720.00	0.55	200	32	0.07	0.38	-
fisher72	232.50	1.90	135	22	0.09	0.31	-

Table 13: Greedy heuristic on LITLIB

Name	Max of LB	GAP	Cuts	Iter	SEP	CPLEX	OGAP Diff
cmt101	816.66	0.41	2,388	54	22.56	23.39	-
eil30	508.50	4.78	509	23	0.05	0.13	(0.18)
eil33	829.55	0.65	1,027	33	0.72	0.43	-
eil51	506.08	2.86	843	38	5.99	1.17	-
eilA101	770.91	5.64	1,468	37	18.73	6.78	(0.23)
eilA76	770.13	7.66	1,510	46	22.14	5.83	(0.45)
eilC76	679.56	7.54	1,141	38	13.64	2.81	-
eilD76	648.23	5.64	811	21	4.50	1.03	(0.69)
fisher135	1,088.11	6.60	3,869	116	54.97	44.72	(0.24)
fisher45	723.60	0.06	800	30	0.40	0.59	-
fisher72	232.50	1.90	773	25	0.70	1.08	-

Table 14: Tabu1 heuristic on LITLIB

Name	L.B.	GAP	Cuts	Iter	SEP	CPLEX	OGAP Diff
cmt101	819.50	0.06	2,312	55	13.23	20.53	-
eil30	508.50	4.78	238	25	0.03	0.08	(0.18)
eil33	833.50	0.18	650	22	0.18	0.21	-
eil51	514.16	1.31	868	27	0.92	0.93	-
eilA101	794.42	2.76	1,827	44	9.48	12.99	(0.24)
eilA76	787.05	5.63	2,104	54	10.37	12.40	(0.45)
eilC76	709.58	3.46	1,689	43	6.71	6.85	-
eilD76	660.12	3.91	1,224	34	3.27	3.67	(0.70)
fisher135	1,154.06	0.94	3,521	90	29.50	33.46	(0.26)
fisher45	724.00	-	461	22	0.11	0.24	-
fisher72	232.50	1.90	392	16	0.14	0.37	-

Table 15: Tabu2 heuristic on LITLIB

Name	L.B.	GAP	Cuts	Iter	SEP	CPLEX	OGAP Diff
cmt101	819.45	0.07	2,222	54	10.60	24.69	-
eil30	508.00	4.87	206	22	0.03	0.07	(0.18)
eil33	832.54	0.30	676	23	0.16	0.22	-
eil51	514.50	1.25	770	26	0.55	0.83	-
eilA101	792.97	2.94	1,648	35	6.57	8.42	(0.24)
eilA76	787.12	5.62	3,117	73	9.21	18.72	(0.45)
eilC76	709.75	3.43	2,497	60	6.30	13.78	-
eilD76	660.40	3.87	1,315	36	2.78	3.98	(0.70)
fisher135	1,152.77	1.05	3,100	74	17.55	30.33	(0.26)
fisher45	724.00	-	402	21	0.09	0.22	-
fisher72	232.50	1.90	411	20	0.15	0.47	-

Table 16: Tabu3 heuristic on LITLIB

Name	L.B.	GAP	Cuts	Iter	SEP	CPLEX	OGAP Diff
cmt101	819.50	0.06	2,108	53	14.08	28.74	-
eil30	508.50	4.78	246	28	0.03	0.10	(0.18)
eil33	833.50	0.18	821	26	0.27	0.32	-
eil51	514.38	1.27	875	32	1.10	1.29	-
eilA101	793.61	2.86	1,583	35	8.31	8.99	(0.24)
eilA76	787.07	5.63	2,159	62	11.99	16.07	(0.45)
eilC76	709.77	3.43	1,591	39	6.58	5.72	-
eilD76	659.58	3.99	1,151	32	3.36	3.16	(0.70)
fisher135	1,149.78	1.31	2,614	67	20.29	24.01	(0.25)
fisher45	724.00	-	470	22	0.13	0.24	-
fisher72	232.50	1.90	510	21	0.20	0.61	-

Table 17: Genetic heuristic on LITLIB

Name	L.B.	GAP	Cuts	Iter	SEP	CPLEX	OGAP Diff
cmt101	809.05	1.34	444	114	11.78	12.53	-
eil30	507.83	4.90	101	35	0.08	0.10	(0.18)
eil33	831.00	0.48	157	43	0.34	0.18	-
eil51	510.22	2.07	85	35	0.91	0.21	-
eilA101	780.37	4.48	240	31	2.57	1.59	(0.23)
eilA76	771.56	7.49	322	108	10.09	3.15	(0.45)
eilC76	701.30	4.58	209	76	6.93	2.20	-
eilD76	649.71	5.43	163	58	3.89	1.26	(0.69)
fisher135	1,095.71	5.95	707	113	23.65	37.51	(0.24)
fisher45	722.31	0.23	117	41	0.20	0.25	-
fisher72	232.50	1.90	108	28	0.23	0.31	-

