Bachelor Project

Czech
Technical
University
in Prague

F4

Faculty of Nuclear Sciences and Physical Engineering
Department of Mathematics

My Favourite Thesis; Just the Title is Soooooooo Loooooong

Journey to the who-knows-what wondeland

Matouš Pelikán

Supervisor: Prof. Krutoš Spravedlivý
Supervisor–specialist: John Doe
Field of study: Mathematical Engineering
Subfield: Mathematical Modelling
February 2017

Acknowledgements

Declaration

Abstract

Keywords: word, key

Supervisor: Prof. Krutoš Spravedlivý
Ústav X,
Uliční 5,
Praha 99

Abstrakt

Klíčová slova: slovo, klíč

Překlad názvu: Moje bakalářka se strašně, ale hrozně dlouhým předlouhým názvem — Cesta do tajů kdovíčeho

Contents

Figures

Tables

Chapter 1
Introduction

Combinatorial optimization problems are a class of challenging tasks that involve finding the best arrangement or combination of discrete elements from a large set of possibilities. These problems arise in various fields, such as logistics, scheduling, network design, and resource allocation. Examples of combinatorial optimization problems include the traveling salesman problem (TSP), the knapsack problem, or the graph coloring problem to name only a few. The complexity of these problems lies in the exponential growth of possible solutions as the problem size increases, making it computationally infeasible to search the entire solution space.

Metaheuristics have emerged as a powerful technique to face the computational challenges posed by complex combinatorial problems. Metaheuristics provide a flexible and robust framework for addressing optimization challenges by operating at a higher level of abstraction. These algorithms offer a unique approach to problem-solving by exploring large solution spaces efficiently and effectively. They are particularly well-suited for combinatorial optimization problems where traditional methods, such as linear programming, integer programming, etc., struggle due to the high-dimensional and non-linear nature of the search space.

Among many other, evolutionary algorithms (EAs) represent a powerful class of metaheuristics that have demonstrated remarkable effectiveness in solving combinatorial optimization problems. Inspired by the principles of natural evolution, these algorithms emulate the process of natural selection and adaptation to guide the search for optimal or near-optimal solutions. They typically work with a population of individuals, where each individual represents a possible solution to the problem at hand. On top of what EAs can offer, it is common practise to integrate local search techniques in order to enhance the performance. Local search focuses on exploring the neighborhood of a given solution to find better nearby solutions. By combining local search with evolutionary algorithms, the search process benefits from both global exploration and local exploitation, leading to improved solution quality and convergence.

An example of such evolutionary based algorithm is IREANN, introduced by Kubalík and Snížek in [KS14]. IREANN uses an indirect representation and a so-called nearest neighbor heuristic, which is a constructive procedure suited for routing problems. Both of these concepts used in IREANN are heavily exploited in this thesis. Local search heuristics might be incorporated to IREANN to yield even better performance.

However, given the nature of IREANN's indirect representation functionality, improvements made by local search heuristics would only affect the single individual whose neighborhood of solution space was searched. The information about local improvements can not be easily passed between other individuals in a population.

The objective of this thesis is to propose an extension to the IREANN algorithm that enables the propagation of valuable information about high-quality features of individual solutions across the entire population during computation. This extension aims to ensure that the entire population can potentially benefit from the insights gained from individually discovered superior features, thereby enhancing the overall performance and optimization capabilities of the algorithm.

The principle of enhancing the algorithm si to incorporate a mechanism that during computation captures and retains information about the features that contibute the most to high-quality solutions. The mechanism involves periodically storing the relevant information which is subsequently used in the nearest neighbor heuristic, and serves as a proxy to information about the actual distance. The whole algorithm was specifically designed to solve the Capacitated arc routing problem (CARP), which is more challenging than the famous TSP and introduces more constraints.

Several slight modifications of above mentioned approach have been implemented as a result of this thesis. The effects of proposed extension on solution quality were empirically verified by testing on standard available CARP benchmark datasets.

The thesis is structured as follows...(TODO)

Chapter 2

Problem definition

The capacitated arc routing problem (CARP), firstly introduced by Golden and Wong [GW81], is a subject in combinatorial optimization, commonly appearing in operations research and transportation logistics. In this chapter, we will provide a formal definition of the CARP, establish relevant terminology, and underline the basic properties that characterize this complex problem.

The Capacitated Arc Routing Problem (CARP) is a variant of the arc routing problem where a fleet of vehicles of uniform capacity is used to service a set of arcs or edges in a network. The fundamental challenge is to design the minimum cost set of routes such that each vehicle originates and terminates at a depot, each edge in the network requiring service is traversed by exactly one vehicle, and the total demand serviced by any vehicle does not exceed its capacity.

The problem is defined on a connected, undirected graph $G = (V,\ E)$, where V is the set of vertices and E is the set of edges. Every edge $e \in E$ has a non-negative cost or length c_e and a non-negative demand for service d_e. The edges with positive demand make up the subset of the required edges E_R. In CARP, the graph is typically undirected, meaning that each edge can be traversed in either direction with equal cost. Throughout this thesis, the terms cost and distance are used interchangibly. The demand d_e of an edge e represents the quantity of some resource or service that must be delivered along that edge. Each vehicle has a maximum capacity Q and the total demand of all edges in its route cannot exceed this capacity. Given a vehicle capacity Q, the CARP consists of finding a set of vehicle routes of minimum cost, such that every required edge is serviced by exactly one vehicle, each route starts and ends at a prespecified vertex $v_0 \in V$ (the depot) and the total demand serviced by a route does not exceed the vehicle capacity Q. The number of maximum vehicles K used is also constrained. Golden and Wong [GW81] show that the CARP is NP-hard.

Chapter 3
Related work

Capacitated Arc Routing Problems (CARP) are known to be NP-hard problems. Due to its complexity, it is possible to solve it exactly only for small-sized instances. Instances of larges size usually make use of heuristic, more specifically metaheuristic approaches.

This chapter gives an overview of known

3.1 Exact and lower bound methods

Lower bound methods provide a tight lower bound on its optimal cost. Such a bound is helpful when evaluating larger CARP instances, where heuristic approach has to be employed, since solving them exactly would be computationally too demanding and not feasible at all. Thus, achieving a solution which is close to a lower bound might be a good measure of quality for heuristic algorithms. A simplified integer linear model was proposed by Belenguer and Benavent [BB03]. The sparse formulation used does not lead to a valid CARP solutions, but presents very tight lower bounds for the problem. Only one integer is used for each edge, which results in not being able to say which vehicles service which edges.

First possible way of solving CARP is based on transforming the problem into a node routing problem and then using existing VRP methods to solve it. Quality of the solution depends on how well, meaning how compact such a transformation can get. The goal is for the dimension to not increase drastically. First transformation of its kind was introduced by Pearn, Assad and Golden [PAG87] which reduced the CARP problem into CVRP problem, but was regarded as unpractical, since the transformed CVRP problem had a graph with 3e + 1 vertices, where e is the number of required edges in CARP. Similar transformation was then proposed by Longo, Aragao and Uchoa [LdU06], which further reduced the number of vertices to 2e + 1. Combined with a branch-and-cut-and-price algorithm, they observed effective results, solving all gdb instances for the first time and finding new optimum for val files and two for egl files. More recently, a compact transformation was introduced recently by Les Foulds et al. [FLM15] where the number of nodes is at most larger only by one than the number of edges. Again, an adapted version of branch-and-cut-and-price algorithm for CVRPs was used to obtain the results. The authors managed to solve all of the instances from the gdb dataset, however some instances in the larger dataset egl were not solved satisfactorily.

As mentioned by Bode and Irnich [BI12], converting arc routing problems into node routing ones has significant drawbacks, which are models with inherent symmetry, dense underlying networks or models with huge number of vertices. Therefore, it is worth considering specialized CARP methods to address these issues. Specialized exact algorithms for CARP often involve solving integer programs using branch-and-cut or branch-and-bound combined with column generation. Branch-and-cut uses a cutting plane approach, in which inequalities are added to the optimization problem. By adding these cuts, the feasible region of the subproblems can be further restricted, which can help the algorithm find the optimal solution more efficiently. Column generation, on the other hand, involves iteratively generating and adding columns (variables) to the problem's constraint matrix until an optimal solution is found. Mentioned algorithms are used in [BI12], [BCL13] to solve instances with up to 190 nodes to optimality. Instances with number of nodes greater than 200 remain unsolved by exact approaches.

3.2 Heuristics

Heuristics algorithms are approximate methods that are used to quickly find a solution to an optimization problem that is likely to be close to the optimal solution. They are typically used when the exact optimization problem is too computationally expensive to solve in a reasonable amount of time, which is the case for solving larger instances of the CARP. The main focus will be on meta-heuristics, which represent more general techniques applicable to wide range of optimization problems.

One of the most famous algorithms for solving CARP is a tabu search algorithm called CARPET proposed by Hertz, Laporte and Mitaz [HLM00]. In CARPET, a solution in represented by a set of nodes representing all traversed edges. Solutions violating vehicle capacity are accepted but penalized. The search process in a tabu search is guided by the tabu rules, which specify which moves are allowed and which are "tabu" (forbidden) at each step. Number of improvement procedures which are used in the search process are presented in CARPET (Shorten, Drop, Add, Paste, Cut, Switch and Postopt).

Subsequently, Hertz and Mittaz in [HM01] applied a new algorithm to solve the CARP, which is the Variable Neighborhood Descent algorithm (VND). It replaces the framework of the tabu search with the framework of the variable neighborhood search and achieves slightly better solutions. It involves exploring a sequence of neighborhoods around the current solution. Several descents with different neighborhoods are performed until a local optimum for all considered neighborhoods is reached. However successful the solutions, the encoding used by CARPET and VND leads to intricate improving procedure, thus potentially making the search space vast.

Lacomme, Prins and Ramdane-Chérif [LPRC01] proposed a memetic algorithm (MA), a genetic algorithm hybridized with a local search). Genetic part of the algorithm is inspired by the process of natural evolution and use techniques such as selection, crossover, and mutation to search for the optimal solution to a problem. Based on this evaluation, the algorithm selects the fittest individuals to survive and reproduce, and combines their genetic material through crossover to create new offspring. Mutation is then used to introduce random changes to the genetic material of the offspring, in order to explore a wider range of potential solutions. MA uses a more compact and natural encoding. Each edge is represented by only two indices, one for each direction. A route can then be defined by a list of such indices. Two consecutive edges in a route are connected by implicit shortest paths, which can be computed in advance. This encoding scheme is very useful when only fraction of edges are required and has been used in almost all metaheuristics published after CARPET and VND. MA achieves is more successful on the standard testing sets than CARPET, while also being twice as fast.

Are recent tabu search algorithm for solving a modified version of the original CARP problem was recently proposed in [LZJQ18]. They consider split-delivery CARP (SDCARP), which generalizes conventional CARP by allowing an arc to be serviced by more than one vehicle. Forest-based tabu search utilizing forest-based neighborhood operators is used in this approach.

A similar memetic algorithm to (MA) with extended neighborhood search (MAENS) was proposed in [TMY09]. This work proposed a novel local search operator, which is capable of searching using large step sizes and is less likely to become trapped in locally optimal solutions.

