Heuristics for Arc Routing Problems and Their Applications

by

Muhilan Ramamoorthy

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved November 2022 by the
Graduate Supervisory Committee:

Violet R. Syrotiuk, Chair
Stephanie Forrest
Pitu Mirchandani
Arunabha Sen

ARIZONA STATE UNIVERSITY

December 2022

ABSTRACT

Arc Routing Problems (ARPs) are a type of routing problem that finds routes of minimum total cost covering the edges or arcs in a graph representing street or road networks. They find application in many essential services such as residential waste collection, winter gritting, and others. Being NP-hard, solutions are usually found using heuristic methods. This dissertation contributes to heuristics for ARP, with a focus on the Capacitated Arc Routing Problem (CARP) with additional constraints.

In operations such as residential waste collection, vehicle breakdown disruptions occur frequently. A new variant Capacitated Arc Re-routing Problem for Vehicle Break-down (CARP-VB) is introduced to address the need to re-route using only remaining vehicles to avoid missing services. A new heuristic PROBE is developed to solve CARP-VB. Experiments on benchmark instances show that PROBE is better in reducing the makespan and hence effective in reducing delays and avoiding missing services.

In addition to total cost, operators are also interested in solutions that are attractive, that is, routes that are contiguous, compact, and non-overlapping to manage the work. Operators may not adopt a solution that is not attractive even if it is optimum. They are also interested in solutions that are balanced in workload to meet equity requirements. A new multi-objective memetic algorithm, MA-ABC is developed, that optimizes three objectives: Attractiveness, makespan, and total cost. On testing with benchmark instances, MA-ABC was found to be effective in providing attractive and balanced route solutions without affecting the total cost.

Changes in the problem specification such as demand and topology occurs frequently in business operations. Machine learning be applied to learn the distribution behind these changes and generate solutions quickly at time of inference. SPLICE is a machine learning framework for CARP that generates closer to optimum solutions

quickly using a graph neural network and deep Q-learning. SPLICE can solve several variants of node and arc routing problems using the same architecture without any modification. SPLICE was trained and tested using randomly generated instances. SPLICE generated solutions faster that are also better in comparison to popular metaheuristics.

ACKNOWLEDGMENTS

I like to thank my advisor, for her excellent support and guidance throughout this PhD journey. She demonstrated what a true mentor-ship is and played a key role in making this into a success.

I thank the committee members for their guidance, valuable suggestions and support. I am grateful for giving their precious time and made themselves available to me in spite of their demanding work and busy schedules.

I thank the staff at ASU for their great support and help throughout this journey.

I am very grateful to ASU for giving me an opportunity to do PhD, as it is very unlikely for some one with a similar background to enter academia to pursue a PhD.

I like to thank my parents for their immense and continued support throughout this long journey. I thank all my friends and family friends in supporting me all along.

The number of people who made this journey possible, either directly or indirectly, are many. When I think about them, the part that I played for this completion is truly insignificant. I am deeply thankful and I enter natural silence on recognising that.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Chapter 1

INTRODUCTION

Transportation and logistics are fundamental to the smooth operation of the world economy [1]. It occupies a major proportion in the GDP (Gross Domestic Product) of every nation. In 2020, transportation services in the US contributed $1.2 trillion, which is around 15.45% of the US GDP [2]. The logistics costs are $1.56 trillion dollars in 2020, which is equal to around 8% of the US GDP [3] with similar figures in 2021 [4]. The share of the transportation services to the GDP is even higher in developing countries. For example, in China it is around 15% of GDP in 2020, nearly twice as high as in the US [3]. In addition to serving and strengthening the economy, transportation and logistics also have their influence on the development of human resources. Access to transportation and availability of the infrastructure helps support the livelihood of communities in rural areas.

The impact of transportation and logistics on environmental resources, conservation and sustainability is also significant [1]. According to the inventory of US Greenhouse Gas (GHG) Emissions and Sinks, transportation accounted for the largest portion (27%) of total US GHG emissions in 2020 [5]. Merely reducing the transportation cost in terms of miles travelled even by a small percentage can contribute a large to the bottom line of the businesses and to the world economy. Routing problems serve a important role in addressing this objective. They not only reduce costs but also help in satisfying customers (through timely and damage-free delivery) and hence increase revenue. They play a significant role in promoting sustainability and in climate change mitigation. Routing problems also play a important role in disaster relief efforts in emergency routing and in resource dispatch and distribution.

1.1 Routing Problems

A routing problem is defined as a problem of finding either a single or a set of routes to cover or service a given set of destinations, with a single or multiple objectives and with some constraints. Some of the typical objectives include minimizing the total cost and reducing the delay in services (or *makespan*). Examples of constraints include routes starting and ending at a fixed location and performing the service with only a limited number of homogeneous vehicles of fixed *capacity*. Routing problems, which are often represented as problems on graphs, are broadly classified into three categories: Node routing problems, arc routing problems (ARPs) and general routing problems.

Node routing problems have their destinations represented as nodes in the graph. The edges connecting the nodes represent the abstract cost such as distance between the locations. The well known Travelling Salesman Problem (TSP), Vehicle Routing Problem (VRP), and the Capacitated Vehicle Routing Problem (CVRP) belong to this category. In contrast, arc routing problems find routes covering the arcs or edges in a graph. Though both type of problems are combinatorial in nature the solution methods and approaches to solve them are often different.

ARPs are used in many essential real world applications. Some of them include postal delivery, winter gritting, solid waste collection, meter-reading, road inspection, mapping, snow removal, and many others [6]. They are also used in less well known applications such as in bridge structure inspection, and by printers, plotters and land cutting machines to determine least cutting paths [7], and others [6].

General routing problems have destinations that include both the nodes and edges. To accommodate mixed graphs and single directional arcs representing one-way streets and roads, Prins and Bouchenoua formulated Node Edge Arc Routing

Problems (NEARP) [8, 9], where the destinations include nodes, directed arcs and undirected edges with turn penalties defined over a mixed graph. Figure 1.1 gives a high level view of the structure of routing problems. This dissertation has a focus on heuristic solutions for arc routing problems and their applications. Hence a brief introduction of arc routing problems, their basic variants and solution approaches is given in Chapter 2.

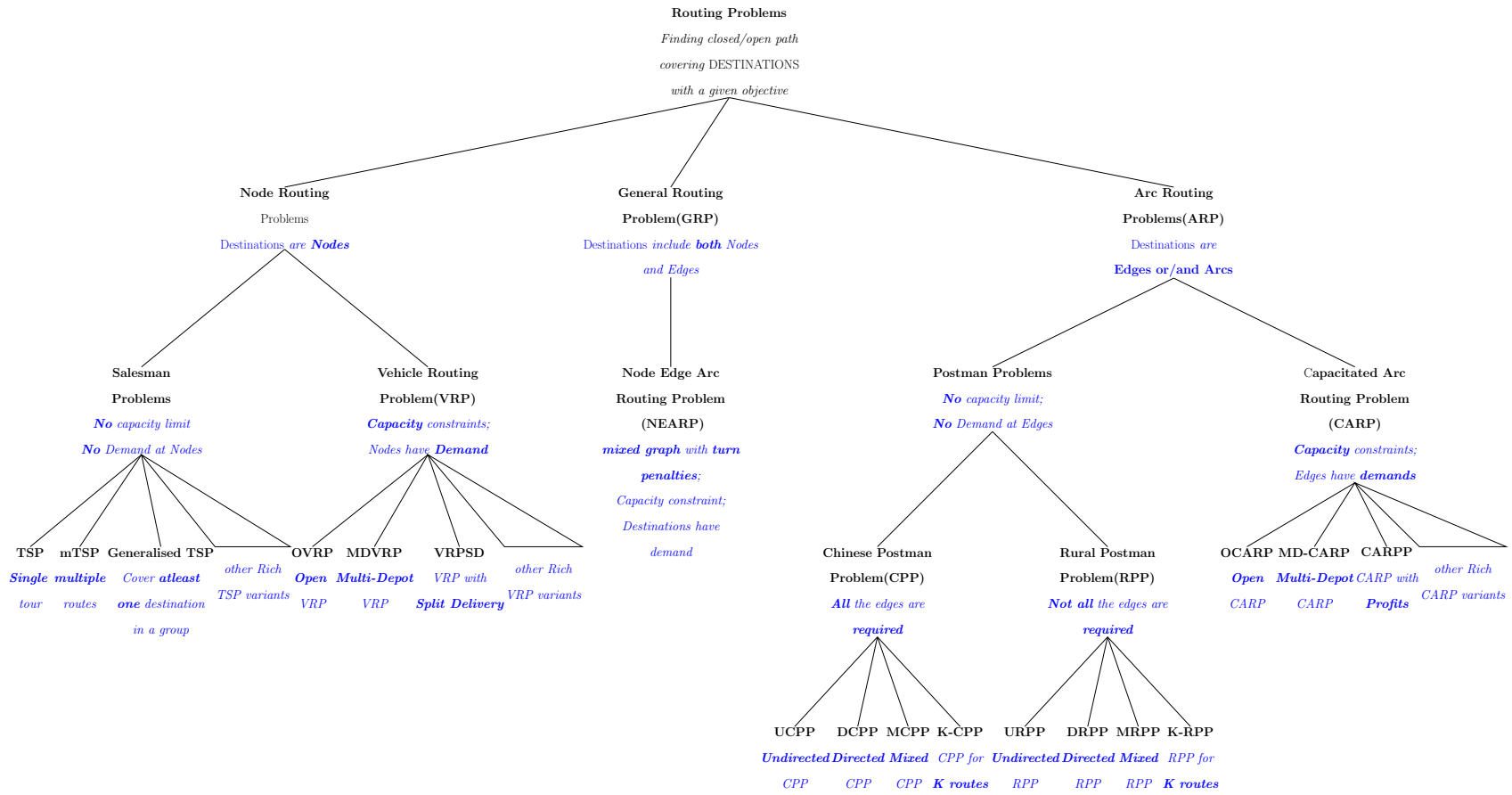


Figure 1.1: Routing Problems -Types and Variants

1.2 Motivation

ARPs are not as well studied as node routing problems, either in academia or in industry. As a result the number of commercial solvers in the market, choices for methods and solution approaches in literature are far fewer when compared to node routing problems. Hence the operators of businesses that are modelled on ARPs find limited options to meet their custom needs and specific requirements.

One example for a custom requirements is the handling of vehicle breakdowns. These happen frequently in residential solid waste collection operations in cities [10]. Often the local government of the city itself manages the operation and they operate with only limited number of vehicles as these vehicles incur high capital cost. When breakdowns happen, the city usually does not have any stand-by vehicles to take-over the operation immediately. They have to complete the assigned work of the broken-down vehicle using the other already assigned vehicles. They encounter the risk of not completing the scheduled operation and may need to ask workers to work overtime to avoid missing services. Problems related to vehicle breakdown in residential waste collection are prevalent across all regions. This motivated the first contribution in this dissertation: A new variant CARP-VB representing this problem and a heuristic algorithm, PROBE to solve it.

The minimum total cost is not the only criterion that operators are interested in. They also look for attractiveness in the route solution, in the sense that the routes in the solution should be compact, non-overlapping with a clear demarcation. It helps operators in work assignment and managing workers. They also give high importance to balanced routes as it is important for worker equity. While there are works exploring attractiveness in node routing problems, they are not explored well in arc routing problems except for a very few [11]. Also there is no work that optimizes

all three important objectives: Attractiveness, route balance and total cost. This motivated the second contribution in this dissertation: development of MA-ABC, a memetic multi-objective algorithm to solve this multi-objective problem.

The motivation behind the third contribution also stems from the same problem, the lack of adequate methods and approaches available for ARPs. The frequency of changes in the problem specifications in arc routing applications such as residential waste collection or postal delivery are high. The same solution is not optimum for the changed specifications. Learning heuristics using machine learning methods enables quick inference by learning the distribution of the solution space with the objective closer to optimum. While there are several works published for node routing problems, there are very few for arc routing problems.

All the methods developed in this dissertation do not require extensive computing infrastructure. They can easily be run on local machines with limited resources or run remotely using cloud facilities.

1.3 Contributions

In summary, the main contributions of this dissertation are :

1. The formulation of a new variant of CARP, 'Capacitated Arc Re-routing Problem for Vehicle Breakdowns' (CARP-VB) to handle vehicle breakdown that occurs in services such as residential waste collection operations. PROBE, an online heuristic algorithm, was developed to solve CARP-VB. Chapter 3 presents the details of PROBE.

2. The development of a new memetic multi-objective optimization algorithm MA-ABC that seeks to optimize three objectives: Attractiveness, route balance, and total cost. Chapter 4 presents the details of MA-ABC.

3. The development of SPLICE, a framework for *learning* the heuristics for CARP based on deep reinforcement learning and graph neural networks. SPLICE can be applied to different variants of arc routing problems and node routing problems with the same architecture. Chapter 5 presents the details of SPLICE.

This dissertation is organized as follows. Chapter 2 introduces preliminaries such as basic variants of ARPs, methods and approaches for solving arc routing problems. It then gives literature review related to our work. Chapter 3 presents PROBE, a heuristic algorithm for disruption management for services modelled on ARPs such as managing vehicle breakdown in residential waste collection operations. Chapter 4 reports MA-ABC, a memetic and multi objective optimization algorithm, that optimizes three objectives: Attractiveness, route-balance and total cost. Chapter 5 presents SPLICE, a machine learning framework for learning the heuristics for CARP and other routing problems using deep reinforcement learning and graph neural networks. Chapter 6 summarizes our work and describes future research directions.

Chapter 2

PRELIMINARIES AND LITERATURE REVIEW

In this chapter, we discuss the complexity of the arc routing problems in §2.1. We introduce the basic variants of arc routing problems and few of its extensions in §2.2. We give the definition of CARP in §2.3. We use CARP in all the works in this dissertation as it is a generic and widely applied variant. We discuss some alternative objectives used in the arc routing literature in §2.4. We use two alternative objectives in addition to '*total cost*' in our work on multi-objective optimization presented in chapter 4. We discuss the approaches and methods used for solving the arc routing problems in §2.5. Finally, we give the literature review related to our work in this dissertation in §2.6.

2.1 Complexity of the Arc Routing Problems

Except for the two variants that are polynomial and few other variants that are NP-complete (see §2.2.1), all other variants of arc routing problems are NP-hard. ARPs are known to be harder than the node-routing problems as many of these ARPs generalize other NP-hard problems [12]. We mention and discuss the complexities of the variants as we introduce them in section §2.2 and when we discuss the different approaches for solving the ARPs in §2.5.

2.2 Basic Variants of ARPs and Its Extensions

Arc routing problem can be traced back to the story of the origin of graph theory itself: Solving the Königsberg bridge puzzle by Euler [13, 14]. We introduce the basic

variants of the arc routing problem and few of its extensions in this chapter. With the recent advancement in computing power, new complex and rich variants have been continuously proposed in the arc routing literature based on the requirements of the applications in the real world; See [6] for more recent and complete updates on the variants of ARPs. In most cases, the many different rich variants of the arc routing problem can be found to have been extended from these basic variants by either adding or modifying the objectives and/or constraints. Figure 1.1 shows the organization of the basic variants of ARP.

2.2.1 Chinese Postman Problem (CPP)

Similar to the salesman problems for uncapacitated variants in node routing problems, the un-capacitated variants in arc routing problems are called *postman problems*, named after the application for which they were used. The Chinese Postman Problem (CPP) (thought to be named after Kwan Mei-Ko [15] who originally studied the problem), or the route inspection problem, is defined as the problem of finding a closed path with minimum total cost covering all the edges in the graph. CPP is the TSP counterpart for the arc routing problem. In relation to the Königsberg bridge problem, it can be interpreted as a problem of finding a subset of edges of minimum cost to be added to the original graph, in the case of graph with nodes of odd degree, to make it an Eulerian graph [14]

Variants of CPP exist based on the type of graph (undirected, directed and mixed graphs) and other factors. CPP is generally defined over undirected graphs and sometimes mentioned specifically as Undirected Chinese Postman Problem (UCPP) in the literature. When defined over directed graphs, it is called *Directed CPP* (DCPP); when defined over mixed graphs that have both edges and directed arcs representing the one-way streets, it is called *Mixed CPP*. If the edges of the graph are asymmetric

with different costs on either direction, it is called *Windy CPP*.

If the problem allows more than one route or is formulated to find a fixed number of routes, then it is called *K-CPP*. With additional constraints on the order for the edges to be visited, used for modelling applications such as snow plowing, the problem is called *Hierarchical CPP*. Variants of CPP have been proposed based on alternative objectives such as MinMax, that are discussed in section 2.4.

Undirected and *directed* CPP can be solved for optimum in polynomial time [15, 16]. If all the nodes in the graph are of even degree, then the solution is obtained by generating an Eulerian circuit using methods such as Hierholzer's algorithm [17]. If there are nodes in the graph with odd degrees, then some edges need to be duplicated or traversed more than once (called *dead heading*). Duplicating the edges for the least cost augmentation can be determined using minimum cost matching of the odd degree nodes, using methods such as Edmonds and Johnson's Blossom algorithm [16], which is polynomial in time complexity. Hierholzer's algorithm [17] can then be applied to the modified graph with duplicated edges to generate an Eulerian circuit.

Only *Undirected* CPP and *directed* CPP are polynomial [15, 16]. *Mixed CPP*, *Windy CPP* and *K-CPP* are NP-complete [18, 19]. All other variants of CPP are NP-hard.

2.2.2 Rural Postman Problem (RPP)

If not all the edges in the graph are *required* to be serviced, with some of the edges as *non-required* that can be used for *deadheading* to reach the nearest task, then the problem is called Rural Postman Problem (RPP). Lenstra and Kan [20] proved RPP to be NP-hard.

Similar to CPP, variants of RPP exist based on graphs such as Directed RPP(DRPP), Mixed RPP(MRPP), Windy RPP (WRPP); based on formulations such as, number

of routes as in *K-RPP* or on order of visit as in *Hierarchical RPP*, and others. Variants of Rural Postman Problem (RPP) based on alternative objectives are discussed in §2.4.

2.2.3 Capacitated Arc Routing Problem (CARP)

The Capacitated Arc Routing Problem (CARP), introduced by Golden and Wong in 1981 [21] is the arc routing counterpart of the vehicle routing problem in node routing problems. It is generally defined over an undirected graph. It is a problem of finding a set of routes with minimum total cost covering a subset of edges called *required* edges that have demands, constrained by vehicle capacity limit. A formal definition of CARP is given in §2.3. Golden and Wong [21] proved that CARP is NP-hard. It is harder than VRP, as it generalizes other NP-hard problems such as RPP. Golden and Wong [21] proved that it is also NP-hard to even find a 3/2 approximation. CARP can have all the edges as *required*, similar to CPP (called *Capacitated Chinese Postman Problem* (CCPP)), or to have only a subset of the edges as *required*, similar to RPP. The demand values for the non-required edges are zero. In fact, Christofides introduced a case where all edges are required, the CCPP in 1973 [22]. Like the VRP in node routing problems, CARP is a widely adopted model used in many real world applications such as postal delivery, winter gritting or salt spreading, residential waste collection and others.

Like the other two basic variants, CPP and RPP, CARP also has many extensions. For example, variants based on the type of graph such as Undirected CARP (UCARP), Directed CARP (DCARP), Mixed CARP (MCARP); variants based on the objective functions such as Min-Max CARP (MMCARP), Multi-objective CARP, Prize collecting CARP, CARP with Profits (CARPP); based on demands and method of handling the problem specifications such as Stochastic CARP, Dynamic CARP;

based on tours and facilities such as Open CARP (OCARP), Multi-Depot CARP (CARP-MD), CARP with Intermediate Facilities (CARP-IF); based on other criteria such as Periodic CARP, CARP with Split-Delivery (CARP-SD) and many others. Adding complex constraints and objectives to meet the real world business requirements leads to many rich variants of CARP. A comprehensive list of CARP variants can be found in [23] and specific chapters in [7] for other variants.

All the work in this dissertation use CARP or its extensions.

2.3 Definition of CARP

The *capacitated arc routing problem* (CARP) [21] is defined on a weighted undirected graph $G = (V, E)$. The streets in a city correspond to the edges E in G, and the vertices V to their intersections. There is a depot $D \in V$ used to store a fleet of k homogeneous vehicles, each with capacity C. *Tasks* correspond to a service required on a subset $T \subseteq E$ of the streets. Each task has a traversal cost $c(t)$ and a demand $d(t)$. Streets not requiring service have zero demand. All edges can be traversed any number of times. The goal of CARP is to find a set of closed routes, one for each vehicle, starting and ending at the depot, of minimum total cost such that: All tasks are serviced, the sum of the demands of serviced edges of each route does not exceed the vehicle capacity, and every serviced edge is in exactly one route. The cost of a route corresponds to the cost of its serviced edges and the cost of *deadheading*, i.e., the traversal cost of any intermediate connecting paths. CARP is NP-hard [21]. As a result, finding an optimum solution for most practical instance sizes is intractable. Therefore a number of heuristics and metaheuristics have been proposed [24].

2.4 Objectives Used in Arc Routing Problems

Minimizing the total cost is the most common objective used in many routing problems and in ARPs. There are also variants in ARPs that are formulated with objectives other than cost. Some of them include maximising profits, min-max route length and problems with multiple objectives (or multi-objectives).

In arc routing problems with profits, there is not a fixed set of customers to be served. Instead the objective is related to maximizing the profits associated with the edges or arcs in ARPs. Variants under this category include Arc Orienteering Problem (AOP), Maximum Benefit Chinese Postman Problem (MBCPP), Prize-Collecting Rural Postman Problem (PCRPP), Team Orienteering Arc Routing Problem (TOARP), Capacitated Arc Routing Problem with Profits (CARPP). Orienteering problems maximise profits with some constraints on the time duration or cost. Prize-Collecting routing problems minimize the route cost of routes that collect profits above a threshold.

The min-max objective minimizes the longest route length or cost in the route solution. It helps to achieve balanced routes. It also helps in achieving customer satisfaction as the route length is proportional to travel time. Some of variants related to min-max objectives include Min-Max K CPP (MMKCPP), Min-max K RPP (MMKRPP) and CARP with min-max objective (MM-CARP).

Visual attractiveness of the route [11] is also an important factor that the operators in real world businesses are concerned about. Route attractiveness in the solution refers to routes being compact, without sharp turns and without crossing each other or overlapping each other. It has been observed that operators even abandon or do not follow the solution of optimum cost if they look unattractive [11]. Different metrics have been suggested in the literature to measure the attractiveness. A comprehensive

review on the metrics used for visual attractiveness can be found in [11]. These metrics can be also used in the objective functions as an item to be optimized. We explore the metrics and objectives of route attractiveness in Chapter 4.

The objectives of attractiveness and total cost are usually conflicting, i.e., an increase or decrease of value in one objective affects the other in the opposite direction. The same relationship holds between min-max and total cost objectives [25]. Hence they are generally formulated and solved as a multi-objective optimization problem. Multi-objective optimization methods attempt to find *Pareto-optimal* solutions: A set of solutions in which none of its objective values can be improved further without affecting the other. In Chapter 4, we develop a multi-objective optimization algorithm that optimizes three objectives: Minimum cost, minimum makespan and maximizing route attractiveness.

2.5 Approaches for Solving Arc Routing Problems

In this section we briefly introduce different approaches and methods to solve ARPs, with a special focus on the CARP variant.

2.5.1 Exact Methods

Exact methods are algorithms that are guaranteed or proven to provide optimum solutions. Mathematical programming and enumeration methods are some common methods used to find the optimum solutions. In the arc routing problems three approaches are generally used: i) Branch-and-Bound based on combinatorial lower bounds; ii) Cutting-plane and Branch-and-Cut methods, iii) Column-Generation and Branch-and-Price methods. More details about the work on exact methods can be found in [26] for CARP and related chapters in the book [7] for other variants.

As mentioned in sections §2.1 and §2.2.1, except for the undirected and directed CPP that are polynomial and some CPP variants such as mixed CPP, windy CPP and MinmaxKCPP (MMKCPP) that are NP-complete, all other variants of ARPs are NP-hard [7], which means that finding an optimum solution for instances of size representing the problems of the real world are intractable to the best of our knowledge. Hence for practical purposes we need to depend on methods other than exact methods.

2.5.2 Approximation Algorithms

Approximation algorithms provide a proven worst case gap from the global optimum, by a factor or an approximation ratio, ϵ. Frederickson's heuristic for undirected RPP [27] is based on the Christofides's approximation algorithm for TSP [28] and has the same worst case ratio of 3/2 for problems that satisfy the triangle inequality. However there is no known 3/2 approximation algorithm for CARP. Golden and Wong [21] proved that it is NP-hard to even find a 3/2 approximation for CARP [29]. Wøhlk proposed A-ALG algorithm for CARP [30] based on the Jansen's *Shortest Optimal Tour Partitioning* (SOTP) algorithm for the General Capacitated Routing Problem (GCRP) [31] with the same approximation ratio of $(7/2 - 3/W)$, where W is the vehicle's capacity.

2.5.3 Heuristic Methods

Heuristic methods are constructive methods that find reasonably good solutions in a reasonable amount of time [32] but cannot guarantee the quality. They are generally faster than other methods. Heuristic methods are also used for generating initial solutions used in population based metaheuristic methods, that are discussed in §2.5.4. A comprehensive review of heuristics for CARP is found in [24] and for

other variants in the related chapters in [7].

Since CARP is comparatively harder to solve, with many of its benchmark instances not solved optimally yet, many heuristics and metaheuristic methods have been proposed to obtain new best known results. *Construct strike* [33], *Path Scanning (PS)* [29], *Augment-Merge* [29], and Ulusoy's *Route first-Cluster second* heuristics [34] are some of the classical heuristics that are still widely used. They were basically developed from the corresponding heuristics in the node routing problems. *Path Scanning (PS)*, *Augment-Merge* and Ulusoys's *Route first-Cluster second* algorithm are still used to design new or improved versions of heuristics and as components in metaheuristics [24]. Many of the recent heuristics of CARP are an improvement of these classical heuristics. Wøhlk [35] in her PhD thesis proposed four constructive heuristics, *Modified Path Scanning* (MPS), *Double Outer Scan*, *Node Duplication Heuristic*, and *A-ALG*. We use a variant of Path Scanning to compare our results and a memetic algorithm, which is an extension of *Route first-Cluster Second* heuristic in our work MA-ABC and Splice in Chapters 4 and 5. Hence we give more details about the two heuristic algorithms here.

The *Path Scanning (PS)* heuristic [29], originally proposed in 1983, is a widely used algorithm. It has the worst case time complexity of $O(t^2)$, where t is the number of *tasks* or required edges. It is one of the fastest algorithms that also gives consistently good results over wide range of instances. It builds routes sequentially, starting from the *depot*, by selecting the nearest task up to the vehicle's capacity limit and returns to depot when the vehicle is full. If there is more than one task at equal distance, it follows five rules to break the tie. The algorithm is run five times based on each rule and the best among the five is selected. Improved versions of Path Scanning include random methods for selecting the next task such as *PS with Random Criterion (PSRC)* [33, 36] and *PS with Random Task (PSRT)* [36]. PSRC breaks the tie by

choosing a random rule among the five rules. PSRT does not follow any rule but selects any of the tasks at random. *Path Scanning with ellipse Rule* [37] follows an additional rule that if the vehicle is near its capacity limit then it allows for selecting the next service edges only that are closer to the shortest path to the depot by a fixed factor. Random versions of the Path Scanning algorithms help to obtain different solutions that can be used as initial solutions for the population based metaheuristic methods discussed in next sub-section §2.5.4.

Ulusoy's *Route first-Cluster second* algorithm [34] has been widely used to design improved algorithms and as a component in metaheuristic algorithms. As the words *Route-first* in its name indicates, it first builds a single tour (called a *giant* tour) without any capacity constraints. It then generates the individual route solution by splitting the tour (*Cluster second*) by constructing an auxiliary graph. Optimum and fast tour splitting procedures with the worst case time complexity of $O(t)$ are available [38], where t is the number of tasks. These fast splitting procedures have enabled this heuristic method to be used as components in many advanced algorithms. Improved algorithms by Prins et al. [39] based on the Ulusoy's heuristic use three different methods of random path scanning for constructing the giant tour and four different methods of splitting, giving rise to twelve randomized heuristics. We use PSRT to compare our work in Chapters 4 and 5.

