Analysis of column-based heuristics for Fixed Order Routing

Sebastiaan Hulst

(2855964)

July 8, 2025

Thesis MSc EOR - Operations Research
Specialization: Operations Research Theory

Thesis committee:
Dr. R.A. Sitters (supervisor)
(MSc.) S.J.G. Miltenburg (co-reader)

Abstract

In this thesis, we compare the performance of Column Generation (CG) and Columnwise Neighborhood Search (CNS) in terms of objective value and speed to an exact ILP algorithm in solving the C-capacitated Fixed Order Routing Problem, a less-studied variant of the Capacitated Vehicle Routing Problem (CVRP). Additionally, we aim to enhance these algorithms by utilizing higher-quality initial solution methods and route pruning techniques. In our experiments, we compute a solution with the CG or CNS algorithm for ten different instances and then compute a solution with the exact ILP algorithm with a time limit. The average ratio of the objective value achieved with the CG or CNS algorithm to the objective value achieved with the ILP algorithm is compared. Our results show that the CG and CNS algorithm outperforms the exact ILP algorithm in terms of the average ratio of objective value in roughly 75% of the instances. We found no significant improvement in solution quality or computational efficiency with higher-quality initial solutions. Our pruning techniques greatly improve the efficiency of the CG algorithm without significantly decreasing the solution quality. However, similar results were not found with the CNS algorithm.

Keywords: Capacitated Vehicle Routing Problem, Fixed Order Routing, Column Generation, Columnwise Neighborhood Search, Set Partitioning Problem.

Contents

1 Introduction

In modern society, routing is an omnipresent problem. Finding the shortest route to a singular destination is a well-researched problem with efficient exact solutions (Taillard, 2023). The problems businesses and other large organisations encounter are more complex with additional constraints and objectives. All of these fall under the class of Vehicle Routing Problems (VRP) where the goal is to find a collection of routes that adheres to a set of constraints and optimizes an objective. The simplest case has no constraints and has the objective of minimizing the total distance for a single vehicle to reach a set of destinations. This reduces to the famous Traveling Salesman Problem (TSP) (Taillard, 2023). The standard VRP also has the objective to minimize total distance but allows the use of multiple vehicles to cover the destinations. To model more real-life scenarios, we can add a capacity constraint to the vehicles or time windows in which a vehicle has to arrive at the destination (Golden et al., 2008). Modern Operations Research has made large advancements in theoretical and practical results for these problems. The standard VRP and its extensions are NP-hard. Therefore in the literature, attention is often devoted to finding heuristics and approximate methods to compute near-optimal solutions in reasonable time frames (Anbuudayasankar et al., 2014).

The standard VRP has no constraints on the order of the destinations in the vehicles. The destinations can be split between the vehicles in any order to optimize the chosen objective. However, there can also be scenarios where the order of the destinations is more rigid. This can manifest as time windows, which give a lower and upper bound for arrival at a location. This variant is referred to as Vehicle Routing Problem with Time Windows (VRPTW). Another way to impose an order is to allow sets of destinations to only be traveled in one way. This can be the case for certain production processes or loading areas where it is infeasible or costly to reorder destinations completely. This is referred to as Fixed Order Routing (FOR). This variant has gotten less attention. In this thesis, we will be studying a problem referred to as C-capacitated Fixed Order Routing, usually shortened to C-cap FOR. This means that each vehicle has a capacity of $C \in \mathbb{Z}_{\geq 2}$ and there is a constraint on the order of the requests. Our goal is to apply two column-based heuristic methods to C-cap FOR and compare their performance to an exact Integer Linear Program approach. Additionally, we will attempt to improve these methods with better initial solution methods and route pruning techniques.

2 Literature Review

2.1 Capacitated Vehicle Routing Problem

The origins of the study of vehicle routing trace back to a paper by Dantzig and Ramser, 1959, in which they studied "*the truck dispatching problem*". In this paper, they formulate what we now would call the Capacitated Vehicle Routing Problem (CVRP) in the context of a fleet of gasoline delivery trucks servicing a number of service stations from a central terminal. Through a method based on a linear programming formulation and hand calculations, they are able to achieve a solution close to the optimal for twelve service stations. A few years later Clarke and Wright, 1964, created an algorithm based on saving distance by combining routes. Many advancements have been made since then in terms of techniques and computing power to allow us to tackle larger problems. These advancements include new heuristic methods that can be applied to CVRP such as *GRASP* and *Simulated Annealing*, first proposed for use in combinatorial optimization by Feo and Resende, 1989, and Kirkpatrick, 1984, respectively. All of these heuristics and a few others were studied in a paper by Sandoya Sánchez et al., 2023. They found that Clarke and Wright heuristics and GRASP metaheuristics generated better quality results than *sweep* and simulated annealing respectively overall. However, the paper also points out that is worth researching the positioning and grouping

of the clients since there were instances where sweep and simulated annealing performed better. Further research into the distribution of the clients could allow a better understanding of the strength of each of the heuristics.

Another area that has seen many advancements are linear relaxations of CVRP. The CVRP is NP-hard in the strong sense, so there does not exist a Linear Program (LP) formulation that directly gives an optimal solution. We do have Integer Linear Program (ILP) formulations of the CVRP which can be relaxed to allow non-integer solutions. These linear relaxations can then be used to generate lower bounds and to design heuristics as can be seen in the work of Huang et al., 2022. A recent paper by Letchford and Salazar González, 2019, explores several existing ILP formulations of the CVRP and proposes two new ones. They find that their new formulations allow better lower bounds, but also increase the time spent solving the relaxations.

All of these advancements allow us to further expand the Capacitated Vehicle Routing Problem to model more real-life scenarios such as non-uniform speeds among the fleet, stochastic demands, and complex geometries for aerial routing in drones (Gørtz et al., 2016, Lei et al., 2011, Gao and Yu, 2021).

2.2 Fixed Order Routing

There are few papers studying a fixed-order framework in vehicle routing. Constraints on order are more common in game theoretical and online combinatorial optimization problems. In congestion and scheduling games, agents want access to limited resources. When multiple agents want a resource like a road or a machine, there needs to be an order in which the agents get access. Ackermann et al., 2008, first applied resources with preferences to the classical congestion game. They restrict their work to a type of congestion game where the agents are interested in singular resources, not combinations of resources. These types of games are called *singleton* congestion games. They prove that these games are potential games with pure Nash equilibria that can be found in polynomial time. Gourvès et al., 2015, extended these results to singleton congestion games with capacities so each resource can only be used a certain amount of times.

In machine scheduling, Christodoulou et al., 2009, introduced job orders in coordination mechanisms to improve the bounds on the price of anarchy. For m machines, they prove that ordering jobs by the Longest Processing Time (LPT) rule gives a price of anarchy of $\frac{4}{3} - \frac{1}{3m}$, which improves on the previous bound of $\Theta(\frac{\log m}{\log \log m})$. Recently, Vijayalakshmi et al., 2021, considered scheduling games where each machine has its priority list of jobs, so there is no global job order. They prove that it is NP-hard to decide if a pure Nash equilibrium exists and give bounds on the inefficiency of Nash equilibria.

A fixed order can make some problems easier. This can be seen in the comparison between the *offline* and *online* versions of the k-server problem. In this problem, we are given a metric space M in which requests will arrive and we have k servers that have to serve the requests (Chrobak et al., 1990). The online version of the problem has requests coming in one-by-one and servers have to be sent to a request without knowledge of the next requests. In the offline version the entire set of requests is known at the start and the best division of servers can be determined with perfect knowledge. The servers have no limit on the amount of requests they can serve in contrast to the problem we will study later. The offline version of the k-server problem is the same as a FOR problem with k vehicles and no capacity constraint. Chrobak et al., 1990, prove that this offline k-server problem can be solved in polynomial time using a minimum cost flow of maximum value formulation of the problem.

Unlike the k-server problem, the CVRP with a fixed order constraint cannot be solved in polynomial time in general. Miltenburg et al., 2025, showed that the Fixed Order version of CVRP is APX-hard. The class of APX-hard problems are a subset of NP-hard problems that admit

polynomial time approximation algorithms with constant approximation ratio (Ausiello et al., 2000). Their paper covers a $(2 - \frac{1}{C})$-approximation algorithm for C-cap FOR, approximation schemes for k-FOR, and complexity of FOR on line graphs and trees. Inspired by this paper, Gouweleeuw, 2024 wrote a bachelor thesis about Fixed Order Routing. She compared a few variations of Column Generation to two different exact algorithms based on ILP formulations of the C-capacitated Fixed Order Routing Problem. In Section 2.3 their work will be discussed in greater detail.

2.3 Column Generation

Many problems in Operations Research can be formulated as Linear Programs. Problems based on graphs will usually construct an LP based on the arcs between vertices. A solution is then a collection of arcs that together form the optimal routes. For many of these problems, there exists an alternative formulation that turns the problem into an instance of the *Set Partitioning Problem* (SPP). This is a variant of the set covering problem with equality constraints. Instead of arcs between vertices, these formulations are based on a set of feasible routes. Garfinkel and Nemhauser, 1969, proposed an enumeration algorithm for the SPP and showed it to be more efficient than corresponding LPs for problems in districting, crew scheduling, and assignment.

A secondary motivation for formulating a problem as a Set Partitioning Problem is that its linear relaxation is usually stronger than the original Integer Linear Program formulation. This makes it more suitable to be used in heuristics. The SPP formulation makes use of a set Ω that contains all feasible routes and this can grow exponentially with the size of the instance. This issue can be handled using the technique of *column generation* (Feillet, 2010). Instead of solving the SPP on Ω, we solve the linear relaxation of the SPP formulation with a small subset $\Omega_1 \subset \Omega$ and add new variables, usually referred to as columns, by solving a subproblem, add them to Ω_1 and solve the relaxation again until no new columns are found by solving the subproblem. Further details on this method are provided in Section 5.2.

Due to this connection between Column Generation and the Set Partitioning Problem, problems where a reformulation into SPP can be applied are also good candidates for Column Generation. The book "Column Generation" by Desaulniers et al., 2005, compared papers where this technique is applied to different problems in scheduling and routing. Kallehauge et al., 2005, applied Column Generation to VRPTW. In this problem, a vehicle must arrive at the location before the upper bound of the time window and if it arrives before the lower bound, it must wait for the time window to open. They present solutions obtained from several exact algorithms for VRPTW based on column generation on a set of instances called "*The Solomon Instances*". These instances consist of 100 customers and they are distributed using a uniform distribution, in clusters or a combination of the two. At the time of the paper, 25% of the instances had no exact optimal solution. Another approach for VRPTW is shown in the paper by Danna and Le Pape, 2005, where they used Column Generation for branch-and-price heuristics. These heuristics make use of branch-and-bound in combination with local search techniques. In the same Solomon Instances, they showed that their heuristics are as effective and sometimes more effective at finding solutions than a pure branch-and-price approach. The field of scheduling also contains many NP-hard problems. When we want to schedule jobs on m parallel machines, we can minimize for the sum of completion times, sum of number of tardy jobs or sum of late work, denoted as $P_m|\cdot|\sum w_jC_j, P_m|\cdot|\sum w_jU_j, P_m|\cdot|\sum w_jV_j$ respectively (Bruno et al., 1974, Ho and Chang, 1995, Chen et al., 2015). van den Akker et al., 2005, explored column generation for these machine scheduling problems and some variations. They first described the SPP formulation and the subproblem for $P_m|\cdot|\sum w_jC_j$ and afterward discussed applying it to the different objectives and variations. A different scheduling problem was explored in a paper by Fügenschuh, 2011. He applied the column generation approach among a few other techniques to a bus scheduling problem. He found that this method performed better where the standard branch-and-cut approach fell short.

Last year a bachelor thesis was written to compare algorithms for the C-capacitated FOR by Gouweleeuw, 2024. It compared the performance of Column Generation to solving its ILP formulation directly on small instances ranging from $n = 8$ to $n = 50$ total destinations and a capacity of $C = 4$. In general, for these small instances, both techniques achieve the same optimal solution even when the ILP is limited in computation time. When removing the time limit, however, they found that the column generation method was more efficient. The ILP methods took longer to compute their final solution compared to the column generation method. From this, it can be concluded that even for small instances exact optimal solutions are hard to compute. It is also mentioned that they were not able to run instances of size $n = 100$ or $n = 150$ because it took too long.

2.4 Columnwise Neighborhood Search

Another approach to dealing with the exponential growth of Ω is to use local search techniques. The SPP formulation is solved on a set of initial solutions $\Omega_1 \subset \Omega$. To generate more columns, new routes are generated using a predefined set of rules for every route that is a part of the optimal solution. Together the new routes are called a neighborhood. That is why this method is referred to as *Columnwise Neighborhood Search* (CNS). We add every optimal route's neighborhood into Ω_1 and solve the SPP again. We keep track of all the neighborhoods in each iteration and stop when all the new routes generated have been seen before. The specifics of the implementation can be seen in Section 5.3.

Neighborhood Search has a long history as a heuristic for solving complex problems, for example *Variable Neighborhood Search* (VNS) and *Large Neighborhood Search* (LNS). VNS was proposed by Mladenović and Hansen, 1997 for TSP and has evolved into a popular heuristic for a wide range of problems from scheduling, districting, and facility layout (Sleptchenko et al., 2023). Shaw, 1998, introduced LNS for the vehicle routing problems. It has now expanded into many different areas like bin-packing, electric vehicle fleet scheduling and many more (Ekici, 2023, Wolfinger, 2021, Winter and Musliu, 2021, Limmer et al., 2023). The general purpose nature of these neighborhood techniques makes them attractive for complex problems that do not have specialised heuristics.

The idea of CNS is relatively simple but only recently was it applied to SPP in the paper by Jagtenberg et al., 2020, for the "*VeRoLog Solver Challenger 2019*". The lack of research on this heuristic is likely due to the computational cost. The ILP grows with each iteration and needs to be solved optimally. Recent developments in the solvers CPLEX and Gurobi by IBM, 2025, and Gurobi Optimization, LLC, 2025, respectively have made this heuristic feasible for more complex problems. To implement Columnwise Neighborhood Search, one needs to have an SPP formulation, methods to generate initial solutions, and rules to generate a neighborhood. Inspired by the work of Jagtenberg et al., 2020, a few master theses have been written about applying this heuristic to different problems. Swart, 2022, applied it to the electric vehicle routing problem, Briefjes, 2023, to the VRPTW and Schel, 2023, to VRP with multiple depots. The general idea of Columnwise Neighborhood Search is quite simple but as can be seen in their work, it is important to design proper initial solutions and rules for generating new feasible routes. They achieved good solutions for their complex problems, so it is a worthy technique to apply to new problems.