To tackle the largest CARP benchmark instances, Mei, Tang and Yao [MLY13] present a mechanism called Random Route Grouping (RRG) designed to decompose the large-scale CARP (LSCARP). RRG is combined with a cooperative co-evolution (CC) model to give yield impressive result on large datasets. The cooperative co-evolution framework is a natural way to implement divide-and-conquer strategy. Generally, CC is a type of evolutionary algorithm that involves the simultaneous evolution of multiple subpopulations, or "species," that are interdependent and work together to find a solution to a problem. A bit later, authors of [MLY14] improve on the decomposition procedure by incorporating information about the quality of the best solution found in the search.

Another group of possible meta-heuristic approaches are ant-colony algorithms, which are inspired by the behaviour of ant colonies. A set of artificial ants is initialized at selected locations of the network, the network is then explored by the ants which are combining local information (the cost of the arc connecting the current node to the next one), with the global information (pheromone levels on the arcs). Pheromone levels store the information about the quality of the solutions found so far. Ants deposit pheromones on the arcs as they traverse, which influences the behaviour of other ants.

Tirkolaee et al. [TAH+19] introduce an ant colony based metaheuristic with some modifications. They use a modified version of the Ant Colony Optimization algorithm derived from Ant System called Max-Min Ant System (MMAS). MMAS was firstly presented by Stützle and Hoos in [SH00], their main contribution was the introduction of upper and lower bounds for the value of the pheromones which avoids stagnation of the search. [TAH+19] further improves the performance of MMAS by utilizing a mechanism called Pheromone Trial Smoothing (PTS), which results in preventing premature convergence, avoiding local optima and increasing efficient search space.

In [MLRR11], a biased random key genetic algorithm is combined with a local search. Optimal or near optimal solutions were obtained while achieving small computation times during testing on sets of CARP benchmark instances. Classical local search methods which “fine-tune” its solutions are used to potentially find better ones. Local search might be applied in different ways. One is to pass the best solution found by RKGA to a local search algorithm to be further optimized, another possibility is to use local search as a mutation operator within RKGA.

Open CARP is a variant of the original CARP problem which releases the constraint which states that tours must begin and end at a depot, which means that the tours in this variant do not have form cycles. A recent work of [AU18] deals with the open CARP by introducing a Hybrid genetic algorithm, whose main features is standard genetic algorithm combined with local search and feasibilization procedure which is responsible for obtaining a feasible solution from chromosome. Feasibilization proved to have substantial role on performance. It also includes a population restart which avoids premature convergence of the population, which happens when the genetic diversity is low and only small are of the search space is being explored.

In some cases, the demand for a product or service may be uncertain or subject to random fluctuations. Such behaviour is modeled by one of the most recently studied variant of the CARP problem, the Uncertain CARP (UCARP). It was proposed to better reflect reality. In UCARP, the travel cost between vertices in the graph and demand of tasks is unknown in advance, and is revealed during the process of executing the services. In this case, a preplanned solution may become worse or even infeasible. Authors of [WMZY21] propose a novel genetic programming approach, which simplifies the routing policies during the evolutionary process using a niching technique, which leads to a more interpretable policies. Niching is a technique used to preserve diversity among populations of solutions. It avoids aforementioned premature convergence, where the algorithm would get stuck in a local optimum. Instead of having a single population of solutions that all evolve together, niching involves dividing the population into subpopulations, or niches. These niches contain solutions that are similar to each other, but distinct from those in other niches. This way, the search space is expanded, increasing the chances of finding the best solution.

Chapter 4

Preliminaries

4.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are a diverse family of optimization techniques rooted in the principles of biological evolution. Mimicking nature's underlying processes, they use mechanisms inspired by natural selection and genetics to solve complex search and optimization problems. The general approach of EAs is to maintain a population of candidate solutions for the problem at hand and to iteratively improve this population over time. They are characterized by their population-based search approach, their utilization of stochastic processes, and their capability of maintaining and exploring a diversity of solutions. This provides an inherent robustness, allowing for a versatile exploration of the search space, and makes EAs well-suited for a wide range of problems, including those with large and intricate search spaces, non-linear relationships, or poorly understood fitness landscapes. The versatility of EAs is showcased in the "Humies" competition, hosted at `https://human-competitive.org/`. This competition serves as a platform to demonstrate how EAs can excel in various domains by producing solutions that are comparable to or even outperform those created by human designers.

The main operators utilized within EAs are selection, crossover (or recombination), and mutation, which emulate the mechanisms of survival of the fittest, mating, and random genetic mutation respectively.

Pseudocode of a general evolutionary algorithm:

```
1: Initialize population $P_0$
2: while termination condition is not met do
3:     Evaluate fitness of individuals in $P_t$
4:     Select parents from $P_t$
5:     Generate offspring by crossover and mutation
6:     Evaluate fitness of offspring
7:     Select individuals for $P_{t+1}$ from parents and offspring
8: end while
```

Above, the provided pseudocode outlines a general evolutionary algorithm in five steps. First, it initializes the initial population of candidate solutions.

Then, the algorithm iteratively evaluates the fitness of individuals in the current population. Next, parents are selected from the population, and offspring are generated through crossover and mutation operations. The fitness of the offspring is then evaluated. Finally, individuals for the next population are selected from both the parents and the offspring. This iterative process continues until the termination condition is met, which usually is the maximum number of generations. The pseudocode serves as a flexible template for implementing various evolutionary algorithms tailored to specific optimization problems.

It is necessary to provide a summary of key terms and concepts commonly used in evolutionary algorithms.

Term	**Description**
Gene	The fundamental unit in an individual's solution, representing a specific piece of information or parameter.
Genotype	The complete set of genes that make up an individual's solution. It represents the encoded solution in the search space.
Phenotype	The manifestation of the genotype in the problem space, denoting the actual solution to the problem.
Individual	Represents a single solution to the optimization problem, characterized by its genotype and associated phenotype.
Fitness	The quality or suitability of a solution, measured by a problem-specific fitness function.
Population	The collection of individuals in a given generation, representing the pool of current solutions in the search space.
Evolution	The iterative process of generating new populations with potentially improved fitness over generations through selection, crossover, and mutation.

Table 4.1: Key Concepts in Evolutionary Algorithms

While all EAs share these commonalities, different types of EAs have emerged, customizing these concepts to particular problem types or application areas. Each type has its unique features and specializations, with notable examples including Genetic Algorithms, Genetic Programming, Differential Evolution, Evolution Strategies, and Evolutionary Programming [M+18]. Genetic algorithms and their memetic extension will be discussed in further detail below.

Genetic Algorithms (GAs) constitute a significant branch of EAs, with a strong emphasis on the mechanisms of natural selection and genetics. Taking inspiration from Charles Darwin's theory of natural evolution, GAs maintain a population of candidate solutions that evolve over generations.

Although GAs are a part of the broader EA family, their distinctive feature lies in their particular implementation of the evolutionary principles. The concrete representation of solutions as chromosomes, the clear distinction of generations, and the straightforward usage of genetic operators make GAs a robust and versatile tool for a wide array of optimization problems.

Each individual solution in a GA is characterized by a set of parameters or variables, encoded in a data structure analogous to a chromosome. These chromosomes can be binary strings, real-valued vectors, or other appropriate structures depending on the specific problem being addressed.

The fitness of each individual is assessed using an objective function specific to the problem, similar to how an individual organism's fitness for survival might be measured in nature. This objective function acts as the primary evaluator and driver for the progression of solutions, pushing the evolution process towards optimal or near-optimal solutions.

GA operates using three main genetic operators:

- Selection: This operator mimics the survival of the fittest principle. It selects the individuals with higher fitness values to pass their genes to the next generation.
- Crossover (or Recombination): It emulates the genetic recombination observed in nature, where offspring inherit genetic information from their parents. In GA, it is a method to create new candidate solutions by combining parts of the 'chromosomes' of two or more selected individuals from the current population.
- Mutation: This operator introduces random modifications in the chromosome of individuals, promoting genetic diversity and enabling the exploration of new areas of the search space.

4.2 Local Search Procedures

Local search optimization is a crucial aspect of evolutionary algorithms, providing an exploration of the neighborhood of solutions to refine the global search. It can greatly enhance the performance of the evolutionary algorithm by allowing it to find better solutions that might be missed in the course of the global search.

A local search is conducted by perturbing the current solution slightly to create a neighboring solution, then comparing the fitness of the new solution to the fitness of the current solution. If the new solution is better (i.e., it has a higher fitness), it replaces the current solution, and the process is repeated. This is commonly known as hill climbing, since it can be visualized as climbing the peak of a fitness landscape.

Given the stochastic nature of evolutionary algorithms, the incorporation of local search techniques adds an additional layer of robustness and effectiveness to the solution process. As a result, evolutionary algorithms provide a strong global search capability, local search on the other hand allows for refinement and exploitation of the best solutions found, which again provides a balance between exploring and exploiting possible solutions.

4.3 Memetic Algorithms

Memetic algorithms (MAs) represent an extension of the traditional genetic algorithms, which on top the genetic framework employ local search operators during the computation. They were introduced by Moscato in [M$^+$89]. MAs are described by Moscato as "a marriage between a population-based global search and the heuristic local search made by the individuals." The word "meme", which was the inspiration for the term memetic algorithms, denotes the idea of a unit of imitation. Moscato uses the analogy of martial arts to describe memes as those undecomposable movements, which when individually composed form a more complex movement. Put simply, memetic algorithms improve genetic algorithm, which rely almost entirely upon recombination mechanisms to improve solution quality, by combining them with some kind of local optimisation of each individual in population.

4.4 IREANN

The foundation for the proposed extension in this thesis is based on the work of Kubalík and Snížek [KS14]. Their research introduces an Evolutionary Algorithm with Indirect Representation and Extended Nearest Neighbor Constructive Procedure (IREANN), specifically designed for solving the Traveling Salesman Problem (TSP). The functionality and effectiveness of the IREANN algorithm in addressing the TSP are demonstrated in their study.

4.4.1 Indirect representation

IREANN uses an indirect representation as a sequence of required nodes which is called a priority list, where the order of nodes define the order in which they will be inserted into an existing tour via extended nearest neighbor heuristic. For example, a priority list 5, 2, 4, 1, 3 represents a solution that is constructed through steps of application of the nearest neighbor heuristic to the cities 5, 2, 4, 1 and 3, in this order.

The optimal solution can be represented by various priority lists, depending on the nature of a specific TSP instance which is being solved. This flexibility is advantageous as it allows the optimal solution to be attracted by multiple different priority lists. However, it is also possible that the given representation may not be able to reach the optimal solution if there is no priority list that accurately represents it.

4.4.2 Extended Nearest Neighbor Constructive Procedure

The central objective of the Constructive Nearest Neighbor Procedure (CNNP) is to formulate a valid tour of length n based on any given priority list. Note, that the terms "Extended Nearest Neighbor Constructive Procedure" and

```
1: Initialize $n$ single node components $start_i = i, end_i = i$ for $i = 1, ..., n$
2: $j \leftarrow 1$
3: do
4:     Take $j$-th city, $P[j]$, from the priority list $P$
5:     Identify component $C_k$ to which city $P[j]$ belongs
6:     if $P[j]$ is either $start_k$ or $end_k$ of $C_k$ then
7:         $N \leftarrow$ nearestNeighbor($P[j]$)
8:         add edge ($N$, $P[j]$)
9:     else
10:        $N_1 \leftarrow$ nearestNeighbor($start_k$)
11:        $N_2 \leftarrow$ nearestNeighbor($end_k$)
12:        if dist($N_1, start_k$) $\leq$ dist($N_1, end_k$) then
13:            add edge ($N_1, start_k$)
14:        else
15:            add edge ($N_2, end_k$)
16:        end if
17:    end if
18:    $j$++
19: while $j \leq n$
```

Figure 4.1: The extended nearest neighbor construction procedure

"Constructive Nearest Neighbor Procedure" are used interchangibly in this thesis. The procedure begins with n separate tour components. Each tour component, denoted as C_k, is marked by its boundary cities $start_k, end_k$. Initially, every city serves as its own distinct component, implying $start_i = i$ and $end_i = i$ for each city i from 1 to n.