2.5.4 Local Search and Metaheuristic Methods

Local search methods are *improvement* procedures that find a solution by improving the solutions originally obtained from heuristics, by searching for a better solution in the solution space, among solutions that are closer to the current solution, called a *neighbourhood*.

Neighbourhood solutions are obtained using `move` operators that make a small

change to the current solution. Even though local search methods give better solutions than heuristic methods, they often get stuck at a local optimum. They too cannot guarantee a global optimum.

Metaheuristics are problem independent methods that help to a escape local minimum and hence able to provide better solutions. They are usually designed to continue their search until a *stopping condition* defined by the user is met. Many of the metaheuristics are stochastic in nature. One disadvantage with metaheuristics is that they may have parameters that need to be tuned to achieve good results.

Metaheuristics can be either *single solution* or *population* based. Single solution based metaheuristics work by improving upon a single solution until the stopping criteria are reached. Population based algorithms such as evolutionary and bio-inspired algorithms work on a set of solutions. The literature of CARP has implementations of most of the popular metaheuristics. On analysing the works of metaheuristics in the arc routing literature, Prins [24] observes a trend of moving from single solution based metaheuristics such as simulated annealing and tabu search to population based methods. He also observes a increase in number of lighter algorithms that give a good trade-off between quality and running time. Many metaheuristics have reported a good amount of success in finding best known solutions at the time of their publications.

Single Solution Based Metaheuristics

Some of the popular single solution based metaheuristic algorithms that had reported good performance in the literature are Tabu Search (TS), Guided Local Search (GLS), *Greedy Randomized Adaptive Search Procedures* (GRASP) and Variable Neighborhood Descent (VND).

Tabu search is a popular single solution based metaheuristic that has reported

good results in the arc routing literature too. The CARPET [40] and TSA [41] algorithms are both based on tabu search and have reported good results on CARP benchmark instances. TSA is fully deterministic and hence can be easily reproduced and verified.

Guided local search (GLS) by Beullens et al. [42] is a local search based metaheuristic that modifies the cost value of the objective function using penalties to escape from the local minima. GLS is one of the fastest metaheuristics along with TSA for the Undirected Capacitated Arc Routing Problem (UCARP). *GRASP with Evolutionary Path Relinking for CARP* [43] is a metaheuristic algorithm based on *Greedy Randomized Adaptive Search Procedures* (GRASP). Even though it takes longer than other metaheuristics, it gives better solutions. For example, it retrieves all the optimal solutions for the `gdb` benchmark instances [29]. Variable Neighborhood Descent (VND) by Hertz and Mittaz [44] is a type of *Variable Neighborhood Search* (VNS) metaheuristic that explores multiple neighborhoods until it finds no better solution in all the neighborhoods.

Population Based Metaheuristics

Evolutionary algorithms such as genetic algorithms and ant colony based algorithms [45], and scatter search [46] algorithms are all population based metaheuristics that use a pool of solutions, the *population*, in searching for better solutions.

Memetic algorithms are population based algorithms that combine the genetic algorithm and local search. Lacomme et al.'s Memetic Algorithm (MA) for CARP [47] [48] is one of the popular population based algorithms and is also widely adopted in other work. MA [48] at the time of its publication improved 26 of the best known solutions for the CARP benchmark instances. One innovative aspect of MA is that it uses giant tours as chromosomes. Hence it need not concern with the inconsistencies

over the capacity limits resulting from the cross-over and mutation operations when actual routes are used as chromosomes. It uses total cost as the fitting function, obtained after splitting the giant tour into route solutions. It uses order crossover (OX) for the cross-over operation [49]. It uses local search in the place of the mutation operation.

More recently, an improved Ant Colony Optimization (ACO) based metaheuristic by Santos et al. [45] has found 6 new best solutions in the larger `egl` benchmark instances [50].

Our work on memetic multi-objective algorithms, MA-ABC presented in Chapter 4, is based on Lacomme et al.'s Memetic Algorithm (MA) for CARP, SPLICE presented in Chapter 5 uses the splitting procedure used by Lacomme et al.

Hybrid Algorithms

Hybrid algorithms are methods that combine different heuristic or metaheuristic algorithms within their implementation. A specific heuristic algorithm may not work well in every type of instance or problem distribution. Since hybrid algorithms combine the beneficial approaches of more than one algorithm, they can perform better in different distribution. The memetic algorithm discussed in §2.5.4 can also be considered a hybrid algorithm since it combines local search with an evolutionary algorithm. Recently Chen et al. [51] published Hybrid Metaheuristic Approach (HMA) for CARP. They have combined effective local refinement and randomized tabu thresholding procedure with an infeasible descent in a memetic framework. They obtained all the best known results of the CARP benchmark instances and have improved results of 15 benchmark instances.

2.6 Related Work

This section provides a literature review related to the works, PROBE, MA-ABC, and SPLICE presented in the forthcoming chapters.

2.6.1 Vehicle Breakdowns and Disruptions Management in Routing Problems

There are few works related to managing the disruptions in routing and scheduling problems in the VRP literature. Mu et al. [52] developed two tabu search algorithms to solve VRP under vehicle breakdown. Mu and Eglese study the disruption due to order release delay for VRP in [53]. Li et al. worked on vehicle re-scheduling problems due to vehicle breakdowns in [54, 55], where new routing plans were developed to reduce the impact of delay in supply reaching the depot. Li et al. study a real time vehicle re-routing with time windows and develop a Lagrangian relaxation based heuristic in [56].

Work on on re-routing and re-scheduling for arc routing problems are very few. Monroy-Licht et al. consider re-scheduling due to vehicle failures in an uncapacitated setting in [57]. To the best of our knowledge PROBE is the first to work on disruption management due to vehicle breakdown in a capacitated arc routing setting.

2.6.2 Visual Attractiveness in Route Solutions

Visual attractiveness in routing problems first came to light when Poot et al. [58] reported that some operators considered the solution generated by the ORTEC [1] vehicle routing software to be poor, even though they were good on the traditional metrics such as total cost, number of vehicles used, and others. They later found the reason to be that the routes were considered visually unattractive. Visually attractive

[1] http://www.ortec.com/

route plans seem to be closer to the traditional way of working, thus generating trust in the plan among the drivers and planners [11, 58].

The objective of minimizing total cost and improving the visual attractiveness are often conflicting [11]. However, it is worth improving the visual attractiveness even when this comes at the expense of other objectives [59] as it enables the ultimate adoption of the route plans by the operators. Visual attractiveness is subjective conveying how well the routes exhibit a set of features such as i) *Compactness* ii) *non-overlapping* or *non-crossing* and iii) *non- complexity* i.e., routes without sharp edges and jagged transitions. Many metrics have been proposed to measure these features. Rossit et al. [11] proposes six measures for *compactness*, three measures for *proximity* (closely related to *compactness*), two measures for *non-overlapping* and *non-crossing* and three measures for *route complexity*.

Most of the methods for enhancing the visual attractiveness are based on heuristic approaches. A comprehensive list of methods and for the list publications on visual attractiveness in routing is found in [11].

2.6.3 Route Balance

Route balance is one of the alternative objectives discussed in §2.4. Its is also the second most important objective after the *total cost* that has been studied extensively [60] and used widely in the real world. Route balance addresses the equity concerns that are different from monetary benefits such as fair workload allocation and resource utilization, customer satisfaction, among others [61, 62]. There are many variants and methods in the node routing literature that use route balance either as a single objective or in combination with total cost, such as *Vehicle routing problem with Route Balance* (VRPRB) [63, 64]. Many metrics, such as *makespan* and *range*, are used either to measure the route balance or use them in an objective function. Lozano et

al. does a statistical analysis of seven objective functions for route balance in VRPRB in [65]. The objectives of route balance and the total cost conflict [25] in nature. Hence a bi-objective formulation such as VRPRB and multi-objective optimization methods [66] are used.

2.6.4 Multi-Objective Evolutionary Algorithms (MOEA)

Multi-objective Evolutionary Algorithms (MOEA) are one of the most widely used and popular approaches for multi-objective optimization. The reason for its wide adoption and comparatively better performance than others can be attributed to its inherent parallel computing nature since it is based on a population based approach. Coello gives more details with a historical view of MOEA in [67]. Among the different methods in MOEA, Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [68] is an efficient and widely studied method. It uses non-dominated sorting in its selection operations and in obtaining the Pareto front. NSGA-II differs from the first generation NSGA by incorporating *elitism* and in using *crowding distance* to improve diversity in the ranking. We use NSGA-II method in our MA-ABC algorithm presented in Chapter 4, where we discuss the evolutionary algorithms in the arc routing literature.

2.6.5 Machine Learning Approaches for Combinatorial Optimization and Routing Problems

As early as 1985, Hopfield networks [69], elastic nets [70], and self organizing maps [71], were applied to problems in combinatorial optimization; see Smith [72] for an overview of these early AI-driven approaches. At the time, the solutions produced were not close enough to the optimum nor practical enough to be applied to real ap-

plications. The approaches were not data driven, and instead attempted to construct the solution for each instance.

More recently there has been renewed interest in exploring AI-driven approaches for combinatorial optimization problems that underlie many real world applications [73]. In 2014, *recurrent neural network* (RNN) based sequence-to-sequence networks [74], and the attention mechanism [75], were successful in machine translation and other applications. In 2015, Vinyals et al. [76] then tried a variation of *long short-term memory* (LSTM) sequence-to-sequence networks, called *pointer networks*, for discrete combinatorial optimization problems. Since then, pointer networks have become the basis of much follow-on work in the field.

Vinyals et al. [76] viewed a combinatorial optimization problem as a permutation problem on sequence data. In their approach, the output *points* to the position of an element in the input sequence. Along with the two-dimensional euclidean *travelling salesman problem* (2D-TSP), they also applied the approach to solve two other discrete optimization problems: Convex Hull and Delaunay Triangulation. For each they used the same supervised learning architecture and hyper-parameter settings. One million data sets were used for training generated randomly based on a uniform distribution. Solutions were obtained for these data sets using commercial or open source solvers, which may not be optimal. Training was conducted on data sets containing up to 20 nodes and then tested with instances having more than 20 nodes to measure the generalization capability. The model generalized well for problems up to 40 nodes but not beyond.

Bello et al. [77] applied the pointer networks of Vinyals et al. [76] but used reinforcement learning for training. More specifically, the actor-critic policy gradient method [78] was used to train the network. With this method, closer to optimum results for up to 100 nodes for 2D-TSP were obtained. The model was also used to

solve the knapsack problem obtaining optimal results for up to 200 nodes.

Based on their previous work `structure2vec` [79], Dai et al. [80] used a graph embedding neural network for solving combinatorial optimization problems. They used Q-learning [81] for training because it is more sample efficient than the policy gradient methods [78]. They tested their model for the problems of maximum vertex cover (MVC), maximum cut (MAXCUT), and 2D-TSP. Their data sets included up to 300 nodes for training and, on testing, achieved cost within 0.07% of the optimum. To check the generalization and scalability of the model, they trained their model for up to 400 nodes and found it generalized for up to 1200 nodes.

Nazari et al. [82] extended the pointer networks of Vinyals et al. [76] to the *vehicle routing problem* (VRP). They handled the demand of nodes as *dynamic data*, that is, data that is updated as the network generates a solution. Their model can be used for solving several variants of VRP as well as TSP. In contrast to pointer networks, Nazari et al. [82] used an embedding layer instead of a recurrent neural network (RNN) for the encoder but used the same LSTM based network with the attention mechanism for the decoder. They trained their network using the actor-critic policy gradient method [78] and the asynchronous advantage actor-critic (A3C) method [83] for stochastic VRP. A smaller optimality gap was achieved compared to the Clarke-Wright (CW) savings heuristics [84], sweep heuristics [85], and Google's optimization tools (OR-Tools) [86]. Even though VRP can be considered an extension of TSP and multiple TSP (mTSP), it is generally more difficult to solve than TSP. Obtaining good results within limited time is challenging even for instances with node sizes of 100.

In 2018, Kool et al. [87] used multi-head attention networks [88] to solve TSP, CVRP, orienteering, and prize-collecting TSP problems. For TSP, they achieved results with the lowest percentage of gap from the optimum. Concurrently, Deudon

et al. [89] used the same multi-head approach for solving TSP, but applied local search to improve the results.

In 2019, Joshi et.al. [90] used a spectral based *graph convolution network* (GCN) [91] for solving TSP. They used supervised methods for learning and additionally used beam search during testing. While their results show a smaller optimality gap compared to the results from auto-regressive models (those based on RNN, LSTM, and transformer networks), their model does not generalize as well as the auto-regressive models.

All of the methods reviewed generate only a single solution to a problem as constructive type of heuristics. Other works approaches modeling improvement heuristics; see [92, 93, 94, 95, 96] for details.

All this work involves only node routing problems, such as VRP or TSP, which are generally represented by a euclidean graph. However ARPs are non-euclidean with their problems parameters defined on edges of the graph rather than on its vertices. We discuss these challenges in Chapter 5

We present our work from the next chapter, starting from PROBE, a heuristic algorithm for vehicle breakdown management in operations such as residential waste collection.

Chapter 3

ONLINE RE-ROUTING FOR VEHICLE BREAKDOWNS

This chapter presents our work on disruption due to vehicle breakdowns encountered frequently in operations such as residential waste collection operation. We propose a new CARP variant, Capacitated Arc Re-routing Problem for Vehicle Breakdown (CARP-VB) and develop a heuristic algorithm PROBE for solving the CARP-VB.

This chapter is organized as follows. §3.1 gives a brief introduction to residential waste collection operation, vehicle breakdown disruptions prevalent in such operations and our contributions for this problem. §3.2 presents a literature review and related works. We introduce and formalize the CARP re-routing problem in §3.3, and our PROBE algorithm to solve it in §3.4. Experimental set-up and results are discussed in §3.5. §3.6 summarizes the work.

3.1 Vehicle Breakdowns in Residential Waste Collection

Residential waste collection is often managed by a city's public works department. Typically, the city is partitioned into areas and each area is serviced on a week day. Pick-ups are done on a weekly basis using special collection vehicles that offload the waste at transfer stations.

The residential collection operation for large metropolitan areas can use as many as a thousand collection vehicles. The cost of collection, transportation, and disposal constitutes 50-70% of the total cost of solid waste management [97, 98, 99]. Reducing the collection route distance can save cost for a city, and also reduce its carbon

footprint. Routes for collection vehicles are often computed offline using optimization software or by commercial applications such as *RouteSmart* [100].

We model the problem of finding the routes for residential waste collection as a Capacitated Arc Routing Problem (CARP). However, the formulation of CARP does not consider the unexpected breakdown of vehicles. These occur with surprisingly high frequency, as often as every day [10]. Due to their cost, the public works departments in cities generally operate with every collection vehicle assigned to a route, performing maintenance and repairs on the vehicles overnight [10]. A common approach to handle a vehicle breakdown is to assign the unserved streets of broken down vehicle to *one* other vehicle after it has completed its own assigned route. This may extend the collection operation late into the day.

As we will see, our re-routing solution takes into account the unserved streets of all routes and equally shares the demands of the costs among all the operational vehicles. The idea is to balance the remaining demand among the operational vehicles in order to minimize the *makespan*, the maximum route length. Re-routing may also be applied to situations such as when a driver is not able to report for work on a given day, or when there is a large difference in the demand across routes causing one collection vehicle to finish much earlier than usual. Our approach provides a chance to complete the residential collection within the shift, reducing the over-time costs for a city.

The contributions of this work are:

1. We formulate the CARP re-routing problem and propose an online algorithm to solve it. Our PROBE algorithm computes new routes for the operational vehicles on the unserved demand when a breakdown occurs. PROBE seeks to minimize the makespan in order to produce more balanced routes.

2. We evaluate PROBE under different breakdown scenarios on the classical CARP benchmark instances. PROBE produces solutions with reduced makespan, range, and deviation, which indicate that the balance of the routes is improved.

3.2 Related Work

The *Capacitated Arc Routing Problem* (CARP) [21] is the arc routing counterpart of the *Vehicle Routing Problem* (VRP), servicing demand associated with edges instead of with nodes. This allows road applications such as winter gritting, salt spreading, and waste collection to be accurately modelled. WE already defined CARP in chapter 2 section2.3.

Path scanning [29] is a classical heuristic algorithm for CARP. We already discussed the Path scanning algorithm when introducing the heuristic methods in chapter1, section 2.5.3. It is constructive, building routes one by one [7]. Starting from the depot, it adds the next nearest task to the route, until it can no longer add any more tasks because it has reached the vehicle capacity limit. When this occurs, the route is closed by returning to the depot. Subsequent routes are built in the same way until there are no tasks remaining. Because there may be more than one choice of the next task to select, tie breaking rules were proposed [29]. There are many variants of path scanning [35, 101], some of which use randomization instead of one of the tie breaking heuristics to choose the best among several solutions [33, 36]. *Path Scanning with Random Task* (PSRT) algorithm [36] picks a task at random instead of following the tie breaking rules when constructing the solution.We have already discussed the Path scanning in details in §2.5.3. We have also discussed other heuristic methods, local search and metaheuristic methods for CARP in §2.5.

The problem of finding the routes for residential waste collection is usually modelled as a CARP. Incorporating additional constraints to address specific requirements

leads to different variants of the problem, such as allowing multiple offloading sites within the vehicle route [102], and incorporating cost restrictions [103].

We have already discussed vehicle breakdowns and disruptions management in routing problems in §2.6.1. There are very few works, if any, on re-routing and re-scheduling in arc routing literature. Monroy-Licht et al. consider re-scheduling due to vehicle failures in an uncapacitated setting such as those used for snow plowing operations [57]. To the best of our knowledge our work is the first that explores vehicle breakdown management in a capacitated arc routing setting.

3.3 The CARP Re-Routing Problem

We model residential waste collection in a city as a CARP. The streets in the city correspond to edges E in the graph G, and the vertices V to their intersections. The depot $D \in V$. Tasks correspond to collecting residential waste on a subset of the city streets $T \subseteq E$, with their demand corresponding to the volume of waste. Streets not requiring service have zero demand. The cost of each edge corresponds to the time to service the street. The city has a fleet of k collection vehicles, each able to collect up to its capacity C in waste.

Suppose that the city has a solution to CARP, providing a set of routes $R = \{r_1, r_2, \ldots, r_k\}$ for each of its collection vehicles for a given day. Each route r_i is a sequence of streets $e_{i1} = (v_{i1}, v_{i2}), e_{i2} = (v_{i2}, v_{i3}), \ldots, e_{ij-1} = (v_{ij-1}, v_{ij})$, that make a closed walk starting and ending at the depot, *i.e.*, $v_{i1} = v_{ij} = D$. A route may include traversing streets that need not be serviced in order to reach those that must be serviced. Such *dead-heading* incurs cost, but services no demand. Deadheading is always assumed to follow the shortest path by distance. The route satisfies the capacity constraints of the collection vehicles, *i.e.*, $\sum_{e \in r_i} c(e) \leq C$ for $1 \leq i \leq k$.

Consider a vehicle breaking down during a shift in waste collection. This requires

us to solve a CARP with $k-1$ collection vehicles on the unserved city streets, starting from their locations at the time of the breakdown, ending at the depot. Using the position of each collection vehicle, readily available from a telematics system, we can compute the streets still requiring service. That is, we partition each route r_i into two concatenated subsequences of edges, $r_i = e_{is}e_{iu}$, where the edges $e_{is} = e_{i1}, \ldots, e_{i\ell}$ have all been serviced hence their demands can be set to zero, and the edges $e_{iu} = e_{i\ell+1}, \ldots, e_{ij-1}$ consisting of streets that remain to be serviced. Either e_{is} or e_{iu} may be empty. The remaining streets to be serviced are therefore $T \setminus e_{is}$ for $1 \leq i \leq k$. The starting vertex for each vehicle i is the intersection associated with the last street it served, *i.e.*, $v_{i\ell}$ because $e_{i\ell} = (v_{i\ell-1}, v_{i\ell})$.

The capacity of each vehicle must take into account that some waste may have been collected. Hence each vehicle i has a capacity $C_i = C - \sum_{e \in e_{is}} d(e)$, $1 \leq i \leq k-1$.

Capacitated Arc Re-routing Problem for Vehicle Breakdown (CARP-VB): Find a set $R = \{r_1, \ldots, r_{k-1}\}$ of closed routes for each operational vehicle, each starting at vertex $v_{i\ell}$, $1 \leq i \leq k-1$, and ending at the depot D, of minimum total cost such that: All remaining tasks are serviced, the sum of the demands of serviced edges of each route must not exceed the remaining vehicle capacity C_i, $1 \leq i \leq k-1$, and every serviced edge must be serviced by exactly one route.

3.4 PROBE: A Proposed Re-Routing Algorithm

One disadvantage of the conventional approach for managing vehicle breakdown is that the cost to complete the re-routing is bounded above by the sum of the cost of the two longest routes. Our interest is in an algorithm to reduce makespan. We propose a new algorithm, PROBE for *Path scanning Re-rOuting under BrEakdown*,

based on the *Path Scanning with Random Task* (PSRT) algorithm [36].

Given an instance of CARP-VB, PROBE starts building routes corresponding for each of the $k-1$ available vehicles from their current locations. If there are collection vehicles that have already completed their routes, they may be redeployed from the depot with their full capacity. In each iteration, PROBE adds the nearest task from the remaining tasks to each route. Thus rather than completing one route at a time as in path scanning, PROBE instead extends each route by one task at a time while at the same time attempting to reduce unbalance in the length of the routes; the goal is to produce routes with shorter makespans. If while adding a task to a route the vehicle is found to have insufficient remaining capacity, it deadheads to the depot, empties its load, and continues building its route from the depot with full capacity; see algorithm 1 for details.

3.5 Evaluation of PROBE

3.5.1 Benchmark Instances and Metrics Measured

We evaluate PROBE on classic CARP benchmark instances: (1) `GDB` is a set of 23 artificial instances, varying between 7–27 vertices and 11–55 edges, all of which are required [104, 29]. (2) `VAL` is a collection of 34 instances based on 10 randomly generated graphs with 25–50 nodes and 34–97 required edges [105]. Each graph has a set of 3 or 4 instances that differ in number of vehicles and their capacities. (3) `EGL` is a collection of 24 instances for winter gritting applications in Lancashire county in the UK [106]. It is based on two graphs, one with 77 vertices and 98 edges and the other with 140 vertices and 190 edges. The number of required edges varies between 51 and 190. (4) `EGL-Large` is a collection similar to `EGL` but on a larger graph with 255 vertices and 375 edges [41]. It has 2 sets of 5 instances, one with 347 edges and

Algorithm 1: PROBE Algorithm

Input: Pending_tasks ; Available_Vehicles; last_Completed_Tasks, $T_L \mid |T_L| = |V_R|$; cur_Vehicle_Capacities, $Q_C \mid |Q_C| < |Q|$,

Output: *route*, *route_cost*

```
1  current_MS ← 0                                   /* current Makespan */
2  set current_loc from last_Completed_Tasks
3  repeat
4      foreach V in Available_Vehicles do
5          if route_cost(V) < 0.5 * current_MS then   /* Add only if more than
            half of the Makespan */
6
7              next_task ← Get_Nearst_task ()
8              if next_task= ∅ then          // Visit Depot to empty & retain Full
                capacity
9
10                 Add Depot(D,D) to route(V)
11                 Q_C ← full Capacity, Q
12                 update route_cost(V)
13                 current_loc ← Depot
14             else
15                 Add next_task to route(V)
16                 update route_cost(V) and Q_C(V)
17                 update current_loc(V)
18     update current_MS
19 until Pending_Tasks = ∅
```

Algorithm 2: GET_NEARST_TASK Algorithm

Input: *current_loc*(V), Q_C(V), Pending_tasks

Output: Nearest_task

```
current_distance ← ∞
foreach task in Pending_tasks do
    distance = Shortest path from current_loc to the task
    if distance < current_distance then
        Nearest_task ← task
        current_distance = distance
    else if distance == current_distance then    /* If more than one task at
     equal distance, pick randomly */
        Assign task to Nearest_task with prob 0.5
    end
end
if current_distance = ∞ then
                                   /* No tasks available with Demand ¡ Q_C */
    Nearest_task ← ∅
end
```

the other with 375 edges, with all edges required. The individual instances within each set are created with different vehicle capacities.

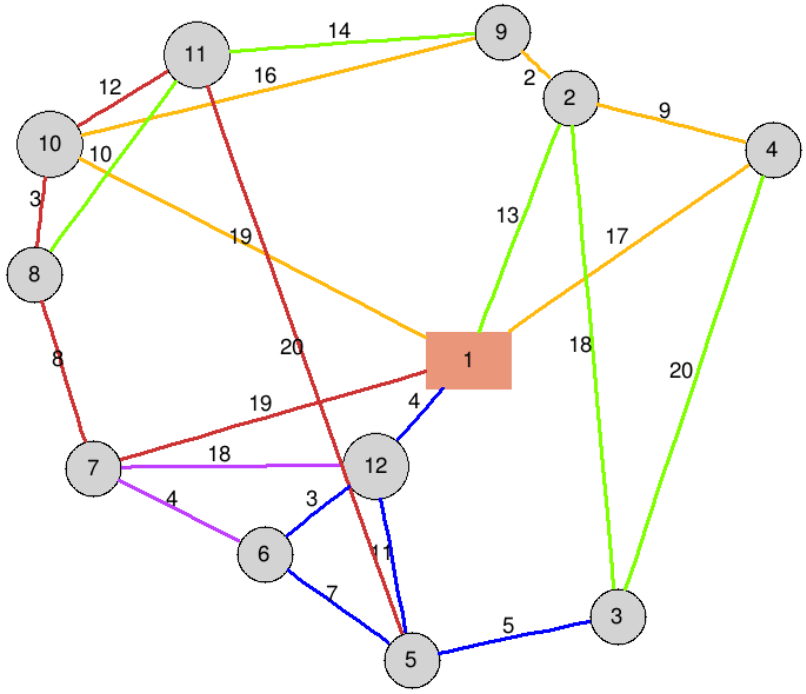
We evaluate the quality of the CARP-VB solutions using three metrics: (1) The maximum length route or *makespan*; (2) The difference between the maximum and minimum route length or *range*; and, (3) The sum of absolute differences of each route length from the mean route length or *discrepancy*.

3.5.2 Simulating Breakdown Events

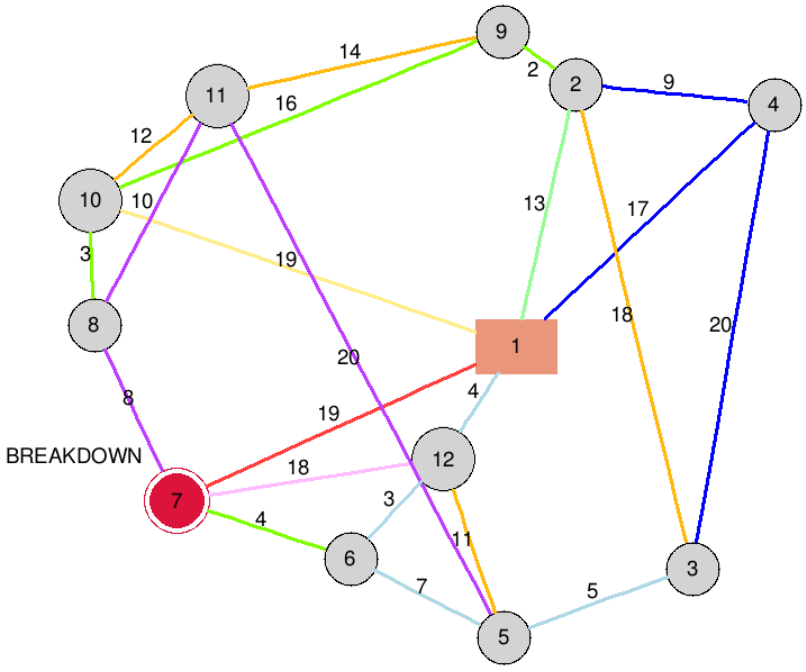
Figure 3.1a shows the first instance of `GDB`. It has 12 vertices and 22 edges, each with demand 1. The depot is at vertex 1. There are 5 collection vehicles, each with capacity 5.