3 Research Questions

We will contribute to the knowledge about the C-capacitated Fixed Order Routing problem by comparing two heuristics, Column Generation and Columnwise Neighborhood Search, to an exact ILP algorithm. Additionally, we attempt to improve these algorithms via different initial solution

methods and route pruning techniques. This leads to the following research questions:

Do column-based heuristics perform better than an exact ILP algorithm?

- *Under what conditions of instance size and capacity do Column Generation and Columnwise Neighborhood Search offer superior performance in terms of objective value and computational efficiency compared to an exact Integer Linear Program for the C-capacitated Fixed Order Routing problem?*
- *To what extent does the quality of initial solutions influence the performance in terms of objective value and computational efficiency of Column Generation and Columnwise Neighborhood Search for the C-capacitated Fixed Order Routing problem?*
- *How do various pruning strategies impact the solution quality and computational efficiency of Column Generation and Columnwise Neighborhood Search for the C-capacitated Fixed Order Routing problem?*

4 Problem Description

4.1 ILP Formulation

To define the C-capacitated Fixed Order Routing problem, we use a directed graph $G = (V, A)$ where V is the set of vertices consisting of n destinations, corresponding to the n people waiting in the queue $\{1, \dots, i, \dots n\} : V = \{v_1, \dots, v_i, \dots v_n\}$. The number given to their destination is equal to their position in the queue. Therefore, any set of destinations, for example $\{v_5, v_2, v_7\}$, will be traversed in order of increasing i. The arcs between destinations is given by $A = \{(v_i, v_j) : v_i, v_j \in V, v_i \neq v_j\}$ and each arc has a cost $d_{ij} \in \mathbb{R}_{>0}$. Each cost d_{ij} is equal to the Euclidean distance between v_i and v_j. Note that this implies $d_{ij} = d_{ji}$ for each pair $(v_i, v_j) \in A$. Furthermore, $d_{ij} \in \mathbb{R}_{>0}$ for all pairs (v_i, v_j) implies that no two vertices are in the same position. Each vehicle has a capacity of C, meaning that they can travel to at most C destinations. Each vehicle starts at the central location and must return to it after visiting its last destination. We can formulate an ILP based on these constraints by modifying the ILP without return described in the bachelor thesis by Paschalidis, 2023:

$$\min \quad \sum_{i=0}^{n} \sum_{j=0}^{n} d_{ij} x_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{j=0}^{i-1} x_{ji} = 1 \qquad 1 \leq i \leq n \tag{2}$$

$$\sum_{j=i+1}^{n} x_{ij} + x_{i0} = 1 \qquad 1 \leq i \leq n \tag{3}$$

$$x_{ij} = 0 \qquad 2 \leq i \leq n, \qquad 1 \leq j \leq i \tag{4}$$

$$(C+1) \cdot (x_{ij} - 1) \leq y_j - y_i - 1 \qquad 0 \leq i \leq n-1, \qquad i+1 \leq j \leq n \tag{5}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i,j) \in A \tag{6}$$

$$y_i \in \{1, 2, \dots, C\} \qquad \forall i \in V \tag{7}$$

The decision variable x_{ij} is equal to 1, also called *active*, if the arc (v_i, v_j) is used by any taxi route. The variable y_i is equal to the amount of people in the taxi when that taxi visits v_i. To ensure that the arcs combine into a route, the constraints (2), (3), (4) ensure that there is only one

arc leading into each vertex, one arc going out of each vertex and that there are no backward arcs respectively. Constraint (5) allows only routes that respect the capacity of each taxi and the order of the queue. At the top of the ILP (1), you can see that the objective is the minimize the sum of the distance of active arcs.

4.2 SSP Formulation

As discussed in Section 2.3, we reformulate the ILP seen in Section 4.1 to a Set Partitioning Problem. The following formulation was described in an assignment given by Sitters, 2023.

$$\min \quad \sum_{t \in T} d_t x_t \tag{8}$$

$$\text{s.t.} \quad \sum_{t \in T \text{ with } i \in t} x_t \geq 1 \qquad \forall i \in \{1, \ldots, n\} \tag{9}$$

$$x_t \in \{0, 1\} \qquad \forall t \in T \tag{10}$$

The problem is defined on the same directed graph as before, but now the decision variable x_t is equal to 1 if a route $t \in T$ is used. This formulation relies on a set T which contains all feasible routes. Only routes that satisfy the same constraints as in Section 4.1 are in T and constraint (9) ensures that each destination gets visited. The objective (8) is the minimize the sum of the distance of the active routes.

4.3 Instance generation

There does not exist a shared set of instances, like the Solomon instances, on which the performance of algorithms can be compared for the C-capacitated Fixed Order Routing problem. That is why we will be generating instances with the random number generator of the NumPy Python package (NumPy Developers, 2024). This will be used to generate n points that are uniformly distributed on the $[0, 1]^2$-plane. The central depot $\mathbf{0}$ is located at $(\frac{1}{2}, \frac{1}{2})$. In Figure 1 you can see an example.

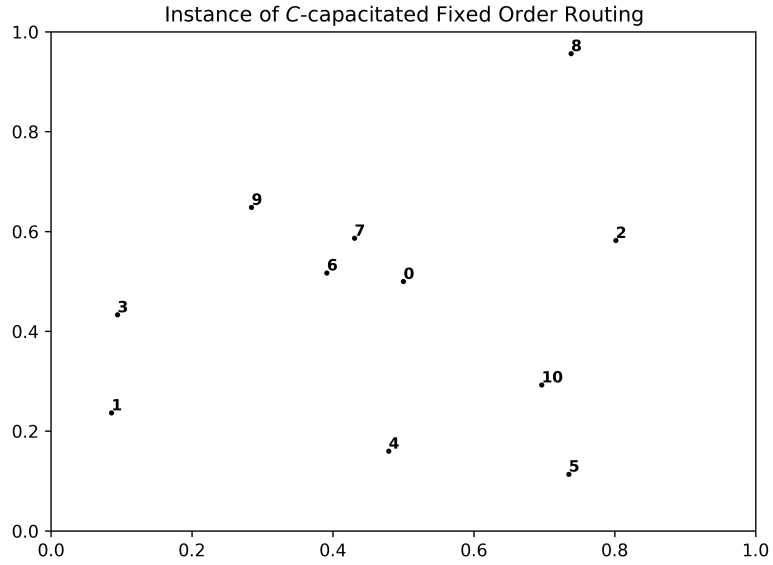


Figure 1: Example of an instance of C-cap FOR with $n = 10$ points on $[0, 1]^2$-plane

.

5 Methodology

5.1 Initial solutions

Column Generation and Columnwise Neighborhood Search both require a set of initial solutions. The only requirement is that the initial solutions are feasible solutions, so they must not violate the capacity or the fixed order constraint. We will be using four different methods to generate the initial solutions.

5.1.1 Simple methods

Two simple methods to generate feasible solutions are to assign customers in increasing order of queue number. To satisfy the capacity constraint, we can assign each customer a taxi or fill a taxi until the capacity is reached and then fill the next taxi. We will refer to these methods as *OneStop* and *MaxCap* respectively. Algorithm 1 and 2 contain descriptions of these methods and Figure 2 and 3 contain examples of initial solutions generated via these methods. The set of initial solutions is denoted by R and this is a set that contains sets of destinations. As noted in Section 4.1, these sets of destinations are traversed in order of increasing queue number i.

Algorithm 1 OneStop method

input: set of destinations $\{v_1, \ldots, v_n\}$
$R \leftarrow \emptyset$
for each destination v_i **do**
    create route r_i with single destination v_i
    add route r_i to R
end for
return R

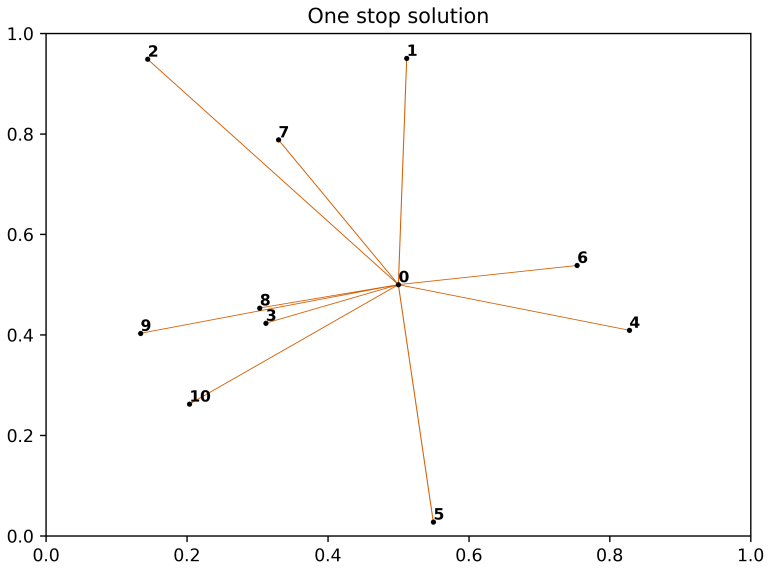


Figure 2: Example of initial solution generated by OneStop Algorithm for an instance of C-cap FOR with $n = 10$ points and $C = 3$

Algorithm 2 MaxCap method

input: set of destinations $\{v_1, \ldots, v_n\}$, capacity C
$R \leftarrow \emptyset$
create empty route $r \leftarrow \emptyset$
for each destination v **do**
  if $|r| < C$ **then**
    add destination v to r
  else
    add route r to R
    $r \leftarrow \emptyset$
  end if
end for
return R

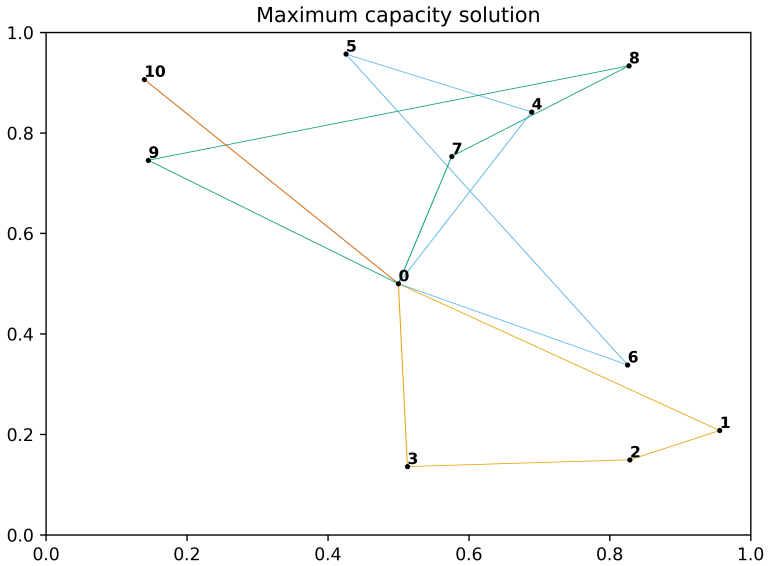


Figure 3: Example of initial solution generated by MaxCap Algorithm for an instance of C-cap FOR with $n = 10$ points and $C = 3$

5.1.2 Constrained k-means methods

The previous methods do not take the positions of the destinations into account. Since the method simply follows the order of the queue, two destinations that are far apart in terms of distance, but are close to each other in the queue, can end up together in a taxi. A smarter approach would be to put customers with destinations near each other in the same taxi. For example, in Figure 2, putting $\{3, 8\}$ together and in Figure 3 putting $\{1, 2, 6\}$ together would result in less total distance traveled. So we could construct a good initial solution by combining groups of C destinations together into routes. This is a similar problem to the well-known *k-means* problem, first defined by MacQueen, 1967. He defined it as partitioning an n-dimensional population into k sets. If we want to evenly divide the n destinations over taxis with a capacity of C, then we would need at least $\frac{n}{C}$ partitions. Therefore, we want $k = \lceil \frac{n}{C} \rceil$ partitions but we also have an additional constraint. The size of each partition may not exceed C. The field of machine learning often wants to make partitions, more

frequently referred to as clusters, of their datasets and thus has further researched this problem. A constrained version of the k-means problem with lower bounds on the size of clusters was first described by Bradley et al., 2000, to avoid the problem of empty clusters. Later Zhu et al., 2010, expanded on their work to also include upper bounds.

Finding the optimal solution to the k-means problem with $k \geq 2$ is NP-hard (Drineas et al., 2004). Therefore, finding an optimal solution to the constrained k-means problem would be computationally intensive whilst not being necessary for creating a set of initial solutions. If we have two solutions to the constrained k-means problem, then it is not necessarily true that a better solution for constrained k-means is also a better solution to the C-cap FOR problem. Therefore, a good approximation of the constrained k-means problem will suffice as a set of initial solutions. We will make use of a greedy approximation approach, described in Algorithm 3.

Algorithm 3 Greedy constrained k-means algorithm

1: **input:** set of destinations $\{v_1, \ldots, v_n\}$, capacity C, number of clusters k, distances between each pair of destinations d_{ij}
2: $R \leftarrow \emptyset$
3: initialise k centroids from set of destinations
4: $H \leftarrow \emptyset$
5: **for** each destination v **do**
6: **for** each centroid c **do**
7: $d_{vc} \leftarrow$ distance between destination v and centroid c
8: add d_{vc} to H
9: **end for**
10: **end for**
11: **for** each destination v **do**
12: **repeat**
13: $d^*_{vc} \leftarrow$ smallest distance of destination v to a centroid c
14: **if** centroid c is not full **then**
15: assign destination v to centroid c
16: **else**
17: remove d^*_{vc} from H
18: **end if**
19: **until** destination v has been assigned
20: **end for**
21: **for** each centroid c **do**
22: create route r_c with all destinations assigned to centroid c
23: add route r_c to R
24: **end for**
25: **return** R

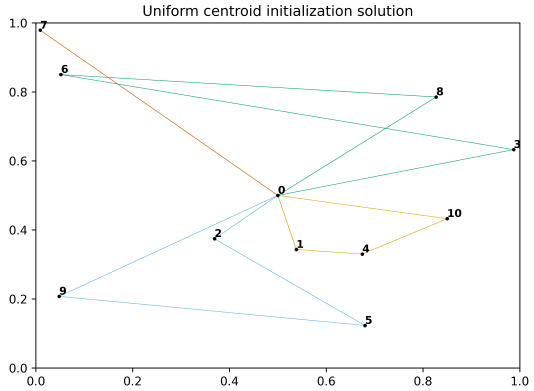
Line 3 of Algorithm 3 mentions initialising the centroids and we can use different methods for this. Lloyd's k-means algorithm did the initialisation step by choosing k points from the dataset uniformly at random (Lloyd, 1982). This method is fast and simple. However, this initialisation method can generate arbitrarily bad clusterings. Arthur and Vassilvitskii, 2007 improve Lloyd's algorithm by changing how the k centroids are initialised. Their method can be seen in Algorithm 4.