Table 18: ConGen1 heuristic on LITLIB

Name	L.B.	GAP	Cuts	Iter	SEP	CPLEX	OGAP Diff
cmt101	818.53	0.18	1,050	61	5.54	10.32	-
eil30	508.50	4.78	153	20	0.25	0.06	(0.18)
eil33	831.11	0.47	276	23	0.47	0.10	-
eil51	514.00	1.34	490	35	4.02	0.79	-
eilA101	793.78	2.84	1,875	81	15.05	13.69	(0.24)
eilA76	780.41	6.43	1,451	81	10.79	8.84	(0.45)
eilC76	706.18	3.92	1,211	71	15.54	6.90	-
eilD76	659.45	4.01	1,314	66	10.54	6.47	(0.70)
fisher135	1,156.52	0.73	2,965	234	34.39	116.30	(0.26)
fisher45	724.00	-	228	18	0.52	0.16	-
fisher72	232.50	1.90	163	16	0.46	0.21	-

Table 19: ConGen2 heuristic on LITLIB

Name	L.B.	GAP	Cuts	Iter	SEP	CPLEX	OGAP Diff
cmt101	818.94	0.13	837	63	3.88	7.73	-
eil30	508.50	4.78	154	17	0.23	0.04	(0.18)
eil33	833.11	0.23	351	31	1.47	0.20	-
eil51	514.00	1.34	353	33	6.33	0.45	-
eilA101	795.12	2.68	2,016	101	17.34	25.30	(0.24)
eilA76	780.97	6.36	1,664	72	9.04	7.47	(0.45)
eilC76	707.47	3.75	1,904	102	17.19	16.86	-
eilD76	660.00	3.93	1,419	76	10.25	6.86	(0.70)
fisher135	1,156.60	0.72	2,946	248	35.43	102.90	(0.26)
fisher45	724.00	-	307	35	0.96	0.38	-
fisher72	232.50	1.90	181	15	0.43	0.20	-

Table 20: Summary of LITLIB replication

Name	Avg GAP	Max GAP	Best	Cuts	Iter	SEP	CPLEX
Constructive	2.62	6.75	3	928.36	69.09	2.20	11.84
Greedy	3.98	7.66	2	1,376.27	41.91	13.13	8.00
Taboo 1	2.27	5.63	5	1,389.64	39.27	6.72	8.34
Taboo 2	2.30	5.62	5	1,487.64	40.36	4.91	9.25
Taboo 3	2.31	5.63	5	1,284.36	37.91	6.03	8.11
Genetic	3.53	7.49	1	241.18	62.00	5.52	5.39
ConGen1	2.42	6.43	3	1,016.00	64.18	8.87	14.89
ConGen2	2.35	6.36	5	1,102.91	72.09	9.32	15.31

Table 21: Summary of ALIB replication

Name	Avg GAP	Max GAP	Best	Cuts	Iter	SEP	CPLEX
Constructive	3.49	5.51	1	785.22	56.48	0.61	2.70
Greedy	5.43	11.77	-	1,268.78	37.74	6.16	2.54
Taboo 1	2.86	5.27	6	1,317.33	35.59	2.30	3.89
Taboo 2	3.08	5.10	2	1,299.78	34.85	1.69	3.49
Taboo 3	2.52	4.54	16	1,357.73	39.62	2.60	4.62
Genetic	4.95	9.35	-	218.88	75.42	3.34	1.61
ConGen1	2.97	5.57	3	784.92	48.92	5.82	2.70
ConGen2	2.83	5.52	3	876.81	58.31	7.75	3.69

Table 22: Summary of BLIB replication

Name	Avg GAP	Max GAP	Best	Cuts	Iter	SEP	CPLEX
Constructive	1.28	3.13	2	645.35	54.17	0.47	3.33
Greedy	1.80	5.26	2	1,564.17	43.83	5.07	5.15
Taboo 1	1.03	2.58	10	1,320.04	40.78	2.17	5.19
Taboo 2	1.41	10.30	3	1,153.74	37.04	1.66	3.37
Taboo 3	0.89	2.59	9	1,360.91	41.65	2.82	4.59
Genetic	2.43	8.01	1	253.35	63.70	2.19	1.85
ConGen1	1.11	3.10	5	745.26	46.61	4.01	3.00
ConGen2	0.96	2.65	7	773.61	51.04	4.75	3.59

Table 23: Average number of cuts in replication

	CONS	GRDY	TBU1	TBU2	TBU3	GEN	CG1	CG2
CONS	-	96.39	290.69	299.03	340.62	7.15	120.54	144.02
GREEDY	332.00	-	436.07	348.48	525.79	20.13	378.62	396.97
TABU1	27.48	32.57	-	65.49	81.69	3.64	61.33	64.02
TABU2	85.93	40.20	139.69	-	173.69	5.44	124.67	140.23
TABU3	23.20	13.78	36.27	27.50	-	1.20	48.58	46.10
GEN	420.28	288.63	593.65	573.97	587.85	-	460.15	520.27
CONGEN1	-	46.50	200.97	194.85	248.87	2.27	-	43.70
CONGEN2	-	49.63	178.07	166.05	199.30	1.80	6.32	-