Throughout the procedure, the algorithm iterates over the priority list, using the nearest neighbor heuristic at each step. At each iteration i, it processes an element from the priority list $P[i]$. Depending on $P[i]$, a component C_k is identified, which may contain one or multiple nodes.

If $P[i]$ turns out to be a boundary node of C_k, the procedure picks the city closest to $P[i]$ and adds it to the tour. Alternatively, if $P[i]$ is not a boundary node, it locates the nearest neighbors of the boundary nodes $start_k$ and end_k of the component C_k, labeled as N_1 and N_2. Of the two possible edges $(N_1, start_k)$ and (N_2, end_k), the one with the shorter distance is chosen and added to the tour.

At every step, the process merges two components into a single one. This sequence of actions repeats until only a single component is left, symbolizing the final route.

Figure 4.2 provides a graphical illustration of this procedure, constructing a route from the priority list P ={A, I, H, F, J, E, C, D, G, B}. As observed, 4.2 a) processes {A, I, H, F}, wherein each step selects the shortest link to a node's neighbor, leading to two components {A, J, I} and {F, G, H}. However, when the node J is considered, it's already part of a component and not a boundary node. Hence, the nearest neighbor is sought from nodes

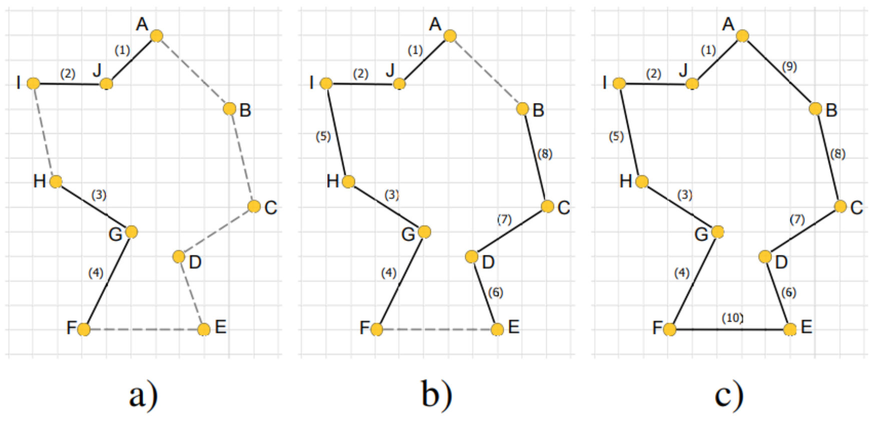

Figure 4.2: Example of the extended nearest neighbor constructive procedure (source: [KS14])

I and A. Since the link between I and H is shorter than that between A and B, the components are connected through the I and H edge. The final route, as shown in 4.2 c), is derived by applying the same rules to the remaining priority list.

As previously mentioned, several priority lists might eventually represent the same route. In this straightforward instance, {A, I, H, F, J, E, C, D, G, B}, {I, C, E, B, H, F, A, J, G, D}, and {F, A, C, H, I, E, D, B, G, J} all lead to the same final route.

Chapter 5
Proposed method

This chapter proposes an extended version of the original IREANN algorithm, whose main contribution is the incorporation of a feature extraction and propagation mechanism. This mechanism aims to leverage high-quality features from individual solutions and propagate the knowledge across the entire population. The primary purpose is to increase the algorithm's effectiveness in dealing with complex combinatorial optimization problems. This mechanism is described in detail in section 5.2.

As a test case for the extended IREANN, an arc routing problem named the Capacitated Arc Routing Problem (CARP) was chosen. However, IREANN was originally designed to solve the Travelling Salesman Problem, a vehicle routing problem. That implies some changes to the core IREANN algorithm were necessary to adapt it to the nuances of CARP. CARP introduces additional complexities not present in the Travelling Salesman Problem. It is a more complex problem that not only requires routing, but also involves servicing edges or arcs with specific demands while defining capacity constraints of the vehicles.

The adjustment of the inner representation of routes for each individual in the population and the modification of the nearest neighbor heuristic is crucial for the domain of the CARP, furthermore the adaptation of local search operators specifically for CARP, which play a crucial role in the algorithm's efficiency, is also required.

On the other hand, the core components of the evolutionary framework, specifically, the selection, crossover, and mutation operators remain largely unchanged. This is because these genetic operators are fundamentally problem-independent and can be applied in the same way across a wide variety of combinatorial optimization problems.

5.1 IREANN customizations to CARP

In CARP, the atomic element shifts from a node to an edge. Instead of cities as it is the case in TSP, the entities to be visited are now the required edges of a graph, each having a specific demand to be serviced. This change in representation has a direct impact on the design of the nearest neighbor heuristic as well, which is a key component in constructing new solutions.

5.1.1 Indirect representation

Proposed customized version of IREANN, uses priority list of edges instead of nodes. Similarly to the original IREANN [KS14] for TSP, the priority list represents the order in which the nearest neighbor heuristic will be applied, but in this case, on edges during the process of developing the set of tours.

The priority list, serving as the genotype within the evolutionary framework, shifts from representing a sequence of nodes or cities in TSP to representing a sequence of required edges in CARP. The overall functionality remains the same as it was the case in the original version, which means that the priority list represent the order in which the nearest neighbor heuristic is applied.

5.1.2 Extended nearest neighbor constructive procedure

In section 4.4.2, the constructive nearest neighbor procedure (CNNP) was introduced and its functionality was demonstrated on the TSP. However, directly applying the same CNNP to the Capacitated Arc Routing Problem is not possible. The transition to CARP requires several modifications to the CNNP to effectively handle the distinct requirements and constraints of this more complex problem.

Procedure is formally described in Fig. 5.1 The process starts with n independent tour components, where n is the number of required edges. Each required edge initially belongs to its own component. Each component C_k is defined by its boundary nodes $start_k$, end_k. Let us define boundary edge as a required edge which on either of its two ends, is not connected to any other required edge. Futher, a boundary node is a node belonging to a boundary edge, through which the edge is not linked to any other required edge. Which means that in the beginning of the CNNP, boundary nodes of all components are the two nodes of each corresponding required edge.

In each step of the constructive procedure, either two components are merged together if their conjunction satisfies the capacity constraints, or nothing is done if there is no pair of route components that would meet the maximum capacity constraint Q after they were merged together.

In particular, in i-th step the corresponding requried edge at i-th position in the priority list, denoted as working edge $P[i]$, is taken and the component to which the edge $P[i]$ belongs, C_i, is identified. Nearest available neigbors to the $start_i$ and end_i boundary nodes of the component C_i, N_1 and N_2, are found. N_1 and N_2 have to meet several constraints imposed by the CARP definition. Firstly, both N_1 and N_2 have to be boundary nodes of some other components C_j, C_k where $i \neq j$ and $i \neq k$.

Secondly, only boundary nodes of C_l, such that the sum of demands along components C_i and C_l is less or equal to the maximum capacity Q, are considered. This is to ensure that the maximum vehicle capacity constraint Q will not be violated.

Finally, the shorter path out of (N_1, $start_i$) and (N_2, end_i) is added to a constructed solution while merging together the component C_i with the component containing the selected boundary node (N_1 or N_2). A path between two arbitrary nodes, v_1 and v_2, refers to the shortest possible path on the entire graph G. It is important to stress, that this path can potentially include passing even a required edge. By definition, the passing of required edges without servicing them is permitted. This allows for more flexibility in finding the shortest path between the specified nodes.

Note, that it is possible that no other node in the entire graph with its corresponding component satifies the vehicle capacity constraint. That situation would result in continuing to (i+1)-th step right away, without extending any component. The procedure ends after all n required edges from priority list have been processed. At the end, we are left with a certain number of components $|\mathcal{C}| \leq n$, i.e. multiple separate routes, where each route is served by a single vehicle. $|\mathcal{C}|$ might be greater than the maximum number of vehicles allowed K, which would make the constructed set of routes an invalid solution, this drawback is discussed in further detail in section 5.1.3.

To sum up the differences between the original constructive procedure for TSP and this modifed version for CARP, most notably we have to check each candidate whether is satisfies the constraint imposed by the CARP definition. The rest of the procedure is very similar, we are just dealing with edges instead of individual nodes.

5.1.3 Solution feasibility

Because of the nature of the capacitated arc routing problem, the challenge of solution feasibility arises. After the evaluation of given individual using the nearest neighbor constructive procedure, it is possible the resulting set of routes violates the constraint of maximum vehicle used. Such problem would not come up if the goal was to solve the Travelling salesman problem using similar heuristic to CNNP, but for the CARP which defines the maximum vehicle contraint, solutions violating this constraint are not acceptable.

Another constraint is the maximum capacity of each vehicle, meaning that the sum of demands of each required edge in a single route must not exceed the defined limit Q, which is same for the whole fleet. The solutions however can not possibly violate this constraint thanks to the way the routes are constructed via the CNNP. (See section 5.1.2) CNNP does not allow a route to be extended with another one which would result in such violation. That results in only the maximum vehicle count constraint being vulnerable to violation, not the maximum capacity one which is always satisfied by the innate design of this approach.

To deal with the infeasibility obstacle, one idea would be to leave every infeasible solution out of the population of candidate solutions and not

```
1: Initialize $n$ single edge components $start_i, end_i$ are the boundary nodes
   of each required edge $i$ for $i = 1, ..., n$
2: $i \leftarrow 1$
3: while $i \leq n$ do
4:     Take $i$-th edge, $P[i]$, from the priority list $P$
5:     Identify component $C_i$ to which edge $P[i]$ belongs
6:     $(N_1, C_{N1}) \leftarrow$ nearestNeighbor$(start_i)$ where sum of $C_i$, $C_{N1}$ satisfies
   $Q$ constraint
7:     $(N_2, C_{N2}) \leftarrow$ nearestNeighbor$(end_i)$ where sum of $C_i$, $C_{N2}$ satisfies
   $Q$ constraint
8:     if dist$(N_1, start_i) \leq$ dist$(N_1, end_i)$ then
9:         add path$(start_i, N_1)$
10:        merge components$(C_i, C_{N1})$
11:    else
12:        add path$(end_k, N_2)$
13:        merge components$(C_i, C_{N2})$
14:    end if
15:    $i{+}{+}$
16: end while
```

Figure 5.1: Extended nearest neighbor constructive procedure modified for CARP

consider them at all. However, this approach would cause a lot of trouble, because in order to create the initial population, chromosomes of individuals in the first generation are set arbitrarily. Those chromosomes with randomly ordered genes are very unlikely to generate a valid solution in terms of the number of vehicles used, which would make it almost impossible to generate an initial generation of only individuals with feasible solutions.