Algorithm 4 Distance-based centroid initialisation

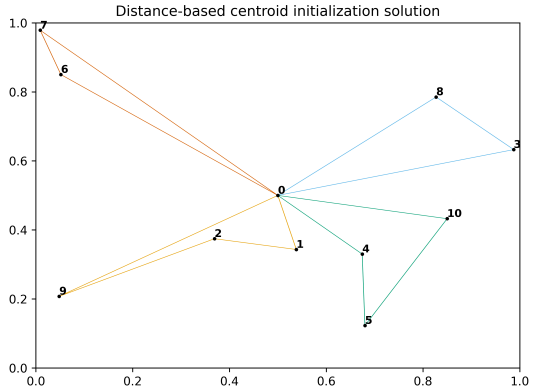
input: set of destinations $\{v_1, \ldots, v_n\}$, number of clusters k, distances between each pair of distances d_{ij}
centroids $\leftarrow \emptyset$
choose destination v uniformly at random from $\{v_1, \ldots, v_n\}$
add destination v to centroids
let D_v be the shortest distance from destination v to its closest centroid c
repeat
    choose next centroid where destination v is chosen with probability $\frac{D_v^2}{\sum_{i=1}^{n} D_{v_i}^2}$
until k centroids have been chosen
return k centroids

The goal of this initialisation method is to spread out the centroids and thus avoid arbitrarily bad clusterings. Arthur and Vassilvitskii were able to prove that it is $O(\log k)$-competitive on all data sets.

We will be using Algorithm 3 with both the uniform centroid initialisation and centroid initialisation defined in Algorithm 4 as methods for generating initial solutions. In Figure 4 we show an example with both initialisation methods on the same instance of C-cap FOR.



(a) Example of initial solution generated by Algorithm 3 for an instance of C-cap FOR with $n = 10$ points and $C = 3$ with uniform centroid initialisation



(b) Example of initial solution generated by Algorithm 3 for an instance of C-cap FOR with $n = 10$ points and $C = 3$ with the initialisation method of Algorithm 4

Figure 4: Example of initial solution generated by Algorithm 3 with both initialisation method on the same instance of C-cap FOR with $n = 10$ points and $C = 3$

The initial routes of Figure 4b look close to an optimal solution of the C-cap FOR problem on this instance. The uniform centroid initialisation in Figure 4a created an initial solution that is more chaotic.

5.2 Column generation

Column generation (CG) consists of two parts, the Master Problem (MP), and its subproblem. The Master Problem is defined as the linear relaxation of the SPP problem we defined in Section 4.2 for C-capacitated Fixed Order Routing. We restrict the Master Problem by considering only a subset

of the set of possible tours, $R \subset T$. This restriction is denoted by MP(R). In Equation 11 and 12 you can see the linear programs and the dual of the linear programs, denoted by D(R).

$$\begin{aligned}
&\textbf{MP} && \textbf{Dual of MP}\\
&\min \quad \textstyle\sum_{t\in T} d_t x_t && \max \quad \textstyle\sum_{i=1}^{n} y_i\\
&\text{s.t.} \quad \sum_{t\in T: i\in t} x_t \geq 1 \quad \forall i \in \{1,\dots,n\} && \text{s.t.} \quad \sum_{i\in t} y_i \leq d_t \quad \forall t \in T\\
&\qquad x_t \geq 0 \quad \forall t \in T && \qquad y_i \geq 0 \quad \forall i \in \{1,\dots,n\}
\end{aligned} \tag{11}$$

$$\begin{aligned}
&\textbf{MP}(R) && \textbf{Dual of MP}(R)\textbf{, D}(R)\\
&\min \quad \textstyle\sum_{t\in R} d_t x_t && \max \quad \textstyle\sum_{i=1}^{n} y_i\\
&\text{s.t.} \quad \sum_{t\in R: i\in t} x_t \geq 1 \quad \forall i \in \{1,\dots,n\} && \text{s.t.} \quad \sum_{i\in t} y_i \leq d_t \quad \forall t \in R\\
&\qquad x_t \geq 0 \quad \forall t \in R && \qquad y_i \geq 0 \quad \forall i \in \{1,\dots,n\}
\end{aligned} \tag{12}$$

The Column Generation algorithm is based on the result in Theorem 1 (Feillet, 2010).

Theorem 1. *Let R be a subset of T and y^* be the optimal solution of D(R). If y^* is feasible for D(T), then y^* is optimal for D(T).*

When we compute MP(R) with a subset $R \subset T$, an optimal solution y^* for D(R) is then also provided. If it is infeasible for D(T), then one or more constraints deriving from routes in the set $T \setminus R$ are violated. The next step is to identify these constraints via a subproblem and integrate the corresponding variables into R. If we find no violated constraints via the subproblem, then the algorithm terminates following the result from Theorem 1. Algorithm 5 contains the complete description.

Algorithm 5 Column Generation

```
input: instance of C-capacitated Fixed Order Routing
generate an initial set of columns R
repeat
    solve MP(R)
    Γ ← column(s) provided by solution to subproblem
    R ← R ∪ Γ
until Γ = ∅
solve MP(R) with x_t ∈ {0, 1}
return optimal routes
```

An essential part of the algorithm is the subproblem which identifies the violated constraints. For many applications of Column Generations, the subproblem is a shortest path problem with resource constraints (SPPRC) or a variation of it (Desaulniers et al., 2005). This is also the case for C-cap FOR and as shown in Sitters, 2023, the subproblem can be expressed as finding tours such that

$$d_t - \sum_{i\in t} y_i < 0. \tag{13}$$

A popular method to solve the subproblems, which we will also be using, is dynamic programming (Desaulniers et al., 2005). The implementation of the dynamic programming solution for the subproblem was provided by Sitters and was not changed. As noted in the work of Feillet, 2010, and Gouweleeuw, 2024, we can choose how many tours with negative reduced cost are added to R. We will add all tours with a negative reduced cost.

5.3 Columnwise Neighborhood Search

For Columnwise Neighborhood Search (CNS), we make use of the SPP ILP formulation we defined in Section 4.2. Instead of the complete set of feasible tours T, we again use a subset $R \subset T$ and expand R until we find no improvement with our search techniques. We will refer to this formulation for a set $R \subset T$, seen in Equation 14, as SPP(R).

$$\begin{aligned} & \textbf{SPP}(R) \\ \min \quad & \sum_{t \in R} d_t x_t \\ \text{s.t.} \quad & \sum_{t \in R : i \in t} x_t \geq 1 \quad \forall i \in \{1, \ldots, n\} \\ & x_t \in \{0, 1\} \quad \forall t \in R \end{aligned} \tag{14}$$

The idea behind CNS is to solve SPP(R) and then perform a local search on each of the routes that are a part of the optimal solution. The local search techniques create a *neighborhood*, $N(r)$, for each route r. The algorithm keeps track of all the routes that have been searched before and only new routes of each neighborhood are added to R for the next iteration. The algorithm terminates when the same set of optimal routes is found for two iterations in a row or when no new routes are found in the neighborhoods.

Let R^0 be the initial set of routes and R^i the set of routes after the i-th iteration of CNS. All the routes from the previous iteration are kept to be able to track all the routes that have been seen before, so R^{i+1} is defined as in Equation 15:

$$R^{i+1} = R^i \cup \Big(\cup_{r \in R^i : x_r = 1} N(r) \Big). \tag{15}$$

The neighborhoods are generated via three simple methods: adding a destination to a route, removing a destination from a route, and swapping a destination from the route with a destination that is not already part of the route. The descriptions of these methods can be found in Appendix B.

Let Remove(r), Add(r), Swap(r) denote Algorithms 10, 11, 12 respectively. The Columnwise Neighborhood Search algorithm is given in Algorithm 6.

Algorithm 6 Columnwise Neighborhood Search

input: instance of C-capacitated Fixed Order Routing
generate initial set of solutions R^0
$i \leftarrow 0$
repeat
  solve SPP(R^i)
  if $\{r \in R^i : x_r = 1\} = \{r \in R^{i-1} : x_r = 1\}$ **then**
    break
  end if
  for $r \in \{r \in R^i : x_r = 1\}$ **do**
    $N(r) \leftarrow \text{Remove}(r) \cup \text{Add}(r) \cup \text{Swap}(r)$
  end for
  $R^{i+1} \leftarrow R^i \cup \Big(\cup_{r \in R^i : x_r = 1} N(r) \Big)$
  $i \leftarrow i + 1$
until $R^i = R^{i-1}$
return $\{r \in R^i : x_r = 1\}$

5.4 Pruning

Both Column Generation and Columnwise Neighborhood Search start with an initial set of routes and then use different methods to expand this set to find a good solution. When the number of destinations grows, the number of routes that are generated also grows. Solving an ILP on a large number of routes is computationally expensive. Since we expand R without discarding any routes, it includes many routes that have no chance of being part of the optimal solution. These routes still increase the computational cost. To tackle this, we will implement a scoring system for the routes. The goal is to discard routes that have a low chance of being part of the final solution. Algorithm 7 describes the general pruning method.

Algorithm 7 Pruning

1: **input:** set of routes R, score function f, percentage p
2: compute $f(R) = \{f(r) : r \in R\}$
3: order $f(R)$ from highest to lowest
4: **return** routes in the top p percentage of scores

To build the scoring function, we make use of the following characteristics of a route within the CG or CNS algorithm:

- **Usage** : the amount of times the route has been a part of the optimal solution of an LP or ILP,
- **Age** : the first iteration that this route was part of an optimal solution,
- **Contribution** : contribution of the route to the objective.

If a route has been part of an optimal solution often, then it is more likely to be a good route. However, relying on usage alone could make the final solution worse. Both our algorithms terminate when no new routes with improvement can be found. This means that in the previous iteration, we found new routes that became part of an optimal solution. These new routes have only been part of an optimal solution once, but they are better than some of the routes that came before. Therefore age can also be an important consideration when choosing between routes. Shorter routes are also more likely to be a part of the final optimal solution than longer routes, so we can also incorporate their length to choose between routes.

From these characteristics, we will create three different score functions. The first one is solely based on usage and can be seen in Equation 16.

$$f(r) = \#\text{times route } r \text{ has appeared in an optimal solution of (I)LP} \tag{16}$$

The second score function will make use of both usage and age to normalize usage across the iterations. The idea is to avoid penalizing new routes that can not accumulate large usage counts because they were generated close to the end of the algorithm. For each route r, let i_r be the first iteration that it was part of an optimal solution and let i be the current iteration count. This score function is given in Equation 17.

$$f(r) = \frac{\#\text{times route } r \text{ has appeared in an optimal solution of (I)LP}}{1 + (i - i_r)} \tag{17}$$

The contribution characteristic is on a different scale compared to usage and age. The minimum and maximum distance between two points on a $[0, 1]^2$-plane is equal to 0 and $\sqrt{2}$ respectively while usage and age can range from zero to the total amount of iterations. To incorporate it alongside

usage and age, we will a technique popular in the field of data mining, *Z-score standardization* (D. T. Larose and C. D. Larose, 2014).

$$Z_X = \frac{X - \text{mean}(X)}{\text{SD}(X)} \tag{18}$$

This transformation will make both values follow a normal distribution with a mean of zero and a variance of one. We will compute the Z-score for the normalized usage and route length and then add the scores together as can be seen in Equation 19.

$$f(r) = Z_{\text{normalized usage}} + Z_{\text{length}} \tag{19}$$

Pruning can be applied to Column Generation by putting the pruning step before the final ILP computation. This decreases the number of variables under consideration for the ILP which can significantly decrease the time to solve it. Algorithm 8 describes Column Generation with the pruning technique.

Algorithm 8 Column Generation with pruning

1: **input:** instance of C-capacitated Fixed Order Routing
2: generate an initial set of columns R
3: **repeat**
4: solve MP(R)
5: $\Gamma \leftarrow$ column(s) provided by solution to subproblem
6: $R \leftarrow R \cup \Gamma$
7: **until** $\Gamma = \emptyset$
8: prune R as described in Algorithm 7
9: solve MP(R) with $x_t \in \{0, 1\}$
10: **return** optimal routes

For Columnwise Neighborhood Search there are more decisions to make since we solve an ILP every iteration. We need to balance allowing the algorithm to explore different routes and the computational cost of solving the ILP. To this end, we will perform several iterations at the start without any pruning and afterward prune after the completion of a certain number of iterations as can be seen in Algorithm 9. It is important to note that pruning also alters the algorithms that generate the neighborhoods. After pruning R^i it is possible that in the next iteration some of the removed routes get generated again. To avoid this, we will be adding an extra set, called *tabu*, that tracks all the routes that have been generated. The versions with the *tabu* list can be found in Appendix B in Algorithms 13, 14, 15.

Algorithm 9 Columnwise Neighborhood Search with pruning

input: instance of C-capacitated Fixed Order Routing, set of iterations where pruning will take place $\{i_1, \ldots, i_k\}$.
generate initial set of solutions R^0
tabu $\leftarrow R^0$
$i \leftarrow 0$
repeat
  solve SPP(R^i)
  if $\{r \in R^i : x_r = 1\} \subseteq$ tabu **then**
    break
  end if
  for $r \in \{r \in R^i : x_r = 1\}$ **do**
    $N(r) \leftarrow \text{Remove}(r) \cup \text{Add}(r) \cup \text{Swap}(r)$
  end for
  $R^{i+1} \leftarrow R^i \cup \left(\cup_{r \in R^i : x_r = 1} N(r) \right)$
  tabu $\leftarrow$ tabu $\cup \left(\cup_{r \in R^i : x_r = 1} N(r) \right)$
  if $i \in \{i_1, \ldots, i_k\}$ **then**
    prune R^{i+1} as described in Algorithm 7
  end if
  $i \leftarrow i + 1$
until $R^i = R^{i-1}$
return $\{r \in R^i : x_r = 1\}$

6 Results

6.1 Performance comparison of Column Generation to ILP

To compare the performance of Column Generation and an exact ILP algorithm based on the formulation in Section 4.1, we will generate ten different instances for different pairs of amount of destinations and capacities, n and C. First, we will let the CG algorithm compute a solution. Afterward, we will compute a solution using the exact ILP algorithm with a time limit equal to the time the CG algorithm took to compute its solution.

For different instances the optimal objective value is different. To be able to compare the performance across all the instances, we will compute the ratio of the objective achieved by Column Generation and the objective achieved by the ILP:

$$\text{ratio}_{\text{CG}} = \frac{\text{Objective achieved by CG algorithm}}{\text{Objective achieved by ILP algorithm}}. \tag{20}$$

If we have an instance where ratio_{CG} is less than 1, then we can conclude that the CG algorithm performed better than the ILP algorithm given the same timespan. Similarly, if ratio_{CG} is equal or greater than 1 we conclude the CG algorithm performed equal or worse than the ILP algorithm.