Table 24: Average maximum slack of cuts in replication

	CONS	GRDY	TBU1	TBU2	TBU3	GEN	CG1	CG2
CONS	-	0.85	1.09	1.14	1.20	0.76	1.07	1.21
GREEDY	2.30	-	1.95	1.97	2.09	2.04	2.32	2.26
TABU1	0.65	0.48	-	0.43	0.59	0.36	0.73	0.73
TABU2	0.89	0.57	0.78	-	0.83	0.66	1.03	1.11
TABU3	0.40	0.28	0.27	0.32	-	0.23	0.55	0.56
GEN	1.87	1.58	2.00	1.94	1.95	-	2.03	2.03
CONGEN1	-	0.47	0.73	0.73	0.87	0.32	-	0.54
CONGEN2	-	0.39	0.57	0.55	0.60	0.21	0.21	-

Table 25: Average CPU time in replication

	CONS	GRDY	TBU1	TBU2	TBU3	GEN	CG1	CG2
CONS	0.12	0.97	2.65	1.99	2.91	0.44	1.89	2.28
GREEDY	4.59	-	4.00	2.43	4.68	2.53	5.97	7.00
TABU1	0.59	0.52	-	0.80	1.21	0.51	1.46	1.79
TABU2	1.27	0.74	1.89	-	2.49	0.52	2.33	2.59
TABU3	0.69	0.41	0.74	0.57	-	0.36	1.53	1.42
GEN	3.42	1.39	3.45	2.27	2.93	-	4.05	5.05
CONGEN1	0.13	0.67	1.93	1.51	2.67	0.37	-	1.11
CONGEN2	0.15	0.71	2.00	1.58	2.47	0.35	0.42	-

Table 26: Callback constructive heuristic on LITLIB

Name	L.B.	GAP	Cuts	Iter	SEP	Cplex	Min LB	Restarts
cmt101	818,05	0,24	1.788	78	2,62	9,23	818,05	1
eil30	508,50	4,78	121	10	0,01	0,03	508,50	1
eil33	832,33	0,32	912	39	0,12	0,35	830,00	2
eil51	511,75	1,78	628	30	0,18	0,37	511,75	1
eilA101	793,85	2,83	4.998	113	3,81	1,66	793,80	1
eilA76	780,36	6,43	12.200	115	4,11	6,61	778,15	3
eilC76	706,59	3,87	4.461	90	1,59	0,93	705,38	2
eilD76	659,87	3,95	7.236	83	2,17	0,89	658,60	3
fisher135	1.155,63	0,80	5.206	192	8,80	15,86	1.155,62	1
fisher45	723,50	0,07	541	26	0,07	0,04	723,00	2
fisher72	231,67	2,25	191	20	0,08	0,03	231,67	1

Table 27: Callback greedy heuristic on LITLIB

Name	L.B.	GAP	Cuts	Iter	SEP	Cplex	Min LB	Restarts
cmt101	817,93	0,25	20.580	73	34,46	37,92	816,25	4
eil30	508,50	4,78	1.148	13	0,11	0,14	505,17	2
eil33	832,50	0,30	1.636	26	0,87	0,12	832,50	1
eil51	509,70	2,17	4.057	33	3,59	1,55	508,02	2
eilA101	791,00	3,18	18.712	82	48,24	59,70	786,29	5
eilA76	771,13	7,54	13.323	64	36,32	46,21	770,44	3
eilC76	697,22	5,14	6.867	45	21,33	45,87	697,01	2
eilD76	647,87	5,70	12.723	56	21,17	34,48	643,95	5
fisher135	1.122,38	3,66	65.106	133	67,95	385,46	1.110,39	9
fisher45	717,67	0,87	1.590	17	0,27	0,06	717,50	2
fisher72	232,50	1,90	2.160	16	0,66	0,12	232,50	1

Table 28: Callback tabu1 heuristic on LITLIB

Name	L.B.	GAP	Cuts	Iter	Sep	Cplex	Min LB	Restarts
cmt101	819,25	0,09	6.019	37	8,32	13,73	819,25	1
eil30	508,50	4,78	241	8	0,02	0,06	508,50	1
eil33	833,50	0,18	1.987	24	0,22	0,35	832,20	2
eil51	514,45	1,26	2.209	28	0,94	0,80	514,45	1
eilA101	795,50	2,63	13.617	69	12,79	4,56	793,91	3
eilA76	787,92	5,53	10.641	67	10,79	3,53	787,27	2
eilC76	710,37	3,35	11.486	58	7,67	2,11	710,06	3
eilD76	661,12	3,77	17.038	61	5,06	2,05	660,24	7
fisher135	1.157,89	0,61	19.939	85	21,35	21,56	1.156,07	3
fisher45	722,20	0,25	972	17	0,16	0,07	722,11	2
fisher72	232,50	1,90	1.170	14	0,22	0,08	232,50	1