Instead, individuals representing infeasible solutions are allowed to exist in the population, but we need a way of telling which infeasible solutions are better than others in order to guide the computation in the correct direction and thus converge to a state where there is a population with feasible individuals.

That is where the fitness evaluation of each individuals comes into play.

5.1.4 Individual Fitness

Using the terminology introduced in table 4.1, in the evaluation process the individual's genotype is taken as input. It is then converted to its corresponding phenotype, which serves as the basis for computing the final fitness, which is the output of this procedure.

To make it clear, in the case of our implementation of extended IREANN algorithm for the CARP, genotype is the priority list of required edges, which is what mainly defines each individual. However, because of the indirect representation aspect (see section 5.1.1), this priority list (genotype) is converted into the actual set of routes (phenotype) the fleet of vehicle has to travel through via the means of CNNP (see section 5.1). It is this set of routes upon which the fitness is computed, not the priority list. Because of the reasons stated in sections 5.1.3, we can not simply define the fitness of an individual by a single number representing the cumulative cost of its set of routes. We also need to capture the count of such set of routes, in order to be able to tell whether the individual violates the maximum vehicle constrained (as discussed in section 5.1.3).

As a result, the fitness is represented by the pair of values being (1) the total cumulative cost of the set of routes computed by the CNNP, and (2) the number of routes in such a set of routes.

Formally, we define the fitness of an individual as a value pair $\{cost,\ vehicle\ count\}$, where

Further clarification on why the evaluation of an individual involves a pair of values, rather than just one, is provided in section 5.1.3.

5.1.5 Comparison of Individuals

For the reasons stated in section 5.1.3, we have to come up with a mechanism which incentivizes the population to converge to a state where there are only individuals with valid number of vehicles.

Ultimately, the desired functionality is to be able to tell which individuals are the fittest with respect to each other. That enables us to create hierarchy within given population of individuals, giving the more fit individuals higher chance of "survival" through the selection operator, which is the main driving force behind the evolutionary process towards finding optimal solutions.

We are not able to establish such a hierarchy just by sorting the population by the measure of the cumulative cost of all the routes in an individual's solution. Although that is the end goal of the whole computation, to find a solution that minimizes this attribute, we can not neglect the number of vehicles used during this process. The reason behind that is the possibility of a situation where a solution with lower cumulative cost and a number of vehicles over the limit K would be considered more fit than a solution with satisfactory number of vehicles and a greater cost. Put simply, a solution that violates the constraints defined by CARP (i.e., one with $|C| > K$) would be considered more fit than a valid one, which is obviously an undesirable behaviour which would disable the evolutionary process from achieving any progress.

Optimization of two variables at the same time is not possible, so the fitness evaluation will prioritize driving the vehicle count down first, and after that optimize the total routes cost when a satisfactory vehicle count has been reached. This desired functionality is achieved by designing a custom comparator, that decides which one of two individuals is "more" fit. This comparator is utilized to sort the population, which results in having an ordered sequence of individuals with the fittest ones at the beginning. In the end, that is what the fitness function is for, to tell how good the solutions are with respect to each other so that during the selection phase (details in section 5.1.6), the right individuals in population are picked to be mated with each other, according to the selection rules.

Comparator is a method which takes two individuals as input and decides which one of them is more fit according to criteria mentioned above. Several options might occur:

- Vehicle count of both individuals is greater than the limit, then the one with lower vehicle count is deemed more fit. The cost is the deciding factor only when the vehicle counts are equal.
- First individual has satisfactory vehicle count, the other has not. In this case, the first individual is preferred no matter the cost of their solution.
- Both vehicle counts are less or equal than the limit, then the individual with lower cost is preferred.

5.1.6 Selection

It is through effective selection that promising individuals are identified and retained, contributing to the improvement of solutions over time. It is the iterative process of selecting and evaluating individuals which guides the evolutionary algorithm towards better performing solutions.

In our implementation, the tournament selection method is utilized as the primary selection mechanism. This method involves selecting a fixed number of individuals, known as the tournament size, from the population. The tournament size is an arbitrary hyperparameter that can be adjusted based on the desired selection pressure. The individual with the best fitness score is the winner of that tournament.

By choosing a larger tournament size, more individuals participate in each tournament, increasing the competition and favoring the selection of fitter individuals. This intensifies the selection pressure and promotes exploitation of promising solutions. Conversely, a smaller tournament size allows weaker individuals to have a better chance of being selected, facilitating exploration and diversifying the population.

In our case, we need to run the tournament selection twice in order to obtain the parent individuals needed to produce an offspring. Our tournament selection procedure has two paramenters t_1 and t_2, which represent the tournament sizes for both selected parents. Both group of individuals in each tournament are sorted according to our comparator described in section **??**, and the first element, considered the winner of a tournament is selected to be one of the parents for the offspring, which is about to generated through means of recombination from the two parent individuals.

5.1.7 Crossover

After performing the tournament selection process on the current population, a pair of parent individuals is selected. For each pair of parents, two offspring are created using crossover and mutation operators.

We used an *order-based crossover*, defined by Syswerda in [Sys91]. The crossover constructs an offspring so that first several cities randomly chosen from the priority list of the first parent are copied to the offspring into the same positions as they appear in the parent. The remaining positions are filled in with the remaining cities, in the same relative order as in the second parent.

5.1.8 Mutation

Mutation operator is very simple, it randomly changes the position of a single edge in the priority list. While some argue that memetic algorithms may not require a mutation operator due to the local search component, we have chosen to retain it in our algorithm. The mutation operator provides an additional source of exploration and helps maintain genetic diversity within the population.

5.1.9 Local search optimization

Local search procedures, introduced in 4.2 play a crucial role in yielding competitive results [M$^+$18].

We implemented several local search procedures, which operate on the phenotype level of solutions. That means, we try to tweak the inner solution representation ever so slightly (i.e., modify the set of routes in one of many ways), while seeking improvement in the objective function. If we come across such an improvement, we incorporate it into the representation of given individual.

Our implementation of local search operators was heavily inspired by the work [TMY09] by Tang, Mei and Yao. They presented one of the most famous algorithms for solving the capacitated arc routing problem, abbreviated the "MAENS". Several local search heuristics utilized in MAENS were in some form adopted for this implementation.

Firstly, we list three traditional move operators, namely the single insertion, reversal and 2-opt moves. All of the moves mentioned operate in a deterministic manner, trying out every possible combination looking for improvement in the fitness score. A greedy approach is favored, which means that if an improvement is found, it is applied right away.

Single insertion

In single insert move, an edge is removed from its current position and reinserted into another position in the sequence of edges. Edge might be reinserted either to the route, in which it originally was, or to any other route is the whole set that given individual possesses.

Reversal moves

Reversal move simply reverses the direction of an edge.

2-opt moves

We differentiate between two types of 2-opt moves, one for a single route and the other for double routes. 2-opt for single route reverses the direction of its whole subroute, and reconnects it accordingly. 2-opt for double routes disconnects two routes, which results in four different subroutes, which are then optimally reconnected.

Merge-Split operator

As the authors of [TMY09] argue, all of the traditional move operators mentioned above adopt a rather simple schemes to generate new solutions, which results in new solutions that are very similar to the original ones. They describe them as having a "small" step size, thus being capable of searching only a "small" neighborhood. It is further discussed, that it would be neat to have a move operator with larger step size. That could be theoretically achieved by extending the traditional move operators, but would be too computationally demanding. For that purpose, the Merge-Split (MS) operator is devised by the authors of [TMY09].

Our implementation takes inspiration from MS operator and uses a simplified version, the improvements it yields are vastly superior to traditional move operators. It basically selects $p(p>1)$ routes of a given individual, merges them together and tries to reconnect them in a way that leads to a decrease in the total cost. When reconnecting this subset of individual's set of routes, the same constructive procedure to the one used during evaluation of offsprings is employed. Which basically means, that we are trying to reconnect all the edges in the selected subset again via the CNNP, but this time, all the edges which are part of routes that were not selected by the MS operator, are not visible at all. That heavily influences the workings of CNNP, which thanks to this functionality is very likely to discover new possible connections, which would be unattainable if the CNNP considered every single edge in the graph as it is the case during standard offspring evaluation.

5.1.10 Dealing with duplicate solutions

Maintaining a diverse population is crucial in evolutionary algorithms to ensure effective exploration of the search space and avoid premature convergence to suboptimal solutions. One aspect of diversity is the absence of duplicate individuals within the population. Duplicates can limit the exploration capability of the algorithm by occupying multiple slots with identical solutions, reducing the diversity of available genetic material. By preventing duplicates, the algorithm is encouraged to explore a wider range of solutions, increasing the chances of finding better and more diverse solutions. Additionally, a diverse population facilitates the exploration of different regions of the search space, enhancing the algorithm's ability to converge to high-quality solutions.

It would be easy to ensure that there are no two individuals with the same genotype, i.e. priority list. But if two individuals have exactly the same priority lists, they might still be different on the phenotype level. Their inner representation might differ thanks to the local search procedures applied. That is why we came up with a way of checking for duplicity on a deeper level, where the inner representation of each individual is considered. Basically, two individuals are deemed identical, if the set of routes each of them possesses, is exactly the same.

There still may be cases where allowing a certain degree of duplicity can be advantageous. To address this, we introduce a parameter called *maxDuplicates*, which specifies the maximum number of identical individuals with the same set of routes that are allowed to proceed to the next generation. This parameter offers a level of control over the tolerance for duplicity within the population. The effects of this tweaking this paramaters are demonstrates in experimental study chapter of this thesis.

5.1.11 Extended IREANN algorithm

All the components discussed in this chapter are combined to form an extended IREANN algorithm, which addresses the capacitated arc routing problem. The proposed algorithm falls under the category of memetic algorithms and builds upon the IREANN algorithm [KS14], incorporating modifications specific to the CARP, as well as a mechanism, which during computation identifies high-quality features and propagates that information across the entire population (described in detail in chapter 5.2).

Figure 5.2 gives an overview of the extended IREANN in pseudocode. As we can see, the computation consists of *maxGenerations* number of total generations. Firstly, an initial population of candidate solutions is created by means of random perturbation of a list of all required edges. All of these individuals in the initial population are expected to be a very poor quality.

During each iteration of the evolutionary process, a new population of offspring individuals, referred to as the "interPopulation," is created. This interPopulation is the same size as the original population. Depending on the value of parameters *probCross* and *probMutation*, a different way of generating an offspring might be employed. *probCross* represent the probability, that offsprings will be, in given iteration, generated via the means of crossover operator. *ProbMutation* represents the probability those offspring are further mutated to achieve more genetic variety. With probability $1-$ *probCross*, the offspring in that single generation will have the same priority lists as their parent, except they are just mutated. It is reasonable to keep the value of parameter *probCross* close to 1.

The fitness of each new offspring created in the interPopulation is then evaluated through a process described in section 5.1.4. This fitness score is then further improved by numerous local search procedures introduced in 5.1.9.

The two populations are then merged into a single one, which is twice the size. This large population is then sorted according to criteria described in section 5.1.5, favoring the "more fit" individuals to be listed closer to the beginning of this sequence. Only the first half this sorted large population survives and “makes it” to the next generation (becoming the new resulting population the same size as in the beginning of this process), the second half of individuals is considered to be of lesser a quality and is tossed away. A mechanism that filteres out duplicate candidates, i.e. identical individuals is incorporated as well, explanation is provided in section 5.1.10.