Let $\mu_{\text{ratio}_{\text{CG}}}$ denote the mean of the ratio_{CG} across all instances for a pair n, C. To investigate the stability of these ratios we will also compute the variance. Let $\sigma_{\text{ratio}_{\text{CG}}}$ denote the variance of ratio_{CG} across all instances for a pair n, C. Let $\mu_{\text{time}_{\text{CG}}}$ denote the average time, in seconds, it took the CG algorithm to compute a solution across all instances for a pair n, C and $\sigma_{\text{time}_{\text{CG}}}$ the variance of these times. Note that these times are only representative of my hardware. If my algorithms were run on a different machine, then the times could be different in a way that is not predictable.

Therefore we will be using these times to broadly compare the computational cost of the different pairs of n, C.

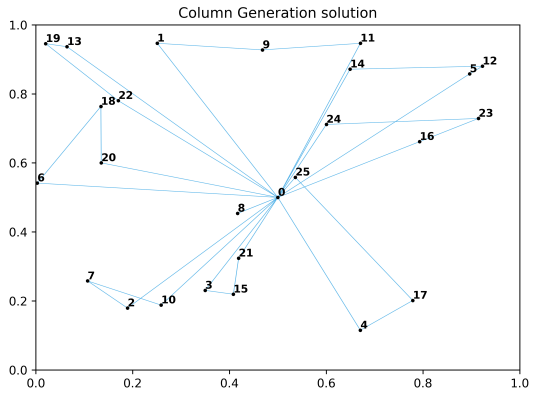
6.1.1 Experiments

In the following simulations, we will be using the CG algorithm defined in Algorithm 5 with the OneStop method, defined in Algorithm 1, to generate the set of initial solutions. In Table 1 the results for instances with $n \in \{25, 50, 100, 150\}$ and $C \in \{2, 3, 4, 5, 6, 7\}$ can be found.

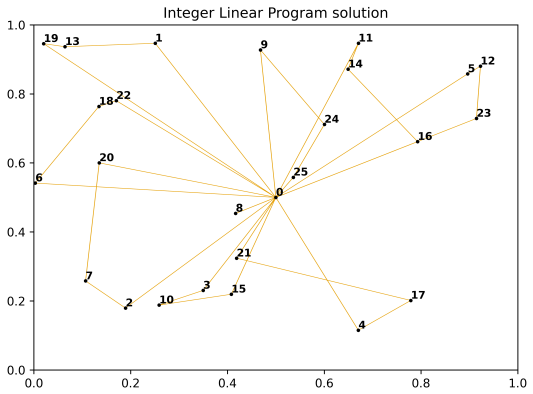
Table 1: Mean and variance of ratio of objectives, $\mu_{\text{ratio}_{\text{CG}}}$, $\sigma_{\text{ratio}_{\text{CG}}}$, between CG and ILP algorithms and mean and variance of time spent in seconds by CG algorithm, $\mu_{\text{time}_{\text{CG}}}$, $\sigma_{\text{time}_{\text{CG}}}$, across ten instances for amount of destinations $n \in \{25, 50, 100, 150\}$ and capacity $C \in \{2, 3, 4, 5, 6, 7\}$.

n	C	$\mu_{\text{ratio}_{\text{CG}}}$	$\sigma_{\text{ratio}_{\text{CG}}}$	$\mu_{\text{time}_{\text{CG}}}$	$\sigma_{\text{time}_{\text{CG}}}$
25	2	0.6414	0.1293	0.0840	0.1657
25	3	0.9730	0.0168	0.0459	0.0031
25	4	1.0001	0.0213	0.0645	0.0031
25	5	1.0134	0.0138	0.0848	0.0071
25	6	1.0052	0.0107	0.0948	0.0088
25	7	1.0086	0.0127	0.1058	0.0193
50	2	0.5655	0.0037	0.0735	0.0031
50	3	0.9253	0.0340	0.1429	0.0153
50	4	0.9643	0.0326	0.2526	0.0655
50	5	1.0093	0.0208	0.3467	0.0510
50	6	1.0083	0.0139	0.4341	0.0466
50	7	1.0124	0.0077	0.5040	0.0693
100	2	0.5462	0.0028	0.2583	0.0080
100	3	0.6457	0.2313	0.6116	0.0920
100	4	0.9287	0.0151	1.0669	0.0920
100	5	0.9641	0.0212	2.2258	0.6715
100	6	1.0053	0.0200	3.4080	1.4541
100	7	1.0235	0.0119	3.6159	0.6875
150	2	0.5363	0.0014	0.5805	0.0180
150	3	0.3873	0.0021	1.5012	0.1916
150	4	0.3182	0.0033	4.1095	1.2403
150	5	0.7209	0.3063	10.1479	7.0226
150	6	0.9758	0.0347	23.0119	12.3224
150	7	0.9926	0.0288	27.2954	17.5483

In Figure 5, you can see an example of solutions for an instance with $n = 25$ and $C = 3$ found by the CG and ILP algorithms. Appendix C contains Figure 7 with an example of solutions found for an instance with for $n = 50$ and $C = 3$.



(a) Example of solution generated by Column Generation for an instance of C-cap FOR with $n = 25$ points and $C = 3$



(b) Example of solution generated by ILP for an instance of C-cap FOR with $n = 25$ points and $C = 3$

Figure 5: Example of solutions generated by Column Generation and ILP for an instance of C-cap FOR with $n = 25$ points and $C = 3$

We also ran the CG algorithm on large instances with $n \in \{200, 300, 400, 500\}$ but it was not feasible to run ten instances with these sizes for all capacities we used before because they took too long to terminate. Table 2 contains the results of instances with $n \in \{200, 300, 400, 500\}$ and $C \in \{2, 3, 4, 5, 6\}$. A combination of n and C is omitted if it takes too long to compute the solutions on ten instances. We still see that the CG algorithm performs well in these large instances.

Table 2: Mean and variance of ratio of objectives, $\mu_{\text{ratio}_{\text{CG}}}$, $\sigma_{\text{ratio}_{\text{CG}}}$, between CG and ILP algorithms and mean and variance of time spent in seconds by CG algorithm, $\mu_{\text{time}_{\text{CG}}}$, $\sigma_{\text{time}_{\text{CG}}}$, across ten instances for amount of destinations $n \in \{200, 300, 400, 500\}$ and $C \in \{2, 3, 4, 5, 6\}$.

n	C	$\mu_{\text{ratio}_{\text{CG}}}$	$\sigma_{\text{ratio}_{\text{CG}}}$	$\mu_{\text{time}_{\text{CG}}}$	$\sigma_{\text{time}_{\text{CG}}}$
200	2	0.5322	0.0016	1.0727	0.2067
200	3	0.6057	0.2532	2.9529	0.6749
200	4	0.9271	0.0156	16.1365	9.2830
200	5	0.9779	0.0194	166.8953	315.3575
200	6	0.9969	0.0266	436.3040	708.3734
300	2	0.5255	0.0008	2.3466	0.1834
300	3	0.3713	0.0012	7.8099	2.7133
400	2	0.5222	0.0006	4.3519	0.2782
400	3	0.3661	0.0009	19.6795	8.5626
500	2	0.5203	0.0010	6.7596	0.1912
500	3	0.5225	0.2570	113.7625	176.4417

Across our two tables, we see that for every instance size n $\mu_{\text{ratio}_{\text{CG}}}$ increases when the capacity C increases, meaning the CG algorithm performs worse in comparison to the ILP algorithm. We see a similar pattern for $\mu_{\text{time}_{\text{CG}}}$. However, time_{CG} does contain stronger outliers as can be seen in the instances with $n = 200$ and capacity $C \in \{5, 6\}$. It is also noteworthy that across both Table 1 and 2, the variance of ratio_{CG} is small generally with a few exceptions.

6.1.2 Interpretation of results

The increase of $\mu_{\text{ratio}_{\text{CG}}}$, $\mu_{\text{time}_{\text{CG}}}$, and $\sigma_{\text{time}_{\text{CG}}}$ for the same instance size with a growing capacity C can be explained by the greater complexity of the routes. The number of routes generated by the CG algorithm on an instance of C-cap FOR with $n = 100$ and $C \in \{3, 7\}$ is equal to, respectively, 788 and 4286. The greater capacity allows for more possible combinations of destinations per route. One of the factors that increases the computation time of the Master Problem is the size of the set R. This explains the increase in both $\mu_{\text{time}_{\text{CG}}}$ and $\sigma_{\text{time}_{\text{CG}}}$. Furthermore, it also gives us a reason why $\mu_{\text{ratio}_{\text{CG}}}$ increases. The extra time allows the ILP algorithm to progress further than before and give better solutions.

Consequently, the best performance of the CG algorithm can be found with the small capacities $C \in \{2, 3, 4\}$. The extremely low ratio_{CG} we find for example in the instances with $n = 150, C = 3$ and $n = 300, C = 3$, can be explained by the speed of the CG algorithm. When we examine the solutions found by the ILP algorithm for some of these instances, we find that it was only able to find a solution equivalent to the solution generated by the OneStop method. These solutions have large objective values while the CG algorithm is able to find a good solution in the same amount of time. This is also part of the reason for the higher $\sigma_{\text{ratio}_{\text{CG}}}$ in the instances with $n = 100, C = 3, n = 150, C = 5$ and $n = 200, C = 3$. Many instances have ratio_{CG} around 0.3 because the ILP returns a solution equivalent to the OneStop solution while it is more competitive in others with ratio_{CG} around 0.9.

In Table 2 we see a few rows with large $\sigma_{\text{time}_{\text{CG}}}$. For the instances with $n = 200, C = 5$ this was caused by one instance taking 1059 seconds to finish while the others finished in 67 seconds on average. In the case of $n = 200, C = 6$, two instances finish after 1565 and 1934 seconds respectively while the others were finished in 107 seconds on average. In both cases, we have a few instances that take an order of magnitude longer to solve than the others. These specific instances were run multiple times and it showed the same performance characteristics. This indicates that it is not a hardware problem but a particularly difficult instance for the Gurobi ILP solver. Appendix C contains Table 16 with information about all three instances.

From these experiments, we can conclude that the CG algorithm performs best for instances with large n and small C. Generally, the CG algorithm can find better solutions than the ILP algorithm when given the same timespan. For instance sizes up to $n = 200$, performance remains reasonable with capacities up to $C = 7$. The CG algorithm remains strong for large instance sizes up to $n = 500$ if the capacities are small with $C \in \{2, 3\}$. When the problem has both a large number of destinations n and a large capacity for each taxi, then both algorithms struggle to find a good solution within a reasonable amount of time. We have also found pairs of n, C where the average $\mu_{\text{time}_{\text{CG}}}$ for the instances is on the order of 100 seconds whilst having outliers that have a time_{CG} on the order of 1000 seconds.

6.2 Performance comparison of Columnwise Neighborhood Search to ILP

We will use the same approach as in Section 6.1 to compare the performance of Columnwise Neighborhood Search and the ILP algorithm. To this end, we define the ratio of objective between the CNS algorithm and the ILP algorithm along with its mean and variance, $\mu_{\text{ratio}_{\text{CNS}}}$, $\sigma_{\text{ratio}_{\text{CNS}}}$ the same way as in Equation (20).

$$\text{ratio}_{\text{CNS}} = \frac{\text{Objective achieved by CNS algorithm}}{\text{Objective achieved by ILP algorithm}}. \tag{21}$$

The mean time spent on the CNS algorithm and its variance, $\mu_{\text{time}_{\text{CNS}}}$, $\sigma_{\text{time}_{\text{CNS}}}$ are also defined the same way.

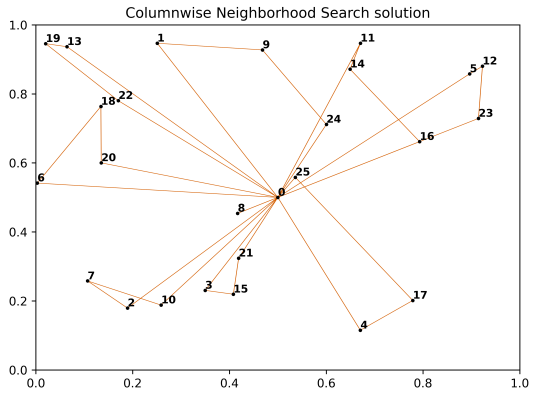
6.2.1 Experiments

In the following simulations, we will be using the CNS algorithm defined in Algorithm 6 with the OneStop method, defined in Algorithm 1, to generate the set of initial solutions. For each pair of n, C the same ten instances are used as in Section 6.1. In Table 3 the results for $n \in \{25, 50, 100, 150, 200\}$ and $C \in \{2, 3, 4, 5, 6, 7\}$ can be found.