The initial solution to CARP is found by the PSRT heuristic [36] run with 1000 random seeds, choosing the routes with shortest length. For `GDB1` PSRT computes 5 routes each shown in a different colour. The length of route $|r_i|, 1 \leq i \leq 5$, is $(63, 76, 105, 39, 33)$; the makespan is 105, the range is 72, and the discrepancy from the mean of 52.58 is 109.80. PROBE is used after a breakdown event.

We assume that the time to service the longest route in the CARP solution is the shift length, usually 7 hours excluding breaks. Secondly, we assume that the speed of servicing a route is the same for each vehicle throughout its shift. Because the `GDB1` instance has a makespan of 105 units, we compute the vehicle speed as $\frac{105}{7\times60} = 0.25$ units/minute. Suppose that vehicle 2 breaks down 1.5 hours into its shift. Then the distance covered by each vehicle is 0.25 units/minute $\times$ 90 minutes $\approx$ 22 units. This means that vehicle 2 has only completed one task in its route, $r_2 = ((1,7),(7,8),(8,10),(10,11),(11,5))$, namely $(1,7)$ with cost 19 depicted by the red edge in Fig. 3.1. The breakdown location is therefore vertex 7, indicated by the red node in Fig. 3.1b. The four unserved edges (tasks) in r_2 must be completed by the



(a) Initial CARP routes for `GDB1` by PSRT



(b) CARP-VB routes for `GDB1` by PROBE

Figure 3.1: Initial Routes for `GDB1`, and after PROBE Re-routing.

remaining operational vehicles.

Similarly, vehicle 1 has also only completed one task $(1, 10)$ in its route r_1. After the break down, vehicle 1 starts its new route at vertex 10, with left-over capacity of 4. The routes computed for each the remaining vehicles by PROBE is depicted by a different colour in Fig. 3.1b, with route lengths of $(71, 59, 70, 60)$. Hence, the makespan of the CARP-VB solution is 71 from the breakdown event. Dividing by the speed, we obtain the route completion times in hours as $(4.73, 3.93, 4.67, 4.00)$. Accounting for the vehicles operating for 1.5 hours before the breakdown, the completion time in hours for the remaining vehicles is $(6.23, 5.43, 6.17, 5.50)$.

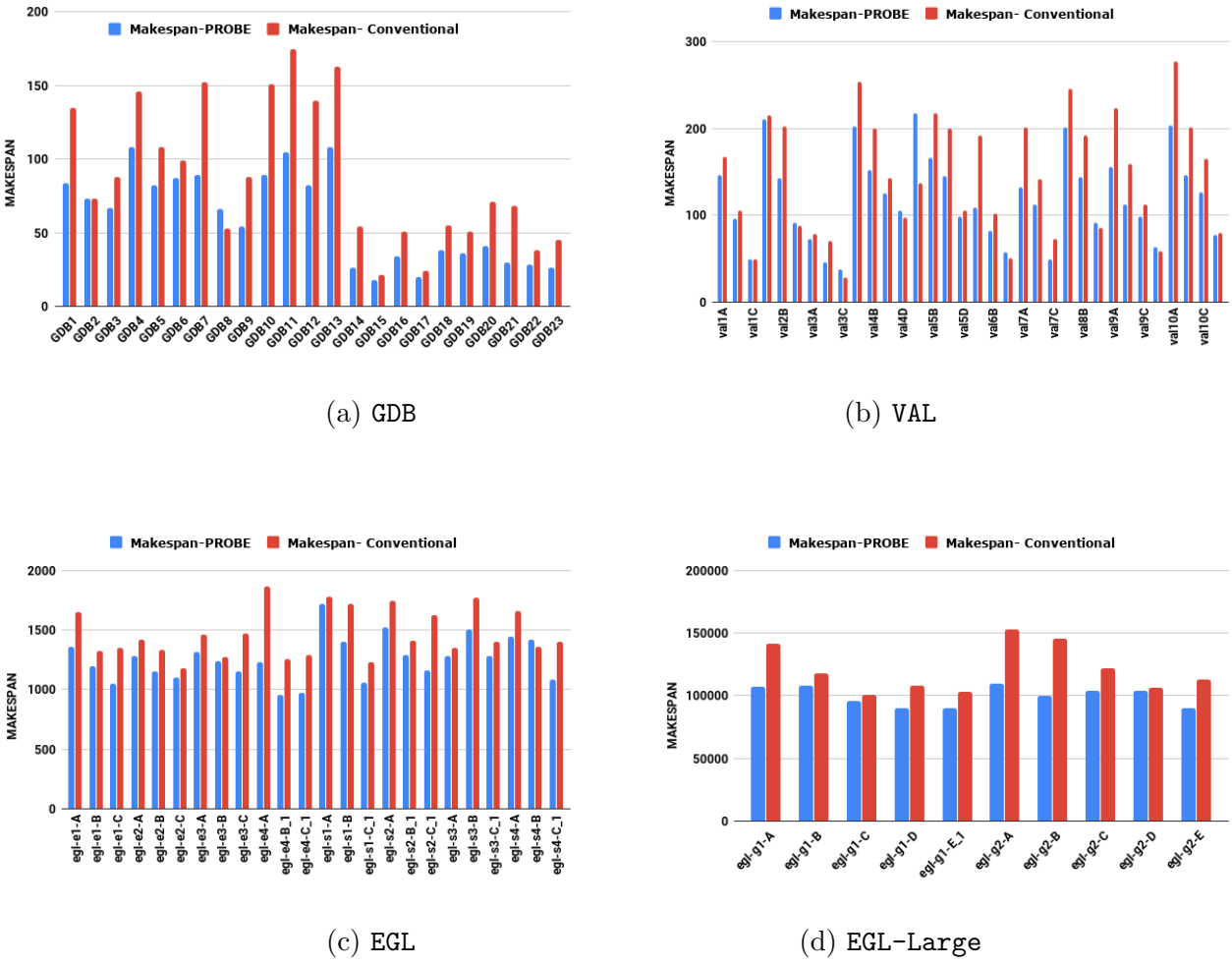
3.5.3 Results for the CARP Benchmark Instances

The initial solutions for the benchmark instances in `GDB` and `VAL` are generated using the PSRT heuristic [36]. A custom heuristic is used for `EGL` and `EGL-Large` that generates solutions with reduced makespan and balanced routes; see [107]. Then a breakdown event is generated and PROBE is run for CARP-VB. We compare our results with those generated using conventional solution. We present here the results of vehicle 2 breaking down after 1 hour of service, however these results are representative of other breakdown scenarios; see [107].

Makespan

Figures 3.2a-3.2d show the makespan for the two CARP-VB solutions on `GDB`, `VAL`, `EGL`, and `EGL-Large`, respectively. The average reduction in makespan for `GDB` is 33.49%. Only for `GDB8` is the makespan larger than the conventional method. The average reduction in makespan for `VAL` is 21.29%. Even though there are increases in makespan for eight instances, only for three instances is the increase more than 10%. The average reduction in makespan for `EGL` and `EGL-Large` is 15.02% and 16.48%,

respectively. Only for EGL-S4-B in EGL does PROBE result in a higher makespan. In general, higher makespans in PROBE result from unbalance in the initial CARP solutions.



(a) GDB (b) VAL

(c) EGL (d) EGL-Large

Figure 3.2: Makespan: PROBE vs. Conventional Method.

Range

Figures 3.3a-3.3d show the results for range for the two re-routing solutions on GDB, VAL, EGL, and EGL-Large, respectively. The range for GDB and VAL are lower on average by 88.48% and 74.75%, which is significant. The number of instances for which

Table 3.1: Makespan for GDB Instances: PROBE vs. Conventional Method

Instance	**Makespan Conventional**	**Makespan Probe**	**Percentage Reduction**
GDB1	135	84	37.78
GDB2	73	73	0.00
GDB3	88	67	23.86
GDB4	146	108	26.03
GDB5	108	82	24.07
GDB6	99	87	12.12
GDB7	152	89	41.45
GDB8	53	66	-24.53
GDB9	88	54	38.64
GDB10	151	89	41.06
GDB11	175	105	40.00
GDB12	140	82	41.43
GDB13	163	108	33.74
GDB14	54	26	51.85
GDB15	21	18	14.29
GDB16	51	34	33.33
GDB17	24	20	16.67
GDB18	55	38	30.91
GDB19	51	36	29.41
GDB20	71	41	42.25
GDB21	68	30	55.88
GDB22	38	28	26.32
GDB23	45	26	42.22
		Avg	33.49

Table 3.2: Makespan for `val` Instances: PROBE vs. Conventional Method

Instance	**Makespan Conventional**	**Makespan Probe**	**Percentage Reduction**
val1A	167	147	11.98
val1B	106	96	9.43
val1C	49	50	-2.04
val2A	215	211	1.86
val2B	203	143	29.56
val2C	88	92	-4.55
val3A	79	73	7.59
val3B	70	46	34.29
val3C	28	38	-35.71
val4A	254	203	20.08
val4B	200	152	24.00
val4C	143	126	11.89
val4D	97	106	-9.28
val5A	137	218	-59.12
val5B	218	166	23.85
val5C	200	145	27.50
val5D	105	98	6.67
val6A	192	109	43.23
val6B	102	82	19.61
val6C	51	58	-13.73
val7A	201	133	33.83
val7B	142	112	21.13
val7C	73	49	32.88
val8A	246	201	18.29
val8B	192	144	25.00
val8C	86	92	-6.98
val9A	223	156	30.04
val9B	159	113	28.93
val9C	113	98	13.27
val9D	59	64	-8.47
val10A	277	204	26.35
val10B	201	147	26.87
val10C	165	127	23.03
val10D	80	78	2.50
		Avg	21.29

Table 3.3: Makespan for `egl` Instances: PROBE vs. Conventional Method

Instance	**Makespan Conventional**	**Makespan Probe**	**Percentage Reduction**
egl-e1-A	1651	1362	17.50
egl-e1-B	1327	1197	9.80
egl-e1-C	1351	1051	22.21
egl-e2-A	1417	1281	9.60
egl-e2-B	1336	1156	13.47
egl-e2-C	1180	1103	6.53
egl-e3-A	1460	1318	9.73
egl-e3-B	1270	1244	2.05
egl-e3-C	1475	1155	21.69
egl-e4-A	1864	1234	33.80
egl-e4-B_1	1253	953	23.94
egl-e4-C_1	1287	973	24.40
egl-s1-A	1778	1721	3.21
egl-s1-B	1719	1401	18.50
egl-s1-C	1235	1060	14.17
egl-s2-A	1744	1524	12.61
egl-s2-B	1411	1288	8.72
egl-s2-C	1623	1165	28.22
egl-s3-A	1355	1281	5.46
egl-s3-B	1768	1503	14.99
egl-s3-C	1405	1283	8.68
egl-s4-A	1656	1442	12.92
egl-s4-B	1357	1418	-4.50
egl-s4-C	1407	1081	23.17
		Avg	15.02

Table 3.4: Makespan for `egl-large` Instances: PROBE vs. Conventional Method

Instance	Makespan Conventional	Makespan Probe	Percentage Reduction
egl-g1-A	141630	107462	24.12
egl-g1-B	117511	107731	8.32
egl-g1-C	100959	95557	5.35
egl-g1-D	108233	90362	16.51
egl-g1-E	102829	89736	12.73
egl-g2-A	152746	109720	28.17
egl-g2-B	145684	99688	31.57
egl-g2-C	121944	103713	14.95
egl-g2-D	106245	103629	2.46
egl-g2-E	113303	90002	20.57
		Avg	16.48

the range was higher is two in GDB and one in VAL. This indicates that PROBE achieves more balanced routes compared to the conventional method. The average reduction in range for the EGL and EGL-Large instances is 32.86% and 46.06%, respectively, with PROBE always lower in both these sets of instances.

Discrepancy

Figures 3.4a-3.4d show the discrepancy for the two re-routing solutions on GDB, VAL, EGL, and EGL-Large, respectively. The average reduction in discrepancy is 83.28% for GDB and 74.13% for VAL, with higher discrepancy for two instances in GDB and one instance in VAL. The average reduction in discrepancy for EGL and EGL-Large is 25.38% and 39.77%. In EGL-Large, all instances have lower discrepancy using

Table 3.5: Range for GDB Instances: PROBE vs. Conventional Method

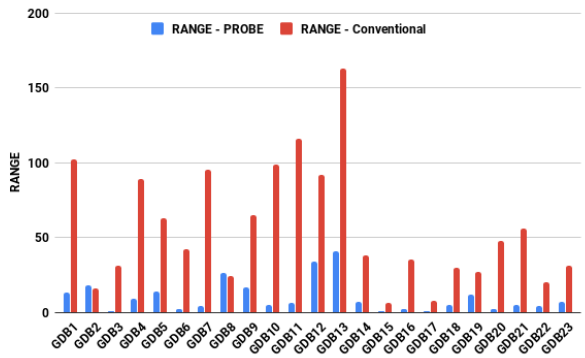
Instance	**Range Conventional**	**Range Probe**	**Percentage Reduction**
GDB1	102	13	87.25
GDB2	16	18	-12.50
GDB3	31	1	96.77
GDB4	89	9	89.89
GDB5	63	14	77.78
GDB6	42	2	95.24
GDB7	95	4	95.79
GDB8	24	26	-8.33
GDB9	65	17	73.85
GDB10	99	5	94.95
GDB11	116	6	94.83
GDB12	92	34	63.04
GDB13	163	41	74.85
GDB14	38	7	81.58
GDB15	6	1	83.33
GDB16	35	2	94.29
GDB17	8	1	87.50
GDB18	30	5	83.33
GDB19	27	12	55.56
GDB20	48	2	95.83
GDB21	56	5	91.07
GDB22	20	4	80.00
GDB23	31	7	77.42
		Avg	84.48

Table 3.6: Range for `val` Instances: PROBE vs. Conventional Method

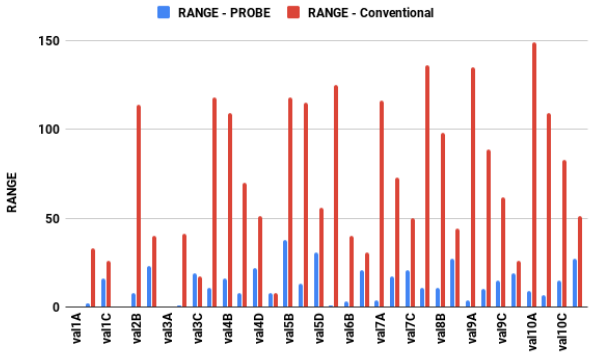
Instance	Range Conventional	Range Probe	Percentage Reduction
val1A	0	0	NA
val1B	33	2	93.94
val1C	26	16	38.46
val2A	0	0	NA
val2B	114	8	92.98
val2C	40	23	42.50
val3A	0	0	NA
val3B	41	1	97.56
val3C	17	19	-11.76
val4A	118	11	90.68
val4B	109	16	85.32
val4C	70	8	88.57
val4D	51	22	56.86
val5A	8	8	0.00
val5B	118	38	67.80
val5C	115	13	88.70
val5D	56	31	44.64
val6A	125	1	99.20
val6B	40	3	92.50
val6C	31	21	32.26
val7A	116	4	96.55
val7B	73	17	76.71
val7C	50	21	58.00
val8A	136	11	91.91
val8B	98	11	88.78
val8C	44	27	38.64
val9A	135	4	97.04
val9B	89	10	88.76
val9C	62	15	75.81
val9D	26	19	26.92
val10A	149	9	93.96
val10B	109	7	93.58
val10C	83	15	81.93
val10D	51	27	47.06
		Avg	74.75

Table 3.7: Range for `egl` Instances: PROBE vs. Conventional Method

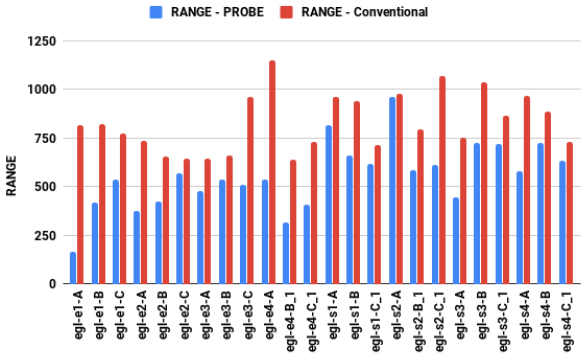
Instance	**Range Conventional**	**Range Probe**	**Percentage Reduction**
`egl-e1-A`	815	166	79.63
`egl-e1-B`	822	416	49.39
`egl-e1-C`	771	534	30.74
`egl-e2-A`	735	372	49.39
`egl-e2-B`	655	424	35.27
`egl-e2-C`	644	570	11.49
`egl-e3-A`	641	476	25.74
`egl-e3-B`	657	537	18.26
`egl-e3-C`	961	510	46.93
`egl-e4-A`	1150	535	53.48
`egl-e4-B`	638	314	50.78
`egl-e4-C`	727	405	44.29
`egl-s1-A`	962	816	15.18
`egl-s1-B`	939	661	29.61
`egl-s1-C`	713	618	13.32
`egl-s2-A`	975	962	1.33
`egl-s2-B`	796	584	26.63
`egl-s2-C`	1071	610	43.04
`egl-s3-A`	750	444	40.80
`egl-s3-B`	1034	726	29.79
`egl-s3-C`	866	718	17.09
`egl-s4-A`	965	579	40.00
`egl-s4-B`	888	726	18.24
`egl-s4-C`	732	631	13.80
		Avg	32.86



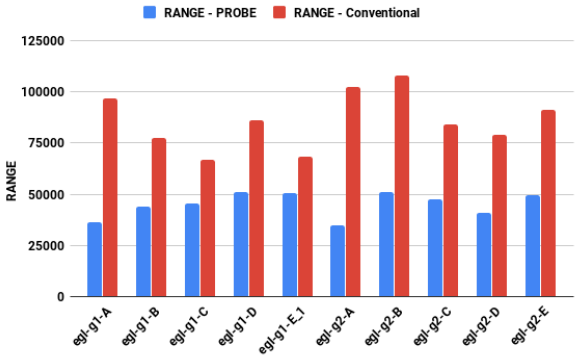
(a) GDB



(b) VAL



(c) EGL



(d) EGL-Large

Figure 3.3: Range: PROBE vs. Conventional Method.

PROBE. In EGL there are 11 instances for which the discrepancy is higher than in the conventional method. The high reduction in *range* also supports that PROBE achieves more balanced routes.

Run Time

Run time is critical for re-routing in breakdown management, so that the waste collection is not delayed. Figures 3.5a-3.5d plot run time as a function of the number of required edges, collected on a Dell desktop with an Intel i5 processor and 16 GB

Table 3.8: Range for `egl-large` Instances: PROBE vs. Conventional Method

Instance	Range Conventional	Range Probe	Percentage Reduction
egl-g1-A	96639	36118	62.63
egl-g1-B	77552	43801	43.52
egl-g1-C	66665	45367	31.95
egl-g1-D	86319	51052	40.86
egl-g1-E	68448	50413	26.35
egl-g2-A	102310	34997	65.79
egl-g2-B	107886	51043	52.69
egl-g2-C	84297	47618	43.51
egl-g2-D	78788	41111	47.82
egl-g2-E	91292	49742	45.51
		Avg	46.06

main memory, running Linux. The run time grows linearly with number of required edges, indicating that PROBE is scalable for real-world instances of CARP-VB.

Impact of Probe's Re-routing on Total Cost

PROBE is designed to get good min-max objective value required in disruptions context by adding the tasks to each of the vehicles in parallel. However, the *Minmax* and *Minsum* or total cost objectives are conflicting and hence optimizing for one would result in poor value for the other [25]. We compared the total cost of PROBE with that of conventional method.The results are given in the table 3.13, 3.14, 3.15, 3.16. The results indicates that the PROBE does not necessarily degrades the total cost

Table 3.9: Discrepancy for GDB Instances: PROBE vs. Conventional Method

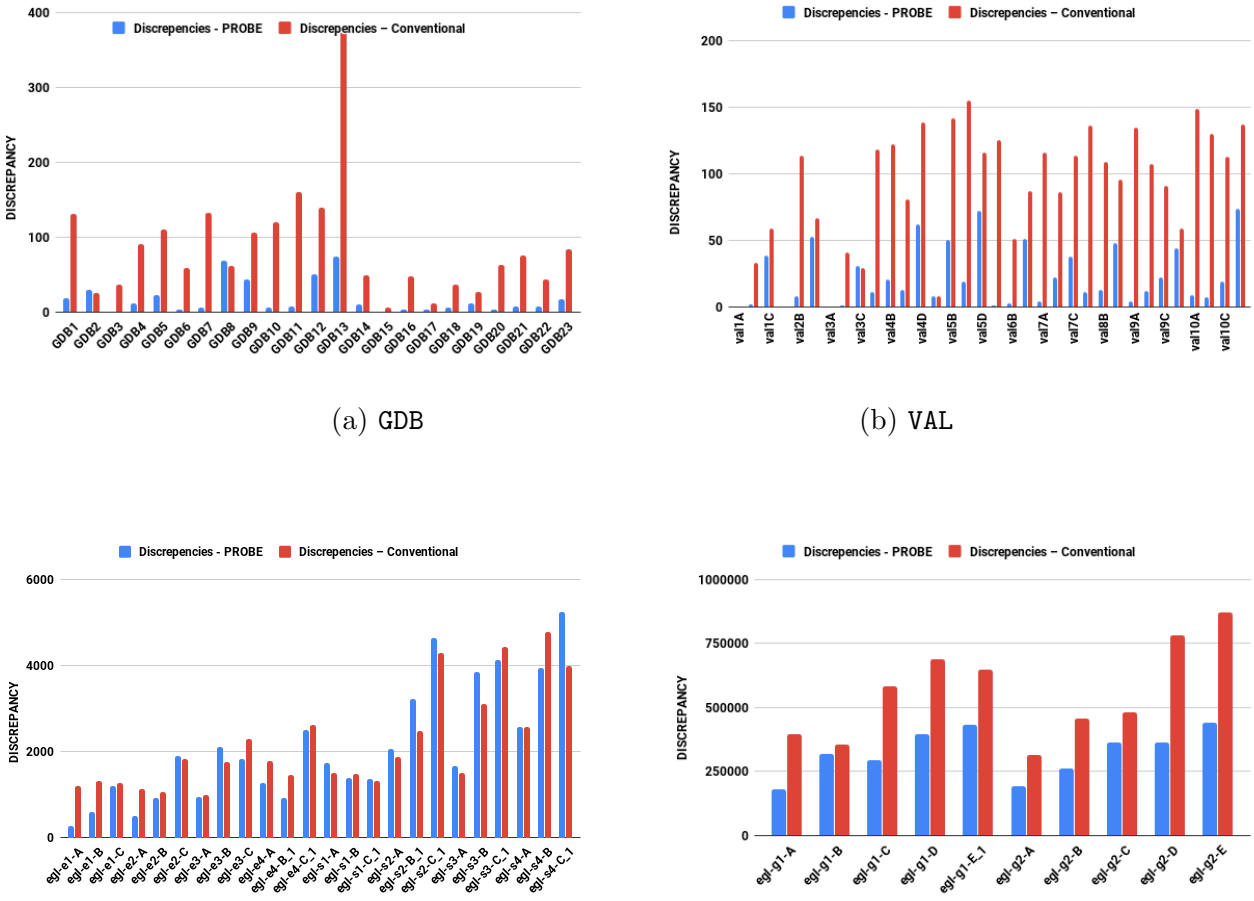
Instance	**Discrepancy Conventional**	**Discrepancy Probe**	**Percentage Reduction**
GDB1	131	18	86.26
GDB2	26	30	-15.38
GDB3	37	1	97.30
GDB4	90	11	87.78
GDB5	110	22	80.00
GDB6	58	3	94.83
GDB7	132	6	95.45
GDB8	61	69	-13.11
GDB9	106	44	58.49
GDB10	120	6	95.00
GDB11	160	8	95.00
GDB12	139	50	64.03
GDB13	381	74	80.58
GDB14	49	10	79.59
GDB15	6	1	83.33
GDB16	47	3	93.62
GDB17	11	3	72.73
GDB18	36	6	83.33
GDB19	27	12	55.56
GDB20	63	3	95.24
GDB21	76	8	89.47
GDB22	44	8	81.82
GDB23	83	17	79.52
		Avg	83.28

Table 3.10: Discrepancy for `val` Instances: PROBE vs. Conventional Method

Instance	Discrepancy Conventional	Discrepancy Probe	Percentage Reduction
val1A	0	0	NA
val1B	33	2	93.94
val1C	59	39	33.90
val2A	0	0	NA
val2B	114	8	92.98
val2C	67	53	20.90
val3A	0	0	NA
val3B	41	1	97.56
val3C	29	31	-6.90
val4A	118	11	90.68
val4B	122	21	82.79
val4C	81	13	83.95
val4D	139	62	55.40
val5A	8	8	0.00
val5B	142	50	64.79
val5C	155	19	87.74
val5D	116	72	37.93
val6A	125	1	99.20
val6B	51	3	94.12
val6C	87	51	41.38
val7A	116	4	96.55
val7B	86	22	74.42
val7C	114	38	66.67
val8A	136	11	91.91
val8B	109	13	88.07
val8C	96	48	50.00
val9A	135	4	97.04
val9B	107	12	88.79
val9C	91	22	75.82
val9D	59	44	25.42
val10A	149	9	93.96
val10B	130	7	94.62
val10C	113	19	83.19
val10D	137	74	45.99
		Avg	74.13

Table 3.11: Discrepancy for `egl` Instances: PROBE vs. Conventional Method

Instance	**Discrepancy Conventional**	**Discrepancy Probe**	**Percentage Reduction**
egl-e1-A	1193	272	77.20
egl-e1-B	1309	588	55.08
egl-e1-C	1271	1193	6.14
egl-e2-A	1138	515	54.75
egl-e2-B	1072	922	13.99
egl-e2-C	1820	1910	-4.95
egl-e3-A	994	953	4.12
egl-e3-B	1763	2100	-19.12
egl-e3-C	2285	1833	19.78
egl-e4-A	1787	1282	28.26
egl-e4-B	1453	934	35.72
egl-e4-C	2618	2495	4.70
egl-s1-A	1510	1734	-14.83
egl-s1-B	1489	1398	6.11
egl-s1-C	1321	1368	-3.56
egl-s2-A	1884	2067	-9.71
egl-s2-B	2487	3229	-29.84
egl-s2-C	4303	4638	-7.79
egl-s3-A	1501	1668	-11.13
egl-s3-B	3103	3856	-24.27
egl-s3-C	4426	4123	6.85
egl-s4-A	2563	2569	-0.23
egl-s4-B	4778	3951	17.31
egl-s4-C	4001	5245	-31.09
		Avg	25.38



(a) GDB (b) VAL

(c) EGL (d) EGL-Large

Figure 3.4: Discrepancy: PROBE vs. Conventional Method.

objective.

Figures 3.6a-3.6d show the results for total cost for the two re-routing solutions on GDB, VAL, EGL, and EGL-Large, respectively. The average rise in the total cost value is 6.9% in GDB, 16.26% in val, 10.47% in egl and 11.29% in egl-large. PROBE has only a low level of impact on the total cost. This can be considered as one of the important advantage to adopt PROBE over conventional method.