Is important to mention, that all of the terms *popSize, maxGenerations, probCross, probMutation, tournamentSize,* M and k are all parameters of the main evolutionary method, whose values need to be carefully chosen as they have a significant influence on the performance of the algorithm. As hyperparameters, there is not one universally optimal set of values for them that will work best for every problem. Instead, their optimal values need to be determined through a process of experimentation, observation, and adjustment. Parameters M and k are specific to the IREANN extension and their effect is dicussed in section 5.2.

5.2 IREANN extensions

We have proposed an extension to the IREANN algorithm, specifically adapted for the CARP.

Firstly, a motivation behind this extension is described. Local search operators have the potential to substantially improve the fitness of an individual. They are applied during the evolution process to every newly generated offspring (see line 36 of 5.2). Local search procedures operate on the inner representation of individual’s routes, constructed by the CNNP from the priority list. They improve the fitness of an individual by reconnecting required edges within the set of existing routes. Hence, it is impossible to mirror back the information about local improvements into the individual’s priority list and benefit from that information later on. In other words, there is no way to share the local improvements applied in one individual to other individuals in the current and subsequent populations.

Each time a new offspring is evaluated, the construction of a new set of routes through the CNNP is implied. The extension we introduce incorporates a process we’ve named “Analysis” of a population. This mechanism periodically computes and retains data regarding the characteristics contributing to high-quality solutions. This information is then incorporated into the inner workings of CNNP, acting as an intermediary to the information about actual distance between two nodes. In essence, this extension provides the CNNP algorithm with a more focused information, enabling it to construct better solutions by utilizing insights gained from well-performing individuals produced in the past generations.

The proposed extension is incorporated into the pseudocode provided in figure 5.2. The core of the extension is on lines 8 to 22. It relies on the analysis procedure that is conducted with a certain frequency, see lines 11 and 16. It scans the population for high-quality features among the fittest individuals. Using that information, it then alters the data in a data structure we have nicknamed the “journal”. Journal serves as an augmented version of the distance table used in the CNNP. It adds a layer of additional information on top of the actual distances between nodes. This information isn't about physical distances between nodes in the graph G, but rather about the “value” of connecting two nodes based on previous successful solutions. For every node, journal keeps track of an ordered list of nearest neighbors. The metric for a “nearest” neighbor has shifted from the original distance to a more insightful measure. This measure takes into account not only the spatial proximity of the two nodes, but also the success ratio of the link between the two nodes as justified by the evolutionary process so far.

For every two boundary nodes on the graph G, the journal incorporates the information about the average fitness of a solution that realizes the connection between these two nodes. The order of nearest neighbors for a particular node u is thus determined by the average quality of the solutions in which the neighbors are linked to the node u, rather than only by the actual distance. This shift in metric combines both spatial and solution-quality information, making it a composite indicator.

The main idea behind using the journal derived from analysis, is that final routes present only in elite individuals are very likely to carry important information. This information, if harnessed effectively, holds significant potential in constructing high-quality solutions in subsequent generations. That is why we introduce a new parameter N, which determines the number of best individuals, which will be analyzed.

After assembling the journal from the best N individuals in population, it is then passed to the CNNP, where it substitutes the original distance table during the route construction process. In other words, the journal acts as an augmented distance table that not only includes the original distance information but also represents the accumulated wisdom from previous generations' successful solutions. It should be noted that the journal doesn't replace the original distance table completely. If a connection to a given node is not present in any of the elite individuals (i.e., it was not observed in their routes), priority of such nodes is derived from the distance table. But while keeping the priority of all the nodes, to which the connections were observed in elite individuals, higher.

The CNNP than uses the journal in the following fashion. When CNNP evaluates the nearest neighbor of a given boundary node u (lines 6 and 7 of 5.1), it returns the first element of the priority-ordered sequence of all the other nodes, provided by journal.

This way, the algorithm does not lose the original structural information about the problem while benefiting from the additional insights gained through the analysis of past solutions.

Journal derived from analysis is not immediately used from the first generation of the population. Instead, we introduce a *warm-up* phase, during which the journal derived from the original distance table is used. We define a new parameter *M*, which tells how many generations from beginning this warm-up phase spans over. The motivation behind this decision is to allow the evolution enough time to manifest the original distance information at first, and use the journal later when it gets harder to make any improvements.

After the *warm-up* phase, journal derived from analysis is employed in subsequent generations and is rederived periodically every *k* generations. This *k* number of generation is called a *period.* In each generation of one period, the same journal is used to evaluate individuals. Separating the evolution process into periods serves two main purposes: it mitigates the risk of oscillation of the computation and provides a sufficient number of generations so that the evolutionary process is able to converge toward increasingly better solutions.

Therefore, the journal not only collects the wisdom of the past generations but also adapts to new knowledge that is continuously being generated as the algorithm evolves the population. By doing so, newly emerging patterns in the elite portion of population are recognized and incorporated into the journal's understanding of beneficial connections between nodes.

However, if the information in the journal becomes too dominant, the algorithm runs into the risk of becoming overly exploitative, potentially converging prematurely to sub-optimal solutions and losing its explorative capabilities.

Moreover, in the worst-case scenario, the dominance of analysis derived information in journal could potentially lead to a diverging loop. This could occur if the algorithm is not able to find a new best solution within a single period. That would imply a new journal being derived from increasingly worse solutions. Using such a journal, we are not likely to discover a new best solution. If this cycles repeats, it inevitably leads to a downward spiral of performance.

To counter this, a mechanism allowing to get the computation back on track is incorporated into the algorithm. The algorithm stores the best individual found so far in *BSF_solution* (see lines 6 and 10 of 5.2), along with the journal *BSF_journal*, using which this *BSF_solution* was constructed. Whenever a new *BSF_solution* is discovered, *BSF_journal* gets updated accordingly (see lines 9-13 of 5.2)

Furthermore, the whole process runs in *epochs.* An epoch is a sequence of periods during which the algorithm is permitted to continue without finding a new best solution. The parameter *maxEpochSize* is introduced to set the maximum size of the epoch. This value is essentially a patience parameter that dictates how long the algorithm will persist with the current journal before considering it unproductive. As a result, after these *maxEpochSize* periods have taken place, the journal is reset to *BSF_journal* and a new epoch starts, see lines 18-19 of 5.2.

At the start of each new period, a new journal is derived from the analysis of elite individuals. Hence, the whole population is reevaluated using this new journal. Additionally, the whole population is perturbed (see line 21) . This means that the priority lists of every individual in population are randomly shuffled. At the end of each period, the population is not really diverse (i.e., priority lists of individuals are similar), due to the effects of the evolutionary process. By shuffling the whole population, we aim to incentize more exploration.

Reason behind shuffling the priority lists of the whole population is the fact, that at the end of each period, their diversity is limited due to the evolutionary process.

In this work, we implemented two variants of the analysis procedure that will be described below.

The first version of analysis only considers individual nodes on the graph G and is referred to as “node analysis”. As mentioned above, when analyzing a set of routes of a given individual, the average fitness of every connection is used as a metric for the nearest neighbor heuristic. A single connection is understood as a link between two required edges. In this version, such a connection is defined by a pair of the boundary nodes, which are linked. This implies that during the CNNP, the nearest neighbor heuristic used only the boundary node as a key, upon which the journal returns the sequence of ordered nearest neighbors.

This version, as shown in chapter 6 yields better performance, but only to a limited degree. The reason is that the node analysis only considers the boundary nodes when assembling the journal and when constructing a new route. It does not consider the corresponding boundary edges at all. The reason behind its poor performance is the fact, that by the design of CARP, a single node might be a part of multiple edges. Implying that it does not differentiate between two candidate components, which have the same boundary node, but a different boundary edge. That inevitably leads to a loss of precision

The second version, referred to as “edge analysis”, corrects this behaviour by changing what defines a single connection. Edge analysis considers a single connection to be not just a pair of boundary nodes. Rather, a connection is defined by a pair of a combination of boundary node and the boundary edge it belongs to. This approach aligns more closely with the specific structure of the CARP. It does not neglect to which boundary edge does the boundary node belong to, as it is the case with node analysis.

As a result, the journal captures much more precise information. It contains far more entries, because now the keys to the table provided by journal, are the pairs of nodes and edges, instead of solely nodes. This enables the CNNP to make better decisions when it comes to selecting the nearest neighbor of a component’s boundary node. Performance of both version of analyses is displayed in chapter 6.

...tenhle odstavec je nejspis zbytecny... Let’s illustrate on an example. Let $r = (e_1, e_2)$ be a single route r consisting of only two required edges in this order e_1, e_2. Let's look at the connection between edges e_1 and e_2. Each edge e is further described by its two boundary nodes u_e and v_e, rewriting the sequence of (e_1, e_2) to $((u_{e_1}, v_{e_1}), (u_{e_2}, v_{e_2}))$. This notation implies, that edges e_1 and e_2 are connected through nodes v_{e_1} and u_{e_2}. ....

```
function EXTENDEDIREANN($popSize, maxGenerations, probCross, probMutation, tournament1, tournament2, maxDuplicates, M, k$)
    $j \leftarrow 0$
    $period \leftarrow 0$
    $journal \leftarrow$ deriveFrom(distanceTable)
    population $\leftarrow$ createInitialPopulation($popSize$)
    $BSF\_solution \leftarrow$ population[0]
    for $i = 0$ to $maxGenerations$ do
        if $i \geq M \wedge j \bmod k == 0$ then
            if population[0] is better than $BSF$ then
                $BSF\_solution \leftarrow$ population[0]
                $journal \leftarrow$ ANALYSIS(population)
                $BSF\_journal \leftarrow journal$
                $period \leftarrow 0$
            else if $period < maxEpochSize$ then
                $period{+}{+}$
                $journal =$ ANALYSIS(population)
            else
                $period = 0$
                $journal = BSF\_journal$
            end if
            perturb(population)
            evaluate(population, $journal$)
        end if
        j++
        interPop $\leftarrow$ empty list
        while |interPop| < popSize do
            parent1, parent2 $\leftarrow$ tournamentSelection(population, tournament1, tournament2)
            if rand() < probCross then
                child1, child2 $\leftarrow$ crossover(parent1, parent2)
                if rand() < probMutation then
                    mutate(child1, child2)
                end if
            else
                child1 $\leftarrow$ parent1.mutate()
                child2 $\leftarrow$ parent2.mutate()
            end if
            evaluate(child1, child2, $journal$)
            localOptimization(child1, child2)
            interPop.add(child1, child2)
        end while
        population = population $\cup$ interPop
        population = sortPopulation(population)
        population $\leftarrow$ deleteDuplicates(population, $maxDuplicates$)
    end for
end function
```

Figure 5.2: Pseudocode of the Extended IREANN

Chapter 6
Experiments

6.1 Compared algorithms

The primary goal of this chapter is to validate the effectiveness of proposed extension to the IREANN algorithm. This will be done by comparing the performance of both versions of extended IREANN, described in detail in section 5.2. First version uses the node analysis and is labeled as the "node version", the second one uses edge analysis and is referred to as "edge version". The perfomance of both versions will be compared against the original IREANN, one which does not incorporate any analyzing functionality and its journal relies solely on the information from the distance table. This version is referred to as the "vanilla version".