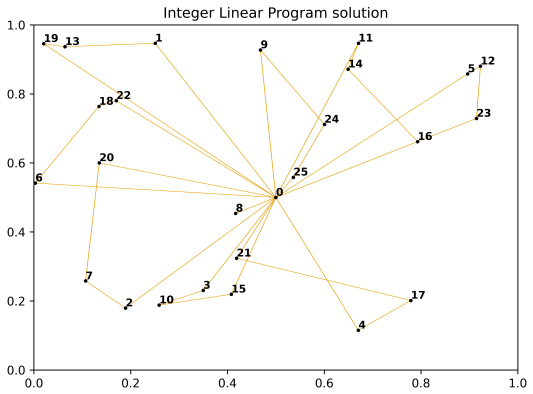
Table 3: Mean and variance of ratio of objectives, $\mu_{\text{ratio}_{\text{CNS}}}$, $\sigma_{\text{ratio}_{\text{CNS}}}$, between CNS and ILP algorithms and mean and variance of time spent in seconds by CNS algorithm, $\mu_{\text{time}_{\text{CNS}}}$, $\sigma_{\text{time}_{\text{CNS}}}$, across ten instances for amount of destinations $n \in \{25, 50, 100, 150, 200\}$ and capacity $C \in \{2, 3, 4, 5, 6, 7\}$.

n	C	$\mu_{\text{ratio}_{\text{CNS}}}$	$\sigma_{\text{ratio}_{\text{CNS}}}$	$\mu_{\text{time}_{\text{CNS}}}$	$\sigma_{\text{time}_{\text{CNS}}}$
25	2	0.5942	0.0108	0.0268	0.0033
25	3	0.9852	0.0122	0.0733	0.0153
25	4	1.0050	0.0170	0.1119	0.0145
25	5	1.0274	0.0213	0.1502	0.0387
25	6	1.0473	0.0234	0.1955	0.0779
25	7	1.0517	0.0366	0.1676	0.0332
50	2	0.5634	0.0049	0.0801	0.0093
50	3	0.9590	0.0232	0.2904	0.0600
50	4	0.9926	0.0197	0.6728	0.1851
50	5	1.0108	0.0157	0.8882	0.1918
50	6	1.0267	0.0158	1.1544	0.2784
50	7	1.0369	0.0187	1.1451	0.2481
100	2	0.5439	0.0029	0.3705	0.0159
100	3	0.9445	0.0124	2.1268	0.8168
100	4	0.9544	0.0146	4.3649	1.6201
100	5	0.9789	0.0111	7.3130	1.8860
100	6	1.0095	0.0189	8.5249	1.9308
100	7	1.0288	0.0200	9.9574	1.9163
150	2	0.5349	0.0014	1.0020	0.0503
150	3	0.6643	0.2415	6.1854	1.2161
150	4	0.9007	0.0092	13.1893	2.9671
150	5	0.9414	0.0112	26.0816	6.0323
150	6	0.9756	0.0139	30.1429	8.4337
150	7	0.9997	0.0110	42.3460	8.8574
200	2	0.5306	0.0012	2.2000	0.1582
200	3	0.9423	0.0040	13.2180	2.7427
200	4	0.9321	0.0132	31.0211	11.7378
200	5	0.9499	0.0113	61.0338	22.3251
200	6	0.9709	0.0114	80.0165	18.3995
200	7	0.9949	0.0089	109.9179	30.2607

In Figure 6, you can see an example of solutions for an instance with $n = 25$ and $C = 3$ found by the CNS and ILP algorithms. Appendix C contains Figure 8 with an example of solutions found for an instance with $n = 50$ and $C = 3$.



(a) Example of solution generated by Columnwise Neighborhood Search for an instance of C-cap FOR with $n = 25$ points and $C = 3$



(b) Example of solution generated by ILP for an instance of C-cap FOR with $n = 25$ points and $C = 3$

Figure 6: Example of solutions generated by Column Generation and ILP for an instance of C-cap FOR with $n = 25$ points and $C = 3$

We also attempted to use the CNS algorithm for larger instances with $n \in \{300, 400, 500\}$ and $C \in \{2, 3, 4\}$. For $n = 300$, we are able to obtain results with all capacities but not for $n \in \{400, 500\}$ with capacities greater than $C = 2$. Many instances were terminated before completion due to having insufficient memory and thus these instances are not included. The rest of the results can be found in Table 4. The CNS algorithm performs well in these cases and is also able to compute ten instances with $n = 300$ and $C = 4$ which we were not able to do with the CG algorithm in a reasonable amount of time.

Table 4: Mean and variance of ratio of objectives, $\mu_{\text{ratio}_{\text{CNS}}}$, $\sigma_{\text{ratio}_{\text{CNS}}}$, between CNS and ILP algorithms and mean and variance of time spent in seconds by CNS algorithm, $\mu_{\text{time}_{\text{CNS}}}$, $\sigma_{\text{time}_{\text{CNS}}}$, across ten instances for amount of destinations $n \in \{300, 400, 500\}$ and capacity $C \in \{2, 3, 4\}$.

n	C	$\mu_{\text{ratio}_{\text{CNS}}}$	$\sigma_{\text{ratio}_{\text{CNS}}}$	$\mu_{\text{time}_{\text{CNS}}}$	$\sigma_{\text{time}_{\text{CNS}}}$
300	2	0.5244	0.0006	6.1078	0.2702
300	3	0.3689	0.0009	41.9949	20.8782
300	4	0.8492	0.0261	78.5368	23.1413
400	2	0.5209	0.0005	13.9810	1.0512
500	2	0.5188	0.0003	25.0125	1.1371

Again, we see that for every instance size n, $\mu_{\text{ratio}_{\text{CNS}}}$ increases as the capacity C increases. We also find that only instances with a capacity $C = 2$ or $n = 150, C = 3$ have a $\mu_{\text{ratio}_{\text{CNS}}}$ smaller than 0.9. We see that $\sigma_{\text{ratio}_{\text{CNS}}}$ is small across all pairs of n, C. The highest variance is found in the instances with $n = 150, C = 3$.

For the instances with $n \in \{25, 50, 100, 150\}$ and $C \in \{2, 3, 4, 5, 6, 7\}$ we see that $\mu_{\text{time}_{\text{CG}}}$ and $\sigma_{\text{time}_{\text{CG}}}$ gradually increases as the complexity of the instances grows. In contrast for $n = 300$ and $C \in \{2, 3\}$ the increase is sharper, going from an average of 6 seconds to 42 seconds and an average of 0.3 seconds to 21 seconds for $\mu_{\text{time}_{\text{CNS}}}$ and $\sigma_{\text{time}_{\text{CNS}}}$ respectively.

6.2.2 Interpretation of results

We see that the CNS algorithm and ILP algorithm produce solutions that are within 10% of each other with respect to the objective value for most pairs of n, C. If we look at $\mu_{\text{time}_{\text{CG}}}$ and compare it to $\mu_{\text{time}_{\text{CNS}}}$, we see that generally, the CNS algorithm takes longer to finish. This can be an explanation for the higher $\mu_{\text{ratio}_{\text{CNS}}}$ because this extra time allows the ILP algorithm to find better solutions.

The performance of the CNS algorithm is stable with low variance across both of our tables. The variance is only higher for the instances with $n = 150, C = 3$. This can be explained by the phenomenon we saw before where the ILP algorithm is only able to return a solution equivalent to the OneStop method while our algorithm found a good solution.

From these results, we see that the CNS algorithm also achieves better solutions than the ILP algorithm given the same amount of time generally. However, the gap between the CNS algorithm and ILP for instances with a capacity C greater than 2 is smaller than 10% in most cases. The CNS algorithm did allow us to compute all ten instances for $n = 300$ and $C \in \{2, 3, 4\}$ which we were not able to do before. For instances with $n \in \{400, 500\}$ and $C \geq 3$ however, we were unable to complete all instances due to having insufficient available memory.

6.2.3 Comparison to Column Generation

The ten instances used for each pair n, C are the same in Section 6.1.1 and 6.2.1. So we can compare the results in the tables. The biggest difference we see is that the CG algorithm has many pairs of n, C with $\mu_{\text{ratio}_{\text{CG}}}$ lower than 0.75 while the CNS algorithm only has a few instances with $\mu_{\text{ratio}_{\text{CNS}}}$ below 0.9. However, it is important to note that $\mu_{\text{ratio}_{\text{CG}}}$ and $\mu_{\text{ratio}_{\text{CNS}}}$ are not the same ratio since the ILP algorithm has a different time limit depending on the algorithm. To see the difference between the performance of the CG and CNS algorithm we need to look at the objective values directly. In Table 5 the objective values of ten instances with $n = 150, C = 4$ of the CG, CNS, and their respective ILP solutions can be found.

Table 5: Objectives achieved by the CG and CNS algorithm compared to the ILP algorithm with the time limit equal to their respective algorithms on the ten instances with amount of destinations $n = 150$ and capacity $C = 4$

(a) Objective values of CG and ILP

Obj. value of CG	Obj. value of ILP
37.3212	119.1146
35.8194	112.2004
37.1122	118.0853
35.6218	110.2761
35.4986	111.4233
36.9219	116.8253
35.4283	109.6283
36.7092	115.8728
36.5490	114.2938
34.9326	109.7975

(b) Objective values of CNS and ILP

Obj. value of CNS	Obj. value of ILP
37.3801	41.6403
35.3041	39.4467
36.8451	40.3669
34.5217	38.4614
34.9814	39.4135
36.3729	40.0144
34.5454	38.8766
36.0560	39.5185
36.1902	40.2422
34.5750	38.1315

We see in Table 5a that the ILP algorithm returns solutions with high objective values. As mentioned in Section 6.1.2, these high objective values are caused by the ILP algorithm only being able to return a solution equivalent to the solution produced by the OneStop method within the time

limit. This does not happen in Table 5b. When we compare the objective values of the solutions found by the CG and CNS algorithms, we see that both algorithms sometimes produce a better solution than the other.

The large disparity between the objective values achieved by the ILP algorithm can be explained by the $\mu_{\text{time}_{\text{CG}}}$ and $\mu_{\text{time}_{\text{CNS}}}$. In Table 1 we see that for instances with $n = 150$ and $C = 4$, the CG algorithm takes an average of 4 seconds to finish while the CNS algorithm takes an average of 13 seconds. The extra time the CNS algorithm takes allows the ILP algorithm to find a better solution too.

Across the different pairs of n, C we see similar results as above. For small capacities, the CG algorithm is fast and performs much better than the ILP algorithm. The objective values of its solutions are close to the objective values of the solutions found by the CNS algorithm. Sometimes they're better and sometimes worse. For larger capacities, we do see that the CG algorithm is less stable than the CNS algorithm. The $\sigma_{\text{time}_{\text{CNS}}}$ is much lower than the $\sigma_{\text{time}_{\text{CG}}}$ in the worst cases.

6.3 Impact of initial solution methods

In Section 5.1 we defined 4 different methods to generate a set of initial solutions. These are the OneStop method, found in Algorithm 1, the MaxCap method, found in Algorithm 2, and two methods based on the constrained k-means problem, found in Algorithm 3. The method based on constrained k-means with uniform centroid initialisation will be referred to as the *uniform k-means*. The other method uses probabilities based on distance to initialise the centroids and follows Algorithm 4. We will refer to this method as the *probabilistic k-means*.

6.3.1 Experiments

To compare the performance of the CG algorithm under the different initial solutions, we will be considering instances with $n = 150, C = 6$ and $n = 200, C = 4$. These two pairs of n, C were chosen because $\mu_{\text{ratio}_{\text{CG}}}$ is close to 1, as can be seen in Section 6.1.1. To compare each initial solution, we will compute the solution on ten different instances with the CG algorithm and the ILP algorithm. We will again consider $\mu_{\text{ratio}_{\text{CG}}}$, $\mu_{\text{time}_{\text{CG}}}$, $\sigma_{\text{ratio}_{\text{CG}}}$, and $\sigma_{\text{time}_{\text{CG}}}$ to assess the performance characteristics. Table 6 contains the results for the different initial solutions.

Table 6: Mean and variance of ratio of objectives, $\mu_{\text{ratio}_{\text{CG}}}$, $\sigma_{\text{ratio}_{\text{CG}}}$, between CG and ILP algorithms and mean and variance of time spent in seconds by CG algorithm, $\mu_{\text{time}_{\text{CG}}}$, $\sigma_{\text{time}_{\text{CG}}}$, across ten instances for amount of destinations $n = 150, C = 6$ and $n = 200, C = 4$ with different initial solution methods.

Method	n	C	$\mu_{\text{ratio}_{\text{CG}}}$	$\sigma_{\text{ratio}_{\text{CG}}}$	$\mu_{\text{time}_{\text{CG}}}$	$\sigma_{\text{time}_{\text{CG}}}$
OneStop	150	6	0.9654	0.0233	18.2200	8.5104
MaxCap	150	6	0.9785	0.0283	27.0069	10.7621
uniform k-means	150	6	0.9683	0.0187	22.2828	13.3131
probabilistic k-means	150	6	0.9679	0.0208	21.2685	7.2396
OneStop	200	4	0.9169	0.0219	11.6030	13.1772
MaxCap	200	4	0.9314	0.0186	17.8285	8.2261
uniform k-means	200	4	0.9280	0.0152	14.2027	4.4591
probabilistic k-means	200	4	0.9253	0.0170	12.8044	6.0135

We see that $\mu_{\text{ratio}_{\text{CG}}}$ is lowest with the OneStop method and highest with the MaxCap method across both pairs of n, C. Both k-means methods also have a higher $\mu_{\text{ratio}_{\text{CG}}}$ but the difference is

smaller. The $\sigma_{\text{ratio}_{\text{CG}}}$ is small across both pairs of n, C and all initial solution methods. The $\mu_{\text{time}_{\text{CG}}}$ and $\sigma_{\text{time}_{\text{CG}}}$ show similar results. The OneStop method has the shortest average time, while the MaxCap method has the longest and both k-means methods are longer than OneStop but shorter than the average time of MaxCap.

As we saw in Table 5, a higher $\mu_{\text{ratio}_{\text{CG}}}$ does not mean that the final objective value achieved is worse. Since each of the initial solutions also has a different $\mu_{\text{time}_{\text{CG}}}$, we will look at the objective values of the ten instances with $n = 200, C = 4$. In Table 7 we see that there are two instances where MaxCap has a lower objective value than OneStop, uniform k-means has three instances and probabilistic k-means has five instances.

Table 7: Objectives achieved by the CG algorithm with each initial solution method on ten instances with amount of destinations $n = 200$ and capacity $C = 4$

Objective value with OneStop	Objective value with MaxCap	Objective value with uniform k-means	Objective value with probabilistic k-means
48.6005	49.1138	48.8660	49.3573
46.5140	46.9184	46.7441	46.8294
47.4279	47.4962	47.2859	47.2478
45.9332	46.4232	46.2404	46.2268
48.6076	48.4327	48.2695	48.4445
46.3166	46.3918	46.3833	46.2615
47.1670	46.6820	46.9959	46.9091
47.0824	47.3097	47.5119	47.3195
46.2341	46.2844	46.8074	46.1087
47.2071	47.3231	47.2666	47.2297

We perform the same simulations to determine the effect of the different initial solution methods on the CNS algorithm. Table 8 contains the results for the different initial solutions with the CNS algorithm.

Table 8: Mean and variance of ratio of objectives, $\mu_{\text{ratio}_{\text{CNS}}}$, $\sigma_{\text{ratio}_{\text{CNS}}}$, between CNS and ILP algorithms and mean and variance of time spent in seconds by CG algorithm, $\mu_{\text{time}_{\text{CNS}}}$, $\sigma_{\text{time}_{\text{CNS}}}$, across ten instances for amount of destinations $n \in \{150, 200\}$ and $C \in \{4, 6\}$ with different initial solution methods.

Method	n	C	$\mu_{\text{ratio}_{\text{CNS}}}$	$\sigma_{\text{ratio}_{\text{CNS}}}$	$\mu_{\text{time}_{\text{CNS}}}$	$\sigma_{\text{time}_{\text{CNS}}}$
OneStop	150	6	0.9673	0.0117	30.3478	7.4298
MaxCap	150	6	1.0145	0.0198	97.8191	29.7159
uniform k-means	150	6	0.9777	0.0173	47.8995	31.7508
probabilistic k-means	150	6	0.9755	0.0233	36.2360	18.2240
OneStop	200	4	0.9287	0.0123	31.7629	6.0058
MaxCap	200	4	0.9527	0.0074	88.1311	27.4642
uniform k-means	200	4	0.9436	0.0163	63.8294	33.8389
probabilistic k-means	200	4	0.9396	0.0141	52.9272	28.1590

The performance characteristics of the initial solutions are similar to the CG algorithm in Table 6. Table 9 contains the objective values of each initial solution method for each of the ten instances with $n = 200, C = 4$.