Table 3.12: Discrepancy for `egl-large` Instances: PROBE vs. Conventional Method

Instance	Discrepancy Conventional	Discrepancy Probe	Percentage Reduction
egl-g1-A	395674	181545	54.12
egl-g1-B	353878	319802	9.63
egl-g1-C	583581	295136	49.43
egl-g1-D	687584	394192	42.67
egl-g1-E	648192	431094	33.49
egl-g2-A	312851	191976	38.64
egl-g2-B	455689	263328	42.21
egl-g2-C	482347	363194	24.70
egl-g2-D	780432	363976	53.36
egl-g2-E	870201	440138	49.42
		Avg	39.77

3.5.4 Discussion

With PROBE, our objective was a heuristic to minimize the *makespan* after a breakdown using only the available vehicles. Our strategy added one task at a time to each route in each iteration if it is balanced (less than $0.5 \times$ makespan) or only to those routes whose length is less than $0.5 \times$ makespan. As a result, PROBE obtains more balanced routes and a higher percentage reduction in range and disparity for CARP-VB.

When PROBE obtains worse metrics compared to the conventional method, it is due to the unbalance of the initial CARP solution. This is why `EGL` and `EGL-Large` show less improvement because the custom heuristic produces more balanced initial solutions. In general, initial solutions that produce more balanced solutions to CARP

Table 3.13: Total Cost for GDB Instances: PROBE vs. Conventional Method

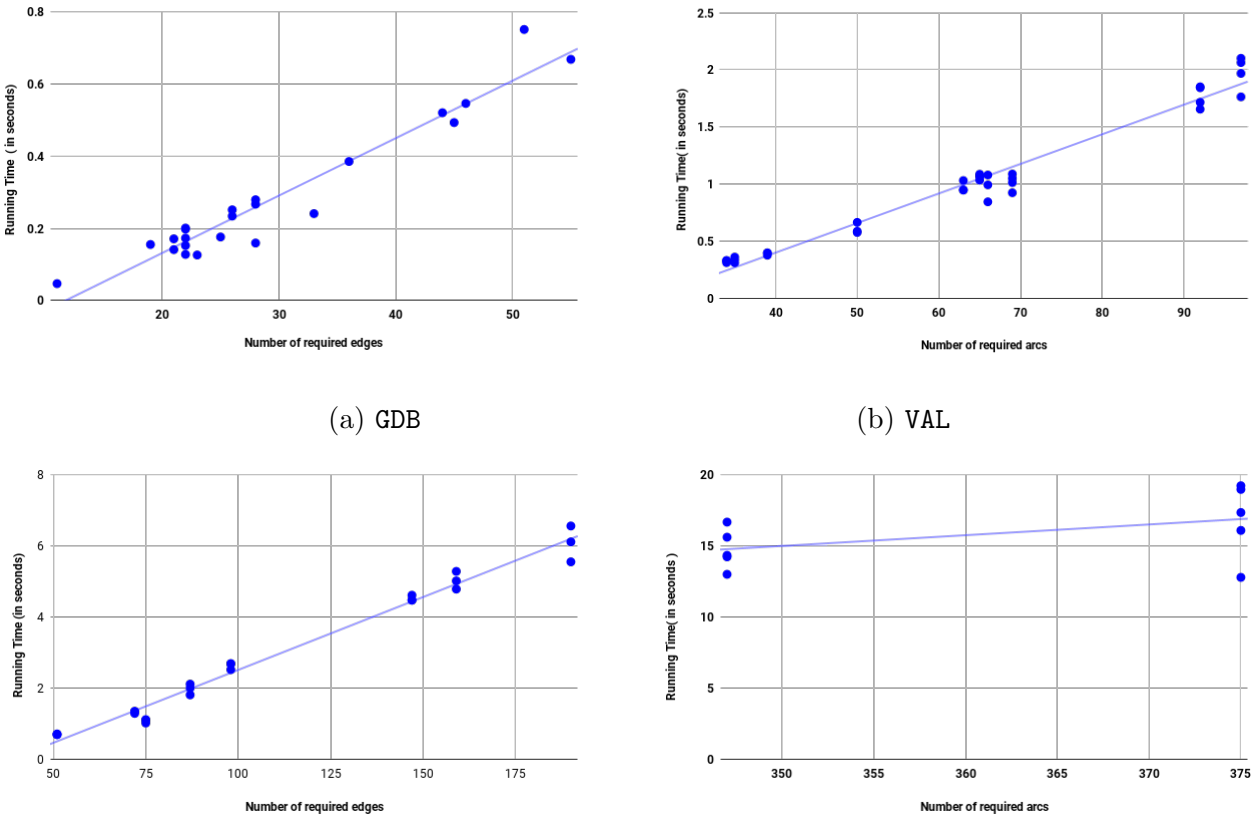
Instance	Total Cost Conventional	Total cost Probe	Percentage Increase
GDB1	318	323	-1.55
GDB2	344	335	2.69
GDB3	265	281	-5.69
GDB4	307	303	1.32
GDB5	387	390	-0.77
GDB6	345	322	7.14
GDB7	348	344	1.16
GDB8	520	358	45.25
GDB9	413	364	13.46
GDB10	258	273	-5.49
GDB11	404	402	0.50
GDB12	434	447	-2.91
GDB13	508	499	1.80
GDB14	94	117	-19.66
GDB15	52	54	-3.70
GDB16	131	127	3.15
GDB17	79	81	-2.47
GDB18	140	158	-11.39
GDB19	60	75	-20.00
GDB20	121	118	2.54
GDB21	139	147	-5.44
GDB22	184	179	2.79
GDB23	212	210	0.95
		Average	6.9
		Better than Conventional	11

Table 3.14: Total Cost for `val` Instances: PROBE vs. Conventional Method

Instance	Total Cost Conventional	Total cost Probe	Percentage Increase
val1A	147	167	-11.98
val1B	190	179	6.15
val1C	302	243	24.28
val2A	211	215	-1.86
val2B	278	292	-4.79
val2C	570	474	20.25
val3A	73	79	-7.59
val3B	91	99	-8.08
val3C	183	139	31.65
val4A	395	390	1.28
val4B	439	416	5.53
val4C	495	407	21.62
val4D	752	579	29.88
val5A	428	266	60.90
val5B	459	441	4.08
val5C	555	489	13.50
val5D	684	622	9.97
val6A	217	259	-16.22
val6B	241	229	5.24
val6C	460	325	41.54
val7A	262	286	-8.39
val7B	317	297	6.73
val7C	330	332	-0.60
val8A	391	356	9.83
val8B	418	412	1.46
val8C	642	570	12.63
val9A	308	311	-0.96
val9B	327	316	3.48
val9C	364	313	16.29
val9D	508	392	29.59
val10A	399	405	-1.48
val10B	430	408	5.39
val10C	479	433	10.62
val10D	602	509	18.27
		Average	16.26
		Better than Conventional	10

Table 3.15: Total Cost for `egl` Instances: PROBE vs. Conventional Method

Instance	Total Cost Conventional	Total cost Probe	Percentage Increase
egl-e1-A	5158	4215	22.37
egl-e1-B	6330	4859	30.27
egl-e1-C	7295	7090	2.89
egl-e2-A	6595	5340	23.50
egl-e2-B	8329	7985	4.31
egl-e2-C	11034	10532	4.77
egl-e3-A	7631	6736	13.29
egl-e3-B	10436	9280	12.46
egl-e3-C	13527	12761	6.00
egl-e4-A	8342	7749	7.65
egl-e4-B	11850	12418	-4.57
egl-e4-C	18569	18148	2.32
egl-s1-A	7602	6134	23.93
egl-s1-B	9139	8753	4.41
egl-s1-C	11612	12291	-5.52
egl-s2-A	13757	13823	-0.48
egl-s2-B	20604	19451	5.93
egl-s2-C	27174	26399	2.94
egl-s3-A	14672	13509	8.61
egl-s3-B	22499	20236	11.18
egl-s3-C	28738	27860	3.15
egl-s4-A	19309	18069	6.86
egl-s4-B	26545	23576	12.59
egl-s4-C	33889	35443	-4.38
		Average	10.47
		Better than Conventional	4



(a) GDB (b) VAL

(c) EGL (d) EGL-Large

Figure 3.5: Run Time vs. Number of Required Edges (Tasks).

will also improve solutions for CARP-VB.

3.6 Conclusion

In this chapter we introduced a daily problem faced in residential waste collection [10], that of completing the collection service after a vehicle breakdown. We used CARP to model for the residential waste collection because it can easily be extended to rich variants addressing the specific needs of cities and businesses.

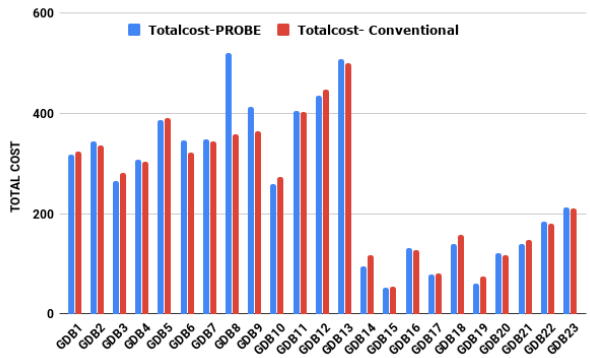
We formulated CARP-VB, a re-routing problem for vehicle breakdown. We proposed a new heuristic algorithm PROBE to compute routes for the remaining oper-

Table 3.16: Total Cost for `egl-large` Instances: PROBE vs. Conventional Method

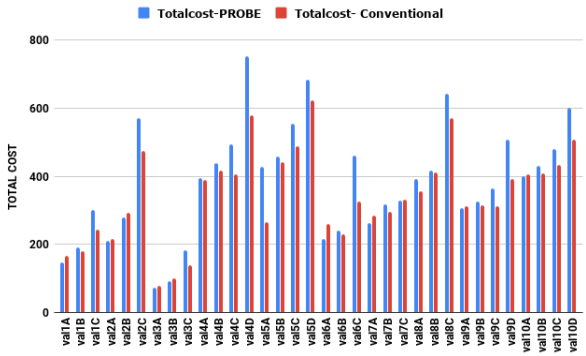
Instance	**Total Cost Conventional**	**Total cost Probe**	**Percentage Increase**
`egl-g1-A`	1744350	1567795	11.26
`egl-g1-B`	2194562	1850526	18.59
`egl-g1-C`	2333422	1973564	18.23
`egl-g1-D`	2491802	2405582	3.58
`egl-g1-E`	2883788	2677949	7.69
`egl-g2-A`	1992362	1880467	5.95
`egl-g2-B`	2275106	2075431	9.62
`egl-g2-C`	2684622	2343911	14.54
`egl-g2-D`	3001260	2651076	13.21
`egl-g2-E`	2920056	2649921	10.19
		Average	11.29
		Better than Conventional	NIL

ational vehicles starting from their locations at the time of breakdown, for all the unserviced streets in all routes, with the objective to minimize the makespan. We evaluated PROBE on the classical CARP benchmark instances `GDB`, `VAL`, `EGL`, and `EGL-Large`, comparing the results on makespan, range, and discrepancy with that of conventional re-routing method. PROBE results in a good average reduction in makespan, high reduction in range and discrepancies, and has run time linear in the number of required edges.

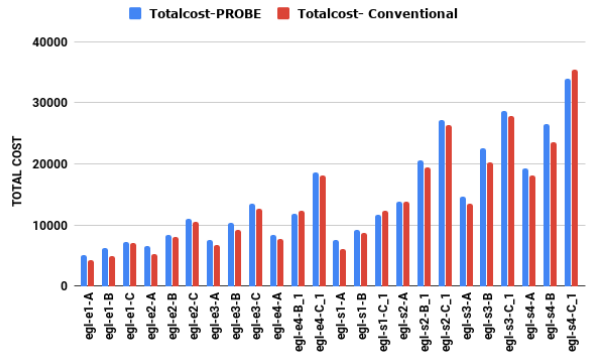
In the next chapter we present MA-ABC, a memetic multi-objective algorithm that optimizes three objectives: attractiveness, route balance and total cost.



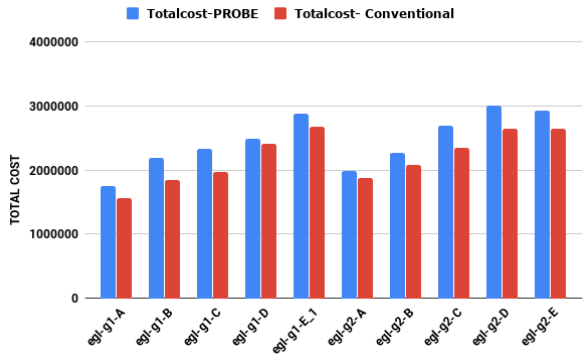
(a) GDB



(b) VAL



(c) EGL



(d) EGL-Large

Figure 3.6: Total Cost: PROBE vs. Conventional Method.

Chapter 4

MA-ABC - A MEMETIC MULTIOBJECTIVE ALGORITHM OPTIMISING FOR ATTRACTIVENESS, BALANCE AND COST

Routing problems are generally defined with the objective of minimizing total cost. Operators in the real world are interested in aspects other than cost and expect those aspects in the routing solutions that they want to use. *Visual attractiveness* is one aspect that operators look for in a routing solution. Balanced routes is another aspect that the operators are equally interested in. This chapter presents MA-ABC, a multi-objective memetic algorithm for CARP that optimizes three objectives: attractiveness, route balance and total cost.

We give a brief introduction to alternative objectives and multi-objective optimization in §4.1. We review related work in §4.2. The design of MA-ABC is provided in §4.3. A description of the benchmark instances, experimental set-up, and an analysis of the results are provided in §4.4, followed by discussion in §4.5. Finally, we summarize our work in §4.6.

4.1 Route Attractiveness, Balance and MA-ABC- An Introduction

Routing problems are among the most widely researched topics in operations research. In arc routing the goal is to cover, i.e., visit, each edge of the graph representing the problem. Applications of arc routing include services such as street sweeping, snow plowing, mail delivery, garbage collection, gritting roads with sand or salt, and the inspection of power lines. Expenditures on such services by public and private entities exceed billions of dollars annually in the U.S., emphasizing their

economic importance [108, 109, 7, 110].

CARP is the widely used arc routing problem to model the real world problems because of its generality. CARP is NP-hard [21]. As a result, an optimum solution for most practical instance sizes is intractable. Therefore a number of heuristics and metaheuristics have been proposed [24]. Many focus on finding a least cost solution for a service. Recently, operators of such services require routes that are balanced and visually attractive in addition to low cost. Routes that are *balanced* are about equal in length and contribute to workload equity and employee satisfaction [61, 62]. The *visual attractiveness* of routes is subjective, but non-crossing routes that provide well-defined service areas are clearly preferred. These additional objectives are important because they help address operational complexities associated with using the routes in practice.

Because it is often impossible to optimize all objectives simultaneously, a solution is called Pareto optimal if no objective can be improved without deteriorating another. *Multi-objective evolutionary algorithms* (MOEAs) use *evolutionary computation* (EC) methods to search for solutions at the Pareto front. These offer service operators a diverse set of solutions that represent different trade-offs among design objectives.

This chapter presents MA-ABC, a memetic algorithm to provide a heuristic solution for CARP to maximize route attractiveness and balance, and to minimize total cost. A memetic algorithm is a hybrid MOEA that incorporates local search. In MA-ABC local search controls the scope of the diversification emphasizing total cost. Non-dominated sorting is used to sort solutions into different ranked fronts. Within the same front, solutions are ranked by crowding distance. We use fitness functions for total cost and for balance, and introduce route continuity and route overlap to assess route attractiveness. To the best of our knowledge, MA-ABC is the first multi-objective optimization approach for CARP that includes attractive-

ness as one of the objectives. We conduct a thorough experimental evaluation of MA-ABC on CARP benchmark instances, comparing to the *path scanning with random task* (PSRT) heuristic [36]. MA-ABC provides heuristic solutions at the Pareto front that have a wide diversity in attractiveness and balance without deviating in the objective space of total cost.

4.2 Related Work

Since the introduction of CARP by Golden and Wong [21], many variants have emerged; see [7] for a comprehensive presentation of applications, resulting CARP variants, and solution methods.

Poot et al. [58] first reported that operators of services considered the heuristic solutions generated by the ORTEC vehicle routing software [111] to be poor, despite ranking highly on the traditional metric of total cost. The reason given was that the routes were not visually attractive. Operators desired routes with more subjective features such as compactness, no crossings, and fewer turns, even at the expense of cost [59]. Operationally, this assigns a route in which each driver has one geographically distinct area of responsibility. Visual preferences may serve as a source for new metrics that can help quantify the visual appeal of a route; Rossit et al. [11] review proposed metrics. Such new metrics may be used as alternatives to, or penalties in, the overall objective function.

A few publications consider visual attractiveness in variants of arc routing problems. Constantino et al. [59] propose a method to bound the number of nodes in more than one route. In partitioning street networks, Lum et al. [112] define a similar *route overlapping index* (ROI) and develop a heuristic for an *uncapacitated* arc routing problem with compact, balanced, and visually appealing routes. Corberan et al. [113] provide a heuristic solution to the same problem also using a multi-objective

approach.

Route balance is another important objective [60] because it contributes to workload equity [61, 62]. Measures of balance have included *range*, the difference between the maximum and minimum route length, and *makespan*, the maximum length route. Several works consider the dual objectives of route balance and total cost in variants of routing problems [66, 114, 64].

Multi-objective evolutionary algorithms (MOEAs) are one of the most popular approaches for solving multi-objective optimization problems to obtain near Pareto optimal solutions [67]. Deb et al. [68] propose NSGA-II, a widely used method of MOEA due to its efficiency. It uses non-dominated sorting to obtain the Pareto fronts and for selection operations. NSGA-II's sorting method is more efficient than other popular MOEA methods such as SPEA [115].

Lacomme et al. [48, 116, 117] propose a memetic algorithm based on the *route first, split second* method to solve CARP. A single *giant tour* covering all required tasks without capacity limits is used to represent individuals. This helps avoid the need to repair routes after crossover, due to capacity overruns. Their method is effective because it replaces the mutation operation with a local search, and this approach has since been adopted for many variants of arc and node routing problems [39, 118]. An efficient splitting algorithm that splits the giant tour is used to evaluate and update the fitness values and to retrieve the final routing solution from the chromosomes.

Other multi-objective optimization methods have been applied to CARP. Mei et al. [119] use a decomposition-based framework for CARP with the dual objectives of minimizing total cost and makespan. Grandinetti et al. [120] use an ϵ-constraint method for CARP with the same objectives.

Tang et al. [121] use a giant tour and introduce a new *merge-split* operator, in addition to other move operators in the local search. This operator merges multiple

routes back into a single tour and then splits it again.

Usberti et al. [122] use a memetic algorithm for a CARP variant. Three types of local search are combined with a stochastic filter to filter out solutions before applying the local search. Chen et al. [51] adopt a hybrid approach for CARP, performing only a single solution update per generation with local refinement. Martinez et al. [123] propose a genetic algorithm for CARP which incorporates local search. To solve larger-scale ($>$ 300 tasks) CARP, Mei et al. use decomposition methods and co-operative co-evolution methods [124]. Stochastic variants of CARP have used memetic algorithms [125, 126, 127]. Wang et al. use an *estimation of distribution algorithm* and a stochastic local search for a stochastic variant of CARP [128]. Handa et al. use an EC method for dynamic route optimization in CARP [129, 130]. Liu et al. [131] use a memetic algorithm with a new splitting scheme to solve dynamic CARP that was improved in [132]. Shang et al. consider a dual objective CARP through a co-operative co-evolutionary method [133].

As we see next, our proposed memetic heuristic algorithm makes use of ideas incorporated in NGSA-II [68] and in the route first, split second method [48, 116, 117], but we believe it is the first to include attractiveness as an objective in CARP.

4.3 Multi-Objective Memetic Algorithm

We propose MA-ABC, a memetic heuristic algorithm for CARP, which maximizes route attractiveness and balance, while minimizing total cost.

MA-ABC pseudocode is given in Algorithm 3. An initial population of μ individuals is generated. Each individual is a *giant tour*. Giant tours can be generated using any heuristic for CARP [27, 24] by ignoring demands and capacity limits; we use the path scanning heuristic [29].

In each of $NGEN$ generations, λ offspring are generated. Parents are chosen

using *tournament selection* in NSGA-II with the ranking determined by dominance and crowding distance. With probability $CXPB$, an order crossover operation is performed. With probability $MUTPB$, a giant tour is split into routes for each of the vehicles. A local search is then performed using a single insertion move operator that emphasizes total cost. The routes are then joined to form a new individual.

After λ offspring have been generated, the fitness metrics of attractiveness, balance, and cost are updated for each. Elitism is incorporated through the $(\mu + \lambda)$ strategy [68]. After $NGEN$ generations, we obtain the Pareto front ranked by non-dominated sorting. We now describe these steps in a more detail.

4.3.1 Selection Using NSGA-II

MA-ABC selects individuals for the crossover operation. It first selects μ of $\mu + \lambda$ individuals from the current generation. It also selects individuals at the Pareto front to return as routing solutions. MA-ABC uses NSGA-II specifically its non-dominated sorting and crowding distance for ranking solutions.

NSGA-II sorts the solutions into different ranked fronts based on *non-dominance* [114, 64]. NSGA-II first finds all non-dominated solutions among the population in the current iteration. These solutions are then removed from the population and the non-dominated individuals among the remaining population are found. The process is repeated until all the solutions are ranked.

Within the same front, solutions are ranked based on *crowding distance* [68]. A solution with higher crowding distance on the same front indicates higher population diversity and is ranked more highly.

Algorithm 3: MA-ABC Algorithm

Input : Graph model $G = (V, E)$ for CARP

Output: Heuristic routing solutions maximizing balance and attractiveness, and minimizing total cost

```
set parameters NGEN, CXPB, MUTPB, μ, λ
create initial population of size μ
for i ← 1 to NGEN do
    for j ← 1 to λ do
        P1, P2 = TournamentSelectionDCD ()
        if prob < CXPB then
            C1 = OrderCrossover (P1,P2)
        else
            C1 = P1
        if prob < MUTPB then
            split = SplitGiantTour (C1)
            split = MutateByLocalSearch (split)
            C1 = JoinRoutes (split)
    update three fitness values for each of the λ offspring
    population =selectNSGA (μ, μ + λ)
Pareto_front = NonDominatedSort (population, μ)
return Pareto_front
```

4.3.2 Crossover

Many crossover operators have been studied in EC, and among these, order crossover (OX) is effective for routing problems [49]. In OX, two sites p and q are randomly selected with $1 \leq p \leq q \leq \tau$, where $\tau = |T|$. For parents P_1, P_2, child C_1 is obtained by copying tasks $P_1(p) \dots P_1(q)$ into $C_1(p) \dots C_1(q)$. $C_1(q+1) \dots C_1(\tau)$ and $C_1(1) \dots C_1(p-1)$ are filled by copying tasks from $P_2(q+1) \dots P_2(\tau)$ and $P_2(1) \dots P_2(q-1)$, taking tasks from P_2 not already in C_1. Child C_2 is created by interchanging the roles of P_1 and P_2.

4.3.3 Splitting Procedure

The splitting procedure plays a central role when a giant tour is used as an individual chromosome. Splitting is based on the *route first, split second* algorithm [34, 48]; see §4.2. The idea is to split the giant tour into routes based on the vehicle capacity limit C. Each route starts at the depot D, and deadheads to D from the vertex where the capacity is reached or prior to it being exceeded. We use this procedure to split the giant tour when reevaluating fitness values, before applying local search, and in retrieving the routes.

4.3.4 Local Search

As in Lacomme et al. [48], MA-ABC uses local search in place of mutation to improve exploitation of the search space and speed convergence. Once a giant tour has been split into routes, a *single insertion* move operator is applied to each pair (t, t') of tasks in the routing solution. Each task t in a route is moved after task t' in every other route. The move that minimizes the total cost is chosen.

Figure 4.1 gives an example of a single insertion move. In this example, the red

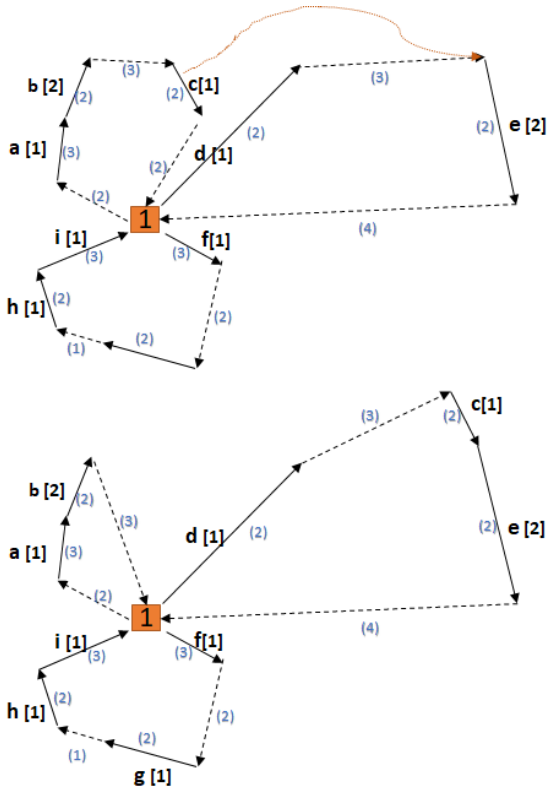


Figure 4.1: Example of the Single Insertion Move Operator

square is the depot, solid edges are tasks, dashed edges are not required, vehicle capacity $C = 4$, integers in square brackets are demands, and integers in parentheses are costs.

The three routes before the move are $((a, b, c)(d, e)(f, g, h, i))$ with costs $(14, 12, 13)$ that sum to a total of 39. After moving c after d the routes are $((a, b)(d, c, e)(f, g, h, i))$ with costs of $(10, 14, 13)$ that sum to 37, a cost savings of 2.

4.3.5 Fitness Functions

Let $R = \{r_1, r_2, \ldots, r_k\}$ be a set of routes for the k vehicles found by splitting a giant tour. Each route r_i is a sequence of streets (tasks) serviced $t_{i1} = (v_{i1}, v_{i2}), t_{i2} = (v_{i2}, v_{i3}), \ldots, t_{ij-1} = (v_{ij-1}, v_{ij})$, that make a closed walk starting and ending at the

depot.

The *total cost* is the sum of the route cost for all routes in R. The cost of a route is the cost of its serviced tasks including any deadheading via shortest paths.

$$\text{total cost} = \sum_{r_i \in R} \left(\sum_{t \in r_i} c(t) + \sum_{\ell=1}^{j-1} DeadHeading(v_{i\ell}, v_{i\ell+1}) \right) \tag{4.1}$$

We use *makespan* as a proxy for route balance. The makespan is the maximum route cost among the routes $r_i \in R$.

A novel contribution in this work is our definition of route attractiveness, a combination of two metrics: The *common node count* (CNC), a measure of route overlap, and the *discontinuity count* (DC), a measure of route contiguity. That is, the preference is for routes of different vehicles to have no overlap, and for sequences of tasks in a route not to be disconnected.

More formally, CNC is a count of vertices $v \in |V \setminus D|$ that exist in each route $r_i \in R$ with one endpoint of a task serviced by r_i.

$$CNC = \sum_{v \in |V \setminus D|} \left(\sum_{i=1}^{k} exists(v, r_i) - |V| - 1 \right) \tag{4.2}$$

This definition differs from the node count used in [59] because we count the nodes that only belong to the tasks serviced by the route. Nodes that belong to the tasks that are not serviced by the route or nodes that belong to non-required edges are not counted. We also count each endpoint of a serviced task unlike [59].

Given two consecutive tasks $t_{i\ell}, t_{i\ell+1}$ serviced by route r_i it is possible that the tasks are not contiguous, i.e., the vehicle must deadhead from the end of $t_{i\ell}$ to the start of $t_{i\ell+1}$. If any edge along the deadheading path has *at least one task that is serviced by another route*, we consider it a *discontinuity* in route r_i and increment DC.

$$DC = \sum_{r_i \in R} \left(\sum_{\ell=1}^{j-1} discontinuity(t_{i\ell}, t_{i\ell+1}) \right) \tag{4.3}$$

Both $exists(\cdot)$ and $discontinuity(\cdot)$ are simple indicator functions.

If the discontinuity count is less than $k = |R|$ then we set the *attractiveness* equal to the CNC, otherwise we set it to infinity:

$$\text{attractiveness} = \begin{cases} \text{CNC}, & \text{if } DC < k \\ \infty, & \text{if } DC \geq k \end{cases} \tag{4.4}$$

This definition of attractiveness forces MA-ABC to reject any solution that has high discontinuity in the selection of parents for offspring of a generation, and the selection of the next generation.