Table 29: Callback tabu2 heuristic on LITLIB

Name	L.B.	GAP	Cuts	Iter	Sep	Cplex	Min LB	Restarts
cmt101	819,50	0,06	14.797	60	9,79	10,14	819,00	3
eil30	508,50	4,78	255	8	0,03	0,10	508,50	1
eil33	833,40	0,19	2.839	27	0,34	0,51	824,87	3
eil51	514,45	1,26	4.363	41	0,76	0,25	514,25	3
eilA101	795,35	2,65	9.423	47	8,66	2,53	795,08	2
eilA76	786,09	5,74	7.058	50	6,69	1,98	786,09	1
eilC76	710,22	3,37	21.283	75	7,21	2,59	709,04	5
eilD76	660,87	3,80	11.623	47	3,53	1,75	660,33	4
fisher135	1.153,70	0,97	34.312	118	24,01	22,40	1.148,51	5
fisher45	724,00	-	1.105	22	0,17	0,05	721,11	2
fisher72	232,50	1,90	1.084	15	0,20	0,08	232,50	1

Table 30: Callback tabu3 heuristic on LITLIB

Name	L.B.	GAP	Cuts	Iter	Sep	Cplex	Min LB	Restarts
cmt101	819,33	0,08	6.474	58	10,22	5,79	819,25	1
eil30	508,50	4,78	241	8	0,04	0,14	508,50	1
eil33	833,50	0,18	1.251	22	0,26	0,36	833,50	1
eil51	514,44	1,26	2.427	28	0,79	0,14	514,43	1
eilA101	794,25	2,78	5.577	36	8,42	1,87	794,25	1
eilA76	788,08	5,51	21.339	89	14,47	5,06	787,27	5
eilC76	710,55	3,33	8.369	64	10,00	2,12	710,40	2
eilD76	660,89	3,80	8.041	44	4,33	1,30	660,61	3
fisher135	1.156,94	0,69	9.015	68	21,16	13,31	1.156,87	1
fisher45	722,20	0,25	972	17	0,19	0,05	722,11	2
fisher72	232,50	1,90	1.111	15	0,21	0,08	232,50	1

Table 31: Callback genetic heuristic on LITLIB

Name	L.B.	GAP	Cuts	Iter	Sep	Cplex	Min LB	Restarts
cmt101	804,16	1,93	2.361	162	20,81	7,67	801,52	4
eil30	508,50	4,78	92	11	0,07	0,16	508,50	1
eil33	831,75	0,39	232	36	0,61	1,33	831,75	1
eil51	512,90	1,55	1.297	73	2,37	1,60	510,65	7
eilA101	778,68	4,69	455	74	8,05	0,74	778,66	1
eilA76	769,30	7,76	2.230	146	17,83	1,45	767,79	6
eilC76	701,36	4,58	1.551	121	12,39	0,84	699,22	5
eilD76	654,66	4,71	1.050	78	6,52	0,40	645,68	6
fisher135	1.032,83	11,35	1.393	107	27,42	3,40	1.021,79	2
fisher45	724,00	-	346	46	0,31	0,06	722,00	2
fisher72	232,50	1,90	480	35	0,75	0,95	232,17	3

Table 32: Callback ConGen1 heuristic on LITLIB

Name	L.B.	GAP	Cuts	Iter	Sep	Cplex	Min LB	Restarts
cmt101	818,83	0,14	3.756	87	8,07	6,36	818,69	2
eil30	508,50	4,78	141	9	0,17	0,14	508,50	1
eil33	832,20	0,34	918	26	1,53	1,24	832,10	2
eil51	514,00	1,34	1.473	46	5,84	4,04	513,44	3
eilA101	794,53	2,75	5.108	101	23,06	14,61	794,52	1
eilA76	781,68	6,27	10.501	120	28,21	35,71	778,00	3
eilC76	707,96	3,68	6.674	119	22,34	37,65	707,76	2
eilD76	660,28	3,89	2.977	81	13,14	5,75	660,25	1
fisher135	1.155,70	0,80	8.884	202	37,69	69,68	1.154,90	2
fisher45	724,00	-	446	17	0,44	0,12	724,00	1
fisher72	232,50	1,90	388	18	0,48	0,12	232,50	1