Table 9: Objectives achieved by the CNS algorithm with each initial solution method on ten instances with amount of destinations $n = 200$ and capacity $C = 4$

Objective value with OneStop	Objective value with MaxCap	Objective value with uniform k-means	Objective value with probabilistic k-means
48.4197	48.3469	49.3364	48.4680
46.2616	46.6327	46.2404	46.5442
46.9963	47.3837	46.9641	46.9565
45.5288	45.8237	45.5893	45.5498
47.4150	47.5631	47.7794	47.4948
45.8380	46.0482	45.7697	46.0079
46.1814	46.3115	46.2119	46.1210
46.5541	46.8384	46.7216	46.7166
45.5940	45.9380	45.7691	45.9828
46.5849	47.1367	47.0576	46.6541

We see that there is one instance where MaxCap has a lower objective value, three instances where the uniform k-means has a lower objective value, and two instances where the probabilistic k-means algorithm has a lower objective value.

6.3.2 Interpretation of results

The initial solution does not have a large impact on the performance of either the CG or CNS algorithm. The only method that stands out is the MaxCap method. It has the highest $\mu_{\text{time}_{\text{CG}}}$ and $\mu_{\text{time}_{\text{CNS}}}$ which causes the highest $\mu_{\text{ratio}_{\text{CG}}}$ and $\mu_{\text{ratio}_{\text{CNS}}}$. For both algorithms, the OneStop initial solution method performed best in terms of $\mu_{\text{ratio}_{\text{CG}}}$ and $\mu_{\text{ratio}_{\text{CNS}}}$. This is likely caused by the fact that the OneStop method is the fastest in terms of $\mu_{\text{time}_{\text{CG}}}$ and $\mu_{\text{time}_{\text{CNS}}}$. The extra time afforded to the ILP algorithm with the other methods allows it to find a better solution which causes a higher $\mu_{\text{ratio}_{\text{CG}}}$ and $\mu_{\text{ratio}_{\text{CNS}}}$. When we looked closer at the individual objective values for each instance, we saw that all methods had instances where they were better than each other. However, the differences between the objective values were small.

6.4 Impact of pruning

As we noted in the previous sections, if $\mu_{\text{time}_{\text{CG}}}$ and $\mu_{\text{time}_{\text{CNS}}}$ increase, $\mu_{\text{ratio}_{\text{CG}}}$ and $\mu_{\text{ratio}_{\text{CNS}}}$ also increase. The performance of the CG and CNS algorithms compared to the ILP algorithm can be improved if we can decrease the time both of the algorithms spend on solving the instance.

Both algorithms consist of two major parts, generating new routes to explore and solving an LP or an ILP with the new routes. In Table 10 the time spent on each part can be seen. Note that in Table 10a route generation includes the time spent on solving the LP relaxations. For the CG algorithm, the route generation is a small part of the algorithm. This means that the performance of the algorithm is mostly determined by the final ILP computation. In Table 10b we see that route generation is a shorter part of the CNS algorithm but not an insignificant part.

Table 10: Time spent on major parts of the CG and CNS algorithm on an instance with $n = 150$ and $C = 5$

(a) Time spent on route generation and ILP

Route generation in seconds	ILP computation in seconds
0.1541s	9.900s

(b) Time spent on route generation and ILPs

Route generation in seconds	ILP computations in seconds
4.5762s	10.5268s

To decrease the number of routes that are under consideration, we will be using the pruning technique, found in Algorithm 7, and the scoring functions discussed in Section 5.4. The scoring functions will be referred to as *usage, normalized, Z-score* for the functions defined in Equation 16, 17, 19 respectively. Besides the scoring function, we also need to decide the percentage of routes we will keep. If we keep the top $p\%$ of routes after pruning, we say that we use the scoring function at $p\%$.

6.4.1 Experiments

To determine the impact of pruning with the different scoring functions, we will compute ten instances with $n = 200$ and $C = 4$ for each of the scoring functions and different percentages of routes that are kept after pruning. As a baseline, we will also compute ten instances with no pruning and use $\mu_{\text{ratio}_{\text{CG}}}$, $\sigma_{\text{ratio}_{\text{CG}}}$ and $\mu_{\text{time}_{\text{CG}}}$, $\sigma_{\text{time}_{\text{CG}}}$ to compare the pruning and scoring techniques. Table 11 contains the results for the three scoring functions and different percentages of routes kept after pruning.

Table 11: Mean and variance of ratio of objectives, $\mu_{\text{ratio}_{\text{CG}}}$, $\sigma_{\text{ratio}_{\text{CG}}}$ between CG and ILP algorithm and mean and variance of time spent in seconds by CG algorithm, $\mu_{\text{time}_{\text{CG}}}$, $\sigma_{\text{time}_{\text{CG}}}$, across ten instances with amount of destinations $n = 200$ and capacity $C = 4$ with different scoring functions and amount of routes kept after pruning

Score function	Percentage of routes kept	$\mu_{\text{ratio}_{\text{CG}}}$	$\sigma_{\text{ratio}_{\text{CG}}}$	$\mu_{\text{time}_{\text{CG}}}$	$\sigma_{\text{time}_{\text{CG}}}$
no pruning	100%	**0.9265**	**0.0193**	**13.5910**	**5.2829**
usage	75%	0.9308	0.0179	13.8910	7.0197
usage	50%	0.9040	0.0392	6.3231	3.6649
usage	25%	0.6478	0.3037	5.0068	3.9322
usage	10%	0.3262	0.0049	1.1123	0.1792
normalized	75%	0.9299	0.0232	14.4154	7.4988
normalized	50%	0.9140	0.0369	9.6680	10.3313
normalized	25%	0.6220	0.3279	7.3258	11.9792
normalized	10%	0.3205	0.0043	1.4227	0.4768
Z-score	75%	0.9266	0.0234	13.5512	8.4117
Z-score	50%	0.8862	0.1111	9.6909	10.7113
Z-score	25%	0.5578	0.3179	4.7942	4.7413
Z-score	10%	0.3206	0.0038	1.0969	0.1157

We see that pruning with a percentage of routes kept at 25% or 10% is effective at lowering $\mu_{\text{ratio}_{\text{CG}}}$ and $\mu_{\text{time}_{\text{CG}}}$ with each scoring function. At 75% however, the $\mu_{\text{ratio}_{\text{CG}}}$ slightly increased

for each of the scoring functions. Note that at 50% $\mu_{\text{time}_{\text{CG}}}$ is half of what it is when we do not apply pruning but the $\mu_{\text{ratio}_{\text{CG}}}$ is still around 0.9 while at 25% $\mu_{\text{time}_{\text{CG}}}$ is only a second lower, but the $\mu_{\text{ratio}_{\text{CG}}}$ is much lower. The Z-score scoring function performs best for all percentages except at 10% where the normalized scoring function is better by 0.001 in terms of $\mu_{\text{ratio}_{\text{CG}}}$.

Whilst a low $\mu_{\text{ratio}_{\text{CG}}}$ is desirable, the goal is also to not significantly worsen the solution found without pruning. Table 12 contains the objective values without pruning and with pruning with each scoring function at 25%. We see that the objective values for each scoring function are quite close to the objective value achieved without pruning. In Appendix C you can find Table 17 for the scoring functions at 10%. For 10% the difference is slightly larger between the objective values without pruning and the objective values with pruning but the solutions are still quite good.

Table 12: Objectives achieved by the CG algorithm with each scoring function at 25% on ten instances with amount of destinations $n = 200$ and capacity $C = 3$

Objective value with no pruning	Objective value with usage function	Objective value with normalized function	Objective value with Z-score function
47.0373	47.9720	47.7156	47.7156
47.2742	48.7695	47.7622	47.7622
48.3003	49.9102	49.7019	49.6922
48.2210	49.5120	49.4981	49.5842
47.1741	48.5203	48.0095	48.0093
47.5116	47.9530	47.9530	47.9530
47.2678	47.8993	47.8993	47.8993
48.6200	49.2133	49.2133	49.2133
48.4096	49.0851	48.9205	48.9205
46.4668	47.5971	47.0916	47.3289

For the CNS algorithm, we will compute ten instances with $n = 200$ and $C = 3$ for each of the scoring functions and different percentages of routes that are kept after pruning. As mentioned in Section 5.4, we need to decide when to apply pruning. For the following simulations, we will perform pruning for every even iteration after the first iteration. Again, we compute ten instances with no pruning as a baseline. Table 13 contains the results.

Table 13: Mean and variance of ratio of objectives, $\mu_{\text{ratio}_{\text{CNS}}}$, $\sigma_{\text{ratio}_{\text{CNS}}}$ between CNS and ILP algorithm and mean and variance of time spent in seconds by CNS algorithm, $\mu_{\text{time}_{\text{CNS}}}$, $\sigma_{\text{time}_{\text{CNS}}}$, across ten instances with amount of destinations $n = 200$ and capacity $C = 3$ with different scoring functions and amount of routes kept after pruning

Score function	Percentage of routes kept	$\mu_{\text{ratio}_{\text{CNS}}}$	$\sigma_{\text{ratio}_{\text{CNS}}}$	$\mu_{\text{time}_{\text{CNS}}}$	$\sigma_{\text{time}_{\text{CNS}}}$
no pruning	100%	**0.9434**	**0.0133**	**16.7178**	**12.5719**
usage	25%	0.9491	0.0117	28.6192	4.8666
usage	10%	0.9441	0.0087	15.7649	4.6420
usage	1%	0.9325	0.0137	10.1947	4.1649
normalized	25%	0.9491	0.0120	29.4519	5.8529
normalized	10%	0.9441	0.0087	15.8307	3.9577
normalized	1%	0.9326	0.0136	10.4302	3.8638
Z-score	25%	0.9491	0.0120	29.4517	6.0588
Z-score	10%	0.9441	0.0086	16.2837	4.3343
Z-score	1%	0.9332	0.0133	10.5099	4.4852

Here we do not see any significant improvement in $\mu_{\text{ratio}_{\text{CG}}}$ despite the aggressive percentages used. At 25% $\mu_{\text{time}_{\text{CNS}}}$ is still greatly increased across all scoring functions. At 10% and 1% the $\mu_{\text{time}_{\text{CNS}}}$ is slightly decreased, however the $\mu_{\text{ratio}_{\text{CNS}}}$ does not decrease significantly. Table 14 contains the baseline objective values and the objective values with pruning at 1%. Similar to before, we see that pruning does not significantly impact the quality of the solutions.

Table 14: Objectives achieved by the CNS algorithm with each scoring function at 1% on ten instances with amount of destinations $n = 200$ and capacity $C = 3$

Objective value with no pruning	Objective value with usage function	Objective value with normalized function	Objective value with Z-score function
57.8445	57.9143	57.9143	57.9143
57.9013	57.9215	57.9215	57.9215
58.9766	59.0695	59.0695	59.0745
58.4741	58.4361	58.4361	58.4361
57.7659	57.8434	57.8434	57.8725
58.1600	58.2018	58.2018	58.2018
57.7643	57.8092	57.8092	57.8222
59.2848	59.3955	59.3955	59.3955
58.8154	58.8896	58.8896	58.9434
56.1206	56.1801	56.1801	56.1989

6.4.2 Interpretation of results

Pruning is an effective technique for improving the CG algorithm. At 25% and below, $\mu_{\text{ratio}_{\text{CG}}}$ is significantly lower than without any pruning. For 75%, pruning does not help improve either $\mu_{\text{ratio}_{\text{CG}}}$ or $\mu_{\text{time}_{\text{CG}}}$. This is likely due to the time it takes to prune. Computing and comparing the scores for all of the routes and the final ILP computation at 75% takes longer than it would take to compute the ILP with all the routes at the start. It is also noteworthy that for each scoring function the difference between $\mu_{\text{time}_{\text{CG}}}$ at 50% and 25% is not large compared to the difference between

$\mu_{\text{ratio}_{\text{CG}}}$. This means that the ILP algorithm requires around 7 seconds on average to be able to produce a better solution than the OneStop method for instances with $n = 200$ and $C = 3$.

For the CG algorithm, we also saw that the objective values achieved with pruning are only slightly higher compared to the objective values without pruning. This indicates that all our scoring functions are effective at ranking good routes. The Z-score scoring function performed better or equal in terms of $\mu_{\text{ratio}_{\text{CG}}}$ to the other functions at each percentage level. Interestingly, the normalized scoring function also performed well but also had the highest $\mu_{\text{time}_{\text{CG}}}$ compared to the usage and Z-score scoring functions. In Table 12 we see that even with the higher $\mu_{\text{time}_{\text{CG}}}$ the normalized scoring function has lower or equal objective values. From this we can conclude that normalizing the usage counts by using the iteration they were first found, is effective at keeping recently found good routes compared to the usage scoring function. Whilst the Z-score scoring function performed better than the normalized scoring function in terms of $\mu_{\text{ratio}_{\text{CG}}}$, the objective values achieved with each scoring function are close to each other. Sometimes the objective value of the Z-score scoring function is lower and other times the normalized scoring function is lower. From our experiments, it is unclear whether adding contribution and Z-score normalization are a significant improvement to the normalized scoring function.

We did not see the same success with pruning when using the CNS algorithm. Pruning at percentages of 25% or higher only increased $\mu_{\text{time}_{\text{CNS}}}$. With the scoring functions at 10% and 1%, we were able to slightly decrease the $\mu_{\text{time}_{\text{CNS}}}$ but this did not significantly impact $\mu_{\text{ratio}_{\text{CNS}}}$. The CNS algorithm creates many more routes than the CG algorithm and thus pruning is much more computationally expensive in the CNS algorithm. We also prune multiple times in the course of the CNS algorithm. This makes it difficult for the pruning to be effective at improving the CNS algorithm.

6.5 Implementation details

All the algorithms mentioned in Section 5 were implemented in Python (Van Rossum and Drake, 2009). For the LPs and ILPs, we made use of the solver created by Gurobi Optimization, LLC, 2025, and the rest of the algorithms were made with the help of NumPy (Harris et al., 2020). All graphs were made using the Matplotlib library (Hunter, 2007).

As noted in Section 4.3, we generated all of the instances with NumPy. The random number generator is controlled with a starting seed. It will produce the same set of numbers given the same seed. This ensured consistency across our experiments.