4.3.6 Elitism

Elitism is essential for the convergence of MOEAs [67]. We incorporated elitism in MA-ABC as implemented in NGSA-II [68].

4.4 Evaluation of MA-ABC

MA-ABC was coded in `python`; source code is provided [134] for reproducibility. We use the DEAP [135] EC framework, which includes an implementation of NSGA-II ranking and selection operations.

We evaluate two classic benchmark CARP instances: `val`, and `egl`. `val` [105] is a collection of 34 instances based on ten randomly generated graphs with 24-50 vertices and 34-97 edges, where all edges are required. Instances of the same graph differ by vehicle capacity. `egl` [106] is a collection of 24 instances generated based on a winter gritting application in Lancashire county in the U.K. It contains two graphs, one with 77 vertices and 98 edges (`egl-s`) and the other with 140 vertices and 190

edges (`egl-e`). Each graph consists of four sets of three instances each with the sets differing by number of required edges. The instances within a set (named with suffix A, B, C) differ by capacity limit of the vehicles.

We report the results of MA-ABC, using the parameter settings in Table 4.1, from a single run to show that our algorithm is efficient, i.e., that operators can use MA-ABC in a real-world scenario where solutions need to be generated in real time such as in a vehicle breakdown scenario [136]. For completeness, we also present a statistical analysis of 30 runs of our algorithm in §4.4.3.

Table 4.1: MA-ABC Parameter Settings

Parameter	**Value**
Number of generations ($NGEN$)	300
Population size (μ)	100
Offspring produced (λ)	100
Crossover rate ($CXPB$)	1
Mutation rate ($MUTPB$)	1

4.4.1 Results

We compare the results of MA-ABC with the solution generated by 1000 iterations of the *path scanning with random task* (PSRT) heuristic. PSRT was chosen because it seeks to optimize total cost. We present results for the objectives of attractiveness, total cost, and balance, in turn. Due to space limitations, see [134] for a presentation of all results, including the data used to generate figures.

Attractiveness

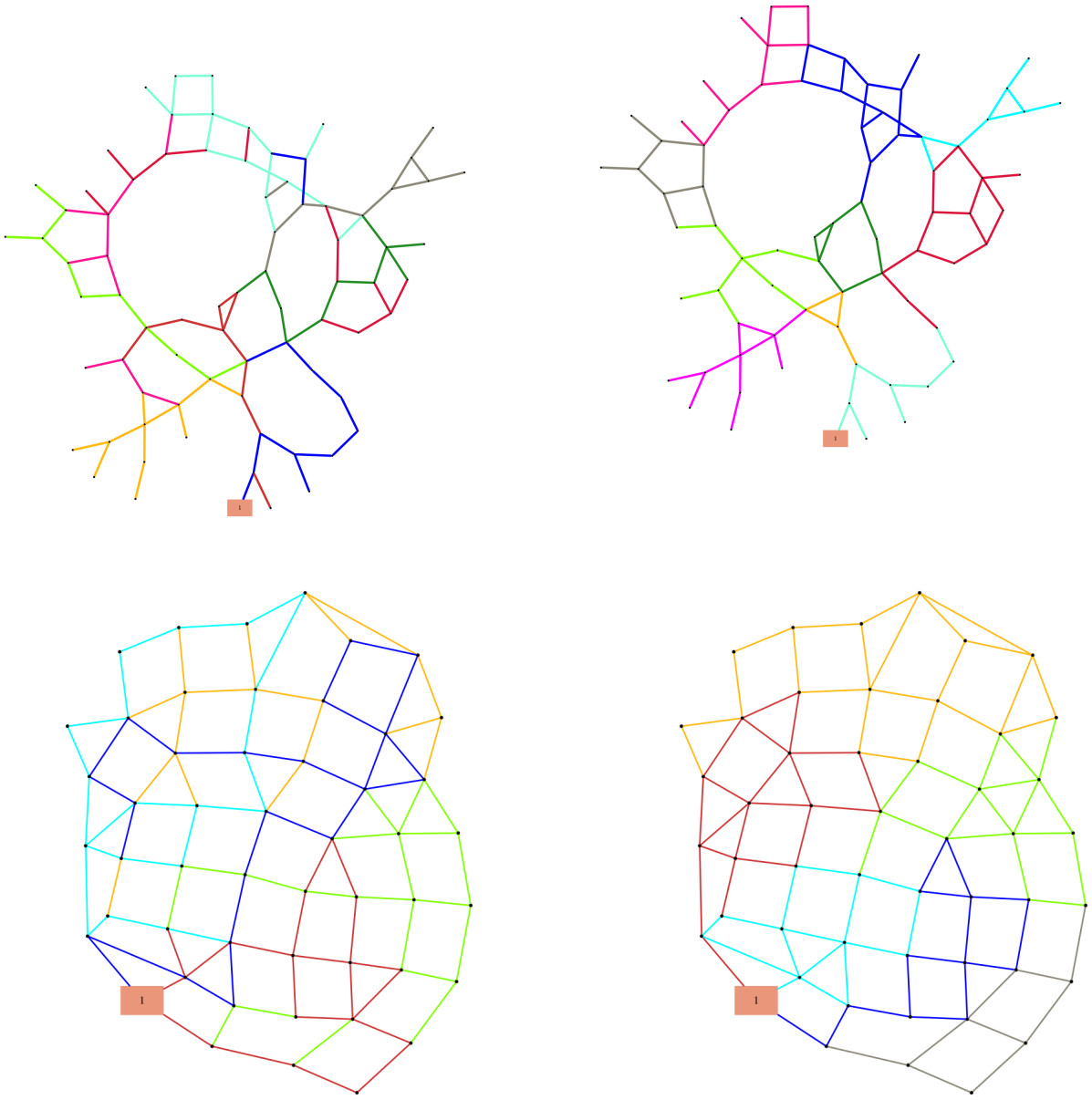
We first provide a qualitative evaluation of MA-ABC for a few CARP instances. Figure 4.2 shows a visualization of the routes generated by PSRT on the left and those produced by MA-ABC on the right, using a different colour for each route, for the `egl-e4-A` and `val10C` instances, respectively. As the figure shows, the routes generated by MA-ABC are compact, do not cross, and have better defined service areas; therefore they considered more attractive than the routes generated by PSRT.

We also provide a quantitative comparison of the algorithms using three metrics: The *connectivity index* (CI), the *average task distance* (ATD), and the *route overlapping index* (ROI) [59]. CI measures the average number of connected components within a route. ATD is a measure of average deadheading cost between task pairs within the same route. ROI measures the node overlap of the current solution with the overlapping of an "ideal" solution. For each metric, smaller is considered more attractive.

We compute these three metrics for the most attractive solution in the Pareto front found by MA-ABC and compare with PSRT. Table 4.2 shows the results for the `egl` instances and Table 4.3 shows the results for the `val` instances. For each metric the smaller value is shown in bold. We see that CI and ROI for MA-ABC are lower than those of PSRT for all instances. ATD is also lower for MA-ABC for all but six instances. Hence our routing solutions are considered attractive by external metrics not used in the algorithm.

Total Cost

The first row of Figure 4.3 plots the total cost of `egl` (a) and `val` (b) instances, produced by MA-ABC and PSRT; the optimal or current best known total cost



(a) PSRT (b) MA-ABC

Figure 4.2: Heuristic Routing Solution Produced By PSRT (*left*) and MA-ABC (right) on Two Instances: `egl-e4-A` (*top row*) and `val10C` (*bottom row*).

The rectangle represents the depot. Vehicles on some routes need to deadhead on the shortest path from the depot to the first task on the route.

Table 4.2: EGL: Attractiveness Metrics for PSRT vs. MA-ABC

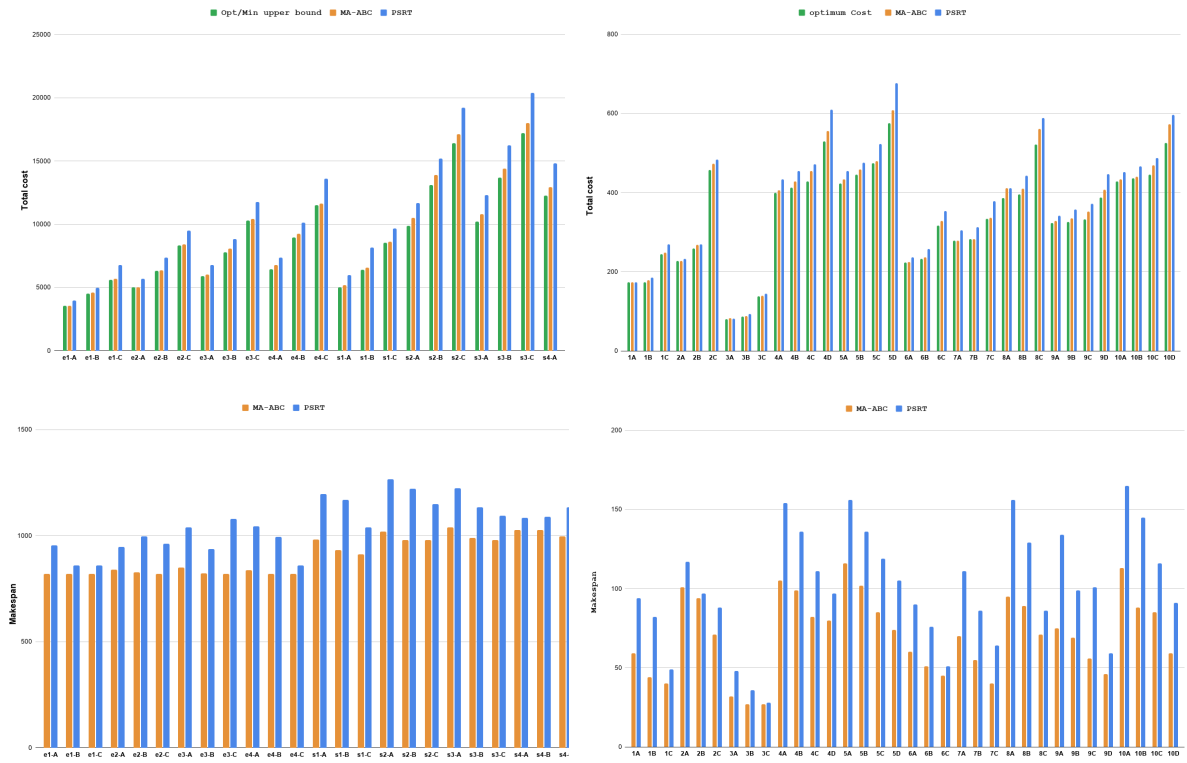
Instance	PSRT CI	PSRT ROI	PSRT ATD	MA-ABC CI	MA-ABC ROI	MA-ABC ATD
egl-e1-A	2.60	0.65	61.19	**2.17**	**0.54**	**43.14**
egl-e1-B	2.71	0.38	**45.35**	**2.29**	**0.38**	57.05
egl-e1-C	2.70	0.35	44.28	**1.30**	**0.35**	**23.40**
egl-e2-A	2.71	0.35	58.67	**2.00**	**0.28**	**58.22**
egl-e2-B	2.60	0.21	60.23	**1.64**	**0.20**	**43.44**
egl-e2-C	2.21	0.23	37.97	**1.40**	**0.22**	**28.91**
egl-e3-A	2.88	0.73	**67.31**	**2.30**	**0.61**	71.95
egl-e3-B	2.25	0.53	51.26	**1.64**	**0.47**	**32.64**
egl-e3-C	2.18	0.62	**33.40**	**1.83**	**0.59**	47.48
egl-e4-A	2.22	0.92	63.86	**1.92**	**0.73**	**67.48**
egl-e4-B	2.36	0.75	41.70	**1.75**	**0.68**	**40.56**
egl-e4-C	1.90	0.59	32.51	**1.29**	**0.57**	**26.79**
egl-s1-A	3.29	1.18	**83.33**	**2.57**	**1.18**	96.80
egl-s1-B	2.90	0.77	**76.66**	**2.18**	**0.71**	83.50
egl-s1-C	2.29	0.59	49.09	**1.60**	**0.56**	**45.60**
egl-s2-A	2.79	0.41	60.95	**2.18**	**0.36**	**44.17**
egl-s2-B	2.85	0.41	44.98	**1.79**	**0.36**	**36.71**
egl-s2-C	2.33	0.50	43.87	**2.03**	**0.48**	**29.16**
egl-s3-A	2.67	0.38	46.59	**1.94**	0.**36**	**35.94**
egl-s3-B	2.27	0.54	**41.67**	**1.96**	**0.51**	56.17
egl-s3-C	2.21	0.38	40.51	**1.44**	**0.36**	**19.95**
egl-s4-A	2.47	0.69	49.65	**1.43**	**0.65**	**39.04**
egl-s4-B	2.04	0.62	34.70	**1.38**	**0.59**	**23.53**
egl-s4-C	2.31	0.59	34.26	**1.28**	**0.54**	**19.97**

Table 4.3: VAL: Attractiveness Metrics for PSRT vs. MA-ABC

Instance	PSRT CI	PSRT ROI	PSRT ATD	MA-ABC CI	MA-ABC ROI	MA-ABC ATD
val1A	**1.50**	2.36	6.46	1.67	**1.69**	**5.94**
val1B	2.00	1.82	5.65	**1.40**	**0.95**	**3.43**
val1C	1.38	0.89	2.98	**1.22**	**0.59**	**1.72**
val2A	**1.00**	1.89	9.22	2.00	**1.30**	**7.00**
val2B	2.33	1.56	**8.14**	**1.67**	**1.56**	9.13
val2C	1.50	0.89	3.60	**1.00**	**0.59**	**1.30**
val3A	**1.00**	1.18	2.61	1.33	**1.04**	**2.57**
val3B	1.67	1.69	3.03	**1.00**	**1.02**	**1.85**
val3C	1.43	0.90	1.10	**1.13**	**0.66**	**0.66**
val4A	2.00	1.72	10.11	**1.20**	**1.44**	**8.29**
val4B	1.75	1.74	7.90	**1.33**	**1.50**	**7.30**
val4C	2.20	2.02	9.52	**1.38**	**0.93**	**5.25**
val4D	2.00	1.38	6.56	**1.10**	**0.99**	**5.15**
val5A	1.40	3.20	6.88	**1.40**	**1.63**	**6.88**
val5B	2.00	1.97	7.87	**1.33**	**1.58**	**6.87**
val5C	**1.80**	2.13	7.87	1.86	**1.74**	**7.11**
val5D	1.89	1.50	5.48	**1.50**	**1.30**	**5.58**
val6A	1.67	1.61	5.93	**1.20**	**0.72**	**3.98**
val6B	1.75	1.65	6.60	**1.00**	**0.60**	**2.57**
val6C	1.50	0.90	2.88	**1.18**	**0.67**	**1.97**
val7A	1.33	1.23	6.55	**1.20**	**0.76**	**3.86**
val7B	1.25	1.03	5.67	**1.00**	**0.54**	**3.11**
val7C	1.44	0.92	3.95	**1.00**	**0.62**	**1.80**
val8A	**1.33**	1.99	7.76	1.40	**1.73**	**6.76**
val8B	2.00	2.09	7.61	**1.67**	**1.72**	**7.38**
val8C	1.33	1.39	3.69	**1.18**	**1.17**	**3.60**
val9A	1.67	2.30	6.09	**1.40**	**1.37**	**4.46**
val9B	**1.75**	2.05	5.60	1.83	**1.37**	**4.08**
val9C	2.00	2.00	5.77	**2.00**	**1.58**	**4.04**
val9D	1.50	1.30	3.11	**1.33**	**1.17**	**3.08**
val10A	1.67	2.57	6.36	**1.60**	**1.68**	**4.62**
val10B	**2.00**	**1.92**	**4.74**	2.67	2.39	7.14
val10C	1.80	2.26	5.16	**1.71**	**1.50**	**4.24**
val10D	**1.60**	1.59	4.53	2.08	**1.42**	**3.89**

solution is also plotted.

In comparison to PSRT, MA-ABC found a lower total cost solution for all 24 instances of egl. The percentage of reduction ranges from 8.81-19.64% with an average reduction of 11.45%. The total cost found by MA-ABC ranges from 0.36-6.18% higher than the optimum (or best known solution), with an average of only 3.31% higher. MA-ABC is able to come quite close to the optimum (or best known) routing solutions with respect to total cost.



(a) egl instances

(b) val instances

Figure 4.3: Comparison of Total Cost (top); Optimum/Best Known Total Cost Results in Green, MA-ABC in Orange, and PSRT in Blue. Comparison of Makespan (bottom); MA-ABC Results in Orange, and PSRT in Blue. Both Objectives are Minimized.

Balance

We use makespan as a proxy for balance; see §4.3.5. The second row of Figure 4.3 plots the makespan of `egl` (a) and `val` (b) instances, produced by MA-ABC and PSRT. Maximizing balance corresponds to minimizing the makespan. The best makespan solutions of MA-ABC are smaller than those of PSRT for all benchmark instances. For `egl` the average reduction is 13.68%, but for `val` the average reduction is much larger, namely 28.27%.

4.4.2 Pareto Efficiency, Spread, and Convergence

An MOEA that produces a diverse set of solutions at the Pareto front is useful for operators by providing a wide set of heuristic solutions to meet design objectives. The approximate Pareto front generated by MA-ABC for the instance `egl-e4-A` is shown in Figure 4.4 with total cost, attractiveness, and balance as axes. The figure shows that the Pareto front is wide and diverse, providing choice for operators.

To further illustrate the range of choice that MA-ABC offers, in Figure 4.5 we use a box-and-whiskers plot for each metric, for each heuristic solution on the Pareto front. In general, these plots show that the boxes for balance and attractiveness are larger than those of total cost. This is not surprising because our algorithm uses only the total cost for the move operator in the local search; this helps to achieve total cost values closer to optimum and maintain the heuristic solutions in the population closer to the optimum total cost (balance and attractiveness considered secondary objectives).

Early convergence is a concern in EC [137]. Convergence can be inferred from the change in the population from generation to generation. A figure that plots of the objective values as a function of generation is also helpful. For example, Figure 4.6

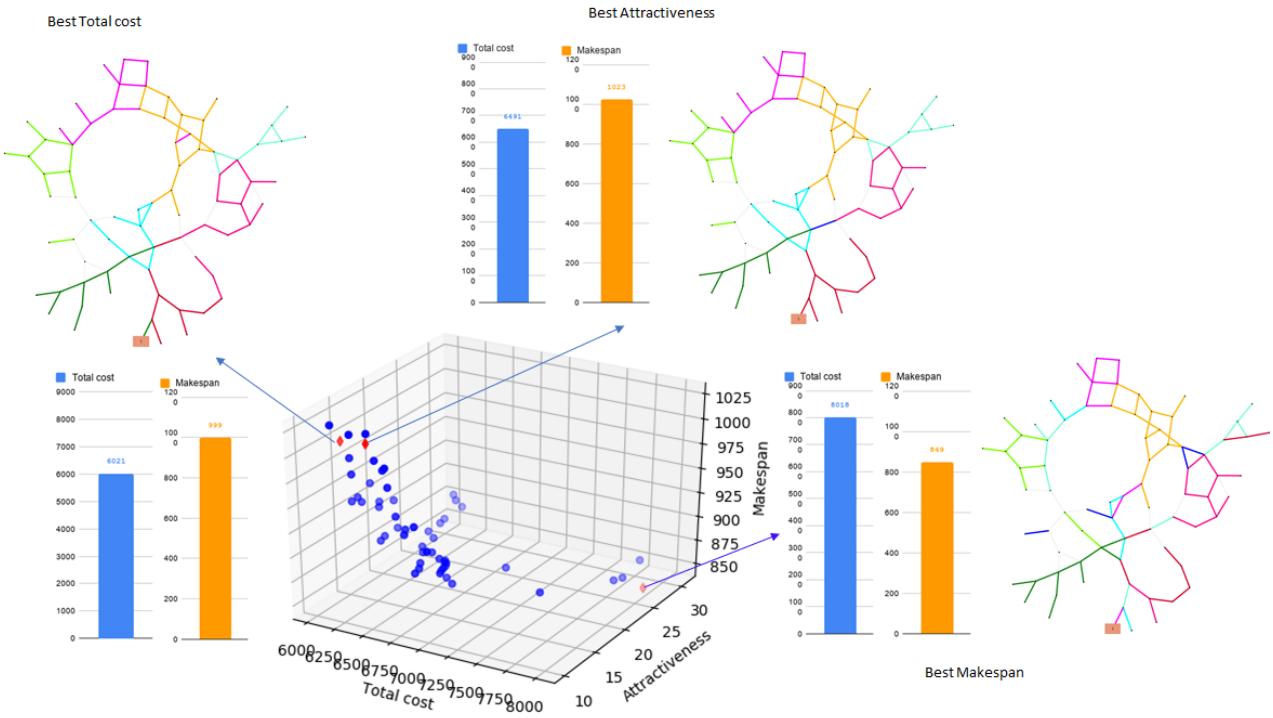
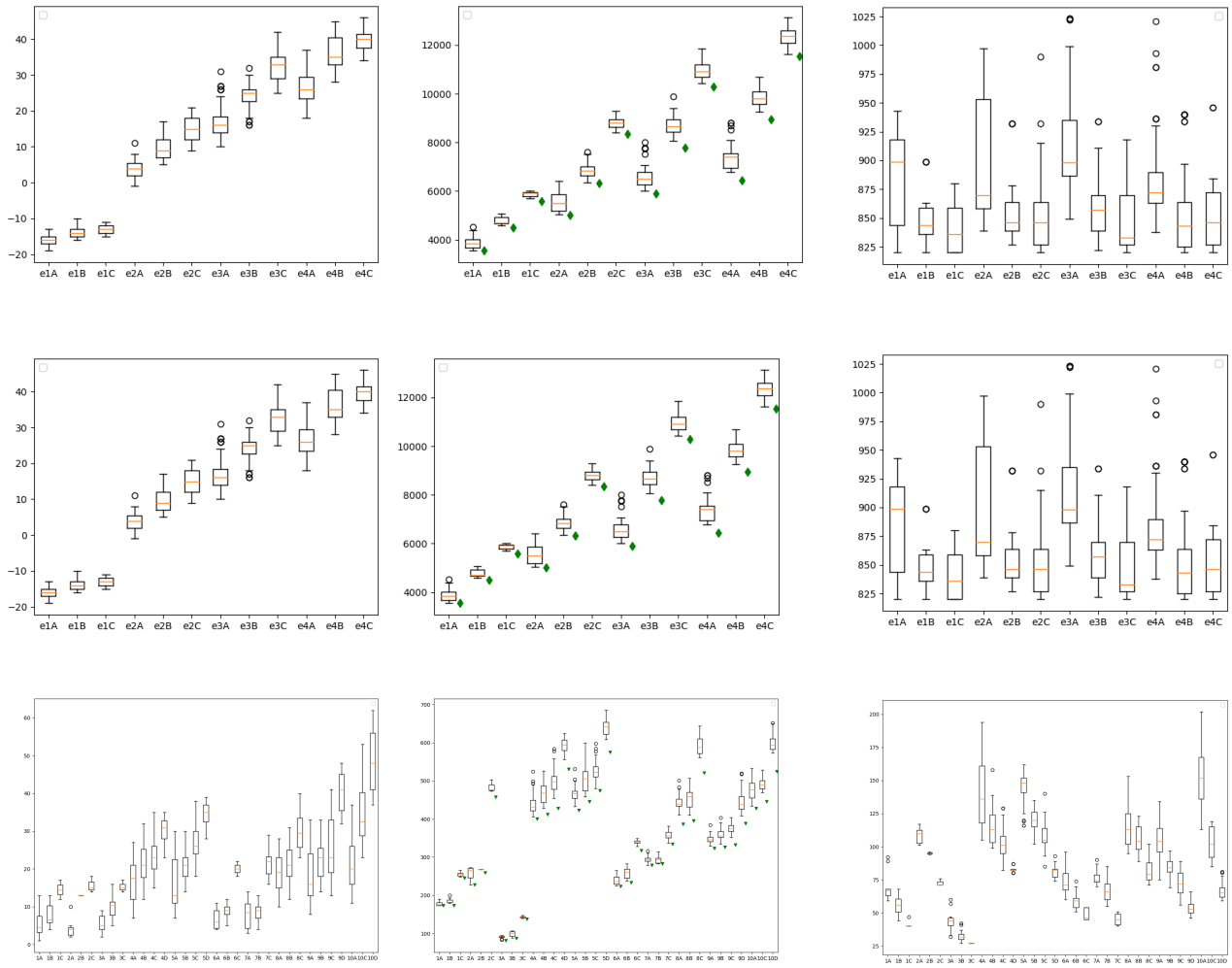


Figure 4.4: Approximate Pareto Front (in Blue) for `egl-e4-A`. A Visualization of the Routing Solution, Total Cost and Makespan are Given for Three Solutions on the Front: From Left to Right, the Best Heuristic Solution with Respect to Total Cost, Attractiveness, and Makespan.

plots the total cost versus generations for the `egl-s4-C` instance. This figure shows that, towards the end of the number of generations, the total cost of the best heuristic solution flattened indicating that it has attained stability and is near convergence. At the same time, the mean fitness varies indicating that the method maintains good diversity.

We also computed the number of unique non-dominated solutions, number of unique dominated solutions, and the number of duplicate solutions in the final iteration of MA-ABC for `egl` and `val`. The number of unique non-dominated solutions ranges from 12-63 for the `egl` instances. For the `val` instances the range is from 7-60,



(a) Attractiveness (b) Total cost (c) Balance

Figure 4.5: Box-and-Whiskers Plots Illustrating Spread of the Heuristic Solutions Found By MA-ABC for (a) Attractiveness, (b) Total Cost, and (c) Balance for the `egl-e` (row 1), `egl-s` (row 2), and `val` (row 3) Instances. The Green Diamond Next to Each Instance of Total Cost Indicates the Optimum (or Best Known) Solution.

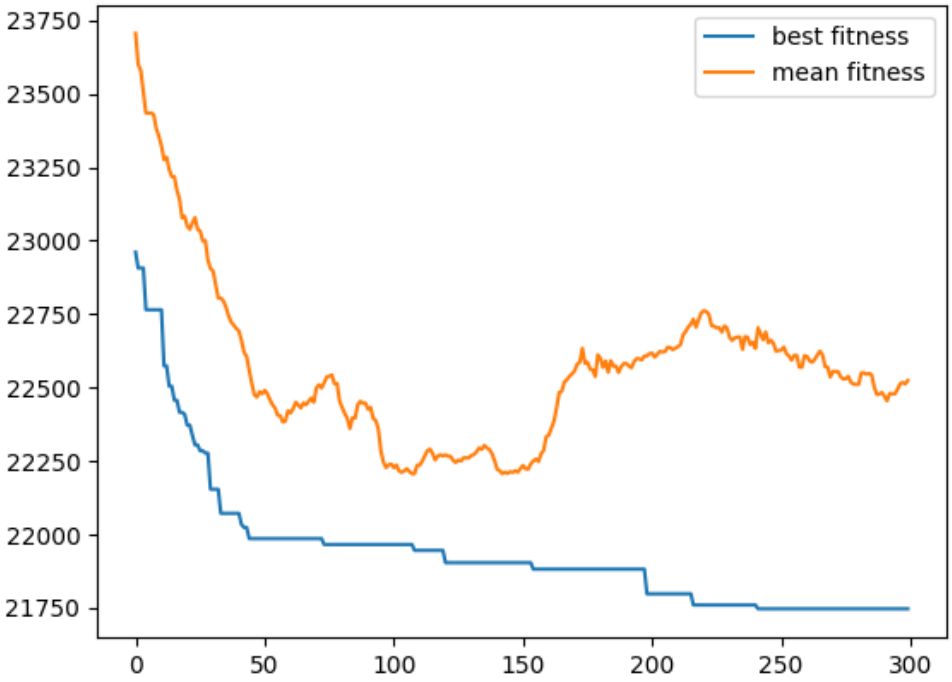


Figure 4.6: Total Cost vs. Generations for `egl-s4-C`.

except for `val1-val3` which range from 2-17 because the graph is smaller. The larger the number of unique non-dominated solutions, the higher the number of choices for an operator; it confirms that diversity is maintained and the early convergence is avoided.