Yet another version of the algorithm was implemented, only for the experimental purposes. This one is referred to as the "basic version". This version is only a slighty modified vanilla version, which has the functionality of population perturbation at the start of each period, just as it is the case in version node and edge. The motivation behind the creation of this basic version is as follows. We wanted to prove, that the superiority of solution quality of versions node and edge over version vanilla was not achieved just because the vanilla does implement perturbation functionality.

To sum up, the performance of four versions of the IREANN algorithms are compared. Namely vanilla and basic versions, none of which utilizes the extension of IREANN this thesis proposes. The remaining two being node and edge versions, which both utilize the extended IREANN algorithm, but differ in the versions of analysis they employ.

6.2 Configuration of algorithms

The goal of this thesis was to propose an extension to the original IREANN algorithm and compare its performance with the original version. Hence, hyperparameter values were not tunes to achieve optimal performance. Instead, we aimed for a common set of hyperparameters that are consistently applied across all versions of the algorithm, ensuring a fair comparison.

All the hyperparameters the values chosen for them are displayed in table 6.1.

Most of hyperparameters are necessary for all four versions, except for *N, M, k* and *maxEpoch*, which are specific to versions node and edge. Node version also uses the parameters M and k. Explanation of the parameters is provided in chapter 5.

The number of generations and periods size vary in each experiment, more details are provided in 6.4

Hyperparameter	**Value**
Population Size (*popSize*)	300
Number of Generations (*maxGen*)	300, 1000
Number of Duplicate Solutions (*duplicates*)	1
Probability of Mutation (*pMutation*)	0.2
Probability of Crossover (*pCross*)	0.9
Tournament Size for Child 1 (*tournament1*)	7
Tournament Size for Child 2 (*tournament2*)	1
Number of Elite Individuals (N)	10
Warm-up Phase Length (M)	100
Period Size (k)	20, 100, 200
Maximum Number of Epochs (*maxEpoch*)	2

Table 6.1: Hyperarameter Configuration

6.3 Datasets

All the experiments were carried out on one benchmark test set of CARP instances, called the *egl* set. We chose this dataset because of its reasonable size and variety of its characteric properties, which enables us to showcase various aspects of the proposed approach. The egl set was generated by Eglese and described in [Egl94]. There are total of 24 instances. These instances are derived from two graphs, each characterized by unique sets of required edges and capacity constraints. For our testing purposes, a subset of 8 of these instance was selected. Properties of each of them are presented in table 6.2. It includes the total number of nodes on the graph G, number of required and unrequired edges and the constrained maximum number of vehicles a the capacity of single vehicle.

6.4 Experiments

Total of 3 experimental setting was created. Every experiment ran on each of 8 selected data instances. In order to ensure statistical reliability, each individual experiment for given data instance was executed 30 times with different seed configuration. The goal of each experiment, the version of

Name	Nodes	Required edges	Unrequired edges	Vehicles	Capacity
egl-e1-A	77	51	47	5	305
egl-e1-C	77	51	47	10	160
egl-e4-A	77	98	0	9	280
egl-e4-C	77	98	0	19	130
egl-s1-A	140	75	115	7	210
egl-s1-C	140	75	115	14	103
egl-s4-A	140	190	0	19	230
egl-s4-C	140	190	0	35	120

Table 6.2: Dataset Information

algorithm compared and the parameter settings are described in individual sections below. The results presented in form of tables and graphs are layed out in individual section 6.5

6.4.1 Experiment A

vanilla vs node and edge

In the first experiment setting, we want to validate the effects of the extended IREANN (represented by versions node and edge) against the vanilla version. Parameters common to all algorithms are listed in table 6.1.

Number of generations was set to 1000 for versions vanilla and node, edge only runs for 300. M is the same for node and edge version, but different k was selected. Parameter k for node version sits at 100, while it is only 20 for the edge version. That puts the edge version into a disadvantage against the node, because it is given less generations to find new optima before reassembling the journal and perturbing the whole population. And secondly, its computation runs only for 300 generation as opposed to 1000.

6.4.2 Experiment B

The objective of the second experiment setting is to find out whether the basic version of the algorithm (vanilla version with population perturbation) can achieve similar results to version node and edge, which implement the extended IREANN.

All three versions share the same parameters, they run for 300 generations, M is set to 100 and k is 20.

6.4.3 Experiment C

In this experiment, versions basic, node and edge are compared again. But this time, with different parameters M and k of versions basic and node. Version edge is the same as in experiment 6.4.2. We set both M and k of versions basic and node to 200. Making these parameters greater, than it was the case in experiment 6.4.2, has two implications. Firstly the warm-up phase is longer, a secondly more generations within a single period are provided to the algorithm. We want to find out, if that might lead to discovering new better solutions which would not be possible in the previous case with shorter periods.

6.5 Results

Results of all experiments are included in comprehensive table 6.4. Detailed description of the algorithm versions used for each experiment is provided in section 6.4. Let us clarify the abbreviations behind each version of algorithm used in headers of each subtable in 6.4. The used abbreviation along with selected parameters for M, k and $maxGenerations$ are listed in table 6.3. Through tweaking parameters M and k, which influence the workings of the analysis component, we achieve different behaviour of the respective version of algorithms. Effects of different values of these parameters are discussed in individual sections for experiment A, B and C.

Thirty independent runs were executed for each instance. Presented quality measures in table 6.4 are:

- median - The median value of the cost of the best solution over 30 runs
- best - The overall best solution over 30 runs
- vehicles - The median value of the vehicles used for the best solution over 30 runs. This found value is the same for each data instance over all algorithm version, that is why it is shown in only single column. This constraint was for every instance and for every version of algorithm satisfied in the very first generation of computation. Except for instances egl-e4-C and egl-s4-C, where a satisfactory solution was not found at all during any run of any version of algorithm.

The last column of the results table 6.4 includes the integer solutions to each data instance taken from literature. The data was obtained from `https://logistik.bwl.uni-mainz.de/forschung/benchmarks/`.

Version	M	k	$maxGenerations$
Vanilla100	-	-	1000
Node100	100	100	1000
Edge20	100	20	300
Basic20	100	20	300
Node20	100	20	300
Basic200	200	200	1000
Node200	200	200	1000

Table 6.3: Algorithm configurations used in experiments

Further, the convergence of all the algorithm versions for every instance is captured in graphs, available in appendix of this thesis. Only a few notable ones are displayed separately in this chapter. Graphs capture the convergence of the median value of the best-so-far solution's cost (BSF_solution) and of the median value

Presented measures in graphs is the convergence of values:

- BSF_solution - median value of the cost of best-so-far solution over 30 runs, full lines
- BOP_solution - median value of the cost of best solution in current population over 30 lines, dashed lines

The full lines in graphs represent the median of the BSF_solution cost over the 30 different seed runs. Whereas the corresponding thin dashed lines represent the medians of best solution of population in given generation, the *BOP_solution*.

6.5.1 Experiment A

Results of the experiments are presented in table **??**.

By observing the results in the second and fourth column of the table, we conclude that the medians of the best solutions' cost found by the edge version are consistently superior to the medians of the best cost found by vanilla on every single instance. It is important to stress, that the edge version runs only for 300 generations, as opposed to the vanilla, which runs for 1000 generations.

The vanilla version was not able to converge to The optimal value was reached for two of the tested instances, egl-e1-A and egl-s1-A by the edge version. In the case of the node version, only one of these optima was reached. The overall perfomance of the node version is not as convincing as the edge version.

As soon as the M-th generation is reached, which implies the first time the journal is derived from analysis of elite individuals, the cost of the BSF_solution of the edge version starts to improve dramatically in contrast with the vanilla version.

6.5.2 Experiment B

6.5.3 Experiment C

Dataset		**Experiment A** **Vanilla100**		**Node100**		**Edge20**		**Optimal**
	vehicles	median	best	median	best	median	best	
egl-e1-A	5	3561	3548	3548	3548	3548	3548	3548
egl-e1-C	10	5760	5668	5687	5603	5680	5613	5595
egl-e4-A	9	6726	6578	6721	6617	6611	6507	6395
egl-e4-C	20	11956	11764	11966	11790	11853	11649	11529
egl-s1-A	7	5143	5035	5019	5018	5018	5018	5018
egl-s1-C	14	8690	8545	8605	8519	8593	8518	8518
egl-s4-A	19	13192	12951	13272	12982	12678	12561	12140
egl-s4-C	36	21720	21245	21909	21626	21210	21071	20380
		Experiment B						
Dataset		**Basic20**		**Node20**		**Edge20**		**Optimal**
	vehicles	median	best	median	best	median	best	
egl-e1-A	5	3610	3548	3561	3548	3548	3548	3548
egl-e1-C	10	5791	5703	5719	5639	5680	5613	5595
egl-e4-A	9	6793	6644	6808	6644	6611	6507	6395
egl-e4-C	20	12008	11841	12022	11841	11853	11649	11529
egl-s1-A	7	5183	5066	5074	5038	5018	5018	5018
egl-s1-C	14	8766	8634	8748	8562	8593	8518	8518
egl-s4-A	19	13354	13221	13336	13221	12678	12561	12140
egl-s4-C	36	22000	21646	22000	21646	21210	21071	20380
		Experiment C						
Dataset		**Basic200**		**Node200**		**Edge20**		**Optimal**
	vehicles	median	best	median	best	median	best	
egl-e1-A	5	3561	3548	3548	3548	3548	3548	3548
egl-e1-C	10	5728	5677	5687	5634	5680	5613	5595
egl-e4-A	9	6689	6598	6732	6594	6611	6507	6395
egl-e4-C	20	11941	11783	11960	11783	11853	11649	11529
egl-s1-A	7	5128	5054	5027	5018	5018	5018	5018
egl-s1-C	14	8660	8587	8652	8518	8593	8518	8518
egl-s4-A	19	13123	12924	13232	12989	12678	12561	12140
egl-s4-C	36	21712	21334	21752	21549	21210	21071	20380

Table 6.4: Experiments

Appendix A
Experiment graphs

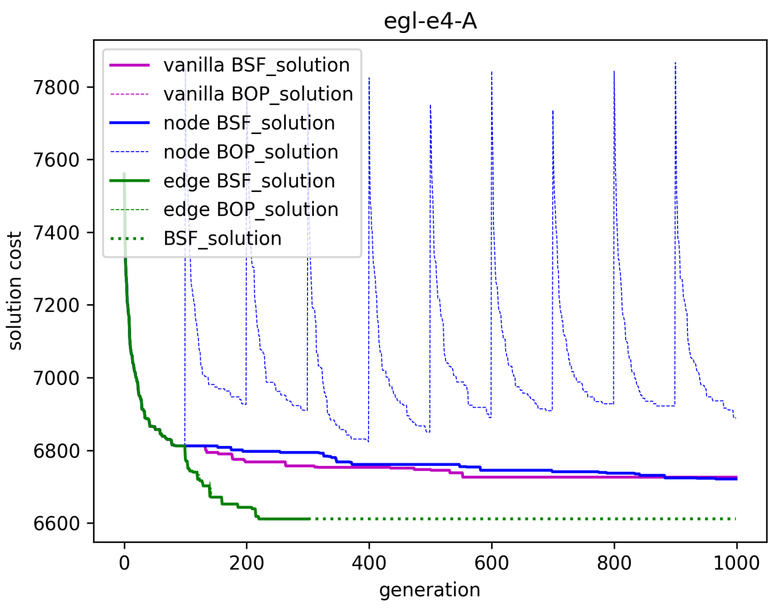

Figure A.1: Caption of the figure.