All simulations were performed on a computer featuring an *AMD Ryzen 6800HS*, 8 cores and 16 threads, with 16.0 GB of internal RAM running on openSUSE Tumbleweed with kernel *Linux 6.15.3-1-default* (openSUSE Project, 2025).

7 Conclusion

Our goal in writing this thesis is to contribute to the less-studied field of Fixed Order Routing. We did this by studying the performance of two heuristic algorithms on the C-capacitated Fixed Order Routing problem. The two heuristic algorithms are based on a Set Partitioning Problem (SPP) formulation of C-capacitated Fixed Order Routing. Each route is called a column within this formulation. Both methods start with an initial set of columns that expands each iteration until no improvement can be found. The methods differ in how they generate new columns. The Column Generation (CG) algorithm makes use of a dynamic program to solve a subproblem which gives new columns while the Columnwise Neighborhood Search (CNS) algorithm uses predefined neighborhood rules to generate a neighborhood of new columns for all optimal columns in the SPP. Due to the lack of a common set of instances to compare algorithms against, we compared our

heuristic algorithms to an exact algorithm based on the ILP formulation of C-capacitated Fixed Order Routing.

In our experiments, we first computed a solution with either the CG or CNS algorithm and limited the exact ILP algorithm to the time the CG or CNS algorithm took. To compare the algorithms to the ILP algorithm across different instances, we computed the ratio between the objective value achieved by the CG or CNS algorithm and the objective value achieved by the ILP algorithm. Our results show that for small capacities of $C \in \{2, 3\}$ both of our algorithms achieve lower objective values than the ILP algorithm in the same time. As the instance size n becomes larger the difference between the objective values greatly increases. For capacities of $C \in \{4, 5, 6, 7\}$, the difference between our heuristic algorithms and the exact ILP algorithm is usually small. The objective values of the heuristic algorithm are on average 10% better than the exact ILP algorithm. For the small instance sizes of $n \in \{25, 50\}$ there are also cases where the exact ILP algorithm performs better on average than our heuristic algorithms. In these cases, the complexity of the instances is low enough that the ILP algorithm is able to find the optimal solution in the time that the CG or CNS algorithm finds a good but not optimal solution. For the large capacities $n \in \{300, 400, 500\}$ fewer capacities are feasible to solve on our hardware. For the CG algorithm only $C \in \{2, 3\}$ were possible and for the CNS algorithm for $n = 300$, $C \in \{2, 3, 4\}$ was possible but for $n \in \{400, 500\}$ only $C = 2$ was able to be completed. The CG algorithm time to solve these instances with greater capacities became unfeasible due to inconsistent computation times. The variance between the time to solve was greater than 300 or 700 seconds in some cases. The CNS algorithm did not suffer from this issue. On average the CNS algorithm took longer to compute its solution but even for the larger instance sizes and capacities, the variance of time to solve did not increase as much as for the CG algorithm. However, the CNS algorithm did suffer from greatly increased memory usage. For a capacity C greater than 2, the amount of memory available on our machine was not enough for the CNS algorithm with instance size $n \in \{400, 500\}$.

The CG and CNS algorithms were compared to the exact ILP algorithm in the same instances so we also compared the objective values achieved by both of the heuristic algorithms. While we found that the ratio of the CG algorithm objective value and its ILP algorithm objective value was lower usually than the ratio of the CNS algorithm and its ILP algorithm, the objective values of the CG and CNS algorithms were not significantly different. Sometimes the CG algorithm had a lower objective value and sometimes the CNS algorithm did. The biggest difference between the CG and CNS algorithm is that the CG algorithm usually found its solution much faster and thus the ILP algorithm had less time to find a good solution. As mentioned before, the CG algorithm exhibited unpredictability in terms of computation time for some instances while the CNS algorithm remained stable throughout our experiments.

The third set of experiments concerned the set of initial solutions both methods use. We consider the simple OneStop initial solution method that sends a vehicle to each destination as the baseline and compared it against the MaxCap method that fills each vehicle sequentially and two methods based on the constrained k-means problem. We found that none of our other methods had a significant positive impact compared to the baseline method. The ratio of the objective values achieved with the OneStop method was lower than the other methods. For each of the initial solution methods, there were instances where they found solutions with the lowest objective value but none of the methods performed consistently better than the rest. Higher-quality initial solution methods did not significantly improve the performance of either the CG or CNS algorithm compared to the simple OneStop initial solution method.

Lastly, we implemented a pruning technique to reduce the number of routes that the most computationally intensive part of both the CG and CNS algorithms has to consider. For the pruning method, we defined three different scoring functions to preserve the best routes and discard routes that are not likely to be part of the final solution. These methods were based on the number of times

the route was used in an optimal solution, when it was first used in an optimal solution, and how much it contributed to the objective value. We found that for the CG algorithm pruning with any of the scoring functions is effective at improving the computational efficiency if we only keep 25% or 10% of the routes. The time to compute a solution drastically reduces at these percentage levels and the ILP algorithm is not able to compute a good solution in the equivalent time. Furthermore, discarding a large percentage of the routes did not significantly impact the objective values. The normalized and Z-score scoring functions performed best. In terms of the ratio of the objective value achieved by the CG algorithm and the ILP algorithm, the Z-score scoring function performed better but the normalized scoring function has a few instances where it achieved the lowest objective value among the scoring functions.

However, the pruning technique did not significantly improve the performance when we used it with the CNS algorithm. Due to the large number of routes generated with the CNS algorithm, the time to compute the scoring function for each function and ranking them took longer for most percentage levels than it would take to compute the ILP with all the routes. Even with an aggressive percentage level of 1% for the pruning, there was only a slight improvement in the ratio of the objective achieved with the CNS algorithm and the ILP algorithm. Furthermore, this slight improvement in the ratio is at the cost of a slight decrease in the objective value achieved compared to applying no pruning.

We have shown that Column Generation and Columnwise Neighborhood Search can be successfully applied to the C-capacitated Fixed Order Routing problem and are able to obtain better results than an exact ILP algorithm with a time limit. Our research mostly focused on small capacities due to the CG and CNS algorithms not terminating within a reasonable amount of time for large capacities. Generally, we saw that the ILP algorithm performed better when the ratio between capacity and instance size became greater. Further research could be done to see if pruning allows the CG algorithm to perform better in these instances. For the CNS algorithm, we did not find ways to improve its performance.

To address this, further research can be done on different neighborhood rules or a smarter algorithm. Better neighborhood generation could find the optimal routes quicker. Additionally, an implementation that makes use of different representations of the routes to make use of boolean operations or SIMD instructions to create neighborhoods and filter the routes could speed up the route generation.

While we found significant improvements with pruning for the CG algorithms, we only tested two different pairs of instance size and capacity. Further research can show whether the pruning technique is able to solve the previously infeasible instances in terms of computation time and more broadly examine the performance characteristics.

When it comes to the generation of the instances, we chose to distribute the points uniformly over the $[0, 1]^2$-plane. This might not be an accurate distribution of requests for different types of problems. The influence of instance size, capacity, initial solution quality, and pruning could also be different when the requests are distributed in clusters. This can be an interesting area of research to further expand the knowledge of the performance characteristics of these column-based heuristics.

A Notation

Table 15: Notation table

Symbol	Description
n	Number of customers
C	Capacity of taxi
k	Number of taxis
V	Set of vertices $\{v_1, \ldots, v_n\}$
A	Set of arcs $\{(v_i, v_j) : v_i, v_j \in V\}$
d_{ij}	Cost of arc (v_i, v_j)
x_{ij}	Decision variable which is 1 if arc (v_i, v_j) is used, 0 otherwise
y_i	Amount of customers visiting v_i
T	Set of feasible routes
R	Subset of feasible routes
d_t	Cost of route t
x_t	Decision variable which is 1 if route t is used, 0 otherwise
$\bar{t}$	Set complement of route t, i.e., destinations not on route t
Z_X	Z-score normalization of X

B Algorithms

Algorithm 10 Remove destination, Remove(r)

input: route r, R^i
$N_{\text{remove}}(r) \leftarrow \emptyset$
for $v_i \in r$ **do**
  $r^{'} \leftarrow r \setminus \{v_i\}$
  if $r^{'} \notin R^i$ **then**
    add $r^{'}$ to $N_{\text{remove}}(r)$
  end if
end for
return $N_{\text{remove}}(r)$

The *Remove destination* algorithm takes in a route r and the set of previously used routes in optimal solutions R^i as input. For every destination on route r, it will create a new route by removing that destination from route r.

Algorithm 11 Add destination, Add(r)

input: route r, set of destinations $\{v_1, \ldots, v_n\}$, R^i
$N_{\text{add}}(r) \leftarrow \emptyset$
for $v_i \in \overline{r}$ **do**
  $r^{'} \leftarrow r \cup \{v_i\}$
  if $r^{'} \notin R^i$ **then**
    add $r^{'}$ to $N_{\text{add}}(r)$
  end if
end for
return $N_{\text{add}}(r)$

The *Add destination* algorithm takes in a route r and the set of previously used routes in optimal solutions R^i as input. For every destination not on route r, it will create a new route by adding that destination to route r.

Algorithm 12 Swap destination, Swap(r)

input: route r, set of destinations $\{v_1, \ldots, v_n\}$, R^i
$N_{\text{swap}}(r) \leftarrow \emptyset$
for $v_i \in r$ **do**
  for $v_j \in \overline{r}$ **do**
    $r^{'} \leftarrow (r \setminus \{v_i\}) \cup \{v_j\}$
    if $r^{'} \notin R^i$ **then**
      add $r^{'}$ to $N_{\text{swap}}(r)$
    end if
  end for
end for
return $N_{\text{swap}}(r)$

The *Swap destination* algorithm takes in a route r and the set of previously used routes in optimal solutions R^i as input. For every destination on route r and every destination not on route r, it will create a new route by swapping the destination on route r with the destination not on route r.

Algorithm 13 Remove destination with tabu list

input: route r, R^i, tabu
$N_{\text{remove}}(r) \leftarrow \emptyset$
for $v_i \in r$ **do**
  $r^{'} \leftarrow r \setminus \{v_i\}$
  if $r^{'} \notin R^i$ **and** $r^{'} \notin$ tabu **then**
    add $r^{'}$ to $N_{\text{remove}}(r)$
  end if
end for
return $N_{\text{remove}}(r)$

This algorithms is the same as *Remove destination* algorithm, but now we also check whether the newly generated route is also contained within the *tabu* list of previously used routes.

Algorithm 14 Add destination with tabu list

input: route r, set of destinations $\{v_1, \ldots, v_n\}$, R^i, tabu
$N_{\text{add}}(r) \leftarrow \emptyset$
for $v_i \in \overline{r}$ **do**
  $r^{'} \leftarrow r \cup \{v_i\}$
  if $r^{'} \notin R^i$ **and** $r^{'} \notin$ tabu **then**
    add $r^{'}$ to $N_{\text{add}}(r)$
  end if
end for
return $N_{\text{add}}(r)$

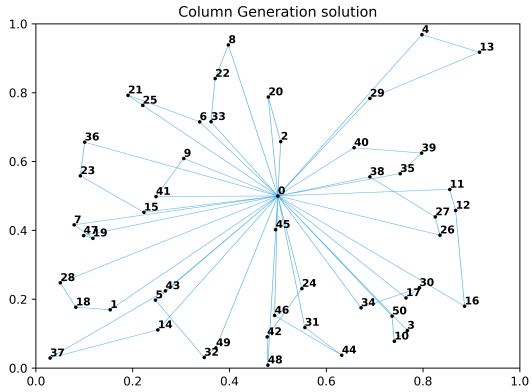
This algorithms is the same as *Add destination* algorithm, but now we also check whether the newly generated route is also contained within the *tabu* list of previously used routes.

Algorithm 15 Swap destination with tabu list

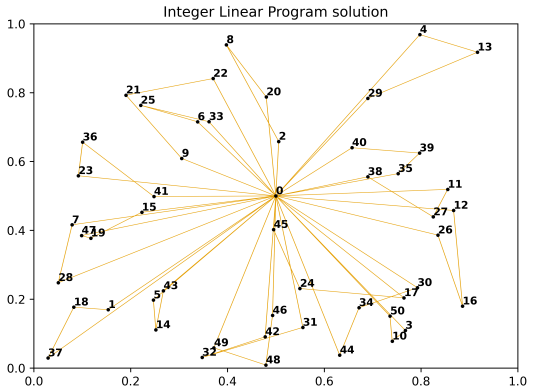
input: route r, set of destinations $\{v_1, \ldots, v_n\}$, R^i, tabu
$N_{\text{swap}}(r) \leftarrow \emptyset$
for $v_i \in r$ **do**
  for $v_j \in \overline{r}$ **do**
    $r^{'} \leftarrow (r \setminus \{v_i\}) \cup \{v_j\}$
    if $r^{'} \notin R^i$ **and** $r^{'} \notin$ tabu **then**
      add $r^{'}$ to $N_{\text{swap}}(r)$
    end if
  end for
end for
return $N_{\text{swap}}(r)$

This algorithms is the same as *Swap destination* algorithm, but now we also check whether the newly generated route is also contained within the *tabu* list of previously used routes.