Table 33: Callback ConGen2 heuristic on LITLIB

Name	L.B.	GAP	Cuts	Iter	Sep	Cplex	Min LB	Restarts
cmt101	818,84	0,14	3.023	84	8,08	5,52	818,53	2
eil30	508,50	4,78	169	11	0,36	0,26	508,50	1
eil33	833,50	0,18	698	34	1,78	0,80	833,50	1
eil51	514,39	1,27	1.646	49	10,06	6,02	514,08	2
eilA101	795,16	2,67	7.194	102	17,44	38,83	793,93	2
eilA76	783,29	6,08	13.236	123	20,97	65,16	782,71	3
eilC76	708,09	3,66	6.550	116	22,19	104,04	707,90	2
eilD76	660,42	3,87	3.503	83	10,11	9,72	660,42	1
fisher135	1.155,65	0,80	4.677	201	17,96	30,00	1.155,55	1
fisher45	724,00	-	467	20	0,60	0,19	724,00	1
fisher72	232,50	1,90	325	17	0,44	0,08	232,50	1

Table 34: Relative differences with iterative using CONS

Name	L.B.	GAP	Cuts	Iter	SEP	CPLEX
cmt101	-0.04%	21.53%	122%	1%	11%	-7%
eil30	-	-	41%	-44%	-50%	-25%
eil33	0.12%	-26.44%	304%	18%	140%	133%
eil51	0.44%	-19.57%	394%	7%	157%	147%
eilA101	0.04%	-1.42%	188%	41%	16%	-89%
eailA76	0.35%	-4.77%	716%	35%	100%	-16%
eilC76	0.05%	-1.17%	348%	36%	26%	-72%
eilD76	0.25%	-5.82%	581%	9%	71%	-81%
fisher135	0.20%	-20.02%	55%	-21%	-36%	-82%
fisher45	0.49%	-87.50%	171%	-19%	-	-89%
fisher72	-0.36%	18.52%	41%	-9%	-11%	-90%

Table 35: Relative differences with iterative using GREEDY

Name	L.B.	GAP	Cuts	Iter	SEP	CPLEX
cmt101	0.15%	-37.81%	762%	35%	53%	62%
eil30	-	-	126%	-43%	120%	8%
eil33	0.36%	-54.17%	59%	-21%	21%	-72%
eil51	0.72%	-24.28%	381%	-13%	-40%	32%
eilA101	2.61%	-43.60%	1175%	122%	158%	781%
eailA76	0.13%	-1.56%	782%	39%	64%	693%
eilC76	2.60%	-31.86%	502%	18%	56%	1532%
eilD76	-0.06%	0.94%	1469%	167%	370%	3248%
fisher135	3.15%	-44.57%	1583%	15%	24%	762%
fisher45	-0.82%	1483.33%	99%	-43%	-33%	-90%
fisher72	-	-	179%	-36%	-6%	-89%

Table 36: Relative differences with iterative using TABU1

Name	L.B.	GAP	Cuts	Iter	SEP	CPLEX
cmt101	-0.03%	50.00%	160%	-33%	-37%	-33%
eil30	-	-	1%	-68%	-33%	-25%
eil33	-	-	206%	9%	22%	67%
eil51	0.06%	-4.34%	154%	4%	2%	-14%
eilA101	0.13%	-4.74%	645%	57%	35%	-65%
eailA76	0.11%	-1.86%	406%	24%	4%	-72%
eilC76	0.11%	-3.12%	580%	35%	14%	-69%
eilD76	0.15%	-3.69%	1292%	79%	55%	-44%
fisher135	0.33%	-35.01%	466%	-6%	-28%	-36%
fisher45	-0.25%	-	111%	-23%	45%	-71%
fisher72	-	-	198%	-13%	57%	-78%

Table 37: Relative differences with iterative using TABU2

Name	L.B.	GAP	Cuts	Iter	SEP	CPLEX
cmt101	0.01%	-8.49%	566%	11%	-8%	-59%
eil30	0.10%	-1.92%	24%	-64%	-	43%
eil33	0.10%	-35.07%	320%	17%	113%	132%
eil51	-0.01%	0.70%	467%	58%	38%	-70%
eilA101	0.30%	-9.93%	472%	34%	32%	-70%
eailA76	-0.13%	2.19%	126%	-32%	-27%	-89%
eilC76	0.07%	-1.86%	752%	25%	14%	-81%
eilD76	0.07%	-1.78%	784%	31%	27%	-56%
fisher135	0.08%	-7.60%	1007%	59%	37%	-26%
fisher45	-	-	175%	5%	89%	-77%
fisher72	-	-	164%	-25%	33%	-83%

Table 38: Relative differences with iterative using TABU3

Name	L.B.	GAP	Cuts	Iter	SEP	CPLEX
cmt101	-0.02%	34.57%	207%	9%	-27%	-80%
eil30	-	-	-2%	-71%	33%	40%
eil33	-	-	52%	-15%	-4%	13%
eil51	0.01%	-1.05%	177%	-13%	-28%	-89%
eilA101	0.08%	-2.73%	252%	3%	1%	-79%
eailA76	0.13%	-2.15%	888%	44%	21%	-69%
eilC76	0.11%	-3.08%	426%	64%	52%	-63%
eilD76	0.20%	-4.78%	599%	38%	29%	-59%
fisher135	0.62%	-47.04%	245%	1%	4%	-45%
fisher45	-0.25%	-	107%	-23%	46%	-79%
fisher72	-	-	118%	-29%	5%	-87%