4.4.3 Statistical Analysis

All results presented so far are for a single run of MA-ABC. To measure the repeatability and precision of MA-ABC we use the *coefficient of variation* (CV). It is defined as the ratio of the standard deviation σ over the absolute mean μ. A CV value that is more than 100% is considered highly variant, while a CV less than 100% is considered to show low variance.

Table 4.4 presents μ, σ^2, and CV for each of total cost, attractiveness, and makespan, computed from 30 runs of MA-ABC. The table shows that the CV is greater than 100% for only one instance in `egl-e` and none in `egl-s`. All `val` instances (see [134]), have CV less than 100%; 10 had more than 33% and only four were more than 50%. This suggests that MA-ABC gives consistent results.

Table 4.4: Mean, Variance, and CV for `egl` Instances

Instance	Total cost			Atttractiveness			Makespan		
	μ	σ^2	$\frac{\sigma}{\lvert\mu\rvert}$	μ	σ^2	$\frac{\sigma}{\lvert\mu\rvert}$	μ	σ^2	$\frac{\sigma}{\lvert\mu\rvert}$
egl-e1-A	3589.73	3988.20	1.76	-19.00	0.69	4.37	819.77	1.63	0.16
egl-e1-B	4566.00	618.90	0.54	-16.63	0.52	4.32	815.77	115.77	1.32
egl-e1-C	5725.57	2497.84	0.87	-14.20	0.51	5.03	799.03	425.90	2.58
egl-e2-A	5122.97	1943.76	0.86	-0.30	2.15	488.57	837.63	124.65	1.33
egl-e2-B	6437.53	1848.53	0.67	3.70	1.11	28.52	826.77	16.94	0.50
egl-e2-C	8529.60	3400.25	0.68	10.27	0.41	6.23	818.40	12.52	0.43
egl-e3-A	6048.33	2804.02	0.88	10.70	1.11	9.86	856.70	87.04	1.09
egl-e3-B	8033.47	5288.26	0.91	17.63	1.14	6.05	826.67	29.26	0.65
egl-e3-C	10422.67	8846.16	0.90	25.37	1.27	4.45	819.73	0.75	0.11
egl-e4-A	6716.30	3202.15	0.84	18.13	0.67	4.52	854.30	150.56	1.44
egl-e4-B	9404.47	6731.50	0.87	28.33	1.61	4.48	822.20	4.65	0.26
egl-e4-C	11775.20	16557.13	1.09	35.33	2.16	4.16	819.80	0.37	0.07
egl-s1-A	5168.37	154.17	0.24	-54.67	0.37	1.11	953.83	260.76	1.69
egl-s1-B	6585.53	1697.57	0.63	-52.00	0.69	1.60	922.70	18.08	0.46
egl-s1-C	8570.50	2717.71	0.61	-47.33	0.37	1.28	914.00	20.69	0.50
egl-s2-A	10454.63	7436.65	0.82	11.07	2.89	15.37	1030.53	187.64	1.33
egl-s2-B	13749.00	15646.07	0.91	21.90	4.58	9.77	992.87	61.64	0.79
egl-s2-C	17058.07	18664.41	0.80	31.17	3.25	5.78	978.70	0.22	0.05
egl-s3-A	10684.20	5807.27	0.71	18.60	3.35	9.84	1031.23	258.74	1.56
egl-s3-B	14368.47	14492.81	0.84	33.20	3.96	5.99	989.40	66.04	0.82
egl-s3-C	17979.63	20183.76	0.79	40.00	3.72	4.82	979.00	4.55	0.22
egl-s4-A	13169.83	13510.83	0.88	52.90	9.33	t 5.78	1031.57	36.19	0.58
egl-s4-B	17076.97	20326.93	0.83	66.77	5.56	3.53	1026.80	0.58	0.07
egl-s4-C	21500.30	41947.25	0.95	80.33	4.71	2.70	1018.50	173.02	1.29

4.4.4 *Run Time Performance*

MA-ABC was run on a desktop system with Intel Core i7-4770S CPU @ 3.10 GHz × 8, with 7.7 GiB of memory, running the Ubuntu 18.04 LTS operating system. Figure 4.7 and figure 4.8 shows the running time for `egl` and `val`instances respectively, plotted as a function of instance size, i.e., the number of required tasks. The running time for `egl` instances ranges from 122.72-1291.12 seconds , and from 42.97-262.34 seconds for `val` instances. As the figure shows, the run time is approximately linear. This indicates that MA-ABC appears to scale well and may be appropriate for real-world instances.

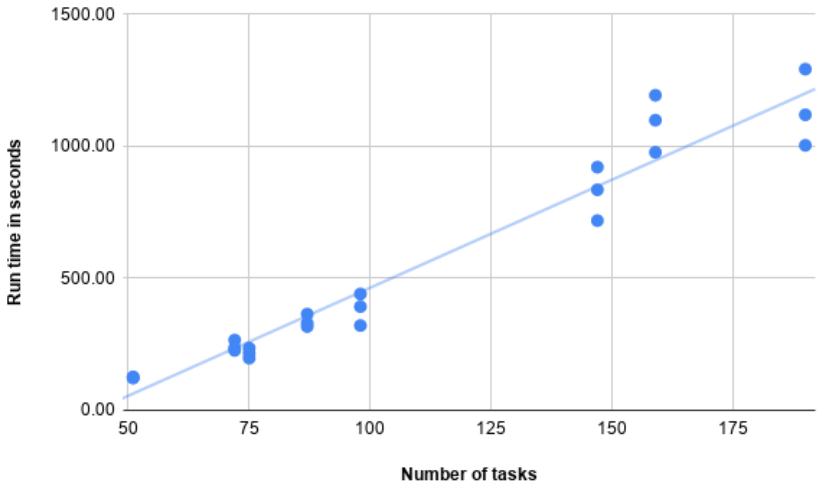


Figure 4.7: Run Time vs. Instance Size for `egl` Instances.

4.5 Discussion

The purpose of the current work is to show that the attractiveness of a heuristic routing solution may be improved using EC algorithms, without compromising total cost. This provides operators of services broad and diverse choice among heuristic solutions to meet their design objectives. Furthermore, MA-ABC produces these

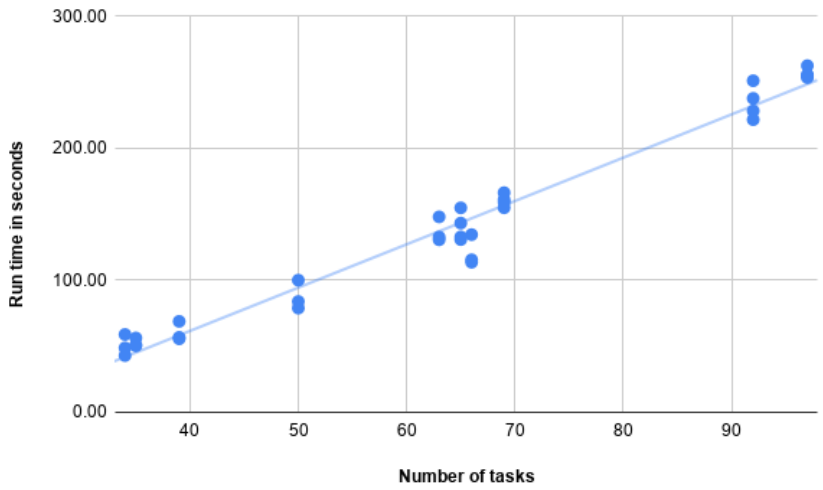


Figure 4.8: Run Time vs. Instance Size for `val` Instances.

heuristic solutions with only one run which is important in case of service interruption which requires a real time response, e.g., rerouting in the event of a vehicle breakdown.

Our new metric of attractiveness could find other application domains, e.g., in node routing problems such as in air route networks, among others.

We want to keep MA-ABC simple to enable easy adoption and reproducibility. Hence no parameter tuning methods are currently integrated into MA-ABC. To enable MA-ABC to be more general so that it can perform well across more types of instances, some form of automatic tuning can be integrated. Recall that the source code is provided at [134].

We also did not use pool update or population restart strategies in order to keep our design simple. The current population size of 100 and number of generations of 300 can be reduced by using some form of pool update strategy which may help in reducing the running time for large-sized instances.

We implemented local search based only on the total cost objective to keep the solutions close to the optimum total cost; this is because total cost is generally con-

sidered of high importance.

We used only one move operator as the local search is expensive and we want to limit the running time. Multi-objective local search and sophisticated move operators such as K-opt may help to improve solution quality in real-world instances in spite of an increase in computation time. The computation time of local search may be reduced by using acceleration mechanisms and by using statistical filters [43].

4.6 Conclusion

In this chapter we introduced attractiveness and route balance objectives that are considered equally important as total cost by operators. We developed MA-ABC, a multi-objective EC algorithm based on NSGA-II for CARP with three objectives: To seek to minimize total route cost and balance, and to maximize route attractiveness. We defined a novel fitness function for attractiveness combining a measure of route overlap and a measure of route contiguity. We evaluated MA-ABC on two benchmark CARP instances. The resulting heuristic solutions are not only visually attractive but also obtain a Pareto front that is diverse with broad choices for the attractiveness and makespan objectives without deviating much from the best total cost objective.

In the next chapter we present SPLICE, an AI- driven, machine learning framework that learns the heuristics and generate close to optimum solution fast at the time of inference.

Chapter 5

LEARNING HEURISTICS FOR ARC ROUTING PROBLEMS

Recently there has been an increased interest in applying machine learning methods to solve combinatorial optimization and routing problems. They enable quick generation of a solution at the time of inference and reduce dependence on human expertise for heuristic solution methods. However, there is little work in arc routing. This chapter presents SPLICE, an AI-driven *machine learning* framework that uses graph neural networks and deep reinforcement learning to learn heuristics for CARP. The design of SPLICE enables it to be used for variants of arc routing and node routing problems without change in network architecture or hyper-parameter tuning.

The chapter is organized as follows. In §5.1, we give a brief introduction to AI-driven approaches in solving routing and combinatorial optimization problems. In §5.2, work related to AI-driven approaches for CARP is reviewed. The SPLICE framework is introduced in §5.3 with the components of each step described. §5.4.4 describes how SPLICE is trained, and then presents the results of applying it to test instances. After some discussion of the framework in §5.5, we summarize our work in §5.6.

5.1 AI Driven Approach for Solving ARPs

Enabled by advances in computer architecture, the performance achieved by artificial intelligence (AI) driven approaches to modelling and solving problems in applications such as image recognition, machine translation, and others, has renewed interest in these methods to tackle problems in combinatorial optimization. Bengio

et al. [73] explore two types of approaches: One is to use the machine learning methods to either augment exact methods for solving the problem as a component, or to assist them in their sub-processes, such as predicting the next efficient branch for a branch-and-bound method [138, 139]. The second is to use machine learning methods as a stand-alone end-to-end solver to generate a heuristic solution faster and closer to the optimum solution at the time of inference. In this approach, the model learns the heuristics by learning the distribution of the parameters of the training instances. It then computes the solution at the inference time assuming that the new instance shares the same distribution.

In their daily operations, operators of services frequently encounter changes in the problem instance. For example in solid waste collection, operators frequently encounter vehicle breakdowns resulting in a change in the availability of vehicles, a change in the streets to service due to events such as local festivals, and a change in demand and cost of servicing the streets due to holidays and seasonal load. Hence there is often a need to run the solver again to obtain a cost effective solution for the changed instance.

As we will see in §5.2, most AI-driven approaches deal with variants of problems represented by a euclidean graph, in which the vertices represent points in the plane, and the edges are assigned lengths equal to the euclidean distance between those points. The graph representation of CARP and its variants is non-euclidian with all parameters associated only with the edges. Hence existing solution methods cannot be applied directly for solving arc routing problems.

This chapter presents SPLICE, an AI-driven *machine learning* (ML) framework to learn heuristics to solve non-euclidean representations of routing problems, and apply it to learn heuristics for CARP. It enables the fast generation of a solution at the time of inference by learning the distribution of the problem parameters from

training instances and, as a result, is also able to adapt to changes in the problem instance. SPLICE learns to generate an appropriate graph embedding using message passing [140], and then learns heuristics through deep Q-learning [81], a form of reinforcement learning. Therefore we do not require labels or an optimum solution to train the network, speeding both the training and inference phases.

Because SPLICE learns a *route-first split-second* heuristic [48, 34], the same architecture can be applied to variants of the arc and node routing problems by using the appropriate splitting procedure. In addition, because SPLICE generates a graph embedding using a Graph Neural Network (GNN), it can learn the structure of the non-euclidean graph extending its applicability to a number of problems in combinatorial optimization.

We run extensive experiments by testing on instances generated with specific distribution settings and report the results of the model's performance. We compare the results of SPLICE with those produced by a path scanning heuristic and by a memetic metaheuristic algorithm. SPLICE finds solutions whose total cost range from 11-30% less in comparison. The heuristics are also shown to generalize well, taking into account the distribution of the topology and the problem parameters such as cost, demand, and the number of edges.

5.2 Related Work

We have already discussed the works related to AI-driven approaches for combinatorial optimization and routing problems in chapter 2 in §2.6.5. All the methods reviewed in §2.6.5 approach to solve as constructive type of heuristics generating only a single solution. There are also works that approach to model as improvement heuristics or local search; see [92, 93, 94, 141, 96] for details.

Those methods are focused only on node routing problems, such as VRP or TSP, where variants of the problem are represented by a euclidean graph. In contrast arc routing problems pose different challenges, notably that the problems are defined on the edges of the graph rather than on its vertices. This requires new methods to transform the input to encode edge features into a suitable representation to learn the non-euclidean graph structure, and to learn heuristics for the problem.

To the best of our knowledge, there are two studies related to learning heuristics for arc routing problems, both dealing with CARP only [142, 143]. Li and Li [142] consider CARP as a *set-to-sequence* problem and use a policy gradient to learn the heuristics. In their network architecture, they first use a *graph convolution network* (GCN) [144] to generate embeddings of the edge set. Then, two pointer networks are used: One outputs a sequence from the edge set embeddings and the second determines the direction of each task in the generated sequence. The demand (state) is updated at every step of the solution generation.

Hong and Liu [143] instead consider CARP as a *sequence-to-sequence* problem and uses only a single *seq-2-seq* based pointer network. Their network architecture has three stages: Presorting, graph embedding generation using `node2vec` [145], and a pointer network to generate the solution. A fourth post-sorting stage is applied to the solution generated in the testing phase. A supervised learning method and the MAENS metaheuristic algorithm [121] are used for generating the solutions.

In both studies, the performance of the model is evaluated by calculating the gap with the results of the MAENS metaheuristic algorithm [121] and with that of the Nazari et al. [82] model for VRP (modified to accept the CARP edge sets as input).

As we will see next, in Splice, we use a message passing *graph neural network* (GNN) [140] to generate the embeddings of the edge set, and we learn heuristics through deep Q-learning [81], which is sample efficient. Our model learns route-

first split-second heuristics and uses splitting procedures that reduce the run-time complexity of handling the demand dynamically, and allows the direction of tasks to be found implicitly from the shortest path. While we focus on CARP, SPLICE can be applied to other variants of arc and node routing problems without any network modification.

We now describe the solution approach used in SPLICE in detail.

5.3 The SPLICE Framework for Learning Heuristics

SPLICE is an AI-driven framework to learn heuristics for CARP that exploits the structure of the graph representation of instances from a distribution $\mathcal{D}$. Figure 5.1 shows a block diagram of the framework incrementally constructing a solution to an instance represented by $G = (V, E)$ from $\mathcal{D}$, with tasks $T \subseteq E$, by combining graph embedding and deep Q-learning. The framework is applied to training instances to learn a model that can be applied to yield solutions for test instances.

At a high level, in each step i of the loop, a message passing graph neural network (GNN) is used to learn the graph embedding, followed by additional layers to learn the functional approximation of a solution through deep Q-learning. The adjacency matrix of the line graph $G_{\mathcal{L}}$, a transformation of G, together with the edge features constitute the input state for the GNN. The output of the GNN in step i is a graph embedding, i.e., action-values of the input state. The action with the maximum embedding corresponds to the selection of a task, which is then appended to the incremental giant tour.

An incremental giant tour is an ordered sequence of i tasks without consideration of vehicle capacity limits. This incremental tour is split into routes for vehicles taking into account their capacity and other constraints of CARP. The meta-algorithm learns

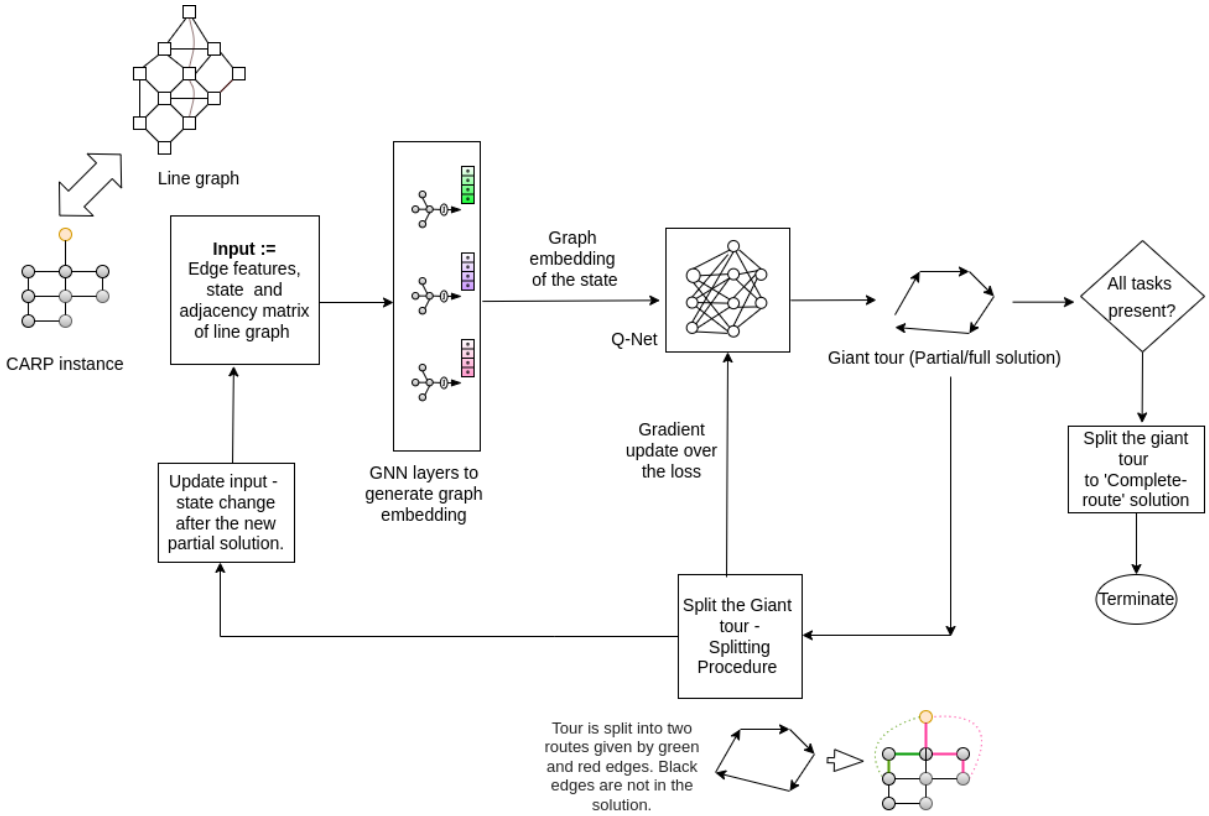


Figure 5.1: Block Diagram of the SPLICE Framework to Learn Heuristics for Arc Routing Problems.

route-first split-second heuristics through deep Q-learning. It estimates a reward for the incremental solution and updates the gradient over the loss. The state of the input is then updated to reflect the results of step i for use in the next step. The loop iterates $1 \leq i \leq |T|$ times, at which point all tasks are included in the giant tour. When the giant tour is split into routes and spliced together, a solution for the CARP instance represented by G is found. This process motivates the name of our framework.

We now describe each block in the SPLICE framework in more detail.

5.3.1 *Input Feature and State Representation*

For node routing problems there are well known methods to learn the graph representation such as random walks [146], or node embedding methods such as `node2vec` [145]. In CARP the problem instance is represented by a weighted undirected graph $G = (V, E)$, together with the set of tasks $T \subseteq E$ each with an associated cost and demand. Because CARP is defined on the edges of G and not its vertices, node embedding methods cannot be applied directly. Therefore, the problem instance is transformed into a line graph to make it more amenable to generate a graph embedding by a graph neural network (GNN).

The *line graph* $G_{\mathcal{L}} = (V_{\mathcal{L}}, E_{\mathcal{L}})$ transformation of G has vertices that correspond to its edge set E, i.e., $V_{\mathcal{L}} = E$. If (u, v) and (v, w) are two edges in E that share a common end-point v, then there is an edge from vertex $(u, v) \in V_{\mathcal{L}}$ to vertex $(v, w) \in V_{\mathcal{L}}$ in the edge set $E_{\mathcal{L}}$ [147, p. 20]. Figure 5.2 shows a graph representation of a CARP instance and its transformation into a line graph.

While the line graph captures the tasks of CARP and their connectivity, it does not capture the other parameters of the problem. Hence to complete the representation of the problem, associated with each edge $e = (u, v) \in E$ is a 7-tuple feature vector $\langle u, v, c(e), d(e), present, first, last \rangle$ that gives the endpoints of the edge, the cost $c(e)$ of traversing e, the demand $d(e)$ of e, and three indicator variables. These indicator variables relate to the incremental giant tour in step i. *present* is true if the task corresponding to e is in the incremental solution and false otherwise. Similarly $first$ and *last* are true if e is the first, respectively last, edge in the incremental giant tour. Initially, the indicator variables of all edges are false. As tasks are added incrementally to the giant tour, the tuples are updated to reflect any state change.

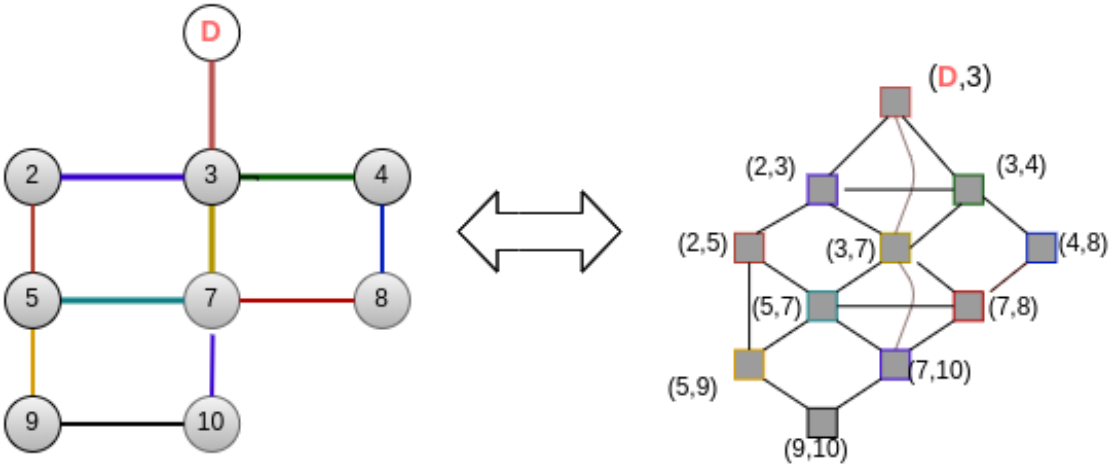


Figure 5.2: A Graph G Representing an Instance of CARP (*left*), and its Line Graph Transformation $G_{\mathcal{L}}$ (*right*).

Each edge in G corresponds to a vertex $G_{\mathcal{L}}$, e.g., the edges $(D, 3)$ and $(3, 4)$ in G correspond to the vertices $(D, 3)$ and $(3, 4)$ in $G_{\mathcal{L}}$. If (u, v) and (v, w) are two edges in G that share a common end-point v, then there is an edge from vertex (u, v) to vertex (v, w) in $G_{\mathcal{L}}$. Continuing the example, because the edges $(D, 3)$ and $(3, 4)$ in G are both incident on vertex 3, there is an edge between vertices $(D, 3)$ and $(3, 4)$ in $G_{\mathcal{L}}$.

5.3.2 Message Passing GNN for Learning the Graph Embedding

In order for subsequent layers to learn the meta-algorithm heuristic function [148], the input state must be transformed into a suitable graph embedding representation. *Graph neural networks* (GNNs) are effective to learn and represent non-euclidean graphs. While many different methods are known, in SPLICE we use a message passing GNN [140] for this purpose.

A message passing GNN learns an embedding by iteratively aggregating the in-

formation of the neighbours of a vertex and then using a non-linear transformation of the aggregated values with itself. In each step i of the loop, there are k layers of message passing in the GNN. In general, the embedding $h_u^{i,k}$ of vertex u in step i after k layers of message passing is given by [149]:

$$h_u^{i,j} = \text{Update}^{j-1}\left(h_u^{i,j-1}, \text{Aggregate}_{v\in N(u)}^{j-1}(h_v^{i,j-1})\right), j = 0, \ldots, k-1 \tag{5.1}$$

where $N(u)$ are the neighbors of u. The Update$(\cdot)$ and Aggregate$(\cdot)$ functions are any differentiable functions that can be implemented in a neural network. Usually a non-linear function is used for the Update$(\cdot)$ function. The aggregation function should be permutation invariant to remove dependence on the input order. Each layer of the GNN extends the learning of neighbors through aggregation.

We now tailor Equation (5.1) in the context of CARP in the SPLICE framework. The vertices of $G_{\mathcal{L}}$ correspond to tasks of the CARP instance, and $h_u^{i,k}$ represents the input features for the edge u at step i after k iterations of GNN message passing. Here, $h_u^{0,0}$ corresponds to the 7-tuple feature vector initialized for each edge. The adjacency matrix of the line graph $G_{\mathcal{L}}$ is used to determine the neighbourhood $N(u)$ of u, for aggregation of messages in the GNN.

Initially the giant tour $\mathcal{G}_0 = \langle\rangle$ is empty, and $\mathcal{T}_0 = T$, the set of all tasks $T \subseteq E$ in the CARP instance. The neighbours $N(u)$ of a task u are the tasks v reachable from u, i.e., v such that $(u, v) \in G_{\mathcal{L}}$. The *rectified linear unit* (ReLU) function [150] is used as the Update$(\cdot)$ function and Aggregate$(\cdot)$ is a summation of the embeddings. We do not normalize after aggregation to avoid any loss of information. Rewriting Equation (5.1) in the context of CARP in SPLICE to update the graph embedding for a task u in the ith step is given by:

$$h_u^{i,j} = ReLU\left(\theta_1 h_u^{i,j-1} + \theta_2 \sum_{v\in N(u)} h_u^{i,j-1}\right), j = 0, \ldots, k-1. \tag{5.2}$$

where θ_1 and θ_2 are the model parameters of the embedding that are learned. The number k of iterations used by the GNN to compute the embedding is bounded by the half the graph diameter because this encompasses all vertices of the graph; more iterations leads to over-smoothing [149].

The subsequent layers use functional approximation to estimate the evaluation function $Q(\mathcal{G}_{i-1}, u, \Theta)$ to measure the quality of vertex u with respect to the incremental giant tour $\mathcal{G}_{i-1}$. These are essentially the action-value pairs for the current input state:

$$Q(\mathcal{G}_{i-1}, u, \Theta) = ReLU(\theta_3 \; h_u^{i,k}) \tag{5.3}$$

An ϵ-greedy strategy [151] is used to aid exploration during the training phase. It selects the action with the maximum embedding value with probability $1 - \epsilon$. This corresponds to the selection of a task u of highest quality:

$$u = \arg\max_{v \in \mathcal{T}_{i-1}} Q(\mathcal{G}_{i-1}, v, \Theta) \tag{5.4}$$

With probability ϵ, the strategy selects a random action.