C Tables & Figures

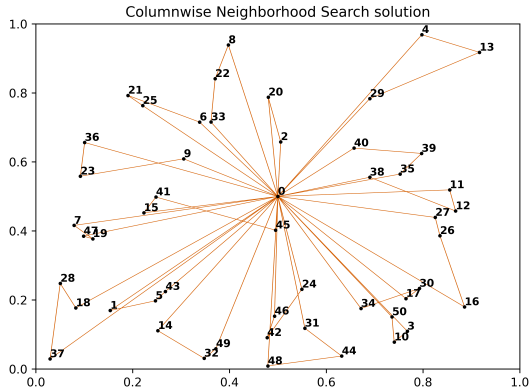


(a) Example of solution generated by Column Generation for an instance of C-cap FOR with $n = 50$ points and $C = 3$

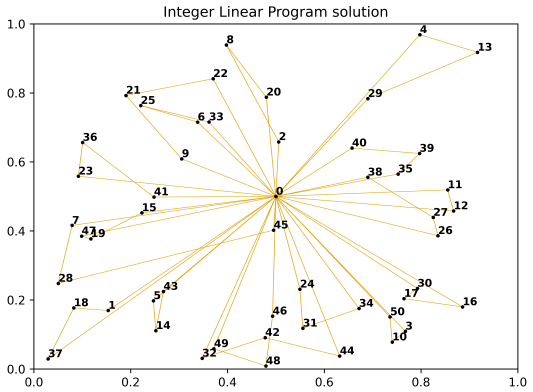


(b) Example of solution generated by ILP for an instance of C-cap FOR with $n = 50$ points and $C = 3$

Figure 7: Example of solutions generated by Column Generation and ILP for an instance of C-cap FOR with $n = 50$ points and $C = 3$



(a) Example of solution generated by Columnwise Neighborhood Search for an instance of C-cap FOR with $n = 50$ points and $C = 3$



(b) Example of solution generated by ILP for an instance of C-cap FOR with $n = 50$ points and $C = 3$

Figure 8: Example of solutions generated by Columnwise Neighborhood Search and ILP for an instance of C-cap FOR with $n = 50$ points and $C = 3$

Table 16: Instances where the time to find a solution with the CG algorithm took significantly longer

seed	n	C	ratio_{CG}	time_{CG}
454433	200	5	1.0246	1059
996365	200	6	1.031	1565
276027	200	6	1.0437	1934

Table 17: Objectives achieved by the CG algorithm with each scoring function at 10% on ten instances with amount of destinations $n = 200$ and capacity $C = 3$

Objective value with no pruning	Objective value with usage function	Objective value with normalized function	Objective value with Z-score function
47.0373	49.5405	49.1032	48.7397
47.2742	49.5965	48.3181	49.2499
48.3003	51.0332	50.3439	49.9548
48.2210	51.5252	50.6818	50.3777
47.1741	50.4901	49.2799	49.5475
47.5116	49.3732	48.9502	48.6980
47.2678	50.2957	49.0595	47.8993
48.6200	50.5998	50.1284	50.1736
48.4096	51.1386	49.6681	50.0413
46.4668	49.7540	48.7966	48.7403

References

Ackermann, Heiner, Paul W. Goldberg, Vahab S. Mirrokni, Heiko Röglin, and Berthold Vöcking (2008). "A Unified Approach to Congestion Games and Two-Sided Markets". In: *Internet Math* 5.4, pp. 439–458.

Anbuudayasankar, S.P., K. Ganesh, and Sanjay Mohapatra (2014). *Models for Practical Routing Problems in Logistics*. 1st edition. Springer. DOI: 10.1007/978-3-319-05035-5.

Arthur, David and Sergei Vassilvitskii (2007). "k-means++: The Advantages of Careful Seeding". In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, pp. 1027–1035. URL: http://theory.stanford.edu/~sergei/papers/kMeansPP-soda.pdf.

Ausiello, G., P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi (2000). "Complexity and approximation: Combinatorial optimization problems and their approximability properties". In: *Computers & Mathematics with Applications* 40.2-3. DOI: 10.1016/S0898-1221(00)90183-4.

Bradley, P. S., K. P. Bennett, and A. Demiriz (2000). *Constrained K-Means Clustering*. Tech. rep. MSR-TR-2000-65. Microsoft Research. URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-2000-65.pdf.

Briefjes, J. (2023). "Columnwise Neighborhood Search for Vehicle Routing Problem with Time Windows". Thesis MSc EOR. Vrije Universiteit Amsterdam.

Bruno, J., E. G. Coffman, and R. Sethi (1974). "Scheduling independent tasks to reduce mean finishing time". In: *Communications of the ACM* 17.7, pp. 382–387. DOI: 10.1145/361011.361064.

Chen, Xin, Malgorzata Sterna, Xin Han, and Jacek Blazewicz (2015). "Scheduling on parallel identical machines with late work criterion: Offline and online cases". In: *Journal of Scheduling* 19.6, pp. 729–736. DOI: 10.1016/0377-2217(93)E0280-B.

Christodoulou, George, Elias Koutsoupias, and Akash Nanavati (2009). "Coordination mechanisms". In: *Theoretical Computer Science* 410.36, pp. 3327–3336. DOI: 10.1016/j.tcs.2009.01.005.

Chrobak, M., H. Karloff, T. Payne, and S. Vishwanathan (1990). "New results on server problems". In: *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '90.

San Francisco, California, USA: Society for Industrial and Applied Mathematics, pp. 291–300. ISBN: 0898712513.

Clarke, G. and J. W. Wright (1964). "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points". In: *Operations Research* 12.4, pp. 568–581. DOI: 10.1287/opre.12.4.568.

Danna, Emilie and Claude Le Pape (2005). "Branch-and-Price Heuristics: A Case Study on the Vehicle Routing Problem with Time Windows". In: *Column Generation*. Ed. by Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon. GERAD 25th anniversary series 5. Springer.

Dantzig, G. B. and J. H. Ramser (1959). "The Truck Dispatching Problem". In: *Management Science* 6.1, pp. 80–91.

Desaulniers, Guy, Jacques Desrosiers, and Marius M. Solomon (2005). *Column Generation*. 1st edition. Springer. DOI: 10.1007/b135457.

Drineas, P., A. Frieze, R. Kannan, S. Vempala, and V. Vinay (2004). "Clustering Large Graphs via the Singular Value Decomposition". In: *Machine Learning* 56.1-3, pp. 9–33. DOI: 10.1023/B:MACH.0000033113.59016.96.

Ekici, Ali (2023). "A large neighborhood search algorithm and lower bounds for the variable-Sized bin packing problem with conflicts". In: *European Journal of Operational Research* 308.3, pp. 1007–1020. DOI: 10.1016/j.ejor.2022.12.042.

Feillet, Dominique (2010). "A tutorial on Column Generation and Branch-and-Price for Vehicle Routing Problems". In: *4OR : A Quarterly Journal of Operations Research* 8.4, pp. 407–424. DOI: 10.1007/s10288-010-0130-z.

Feo, Thomas A. and Mauricio G.C Resende (1989). "A probabilistic heuristic for a computationally difficult set covering problem". In: *Operations Research Letters* 8.2, pp. 67–71. DOI: 10.1016/0167-6377(89)90002-3.

Fügenschuh, Armin (2011). "A set partitioning reformulation of a school bus scheduling problem". In: *Journal of Scheduling* 14.4, pp. 307–318. DOI: 10.1007/s10951-011-0234-0.

Gao, Kai and Jingjin Yu (2021). "Capacitated Vehicle Routing with Target Geometric Constraints". In: *Proceedings of the 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 7925–7930. DOI: 10.1109/IROS51168.2021.9636123.

Garfinkel, R.S and G.L Nemhauser (1969). "The Set-Partitioning Problem: Set Covering with Equality Constraints". In: *Operations Research* 17.5, pp. 848–856. DOI: 10.1287/opre.17.5.848.

Golden, Bruce, Subramanian Raghavan, and Edward Wasil (2008). *The vehicle routing problem : latest advances and new challenges*. 1st edition. Springer. DOI: 10.1007/978-0-387-77778-8.

Gørtz, Inge Li, Marco Molinaro, Viswanath Nagarajan, and R. Ravi (2016). "Capacitated Vehicle Routing with Nonuniform Speeds". In: *Mathematics of Operations Research* 41.1, pp. 318–331. DOI: 10.1287/moor.2015.0729.

Gourvès, Laurent, Jérôme Monnot, Stefano Moretti, and Nguyen Kim Thang (2015). "Congestion Games with Capacitated Resources". In: *Theory of Computing Systems* 57.3, pp. 598–616. DOI: 10.1007/s00224-014-9541-0.

Gouweleeuw, Sanne (2024). "Comparing Exact Algorithms for a Fixed Order Routing Problem". Thesis BSc EOR. Vrije Universiteit Amsterdam.

Gurobi Optimization, LLC (2025). *Gurobi Optimizer Version 12.0.2*. URL: https://www.gurobi.com (Accessed on 8/6/2025).

Harris, Charles R., K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime

Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant (2020). "Array programming with NumPy". In: *Nature* 585.7825. Version 2.3.0 released June 7, 2025, pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

Ho, Johnny C. and Yih-Long Chang (1995). "Minimizing the number of tardy jobs for m parallel machines". In: *European Journal of Operational Research* 84.2, pp. 343–355. DOI: 10.1016/0377-2217(93)E0280-B.

Huang, Taoan, Aaron Ferber, Yuandong Tian, Bistra Dilkina, and Benoit Steiner (2022). "Local Branching Relaxation Heuristics for Integer Linear Programs". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer. DOI: 10.1007/978-3-031-33271-5_7.

Hunter, J. D. (2007). "Matplotlib: A 2D graphics environment". In: *Computing In Science & Engineering* 9.3. Version 3.10.3 released May 8, 2025, pp. 90–95. DOI: 10.1109/MCSE.2007.55.

IBM (2025). *IBM ILOG CPLEX Optimization Studio*. URL: https://www.ibm.com/products/ilog-cplex-optimization-studio (Accessed on 8/6/2025).

Jagtenberg, Caroline J., Oliver J. Maclaren, Andrew J. Mason, Andrea Raith, Kevin Shen, and Michael Sundvick (2020). "Columnwise neighborhood search: A novel set partitioning matheuristic and its application to the VeRoLog Solver Challenge 2019". In: *Networks* 76.6, pp. 273–293. DOI: 10.1002/net.21961.

Kallehauge, Brian, Jesper Larsen, Oli B.G Madsen, and Marius Solomon (2005). "The Vehicle Routing Problem with Time Windows". In: *Column Generation*. Ed. by Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon. GERAD 25th anniversary series 5. Springer.

Kirkpatrick, Scott (1984). "Optimization by simulated annealing: Quantitative studies". In: *Journal of Statistical Physics* 34.5-6, pp. 975–986. DOI: 10.1007/BF01009452.

Larose, Daniel T. and Chantal D. Larose (2014). *Discovering Knowledge in Data: An Introduction to Data Mining*. Second edition. Wiley Series on Methods and Applications in Data Mining. Hoboken: Wiley. DOI: 10.1002/9781118874059.

Lei, Hongtao, Gilbert Laporte, and Bo Guo (2011). "The capacitated vehicle routing problem with stochastic demands and time windows". In: *Computers and Operations Research* 38.12, pp. 1775–1783. DOI: 10.1016/j.cor.2011.02.007.

Letchford, Adam N. and Juan-José Salazar González (2019). "The Capacitated Vehicle Routing Problem: Stronger bounds in pseudo-polynomial time". In: *European Journal of Operational Research* 272.1, pp. 24–31. DOI: 10.1016/j.ejor.2018.06.002.

Limmer, Steffen, Johannes Varga, and Günther Robert Raidl (2023). "Large Neighborhood Search for Electric Vehicle Fleet Scheduling". In: *Energies* 16.12, p. 4576. DOI: 10.3390/en16124576.

Lloyd, S. (1982). "Least squares quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2, pp. 129–137. DOI: 10.1109/TIT.1982.1056489.

MacQueen, J. (1967). "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability* 1, pp. 281–297. URL: http://projecteuclid.org/euclid.bsmsp/1200512992.

Miltenburg, Steven, Tim Oosterwijk, and René Sitters (2025). "Complexity of Fixed Order Routing". In: *International Workshop on Approximation and Online Algorithms*. Ed. by Marcin Bieńkowski and Matthias Englert. Cham: Spinger Nature Switzerland, pp. 183–197.

Mladenović, N. and P. Hansen (1997). "Variable neighborhood search". In: *Computers and Operations Research* 24.11, pp. 1097–1100. DOI: 10.1016/S0305-0548(97)00031-2.

NumPy Developers (2024). *NumPy Random Number Generation*. URL: https://numpy.org/doc/stable/reference/random/generator.html (Accessed on 8/6/2025).

openSUSE Project (2025). *openSUSE Tumbleweed*. A continuously updated rolling release Linux distribution. URL: https://www.opensuse.org/ (Accessed on 8/6/2025).

Paschalidis, Christos (2023). "Fixed Order Routing". Thesis BSc EOR. Vrije Universiteit Amsterdam.

Sandoya Sánchez, Fernando Francisco, Carmen Andrea Letamendi Lazo, and Fanny Yamel Sanabria Quiñónez (2023). "Comparative Study of Algorithms Metaheuristics Based Applied to the Solution of the Capacitated Vehicle Routing Problem". In: *[s.l]*.

Schel, Ruben (2023). "Columnwise neighborhood search for the Multi-depot Vehicle Routing Problem". Thesis MSc EOR. Vrije Universiteit Amsterdam.

Shaw, Paul (1998). "Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems". In: *Principles and practice of constraint programming*. Ed. by Michael Maher and Jean-Francois Puget. Vol. 1520. Lecture Notes in Computer Science. Berlin: Springer-Verlag Berlin Heidelberg, pp. 417–431. DOI: 10.1007/3-540-49481-2.

Sitters, René (2023). *Assignment 2*. Operations Research III, Vrije Universiteit Amsterdam.

Sleptchenko, Andrei, Angelo Sifaleras, and P. Hansen, eds. (2023). *Variable Neighborhood Search: 9th International Conference, ICVNS 2022, Abu Dhabi, United Arab Emirates, October 25–28, 2022, Revised Selected Papers*. Vol. 13863. Lecture Notes in Computer Science. Cham, Switzerland: Springer. DOI: 10.1007/978-3-031-34500-5.

Swart, A. (2022). "Columnwise neighborhood search for Electric vehicle routing problems with non-linear charging functions". Thesis MSc EOR. Vrije Universiteit Amsterdam.

Taillard, Eric D. (2023). *Design of Heuristic Algorithms for Hard Optimization*. 1st edition. Springer. DOI: 10.1007/978-3-031-13714-3.

van den Akker, Marjan, Han Hoogeveen, and Steef van de Velde (2005). "Applying Column Generation to Machine Scheduling". In: *Column Generation*. Ed. by Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon. GERAD 25th anniversary series 5. Springer.

Van Rossum, Guido and Fred L. Jr. Drake (2009). *Python 3 Reference Manual*. Version 3.13.3, released April 8, 2025. Scotts Valley, CA: CreateSpace. URL: https://docs.python.org/3/reference/.

Vijayalakshmi, Vipin Ravindran, Marc Schröder, and Tami Tamir (2021). "Scheduling games with machine-dependent priority lists". In: *Theoretical Computer Science* 855, pp. 90–103. DOI: 10.1016/j.tcs.2020.11.042.

Winter, Felix and Nysret Musliu (2021). "A large neighborhood search approach for the paint shop scheduling problem". In: *Journal of Scheduling* 25.4, pp. 453–475. DOI: 10.1007/s10951-021-00713-7.

Wolfinger, David (2021). "A Large Neighborhood Search for the Pickup and Delivery Problem with Time Windows, Split Loads and Transshipments". In: *Computers and Operations Research* 126. DOI: 10.1016/j.cor.2020.105110.

Zhu, Shunzhi, Dingding Wang, and Tao Li (2010). "Data clustering with size constraints". In: *Knowledge-Based Systems* 23.8, pp. 883–889. DOI: 10.1016/j.knosys.2010.06.003.