Table 39: Relative differences with iterative using GEN

Name	L.B.	GAP	Cuts	Iter	SEP	CPLEX
cmt101	-0.60%	44.70%	432%	42%	77%	-39%
eil30	0.13%	-2.55%	-9%	-69%	-13%	60%
eil33	0.09%	-18.75%	48%	-16%	79%	639%
eil51	0.53%	-24.85%	1426%	109%	160%	662%
eilA101	-0.22%	4.63%	90%	139%	213%	-53%
eailA76	-0.29%	3.61%	593%	35%	77%	-54%
eilC76	0.01%	-0.19%	642%	59%	79%	-62%
eilD76	0.76%	-13.26%	544%	34%	68%	-68%
fisher135	-5.74%	90.74%	97%	-5%	16%	-91%
fisher45	0.23%	-100.00%	196%	12%	55%	-76%
fisher72	-	-	344%	25%	226%	206%

Table 40: Relative differences with iterative using CONGEN1

Name	L.B.	GAP	Cuts	Iter	SEP	CPLEX
cmt101	0.04%	-20.37%	258%	43%	46%	-38%
eil30	-	-	-8%	-55%	-32%	130%
eil33	0.13%	-28.00%	233%	13%	226%	1136%
eil51	-	-	201%	31%	45%	411%
eilA101	0.09%	-3.21%	172%	25%	53%	7%
eailA76	0.16%	-2.38%	624%	48%	161%	304%
eilC76	0.25%	-6.17%	451%	68%	44%	446%
eilD76	0.13%	-3.00%	127%	23%	25%	-11%
fisher135	-0.07%	9.66%	200%	-14%	10%	-40%
fisher45	-	-	96%	-6%	-15%	-25%
fisher72	-	-	138%	13%	4%	-45%

Table 41: Relative differences with iterative using CONGEN2

Name	L.B.	GAP	Cuts	Iter	SEP	CPLEX
cmt101	-0.01%	9.14%	261%	33%	108%	-29%
eil30	-	-	10%	-35%	57%	544%
eil33	0.05%	-20.59%	99%	10%	21%	300%
eil51	0.08%	-5.56%	366%	48%	59%	1238%
eilA101	0.01%	-0.21%	257%	1%	1%	53%
eailA76	0.30%	-4.38%	695%	71%	132%	772%
eilC76	0.09%	-2.25%	244%	14%	29%	517%
eilD76	0.06%	-1.57%	147%	9%	-1%	42%
fisher135	-0.08%	11.31%	59%	-19%	-49%	-71%
fisher45	-	-	52%	-43%	-38%	-51%
fisher72	-	-	80%	13%	2%	-58%

Table 42: Summary of callback heuristics on LITLIB

Name	Avg GAP	Max GAP	Best	Cuts	Iter	SEP	CPLEX	Restarts
CONS	2.72	6.43	2	3,774.10	77.00	2.35	3.60	1.60
GREEDY	3.23	7.54	1	13,361.91	49.45	21.34	55.60	3.27
TABU1	2.21	5.53	5	7,756.27	42.55	6.14	4.45	2.36
TABU2	2.23	5.51	5	5,892.45	40.82	6.37	2.75	1.73
TABU3	2.25	5.74	4	9,831.09	46.36	5.58	3.85	2.73
GEN	3.97	11.35	1	1,044.27	80.82	8.83	1.69	3.45
CONGEN1	2.35	6.27	3	3,751.45	75.09	12.82	15.95	1.73
CONGEN2	2.30	6.08	4	3,771.64	76.36	10.00	23.69	1.55

Table 43: Summary of callback heuristics on ALIB

Name	Avg GAP	Max GAP	Best	Cuts	Iter	SEP	CPLEX	Restarts
CONS	3.18	5.79	1	3,674.31	63.46	0.80	0.65	2.31
GREEDY	3.77	7.59	-	8,551.12	51.15	13.15	16.04	2.77
TABU1	2.44	4.56	10	5,791.19	40.50	2.55	1.16	2.04
TABU2	2.43	4.58	15	6,252.54	43.12	3.03	1.20	2.08
TABU3	2.52	4.61	3	9,568.04	48.54	2.43	1.61	3.19
GEN	4.16	6.95	-	1,467.81	116.69	6.71	0.59	4.15
CONGEN1	2.82	5.56	-	4,625.00	62.81	8.37	3.63	3.04
CONGEN2	2.74	5.76	4	4,200.46	70.92	10.94	12.88	2.50