This vertex u selected is appended to the incremental giant tour, i.e., $\mathcal{G}_i = \langle \mathcal{G}_{i-1}, u \rangle$, an ordered list of vertices. In addition u is removed from the set of tasks remaining $\mathcal{T}_i = \mathcal{T}_{i-1} \setminus u$.

5.3.3 Deep Q-learning for Learning the Heuristics

For each task u, the model parameters $\Theta = \{\theta_1, \theta_2, \theta_3\}$ of the quality evaluation function $Q(\mathcal{G}_i, u, \Theta)$ are learned using deep Q-learning [152], a form of reinforcement learning. Q-learning [153] is chosen because it is sample efficient and is more stable than other methods [151]. SPLICE combines n-step fitted Q-iteration [154] and deep Q-learning [81] to learn the parameters.

The target value y is the sum of the reward and the maximum action-value of the next state scaled by discount factor γ [151], in step i:

$$y = \text{reward}(\mathcal{G}_i) + \gamma max_{u'} Q(\mathcal{G}_i, u', \Theta). \tag{5.5}$$

The mean squared loss is defined as the square of the difference between the target value y and the current state value given by:

$$loss = (y - Q(\mathcal{G}_i, u, \Theta))^2. \tag{5.6}$$

Fitted Q-iteration uses *experience replay*, performing batch updates from a replay memory at the gradient steps. This helps to mitigate the correlation issues faced by performing updates of loss from sequential samples [81].

In step i, the giant tour $\mathcal{G}_i$ is a sequence of i tasks. That is, we have generated a tour ("route-first") and now we must split the tour into a set of closed routes ("split-second"). Then a reward estimated for the solution, and the loss propagated; these are discussed next.

Splitting the Giant Tour into Routes and Reward Estimation

Methods for splitting a giant tour into routes are well studied in arc routing [34, 38, 118, 39]. A giant tour must be split into individual routes respecting the capacity limit C of the vehicles and other CARP constraints.

In SPLICE, the splitting procedure is based on the route-first split-second heuristic proposed by Ulusoy [34]. First, an auxiliary *directed acyclic graph* (DAG), is constructed based on the giant tour $\mathcal{G}_i$. Once the DAG has been constructed, it is split into individual routes by finding the shortest path for each task in tour order from the depot and back, subject to the vehicle capacity and other constraints.

Figure 5.3a illustrates the splitting procedure for a sample CARP instance with five tasks $T = \{a, b, c, d, e\}$ with the demand for each indicated inside a square; recall

non-required edges have zero demand. Each edge also has an associated traversal cost. Figure 5.3b shows a giant tour $\mathcal{G}_5 = \langle b, c, a, e, d \rangle$ containing all tasks. Figure 5.3c shows, for each task in the sequence, the cost of the shortest path from the depot to the task returning back to the depot including dead-heading, in addition to the cost of the shortest path between tasks via dead-heading, if required. In this example, task c is adjacent to task b hence no dead-heading is required to service task b after servicing task c.

From this graph, the auxiliary DAG is constructed. While $\mathcal{G}_i$ has i vertices, the DAG has one additional vertex for the depot. The vertices in the DAG correspond to the cost of servicing each subsequence t_k, $k \leq j, j = 1, \ldots, i$ of tasks in $\mathcal{G}_i = \langle t_1, t_2, \ldots, t_i \rangle$, while the arcs are labelled by the feasible subsequences of tasks. Feasibility is dictated by sequence order in the giant tour. All routes in CARP start and end at the depot, hence the cost of the first vertex is zero.

Suppose that the vehicle capacity $C = 7$. Because the first task in $\mathcal{G}_5$ is task b, all arcs out of the depot must start by servicing task b. As Figure 5.3c shows, servicing task b only has a cost of 12, hence the auxiliary graph has a vertex with cost 12 for servicing task b. However the demand of task b is only 3, leaving capacity $C = 7 - 3 = 4$ with the possibility to service another task. Only task c is feasible. The route bc shown as arc bc with cost 16 in the DAG is obtained from Figure 5.3c by including the cost of tasks b and c, the cost of the deadheading between them, and cost of dead-heading from the depot to b, and from c to the depot: $4 + 2 + 0 + 2 + 8 = 16$. The route bc cannot be extended because the remaining capacity is $C = 4 - 2 = 2$ and the demand 4 of the next task in the sequence, a, exceeds the capacity.

Each vertex in the auxiliary graph is extended in all feasible ways subject to capacity constraints. The last vertex in the DAG is then the total cost of servicing all tasks in $\mathcal{G}_i$. After the construction of the DAG is complete, the shortest path

from the depot to the last vertex represents the minimum cost of servicing all tasks. Figure 5.3e shows a shortest path with three arcs in bold with a total cost of 48 for G_5. Each arc corresponds to a closed route that includes the tasks serviced by that route. In this example, there are three routes from the depot with one servicing task b followed by c, a second servicing only task a, and a third servicing task e followed by d. These tasks are spliced together to give a solution for $\mathcal{G}_5$.

After splitting and computing the *total cost* for $\mathcal{G}_i$ at step i, we compute the reward for the current action of appending task u to $\mathcal{G}_i$ as:

$$reward(\mathcal{G}_i) = -\left(total\_cost(\mathcal{G}_i) - total\_cost(\mathcal{G}_{i-1})\right), \tag{5.7}$$

where the $total\_cost(\mathcal{G}_0) = 0$.

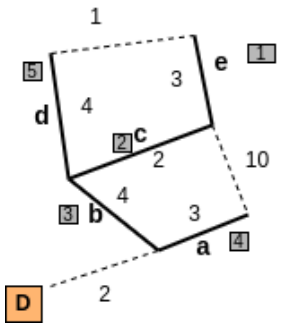
The input features of all edges are then updated for the next step of the framework.

The run-time and space complexity of the splitting procedure is $O(|T|^2)$, where $|T|$ is the number of tasks. The splitting procedure described can be performed without constructing the DAG explicitly as done by Lacomme et al. [48]. This reduces the space complexity to $O(|T|)$.

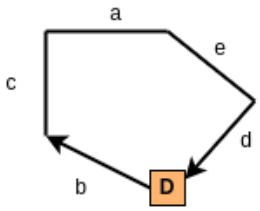
A giant tour of minimum cost that is split into a routing solution does not necessarily result in a solution to CARP of minimum total cost. This is one way in which Splice differs from the learning methods for solving the TSP. Splice does not learn to construct the tour of minimum cost but a tour that yields minimum total cost upon optimal splitting.

5.3.4 The Splice Q-learning Algorithm

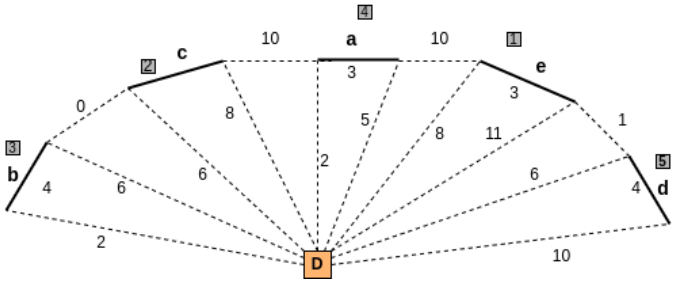
The Splice framework is now summarized using pseudocode in Algorithm 4. In line 3, the algorithm iterates for N episodes to train the network. Each episode trains



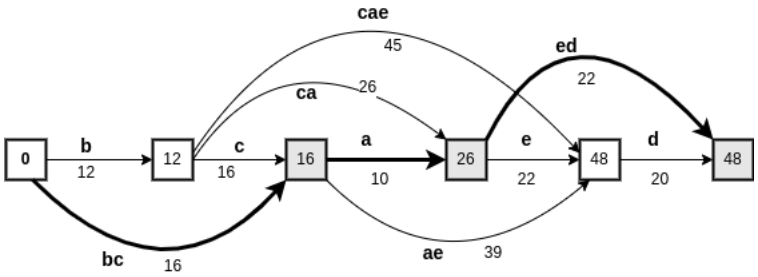
(a) CARP instance.



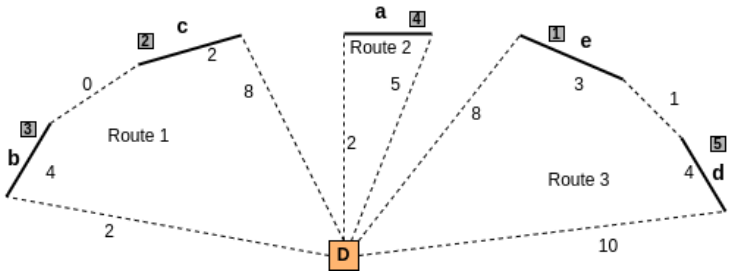
(b) Giant tour $\mathcal{G}_5$.



(c) Prelude to DAG construction.



(d) Auxiliary DAG taking into account capacity.



(e) Routes resulting from splitting the giant tour.

Figure 5.3: An Illustration of the Splitting Procedure for CARP.

using a single graph.

In line 4, a graph G representation of a CARP instance is selected at random from a training data set with distribution $\mathcal{D}$. G is transformed into a line graph $G_{\mathcal{L}}$, and the input features for each task are collected.

The outer loop of the algorithm in line 9, which corresponds to the outer loop in the block diagram in Figure 5.1, iterates for $|T|$ steps. In each step, a message passing GNN generates a graph embedding (line 10). The embedding is used as input to the Q-Net. Using the ϵ-greedy strategy, a task u maximizing the embedding is selected (line 12) and added into the giant tour. Then after splitting the giant tour, a reward is computed for the solution (line 18) and stored in the experience replay memory (line 19). In fitted Q-iteration, loss is not performed on sequential samples but instead performed on batches at a gradient step to avoid correlation (line 22).

The state is then updated (starting at line 23) for the next step of the outer loop. After training, the algorithm returns the model Θ which can then be used to apply the heuristics learned to obtain a solution to a new CARP instance from distribution $\mathcal{D}$ in a similar manner.

5.4 Experimentation and Results

In this section we describe how instances of CARP are generated for training and testing in the SPLICE framework, and the selection and tuning of hyper-parameters. Then the experimental set-up is described, followed by the results obtained.

Algorithm 4: SPLICE Q-learning Algorithm

Input : CARP instance $G = (V, E)$, $T \subseteq E$, and $c(t)$ and $d(t)$ for each $e \in E$
Output: SPLICE model, Θ

Initialize experience replay memory M capacity
Set hyper-parameters as in Table 5.1

/* Iterate for N episodes */
for *episode* $\leftarrow 1$ **to** N **do**
  /* Select a graph representation of a CARP instance */
  Select a graph G at random for training from distribution $\mathcal{D}$
  Transform G into its corresponding line graph $G_{\mathcal{L}}$
  Initialize input state: Adjacency matrix of $G_{\mathcal{L}}$ and feature vector for each edge
  Initialize incomplete tasks $\mathcal{T}_0 \leftarrow T$
  Initialize giant tour $\mathcal{G}_0 \leftarrow \langle\rangle$
  /* Use *one*-step Q-learning to learn the parameters */
  for $i \leftarrow 1$ **to** $|T|$ **do**
    Use message passing to learn the graph embedding
    /* Use ϵ-greedy strategy to select next task, u */
    Choose probability p uniformly at random
    if $p < \epsilon$ **then**
      $u \leftarrow$ choose task uniformly at random from T_{i-1}
    else
      $u \leftarrow \arg\max_{v \in \mathcal{T}_{i-1}} Q(\mathcal{G}_{i-1}, v, \Theta)$
    /* Append u to incremental giant tour; update task set */
    $\mathcal{G}_i \leftarrow \langle \mathcal{G}_{i-1}, u \rangle$
    $\mathcal{T}_i \leftarrow \mathcal{T}_{i-1} \setminus u$
    Split $\mathcal{G}_i$ in routes and compute reward($\mathcal{G}_i$)
    Add tuple into replay memory M
    /* Use fitted Q-iteration with experience replay */
    if $i \bmod \mathcal{N} == 0$ **then**
      Obtain a random sample batch from replay memory, $B \overset{iid}{\sim} M$
      Perform gradient update of Θ over loss (Equation (5.6)) for the batch B
    /* Update state for next iteration of outer loop */
    update feature vectors for each edge
return Θ

5.4.1 Sample Instances for Training and Testing

Arc routing problems are defined on road networks that are commonly grid-shaped. Hence the graph representation of the road network has a grid topology. We create a distribution $\mathcal{D}$ of grid topologies, deleting at most k edges at random, to create variations in topology and in vertex degree. These were then augmented with costs and demands to generate a CARP instance.

To generate a training instance, first graph $G = (V, E)$ in the form of an $m \times n$ grid of vertices with $2mn - m - n$ edges is generated; vertex 1 is designated as the depot. At most k edges are deleted at random. As edges are deleted, if a vertex becomes isolated it is deleted from the instance; if the depot becomes isolated the instance is abandoned and generation is restarted. Then, costs and demands are assigned to the edges from a distribution according to the specific arc routing problem. For CARP, traversal costs are assigned to all edges, with non-zero demand assigned to $t \in T$ and zero demand assigned to $t \in E \setminus T$. The Capacitated Chinese Postman Problem (CCPP) is a variant of CARP in which $T = E$. The vehicle capacity C does not vary across instances.

For training Splice all instances created start as a 5×4 grid, which have $k = 2$ deleted at random yielding 29 edges. The cost of each edge is varied uniformly between 4 to 6; the cost is not varied much to ensure that the topology of the graph remains planar. The demand of each is varied uniformly between 0 and 5 for CARP (and between 1 and 6 for CCPP). Figure 5.4 shows one of the instances generated for CARP by this method.

A total of 20,000 distinct instances with 29 edges were generated. These are divided into training, validation, and testing data sets with a ratio of 80:10:10, respectively.

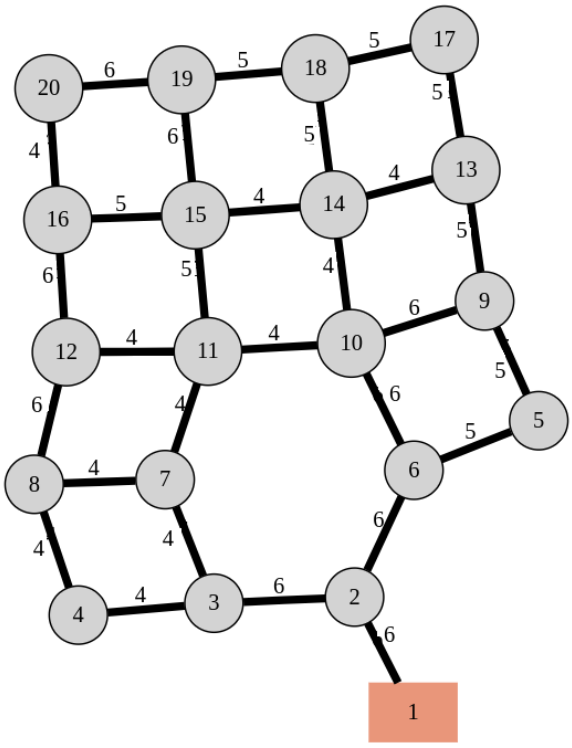


Figure 5.4: Sample Training Instance Generated for CARP.

5.4.2 Hyper-Parameter Tuning and Selection

The typical hyper-parameter settings used in the many general reinforcement learning for control tasks are found to fit our experiments without the need for extended tuning. From monitoring the metrics of loss, the running average of the route cost, and observing their stability, 10,000 training episodes were found to suffice. An embedding dimension of five was found to be adequate for models of CARP instances with both 15 and 29 edges with a 7-tuple feature vector. Extending the embedding dimension beyond five does not provide any advantage. The replay buffer capacity plays a role in how long the transitions are kept and used for learning [155, 156]. We found a capacity of 4000 fit our architecture, input problem size, and the number of episodes. The learning rates, and the learning decay rate for the Adam optimizer

[157], were set to the standard values typically used and did not require any further tuning. The settings used are summarized in Table 5.1.

Table 5.1: Hyper-parameter Settings Used in SPLICE

Hyper-parameter	**Value**
Number of episodes	10,000
Embedding dimension	5
Discount factor, γ	0.9
Number of layers	5
Learning rate, α	0.005
Learning decay rate, η	0.99998
ϵ in ϵ-greedy strategy	0.1
ϵ decay rate	0.0006
n-step reward update, $\mathcal{N}$	2
Replay buffer memory capacity	4000
Batch size	16

The Pytorch [158] framework was used for creating the neural network layers in the `python` programming language. The `Networkx` [159] package was used for generating instances, and for line graph transformations.

5.4.3 Experiment Set-Up

We performed three experiments. The first one is for the CCPP variant of CARP in which all the edges are required, i.e., $T = E$. The second experiment is performed on CARP. In both these experiments, the vehicle capacity $C = 16$. All instances created had 29 edges, with demands as described in §5.4.1.

The third experiment is to evaluate the generalization capability of the model. In this experiment we train the SPLICE model on CARP instances with 15 edges but test it on instances with 29 edges. For both the 15- and 29-edge instances, we set the vehicle capacity to $C = 8$.

We compare the results of SPLICE on test instances with those produced by the *path scanning* (PS) heuristic [29], and the *memetic algorithm metaheuristic* (MA) [48]. The hyper-parameters of MA are set to 300 generations, a population size of 100, and the probability of cross-over and mutation to be certainty.

5.4.4 Results and Analysis

After training, inference was performed on test instances without learning or updating the gradients. In general, the total cost achieved by the memetic algorithm is less than that achieved by path scanning; this is not surprising as metaheuristics generally outperform heuristics [24].

For CCPP, the total cost achieved by SPLICE is less than path scanning and the memetic algorithm for ten instances selected at random from the 2,000 instances run. As Figure 5.5 shows, the average percentage of reduction in total cost is 29.03% when compared to path scanning and 23.43% when compared to the memetic algorithm.

For CARP, the results are similar to those of CCPP except that, for two of the ten instances selected at random from 2,000 instances run, the memetic algorithm has lower total cost. For all other instances, SPLICE has lower total cost, and has lower total cost than all PS instances. On average the percent reduction in total cost is 15.60% over path scanning and 10.75% lower than the memetic algorithm. Figure 5.6 tabulates and plots results for these ten CARP instances.

To check the generalization capability of SPLICE, we train the model on CARP

Instance	S	PS	%↓	MA	%↓
29E_19160	228	304	25.00	296	22.97
29E_12689	231	329	29.79	305	24.26
29E_8479	234	333	29.73	312	25.00
29E_17805	238	326	26.99	308	22.73
29E_12410	228	366	37.70	330	30.91
29E_5463	241	365	33.97	316	23.73
29E_4942	249	387	35.66	343	27.41
29E_9657	224	270	17.04	254	11.81
29E_12271	239	341	29.91	329	27.36
29E_11712	249	330	24.55	304	18.09
		Avg:	29.03	*Avg:*	23.43

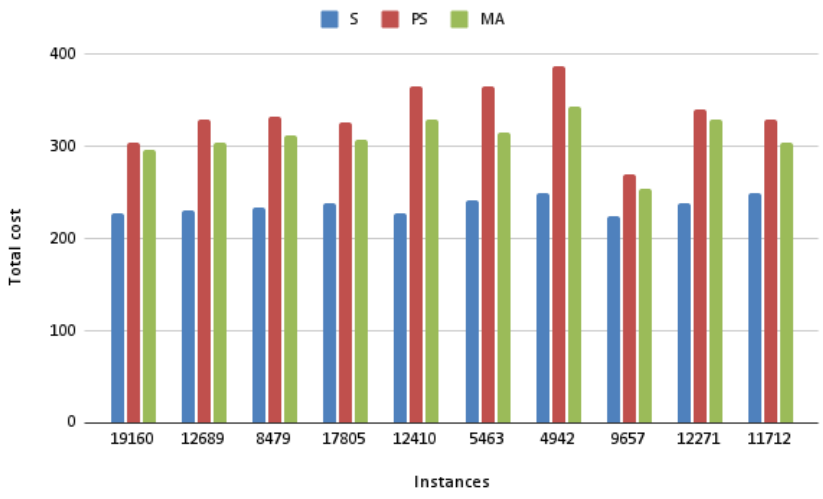


Figure 5.5: Total Cost for CCPP on Instances with 29 Edges and $C = 16$ for Splice (S), the Path Scanning heuristic (PS), and the Memetic Algorithm Metaheuristic (MA). The Percentage of Reduction in Total Cost that Splice Achieves Over PS and MA Is Also Given. The Figure on the Right Plots the Results in the Table on the Left for Each Instance.

Instance	S	PS	%↓	MA	%↓
29E_2185	219	233	6.01	216	-1.39
29E_5093	229	282	18.79	264	13.26
29E_155	238	287	17.07	282	15.60
29E_9689	234	287	18.47	285	17.89
29E_7372	228	279	18.28	255	10.59
29E_15979	235	266	11.65	253	7.11
29E_14782	226	288	21.53	276	18.12
29E_11832	228	313	27.16	285	20.00
29E_776	217	232	6.47	208	-4.33
29E_4620	211	236	10.59	236	10.59
		Avg:	15.60	*Avg:*	10.75

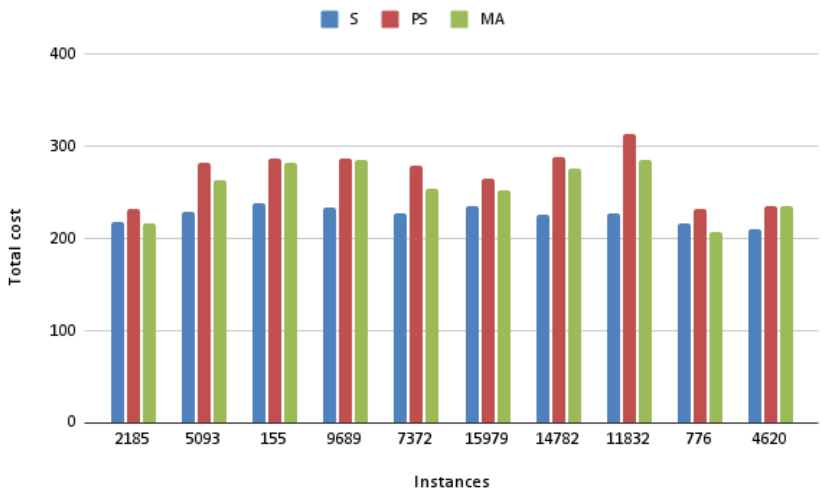


Figure 5.6: Total Cost for CARP on Instances with 29 Edges and $C = 16$ for SPLICE (S), the Path Scanning heuristic (PS), and the Memetic Algorithm Metaheuristic (MA). The Percentage of Reduction in Total Cost that SPLICE Achieves Over PS and MA Is Also Given. The Figure on the Right Plots the Results in the Table on the Left for Each Instance.

instances with 15 edges and with capacity $C = 8$. Then the model is tested on CARP instances with 29 edges and with the same vehicle capacity. As Figure 5.7 shows, SPLICE generalizes well; the ten instances were chosen uniformly at random from the 2,000 test instances. SPLICE shows a 46.60% and 39.62% lower total cost on average compared to path scanning and the memetic algorithm, respectively.

The minimum, maximum, and median percentage reductions for CCPP are 11.81%, 30.91%, and 24.00% over MA, respectively, and 17.04%, 37.70%, and 29.76% over PS, respectively. For CARP they are -4.33%, 20.00%, and 11.93% over MA, respectively, and 6.01%, 27.16%, and 17.68% for PS, respectively. For the generalization experiment the minimum, maximum, and median percentage reduction are 24.62%, 45.31%, and 41.86% over MA, respectively, and 35.36%, 51.56%, and 48.35% over PS, respectively. Box plots illustrating the percentage reductions for PS and MA are given in Figure 5.8. The box plots indicate that the performance of SPLICE does not vary much.

The Run Time of Splice

Training SPLICE on 10,000 episodes where each instance has 29 edges takes less than 20 minutes of CPU time. Once trained, an inference on an instance of the same size takes less than a second. Table 5.2 shows the CPU time used for inference on a CCPP instance, a CARP instance, and a smaller CARP instance used for training in the generalization experiment. The inference time for SPLICE for all the three tests (experiments) is around 0.8 seconds.

As Table 5.5 shows, we obtained 23.43% reduction in total cost for CCPP instances using SPLICE but the running time was a fraction of a second (0.86 seconds versus over 53 seconds for MA). Similarly we obtained an average of 10.75% reduction in total cost for CARP instances with around the same inference time only (0.82 seconds

Instance	S	PS	%↓	MA	%↓
29E_2185	241	440	45.23	374	35.56
29E_5093	253	490	48.37	429	41.03
29E_155	266	524	49.24	463	42.55
29E_9689	268	525	48.95	490	45.31
29E_7372	250	445	43.82	425	41.18
29E_15979	259	460	43.70	412	37.14
29E_14782	264	545	51.56	460	42.61
29E_11832	260	536	51.49	461	43.60
29E_776	245	379	35.36	325	24.62
29E_4620	233	451	48.34	406	42.61
		Avg:	46.60	*Avg:*	39.62

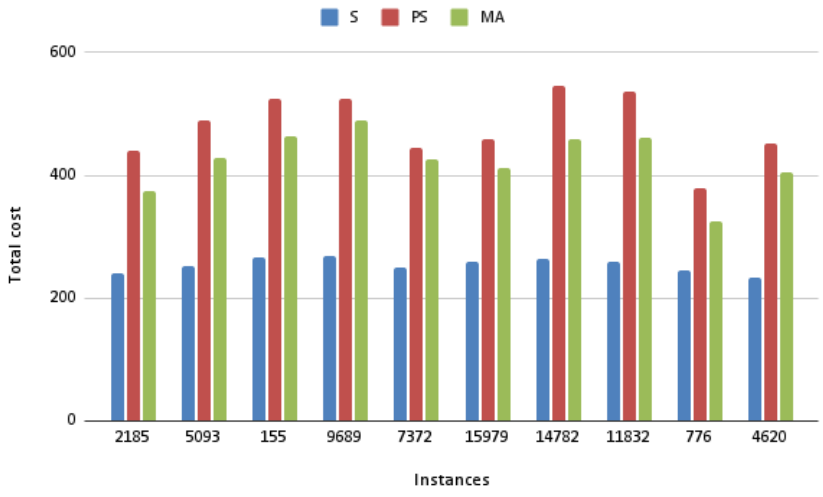


Figure 5.7: Generalization Results. The Total Cost for CARP on Instances Trained on 15 Edges to Instances with 29 Edges and $C = 8$ for Splice (S), the Path Scanning heuristic (PS), and the Memetic Algorithm Metaheuristic (MA). The Percentage of Reduction in Total Cost that Splice Achieves Over PS and MA Is Also Given. The Figure on the Right Plots the Results in the Table on the Left for Each Instance.

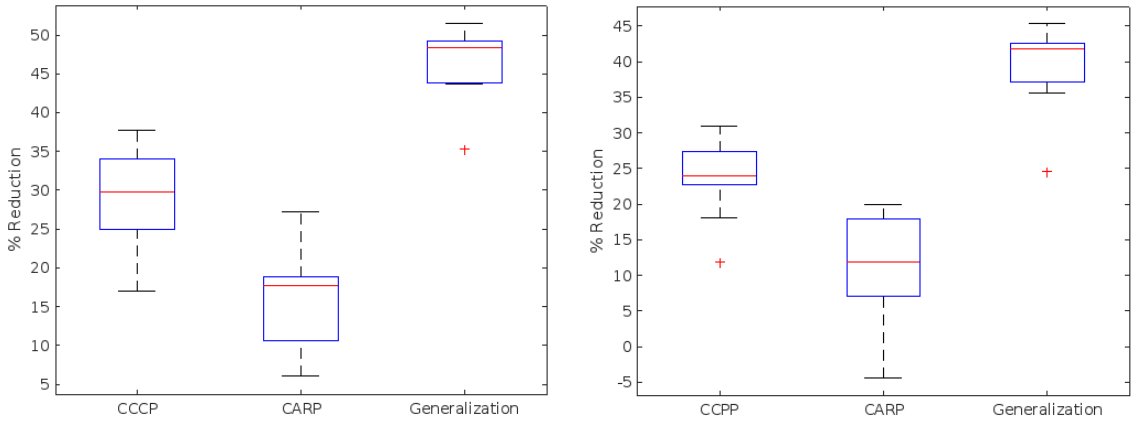


Figure 5.8: Box Plot Indicating the Percentage of Reduction in Total Cost for the CCPP, CARP, and Generalization Experiments Compared to Path Scanning (left) and the Memetic Algorithm (right).