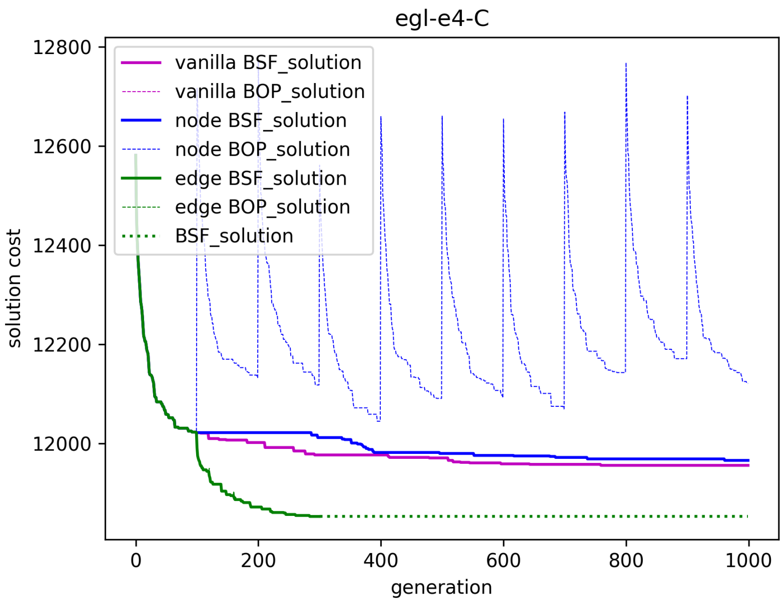

Figure A.2: Caption of the figure.

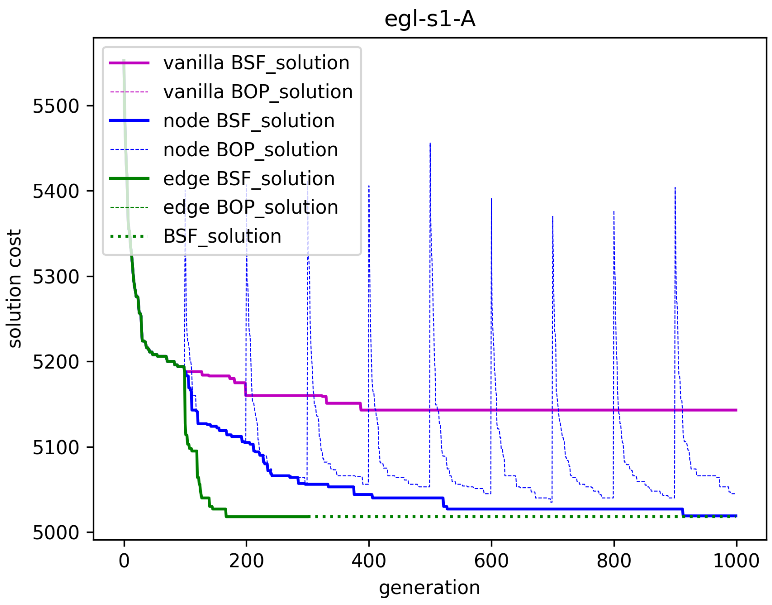

Figure A.3: Caption of the figure.

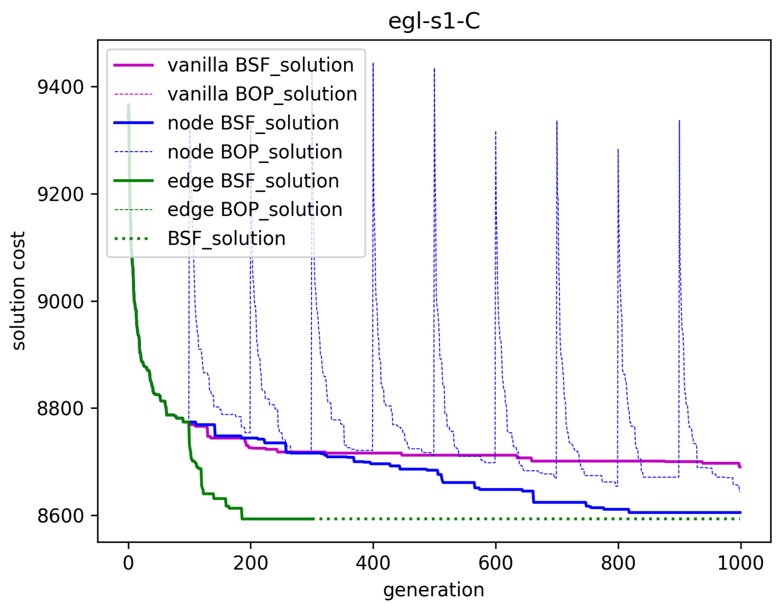

Figure A.4: Caption of the figure.

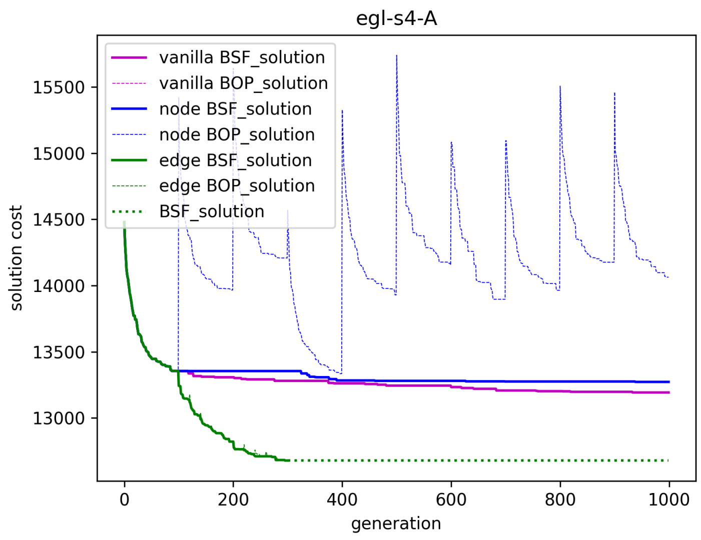

Figure A.5: Caption of the figure.

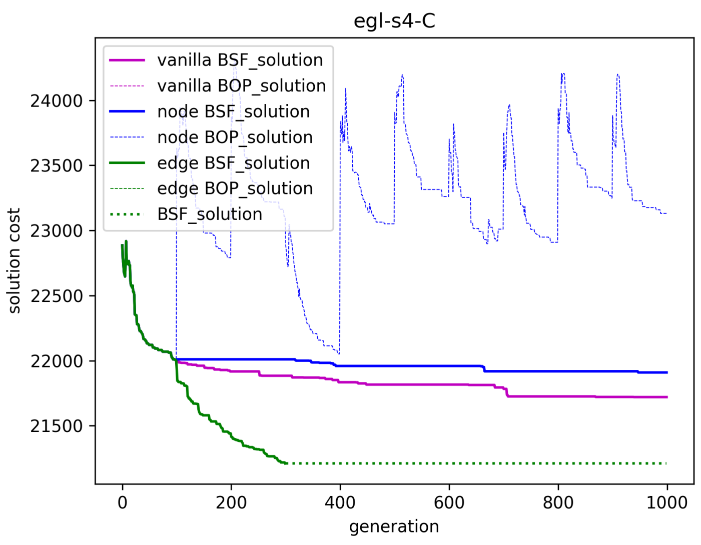

Figure A.6: Caption of the figure.

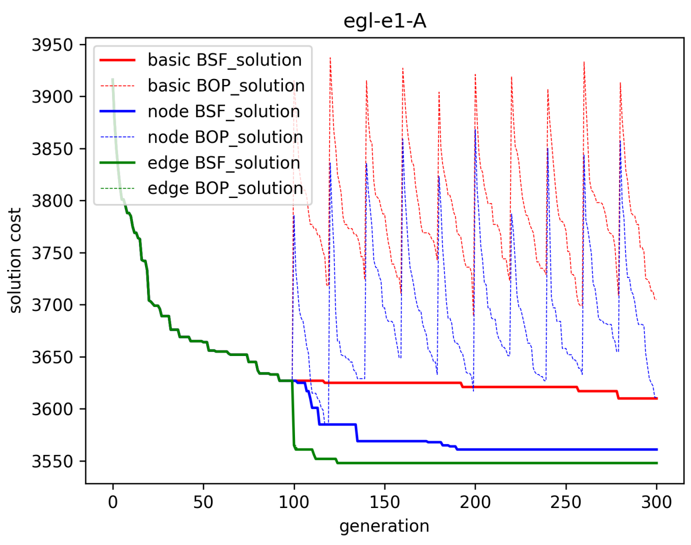

Figure A.7: Caption of the figure.

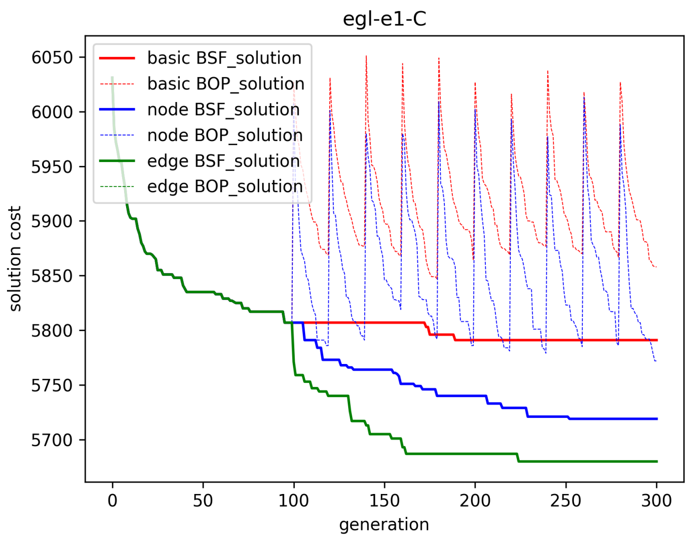

Figure A.8: Caption of the figure.

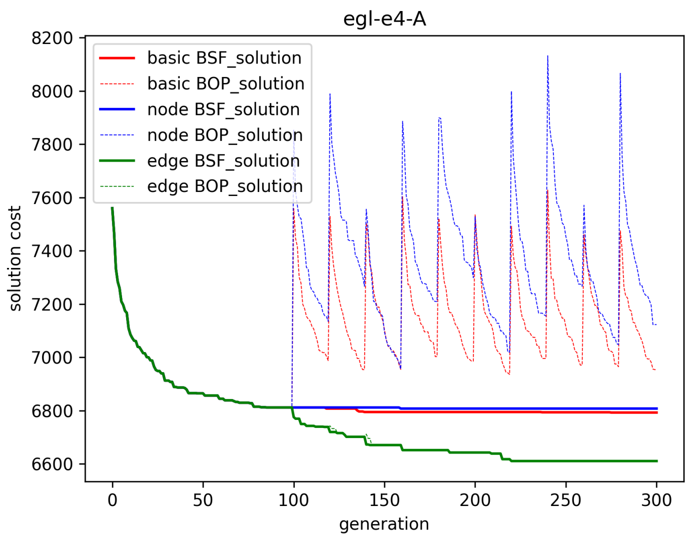

Figure A.9: Caption of the figure.