Table 44: Summary of callback heuristics on BLIB

Name	Avg GAP	Max GAP	Best	Cuts	Iter	SEP	CPLEX	Restarts
CONS	1.15	3.45	2	1,907.00	54.70	0.47	0.39	1.70
GREEDY	1.22	3.90	3	8,993.48	48.43	8.83	7.79	2.39
TABU1	0.82	2.87	11	5,177.13	36.09	1.82	1.18	1.70
TABU2	0.78	2.65	9	5,227.57	38.35	2.13	0.96	1.52
TABU3	0.89	3.39	3	6,924.39	42.78	1.48	0.92	2.52
GEN	2.08	6.88	2	1,230.65	91.96	4.07	0.52	3.04
CONGEN1	1.00	2.70	5	2,783.26	51.04	4.39	1.71	2.04
CONGEN2	0.84	2.61	8	2,697.74	60.22	7.92	3.95	1.96

Table 45: Summary of callback heuristics on all problem instances

Name	Avg GAP	Max GAP	Best	Cuts	Iter	SEP	CPLEX	Restarts
CONS	2.31	6.43	5	3,002.27	62.34	0.93	1.05	1.95
GREEDY	2.69	7.59	4	9,602.67	49.80	13.00	20.13	2.72
TABU1	1.78	5.53	26	5,916.07	39.18	2.92	1.77	1.97
TABU2	1.76	5.51	29	5,793.62	40.87	3.30	1.39	1.80
TABU3	1.85	5.74	10	8,602.87	45.93	2.64	1.76	2.85
GEN	3.33	11.35	3	1,299.25	100.63	6.09	0.76	3.60
CONGEN1	2.03	6.27	8	3,758.85	60.55	7.66	5.15	2.42
CONGEN2	1.94	6.08	16	3,545.80	67.82	9.61	11.44	2.12

Table 46: Relative differences with iterative on LITLIB

Name	Avg GAP	Max GAP	Best	Cuts	Iter	SEP	CPLEX
CONS	4.04%	-4.77%	-33%	307%	11%	7%	-70%
GREEDY	-18.87%	-1.56%	-50%	871%	18%	63%	595%
TABU1	-2.39%	-1.86%	-	458%	8%	-9%	-47%
TABU2	-2.94%	-2.06%	-	296%	1%	30%	-70%
TABU3	-2.70%	2.10%	-20%	665%	22%	-7%	-53%
GEN	12.31%	51.53%	-	333%	30%	60%	-69%
CONGEN1	-2.64%	-2.38%	-	269%	17%	44%	7%
CONGEN2	-1.77%	-4.38%	-20%	242%	6%	7%	55%

Table 47: Relative differences with iterative on ALIB

Name	Avg GAP	Max GAP	Best	Cuts	Iter	SEP	CPLEX
CONS	-8.92%	5.12%	-	368%	12%	31%	-76%
GREEDY	-30.67%	-35.56%	-	574%	36%	114%	532%
TABU1	-14.72%	-13.55%	67%	340%	14%	10%	-70%
TABU2	-21.12%	-10.26%	650%	381%	24%	79%	-66%
TABU3	0.09%	1.58%	-81%	605%	23%	-7%	-65%
GEN	-15.92%	-25.65%	-	571%	55%	101%	-64%
CONGEN1	-5.33%	-0.31%	-100%	489%	28%	44%	34%
CONGEN2	-3.02%	4.40%	33%	379%	22%	41%	249%

Table 48: Relative differences with iterative on BLIB

Name	Avg GAP	Max GAP	Best	Cuts	Iter	SEP	CPLEX
CONS	-9.97%	10.11%	-	195%	1%	1%	-88%
GREEDY	-32.61%	-25.69%	50%	475%	11%	74%	51%
TABU1	-20.35%	11.21%	10%	292%	-12%	-16%	-77%
TABU2	-44.22%	-74.29%	200%	353%	4%	28%	-72%
TABU3	-0.40%	30.94%	-67%	409%	3%	-47%	-80%
GEN	-14.55%	-14.06%	100%	386%	44%	86%	-72%
CONGEN1	-10.19%	-12.99%	-	273%	10%	9%	-43%
CONGEN2	-11.64%	-1.58%	14%	249%	18%	67%	10%

9 References

Aiko, S., Itabashi, R., Seo, T., Kusakabe, T., & Asakura, Y. (2017). Social benefit of optimal ride-share transport with given travelers' activity patterns. *Transportation Research Procedia*, *27*, 261-269.

Anderson, R. (2012). *Object oriented integer programming.* Retrieved 2019-06-01, from `https://app.assembla.com/spaces/oocplex/subversion/source`

Applegate, D., E. Bixby, R., Chvátal, V., J. Cook, W., Espinoza, D., Goycoolea, M., & Helsgaun, K. (2009, 1). Certification of an optimal tsp tour through 85,900 cities. *Oper. Res. Lett.*, *37*, 11-15.

Arnold, G.-M. S. K., F. (2019). A genetic algorithm for the bin packing problem. *Computers Operations Research*, *94*, 32-42.

Augerat, Belenguer, Benavent, Corberin, & Naddef. (1998). Separating capacity constraints in the cvrp using tabu search. *European Journal of Operational Research*, *106*, 546-557.

Baldacci, R., Hadjiconstantinou, E., & Mingozzi, A. (2004). An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations Research*, *52*(5), 723-738.

Beasley, J. (1983). Route first—cluster second methods for vehicle routing. *Omega*, *11*(4), 403-408.

Bullnheimer, B., Hartl, R., & Strauss, C. (1999). An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research*, *89*, 319-328.

Christofides, N., & Eilon, S. (1969). An algorithm for the vehicle-dispatching problem. *Journal of the Operational Research Society*, *20*(3), 309–318.

Clarke, G., & Wright, J. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, *12*(4), 519-643.

Cordeau, J., Gendreau, M., & Laporte, G. (1998). A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, *30*(2).

Creput, J.-C., Hajjam, A., Koukam, A., & Kuhn, O. (2011). Dynamic vehicle routing problem for medical emergency management. *Self Organizing Maps - Applications and Novel Algorithm Design*.

Dantzig, G. B., & Ramser, J. (1959). The truck dispatching problem. *Management Science*, *6*(1), 80-91.

Denis Naddef, D., & Rinaldi, G. (2002). Branch-and-cut algorithms for the capacitated vrp. *The vehicle routing problem*.

Erdogan, S., & Miller-Hooks, E. (2012). A green vehicle routing problem. *Transportation Research Part E: Logistics and Transportation Review*, *48*(1), 100-114.

Gendreau, M., Hertz, A., & Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management Science*, *40*(10), 1207-1393.

Gharib, Z., Bozorgi-Amiri, A., Tavakkoli, R., & Najafi, E. (2018). A cluster-based emergency vehicle routing problem in disaster with reliability. *Transactions E: Industrial Engineering*, *25*(4), 2312-2330.

Hasle, G., Lie, K., & Quak, E. (2007). *Geometric modelling, numerical simulation, and optimization.* Springer.

Java framework for multiobjective optimization. (2012). Retrieved 2019-06-01, from `http://moeaframework.org/`

Java universal network/graph framework. (n.d.). Retrieved from `http://jung.sourceforge.net/, journal=JUNG, urldate = 2019-06-01, year=2012`

Jozefowiez, N., Semet, F., & Talbi, E. (2005). Enhancements of nsga ii and its application to the vehicle routing problem with route balancing. *Artificial Evolution. Lecture Notes in Computer Science*, *3871*, 131-142.

Junkermeier, J. (2015). A genetic algorithm for the bin packing problem.

Kalyanmoy, D. (2001). *Multi-objective optimization using evolutionary algorithms.* Wiley.

Kalyanmoy, D., & Jain, H. (2014). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, *18*(4).

Kumar, S., & Panneerselvam, R. (2012). A survey on the vehicle routing problem and its variants. *Intelligent Information Management*, *4*, 66-74.

Laporte, G., Nobert, Y., & Desrochers, M. (1985). Optimal routing under capacity and distance restrictions. *Operations Research*, *33*(5), 1050-1073.

Lia, R., Xie, X., Augusto, V., & Rodriguez, C. (2013). Heuristic algorithms for a vehicle routing problem with simultaneous delivery and pickup and time windows in home health care. *European Journal of Operational Research*, *230*(3), 475-486.

Ralphs, K.-L. P. W., T. (2003). Certification of an optimal tsp tour through 85,900 cities. *Mathematical Programming*, *94*, 343–359.

Schuijbroek, J., Hampshire, R., & van Hoeve, W.-J. (2017). Inventory rebalancing and vehicle routing in bike sharing systems. *European Journal of Operational Research*, *257*(3), 992-1004.

Soysal, M., Bloemhof-Ruwaard, M., & Bektas, T. (2015). The time-dependent two-echelon capacitated vehicle routing problem with environmental considerations. *International Journal of Production Economics*, *164*, 366-378.

Thangiah, S., Potvin, J., & Sun, T. (1996). Heuristic approaches to vehicle routing with backhauls and time windows. *Computers & Operations Research*, *23*(11), 1043-1057.

Toth, P., & Vigo, D. (2015). *Vehicle routing: problems, methods, and applications.* Society for Industrial and Applied Mathematics.

Ubeda, S., Arcelus, F., & Faulin, J. (2011). Green logistics at eroski: A case study. *International Journal of Production Economics*, *131*(1), 44-51.

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Subramanian, A., & Vidal, T. (2016). New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, *257*(3).

Wayne, K., & Sedgewick, R. (2011). *Algorithms (4th edition).* Addison-Wesley. Retrieved 2019-06-01, from `https://algs4.cs.princeton.edu/`