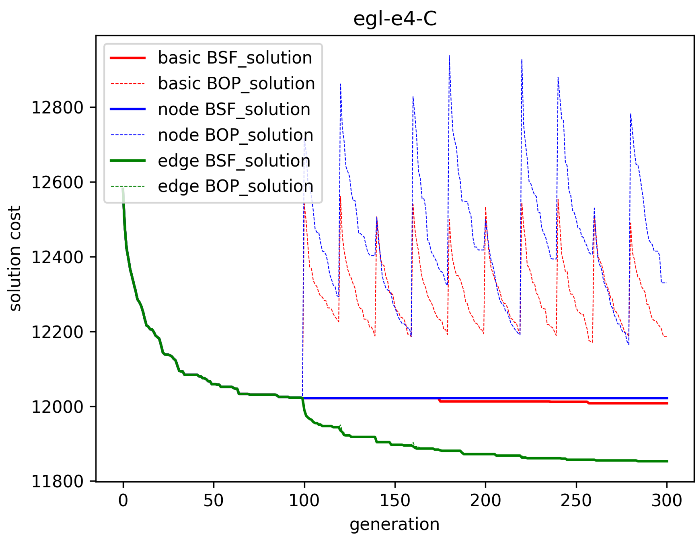

Figure A.10: Caption of the figure.

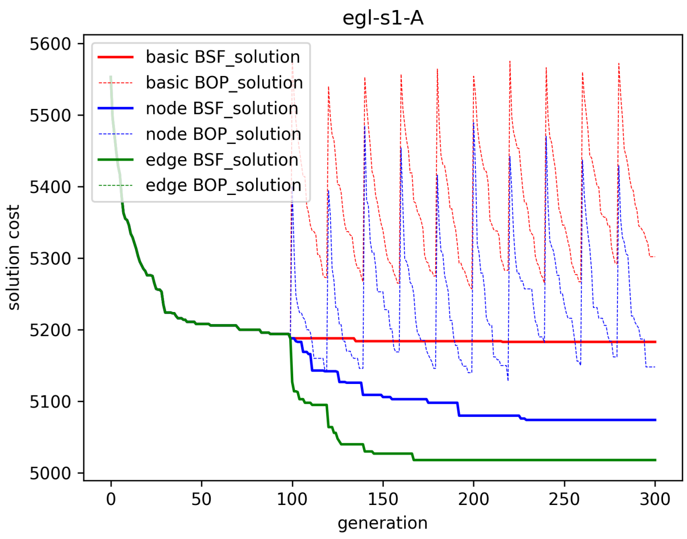

Figure A.11: Caption of the figure.

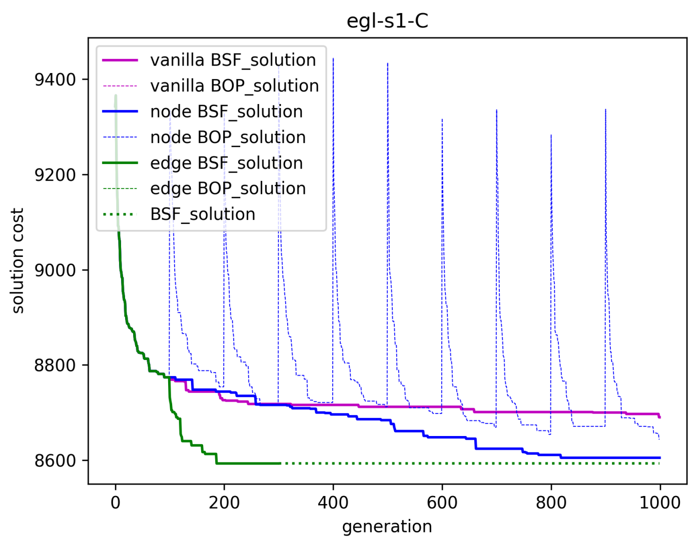

Figure A.12: Caption of the figure.

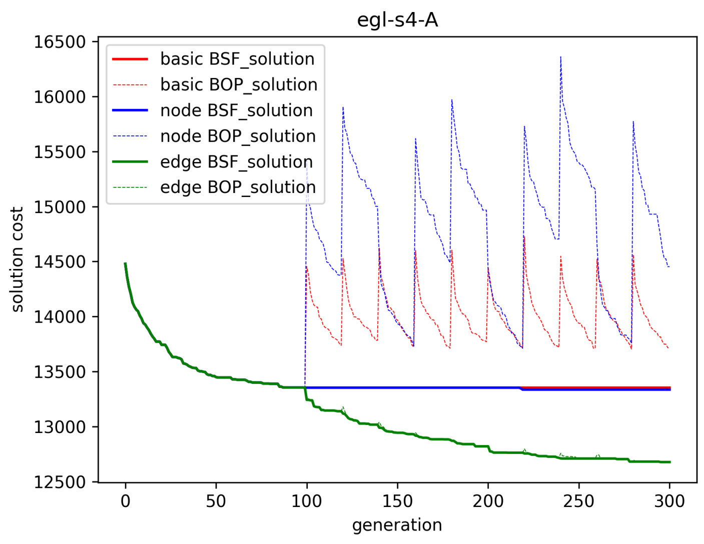

Figure A.13: Caption of the figure.

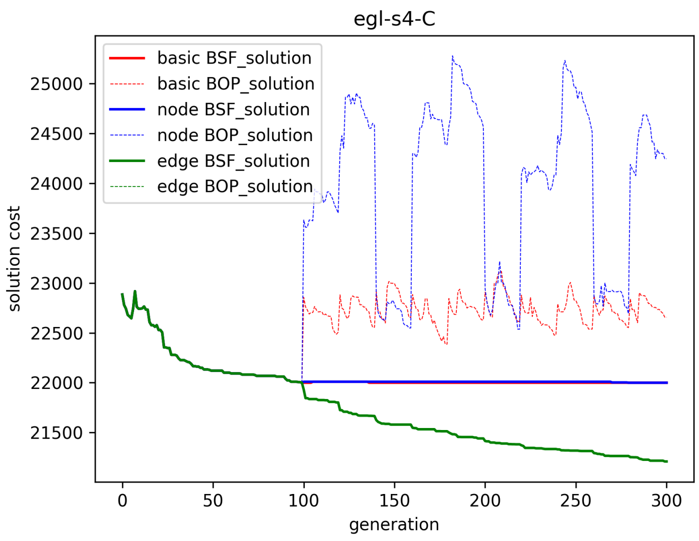

Figure A.14: Caption of the figure.

Appendix B

Bibliography

[AU18] Rafael Kendy Arakaki and Fábio Luiz Usberti, *Hybrid genetic algorithm for the open capacitated arc routing problem*, Computers & Operations Research **90** (2018), 221–231.

[BB03] José M. Belenguer and Enrique Benavent, *A cutting plane algorithm for the capacitated arc routing problem*, Computers & Operations Research **30** (2003), no. 5, 705–728.

[BCL13] Enrico Bartolini, Jean-François Cordeau, and Gilbert Laporte, *Improved lower bounds and exact algorithm for the capacitated arc routing problem*, Mathematical Programming **137** (2013), no. 1, 409–452.

[BI12] Claudia Bode and Stefan Irnich, *Cut-first branch-and-price-second for the capacitated arc-routing problem*, Operations research **60** (2012), no. 5, 1167–1182.

[Egl94] R.W. Eglese, *Routeing winter gritting vehicles*, Discrete Applied Mathematics **48** (1994), no. 3, 231–244.

[FLM15] Les Foulds, Humberto Longo, and Jean Martins, *A compact transformation of arc routing problems into node routing problems*, Annals of Operations Research **226** (2015), no. 1, 177–200.

[GW81] Bruce L Golden and Richard T Wong, *Capacitated arc routing problems*, Networks **11** (1981), no. 3, 305–315.

[HLM00] Alain Hertz, Gilbert Laporte, and Michel Mittaz, *A tabu search heuristic for the capacitated arc routing problem*, Operations research **48** (2000), no. 1, 129–135.

[HM01] Alain Hertz and Michel Mittaz, *A variable neighborhood descent algorithm for the undirected capacitated arc routing problem*, Transportation science **35** (2001), no. 4, 425–434.

[KS14] Jiří Kubalík and Michal Snížek, *A novel evolutionary algorithm with indirect representation and extended nearest neighbor constructive procedure for solving routing problems*, 2014 14th International Conference on Intelligent Systems Design and Applications, IEEE, 2014, pp. 156–161.

[LdU06] Humberto Longo, Marcus Poggi de Aragão, and Eduardo Uchoa, *Solving capacitated arc routing problems using a transformation to the cvrp*, Computers & Operations Research **33** (2006), no. 6, 1823–1837.

[LPRC01] Philippe Lacomme, Christian Prins, and Wahiba Ramdane-Chérif, *A genetic algorithm for the capacitated arc routing problem and its extensions*, Workshops on applications of evolutionary computation, Springer, 2001, pp. 473–483.

[LZJQ18] Qidong Lai, Zizhen Zhang, Xin Jin, and Hu Qin, *Forest-based tabu search to the split-delivery capacitated arc-routing problem*, 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE), IEEE, 2018, pp. 1237–1242.

[M$^+$89] Pablo Moscato et al., *On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms*, Caltech concurrent computation program, C3P Report **826** (1989), no. 1989, 37.

[M$^+$18] Rafael Marte et al., *Handbook of heuristics*, Springer,, 2018.

[MLRR11] Cristian Martinez, Irene Loiseau, Mauricio GC Resende, and S Rodriguez, *Brkga algorithm for the capacitated arc routing problem*, Electronic Notes in Theoretical Computer Science **281** (2011), 69–83.

[MLY13] Yi Mei, Xiaodong Li, and Xin Yao, *Decomposing large-scale capacitated arc routing problems using a random route grouping method*, 2013 IEEE Congress on Evolutionary Computation, IEEE, 2013, pp. 1013–1020.

[MLY14] ______, *Variable neighborhood decomposition for large scale capacitated arc routing problem*, 2014 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2014, pp. 1313–1320.

[PAG87] Wen-Lea Pearn, Arjang Assad, and Bruce L. Golden, *Transforming arc routing into node routing problems*, Computers & Operations Research **14** (1987), no. 4, 285–288.

[SH00] Thomas Stützle and Holger H Hoos, *Max–min ant system*, Future generation computer systems **16** (2000), no. 8, 889–914.

[Sys91] Gilbert Syswerda, *Schedule optimization using genetic algorithms*, Handbook of genetic algorithms (1991).

[TAH$^+$19] Erfan Babaee Tirkolaee, Mehdi Alinaghian, Ali Asghar Rahmani Hosseinabadi, Mani Bakhshi Sasi, and Arun Kumar Sangaiah, *An improved ant colony optimization for the multi-trip capacitated arc routing problem*, Computers & Electrical Engineering **77** (2019), 457–470.

[TMY09] Ke Tang, Yi Mei, and Xin Yao, *Memetic algorithm with extended neighborhood search for capacitated arc routing problems*, IEEE Transactions on Evolutionary Computation **13** (2009), no. 5, 1151–1166.

[WMZY21] Shaolin Wang, Yi Mei, Mengjie Zhang, and Xin Yao, *Genetic programming with niching for uncertain capacitated arc routing problem*, IEEE Transactions on Evolutionary Computation **26** (2021), no. 1, 73–87.