Table 5.2: Average CPU Time in Seconds Taken for Running 10 Instances at Inference

Instance	**Splice**	**PS**	**MA**
CCPP, 29 edges, capacity $C = 16$	0.86	0.37	53.16
CARP, 29 edges, capacity $C = 16$	0.82	0.26	41.70
CARP, 29 edges, capacity $C = 8$	0.80	0.28	57.33

versus 41.7 seconds for MA). For the generalization experiment, we obtained around a 39.62% reduction in total cost with around the same inference time (0.8 seconds versus 57.3 seconds for MA).

These run time results demonstrates the utility of Splice for inference, and also for changes in operational instances.

5.5 Discussion

From the results presented in §5.4.4, the SPLICE framework is effective in learning the distribution $\mathcal{D}$ from which instances are drawn, and yields improved total cost, performing better than metaheuristics (MA) and path scanning heuristics, in a reasonable running time. The memetic algorithm is a hybrid evolutionary algorithm, combining evolutionary computing with local search. Hence it can also be thought of an algorithm that is learning to route using evolutionary algorithms but is unable to leverage its past learning in a data-driven manner. The memetic algorithm [48] also uses the route-first split-second splitting procedure. However, because SPLICE is able to learn the parameters of the distribution $\mathcal{D}$ it is able to give better results. The generalization result in Figure 5.7 shows that SPLICE can generalize well and can scale well for a change in the problem size.

An advantage of machine learning over metaheuristic methods is fast inference. As the results on total cost show, SPLICE performs better than MA and PS because of its capability to learn the distribution. At the same time the running time for SPLICE is 50 to 72 times faster. The model may be useful for and of potential interest when an operational environment experiences changes in problem parameters.

The SPLICE framework is designed for optimizing one objective, the total cost. Since SPLICE is effective in learning the distribution $\mathcal{D}$ in a data-driven manner, we obtain from 10% to 40% improvement in total cost over the metaheuristic memetic algorithm. However since it is trained only for the objective of minimum total cost, the generated routes are not considered attractive (i.e., the routes are not compact nor contiguous); see Figure 5.9 for an example of the solutions found by each of these methods.

The SPLICE framework may be applied to variants of both node and arc routing

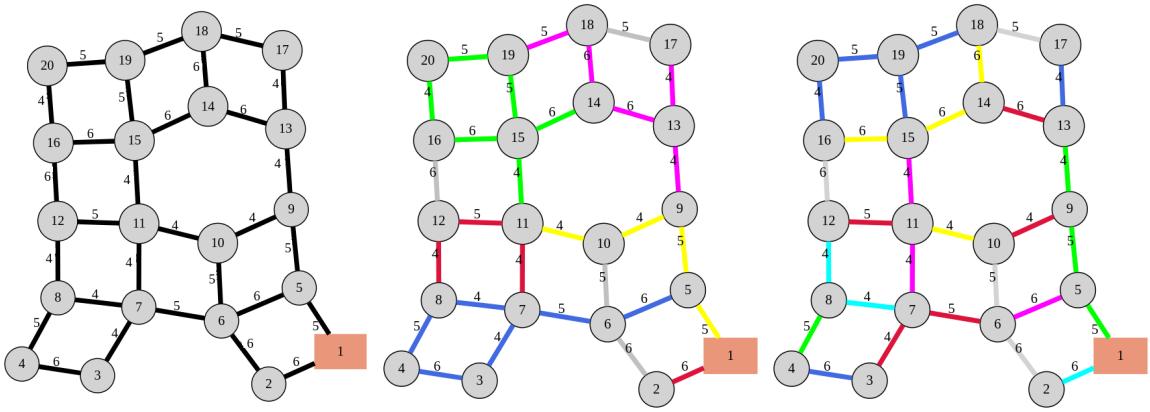


Figure 5.9: Route Attractiveness in the Generated Solution. Problem Instance (left), Routes Found by MA (center), and Routes Found by SPLICE (right).

The gray edges are not required; that is, they are only used for deadheading.

problems. Variants such as the multi-depot arc routing problem, the split-delivery capacitated arc routing problem, and others, can be accommodated using a splitting procedure that corresponds to the variant [7, 160]. The SPLICE network architecture requires no other changes. This simplifies the incorporation of new innovations in machine learning methods into the architecture.

5.6 Conclusion

In this chapter, we developed a deep Q-learning based framework, SPLICE, that learns heuristics for arc routing and other problems. It uses a message passing based graph neural network to learn the graph structure representation of the problem and, from the graph embedding, learns the heuristics using Q-learning.

SPLICE yields improved results over a metaheuristic memetic algorithm with an average 10% to 24% lower total cost with the inference running in around 0.8 seconds.

SPLICE also generalizes well. These results demonstrate that SPLICE has the potential to be useful for real operational requirements.

SPLICE can be applied to different variants of arc routing because it learns route-first split-second heuristics. It can also be applied to node routing problems and its variants.

In the next chapter we summarize the results of this dissertation and provide research directions and future plans.

Chapter 6

CONCLUSION AND FUTURE WORKS

We started this dissertation with the motivation being the lack of adequate methods available to the operators using arc routing problems in their businesses to meet their requirements. These include: i) Managing vehicle breakdown disruptions ii) incorporating additional features and objectives that are equally important as optimum cost, such as visual attractiveness and route balance in their route plan, and iii) methods to generate closer to optimum solutions *quickly* for changes that arise periodically in the problem specification. Chapters 3 to 5 presented our work to address these requirements. This chapter summarizes the results of our work and provides research directions for the future.

6.1 PROBE for Vehicle Breakdown Disruptions

Vehicle breakdown is a prevalent problem in residential waste collection occurring frequently, almost daily in their operations. Without adequate vehicles for replacement and with only limited operating hours, operators face the risk of missing services when breakdown events occur. Chapter 3 proposed online re-routing of remaining *tasks* using only the remaining vehicles, with the new routes starting from their current location and ending at the depot, and taking into account the remaining capacity in each of the vehicles. A new ARP variant, CARP-VB was formulated for this problem of re-routing under vehicle breakdowns. PROBE, a heuristic algorithm was developed for solving CARP-VB.

PROBE was evaluated on the CARP benchmark instances `gdb`, `val`, `egl`, and

`egl-Large` and compared with the conventional method, which is to complete the pending tasks of the broken-down vehicle, using a vehicle that finishes its route first. PROBE's performance was evaluated and compared on the metrics of makespan, range, and discrepancy, as the objective shifts from minimizing total cost to finishing earlier to avoid missing services. PROBE performed better on all the four benchmark instances. The average percentage of reduction on all the three metrics are positive and significant (see §3.5.3), indicating the effectiveness of the algorithm in handling breakdown disruptions.

The increase in the percentage value observed in few of the instances can be attributed to the original solution being followed before the break-down event. Since path scanning heuristic optimizes for total cost, the solution generated can be highly unbalanced with one or two routes being very short. This leads to vehicles finishing quickly and being available at the depot to take-over the operation of the broken-down vehicle when an event occurs break-down occurs. However, it is very unlikely for the operators to follow such plan in the real world.

On measuring and plotting the run time, PROBE is found to be linear with the number of required edges. This implies that the PROBE can scale well with instances of large size that are representative of the real world problems. We also compared the increase in total cost due to re-routing with that of the conventional method. We found that the average increase in total cost is less with 7% in `gdb`, 16% in `val`, 10% in `egl` and 11 % in `egl-large`, indicating that PROBE is an efficient algorithm in both running time and cost.

6.2 MA-ABC for Attractiveness, Route Balance and Cost

Sections §2.6.2 and §4.2 discussed the importance that the operators give to visual attractiveness in the routing solution and the observation that operators are more likely to abandon a solution to their own traditional way of working if the route plans do not look visually attractive. Route balance is another important feature expected by operators in their routing plans to meet the equity concerns such as workload fairness and customer satisfaction.

MA-ABC presented in Chapter 4 optimizes three important objectives: Attractiveness, route balance and total cost. A novel fitness function was used for attractiveness that combines two measures of attractiveness: route over-lap and route contiguity. It contributed results with better attractiveness on all the three features: *Compactness*, *proximity*, and *non-overlapping*.

MA-ABC was evaluated on two benchmark instances `val` and `egl` and its results were compared with that of PSRT. MA-ABC was evaluated on three metrics for attractiveness: *Connectivity index* (CI), the *average task distance* (ATD), and the *route overlapping index* (ROI). MA-ABC scored well on comparing with PSRT. We also compared the performance of MA-ABC on two other objectives, total cost and makespan, with PSRT. MA-ABC performed better than PSRT in those objectives also. MA-ABC had an average gap of only 3% in `val` and 3.31% in `egl` from the optimum or the best known value of *total cost*. MA-ABC is effective in providing solutions of minimum total cost even when optimized with other objectives, attraction and route balance. On plotting the run time, the MA-ABC showed a linear relationship with the number of tasks.

6.3 SPLICE for Learning Heuristics

Chapter 5 discussed the problem of changes happening frequently to the problem specifications for operations such as postal delivery and residential waste collection. The original solution will not be effective for the new changes. Computing new routing solutions for the changes in problem specifications takes longer using existing methods. By learning the distribution of the problem SPLICE an AI driven machine learning framework presented in Chapter 5 generated solutions quickly, that are also closer to optimum.

SPLICE was trained using random instances generated with specific distribution setting for CARP and CCPP with 29 edges. Its results on testing data were compared with the Memetic algorithm for CARP (MA) by Lacomme et al. and with Path scanning (PS) heuristics. SPLICE is better than memetic algorithm, MA in *total cost* by an average of 23% in CCPP and 10% in CARP. On comparing with Path Scanning (PS) algorithm, it is lower by an average of 29% for CCPP and 16% for CARP. We also tested the generalization capability of SPLICE by training the model with instances of size 15 edges and then tested them using instances of size 29 edges. SPLICE generalised well with an average of 40% lower in total cost than MA and 47% lower than PS. The average inference running time for all the experiments is 0.8 seconds, which demonstrates effectiveness of SPLICE and the potential for using it on real world applications.

The main advantage of SPLICE framework is that it can be used to solve different variants of node and arc routing problems, as SPLICE learns Route first-Cluster second type of heuristics. By using an appropriate splitting procedure for the reward estimation, the same model can be trained to solve for different variants of arc routing problems such as Open-CARP, Multi-Depot CARP, Split-Delivery ARP,

stochastic CARP and node routing problems, such as VRP, CVRP, Multi-Depot VRP and others. It does not require any change in the neural network architecture or hyper-parameter settings.

6.4 Future Directions

The PROBE algorithm and the CARP-VB formulation allocates all the remaining vehicles for re-routing after the breakdown. Depending on the time of occurrence of the breakdown, the re-routed plan can extend beyond the normal shift hours. However, the availability of all the remaining vehicles or workers cannot be ensured beyond the shift hours in the real world. Hence accommodating different over-time availability for each of the remaining vehicles while generating the routing plans would be a useful and a potential research direction. The formulation should allow for both the general availability and time period of availability for each of the remaining operating vehicles.

Demands in the route plan are not constant and vary on each collection day. The demands collected until the time of the breakdown event can be indicative of the current distribution of the demand for the pending tasks. Incorporating uncertainty of the demand within the formulation of CARP-VB and using methods such as stochastic and dynamic routing can be another research direction worthy of exploration.

Evaluating MA-ABC on larger, more realistic, data sets would be interesting to understand the implications on running time. Comparison to other heuristics that seek to optimize total cost and balance, not just the total cost as in PSRT, is also worthy to be explored in the future. Incorporating either more or different metrics for attractiveness is worthy of study. Implementing automatic tuning of parameters for each instance can improve the accuracy and speed and is a good subject for future work. Operators in the real world are interested to have many more features in the

solutions in addition to the three objectives used in MA-ABC. Including additional objectives, such as minimizing the number of vehicles and solving them using *many objective optimization* methods such as NSGA-III would be a potential and useful future direction.

Improving SPLICE to solve CARP-VB, re-routing under breakdown scenarios is another plan. As the SPLICE can be applied to different variants of routing problems using appropriate splitting procedures, the challenge is primarily in the design of an appropriate splitting procedure for the CARP-VB variant. Another interesting and useful research direction is to improve SPLICE to be able to solve multiple objectives such as in MA-ABC. A promising area to explore is *transfer learning* in which a model trained for one variant can be fine-tuned to solve a different variant with only limited samples or training (called *Few shot learning*). Other interesting directions to explore include solving for stochastic variants of CARP [7, 160] and extending SPLICE to solve *node, edge, arc routing problems* (NEARP) [8, 9], which is a routing problem defined on mixed graphs with demands requiring services located on nodes, edges, and arcs.

REFERENCES

[1] D. F. Wood, "Transportation economics." https://www.britannica.com/topic/transportation-economics, November 2021.

[2] Bureau of Transportation Statistics, "Transportation Services Contributed 5.4% to US GDP in 2020; a Decline from 5.9% in 2019." https://www.bts.gov/newsroom/transportation-services-contributed-54-us-gdp-2020-decline-59-2019#:~:text=By%20accounting%20for%20the%20in,to%205.4%20percent%20in%202020., May 2022.

[3] A. Kearney, "Change of plans," CSCMP's Annual State of Logistics Report, 2021.

[4] A. Kearney, "Change of plans," CSCMP's Annual State of Logistics Report, 2022.

[5] "Inventory of U.S. Greenhouse Gas (GHG) Emissions and Sinks 1990–2020." https://www.epa.gov/ghgemissions/inventory-us-greenhouse-gas-emissions-and-sinks-1990-2020. Accessed: 2022-10-05.

[6] Á. Corberán, R. Eglese, G. Hasle, I. Plana, and J. M. Sanchis, "Arc routing problems: A review of the past, present, and future," Networks, vol. 77, no. 1, pp. 88–115, 2021.

[7] Á. Corberán and G. Laporte, Arc routing: problems, methods, and applications. SIAM, 2015.

[8] C. Prins and S. Bouchenoua, "A memetic algorithm solving the VRP, the CARP and general routing problems with nodes, edges and arcs," in Recent advances in memetic algorithms, pp. 65–85, Springer, 2005.

[9] T. Vidal, "Node, edge, arc routing and turn penalties: Multiple problems—one neighborhood extension," Operations Research, vol. 65, no. 4, pp. 992–1010, 2017.

[10] C. o. T. Solid Waste Manager, "Tony Miano, Solid Waste Manager, City of Tempe Public Works, Tempe, Arizona, USA." Personal Communication, December 2019.

[11] D. G. Rossit, D. Vigo, F. Tohmé, and M. Frutos, "Visual attractiveness in routing problems: A review," Computers & Operations Research, vol. 103, pp. 13–34, 2019.

[12] R. van Bevern, R. Niedermeier, M. Sorge, and M. Weller, "Chapter 2: The complexity of arc routing problems," in Arc routing: Problems, methods, and applications, pp. 19–52, SIAM, 2015.

[13] L. Euler, "Solutio problematis ad geometriam situs pertinentis," Commentarii academiae scientiarum Petropolitanae, vol. 8, pp. 128–140, 1741.

[14] Á. Corberán and G. Laporte, "A historical perspective on arc routing," in Arc routing: Problems, methods, and applications, ch. 1, pp. 1–16, SIAM, 2013.

[15] K. Mei-Ko, "Graphic programming using odd or even points," Chinese Math., vol. 1, pp. 273–277, 1962.

[16] J. Edmonds and E. L. Johnson, "Matching, euler tours and the chinese postman," Mathematical programming, vol. 5, no. 1, pp. 88–124, 1973.

[17] C. Hierholzer and C. Wiener, "Ueber die möglichkeit, einen linienzug ohne wiederholung und ohne unterbrechung zu umfahren," Mathematische Annalen, vol. 6, pp. 30–32, Mar 1873.

[18] C. H. Papadimitriou, "On the complexity of edge traversing," Journal of the ACM (JACM), vol. 23, no. 3, pp. 544–554, 1976.

[19] M. Guan, "On the windy postman problem," Discrete Applied Mathematics, vol. 9, no. 1, pp. 41–46, 1984.

[20] J. K. Lenstra and A. R. Kan, "On general routing problems," Networks, vol. 6, no. 3, pp. 273–280, 1976.

[21] B. L. Golden and R. T. Wong, "Capacitated arc routing problems," Networks, vol. 11, no. 3, pp. 305–315, 1981.

[22] N. Christofides, "The optimum traversal of a graph," Omega, vol. 1, no. 6, pp. 719–732, 1973.

[23] L. Muyldermans and G. Pang, "Variants of the capacitated arc routing problem," in Arc routing: problems, methods, and applications, ch. 10, pp. 223–253, SIAM, 2015.

[24] C. Prins, "The capacitated arc routing problem: Heuristics," in Arc routing: Problems, methods, and applications, ch. 7, pp. 131–157, SIAM, 2013.

[25] L. Bertazzi, B. L. Golden, and X. Wang, "Min–max vs. min–sum vehicle routing: A worst-case analysis," European Journal of Operational Research, vol. 240, no. 2, pp. 372–381, 2015.

[26] J. M. Belenguer, E. Benavent, and S. Irnich, "The capacitated arc routing problem: Exact algorithms," in Arc routing: Problems, methods, and applications, ch. 9, pp. 183–221, SIAM, 2013.

[27] G. N. Frederickson, "Approximation algorithms for some postman problems," Journal of the ACM (JACM), vol. 26, no. 3, pp. 538–554, 1979.

[28] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," tech. rep., Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[29] B. L. Golden, J. S. DeArmon, and E. K. Baker, "Computational experiments with algorithms for a class of routing problems," Computers & Operations Research, vol. 10, no. 1, pp. 47–59, 1983.

[30] S. Wøhlk, "An approximation algorithm for the capacitated arc routing problem," Open Operational Research Journal, vol. 2, pp. 8–12, 2008.

[31] K. Jansen, "Bounds for the general capacitated routing problem," Networks, vol. 23, no. 3, pp. 165–173, 1993.

[32] E.-G. Talbi, Metaheuristics: From design to implementation. John Wiley & Sons, 2009.

[33] W. L. Pearn, "Approximate solutions for the capacitated arc routing problem," Computers & Operations Research, vol. 16, no. 6, pp. 589–600, 1989.

[34] G. Ulusoy, "The fleet size and mix problem for capacitated arc routing," European Journal of Operational Research, vol. 22, no. 3, pp. 329–337, 1985.

[35] S. Wøhlk, Contributions to arc routing. PhD thesis, University of Southern Denmark, Aarhus, Denmark, 2005.

[36] J.-M. Belenguer, E. Benavent, P. Lacomme, and C. Prins, "Lower and upper bounds for the mixed capacitated arc routing problem," Computers & Operations Research, vol. 33, no. 12, pp. 3363–3383, 2006.

[37] L. Santos, J. Coutinho-Rodrigues, and J. R. Current, "An improved heuristic for the capacitated arc routing problem," Computers & Operations Research, vol. 36, no. 9, pp. 2632–2637, 2009.

[38] T. Vidal, "Split algorithm in o (n) for the capacitated vehicle routing problem," Computers & Operations Research, vol. 69, pp. 40–47, 2016.

[39] C. Prins, N. Labadi, and M. Reghioui, "Tour splitting algorithms for vehicle routing problems," International Journal of Production Research, vol. 47, no. 2, pp. 507–535, 2009.

[40] A. Hertz, G. Laporte, and M. Mittaz, "A tabu search heuristic for the capacitated arc routing problem," Oper. Res., vol. 48, pp. 129–135, Jan. 2000.

[41] J. Brandão and R. Eglese, "A deterministic tabu search algorithm for the capacitated arc routing problem," Computers & Operations Research, vol. 35, no. 4, pp. 1112–1126, 2008.

[42] P. Beullens, L. Muyldermans, D. Cattrysse, and D. V. Oudheusden, "A guided local search heuristic for the capacitated arc routing problem," European Journal of Operational Research, vol. 147, no. 3, pp. 629 – 643, 2003.

[43] F. L. Usberti, P. M. França, and A. L. M. França, "Grasp with evolutionary path-relinking for the capacitated arc routing problem," Computers & Operations Research, vol. 40, no. 12, pp. 3206–3217, 2013.

[44] A. Hertz and M. Mittaz, "A variable neighborhood descent algorithm for the undirected capacitated arc routing problem," Transportation science, vol. 35, no. 4, pp. 425–434, 2001.

[45] L. Santos, J. Coutinho-Rodrigues, and J. R. Current, "An improved ant colony optimization based algorithm for the capacitated arc routing problem," Transportation Research Part B: Methodological, vol. 44, no. 2, pp. 246–266, 2010.

[46] P. Greistorfer, "A tabu scatter search metaheuristic for the arc routing problem," Computers & Industrial Engineering, vol. 44, no. 2, pp. 249–266, 2003.

[47] P. Lacomme, C. Prins, and W. Ramdane-Chérif, "A genetic algorithm for the capacitated arc routing problem and its extensions," in Applications of Evolutionary Computation, pp. 473–483, Springer, 2001.

[48] P. Lacomme, C. Prins, and W. Ramdane-Cherif, "Competitive memetic algorithms for arc routing problems," Annals of Operations Research, vol. 131, no. 1, pp. 159–185, 2004.

[49] I. M. Oliver, D. J. Smith, and J. R. C. Holland, "A study of permutation crossover operators on the traveling salesman problem," in Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application, pp. 224–230, L. Erlbaum Associates Inc., 1987.

[50] L. Y. Li and R. W. Eglese, "An interactive algorithm for vehicle routeing for winter—gritting," Journal of the Operational Research Society, vol. 47, no. 2, pp. 217–228, 1996.

[51] Y. Chen, J.-K. Hao, and F. Glover, "A hybrid metaheuristic approach for the capacitated arc routing problem," European Journal of Operational Research, vol. 253, no. 1, pp. 25–39, 2016.

[52] Q. Mu, Z. Fu, J. Lysgaard, and R. Eglese, "Disruption management of the vehicle routing problem with vehicle breakdown," Journal of the Operational Research Society, vol. 62, no. 4, pp. 742–749, 2011.

[53] Q. Mu and R. W. Eglese, "Disrupted capacitated vehicle routing problem with order release delay," Annals of Operations Research, pp. 1–16, 2013.

[54] J.-Q. Li, P. B. Mirchandani, and D. Borenstein, "The Vehicle Rescheduling Problem: Model and Algorithms," Networks, vol. 50, no. 3, pp. 211–229, 2007.

[55] J.-Q. Li, P. B. Mirchandani, and D. Borenstein, "A lagrangian heuristic for the real-time vehicle rescheduling problem," Transportation Research Part E: Logistics and Transportation Review, vol. 45, no. 3, pp. 419–433, 2009.

[56] J.-Q. Li, P. B. Mirchandani, and D. Borenstein, "Real–time vehicle rerouting problems with time windows," European Journal of Operational Research, vol. 194, no. 3, pp. 711–727, 2009.

[57] M. Monroy-Licht, C. A. Amaya, A. Langevin, and L.-M. Rousseau, "The rescheduling arc routing problem," International Transactions in Operational Research, vol. 24, no. 6, pp. 1325–1346, 2017.

[58] A. Poot, G. Kant, and A. P. M. Wagelmans, "A savings based method for real-life vehicle routing problems," Journal of the Operational Research Society, vol. 53, no. 1, pp. 57–68, 2002.

[59] M. Constantino, L. Gouveia, M. C. Mourão, and A. C. Nunes, "The mixed capacitated arc routing problem with non-overlapping routes," European Journal of Operational Research, vol. 244, no. 2, pp. 445–456, 2015.

[60] D. Applegate, W. Cook, S. Dash, and A. Rohe, "Solution of a min-max vehicle routing problem," INFORMS Journal on computing, vol. 14, no. 2, pp. 132–143, 2002.

[61] P. Matl, R. F. Hartl, and T. Vidal, "Workload equity in vehicle routing problems: A survey and analysis," Transportation Science, vol. 52, no. 2, pp. 239–260, 2018.

[62] P. Matl, R. F. Hartl, and T. Vidal, "Workload equity in vehicle routing: The impact of alternative workload resources," Computers & Operations Research, vol. 110, pp. 116–129, 2019.

[63] N. Jozefowiez, E.-G. Talbi, et al., "From single-objective to multi-objective vehicle routing problems: Motivations, case studies, and methods," in The vehicle routing problem: Latest advances and new challenges, pp. 445–471, Springer, 2008.

[64] N. Jozefowiez, F. Semet, and E.-G. Talbi, "An evolutionary algorithm for the vehicle routing problem with route balancing," European Journal of Operational Research, vol. 195, no. 3, pp. 761–769, 2009.

[65] J. Lozano, L.-C. González-Gurrola, E. Rodríguez-Tello, and P. Lacomme, "A statistical comparison of objective functions for the vehicle routing problem with route balancing," in 2016 Fifteenth Mexican international conference on artificial intelligence (MICAI), pp. 130–135, IEEE, 2016.

[66] M. Ehrgott, Multicriteria optimization. Springer Berlin, Heidelberg, 2005.

[67] C. A. Coello, "Evolutionary multi-objective optimization: a historical view of the field," IEEE computational intelligence magazine, vol. 1, no. 1, pp. 28–36, 2006.

[68] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," IEEE transactions on evolutionary computation, vol. 6, no. 2, pp. 182–197, 2002.

[69] J. J. Hopfield and D. W. Tank, ""Neural" computation of decisions in optimization problems," Biological cybernetics, vol. 52, no. 3, pp. 141–152, 1985.

[70] R. Durbin and D. Willshaw, "An analogue approach to the travelling salesman problem using an elastic net method," Nature, vol. 326, no. 6114, pp. 689–691, 1987.

[71] J. Fort, "Solving a combinatorial problem via self-organizing process: An application of the Kohonen algorithm to the traveling salesman problem," Biological cybernetics, vol. 59, no. 1, pp. 33–40, 1988.

[72] K. A. Smith, "Neural networks for combinatorial optimization: a review of more than a decade of research," INFORMS Journal on Computing, vol. 11, no. 1, pp. 15–34, 1999.

[73] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," European Journal of Operational Research, 2020.

[74] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," Advances in neural information processing systems, vol. 27, pp. 3104–3112, 2014.

[75] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in 3rd International Conference on Learning Representations, ICLR 2015, 2015.

[76] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," Advances in neural information processing systems, vol. 28, pp. 2692–2700, 2015.

[77] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," arXiv preprint arXiv:1611.09940, 2016.

[78] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," Machine learning, vol. 8, no. 3-4, pp. 229–256, 1992.

[79] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," in International conference on machine learning, pp. 2702–2711, 2016.

[80] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," Advances in neural information processing systems, vol. 30, pp. 6348–6358, 2017.

[81] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[82] M. Nazari, A. Oroojlooy, M. Takáč, and L. V. Snyder, "Reinforcement learning for solving the vehicle routing problem," in Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS'18, p. 9861–9871, 2018.

[83] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in International conference on machine learning, pp. 1928–1937, 2016.

[84] G. Clarke and J. W. Wright, "Scheduling of vehicles from a central depot to a number of delivery points," Operations research, vol. 12, no. 4, pp. 568–581, 1964.

[85] A. Wren and A. Holliday, "Computer scheduling of vehicles from one or more depots to a number of delivery points," Journal of the Operational Research Society, vol. 23, no. 3, pp. 333–344, 1972.

[86] L. Perron and V. Furnon, "Google OR-Tools." https://developers.google.com/optimization/.

[87] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!," in International Conference on Learning Representations, 2019.

[88] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in Advances in neural information processing systems, pp. 5998–6008, 2017.

[89] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, and L.-M. Rousseau, "Learning heuristics for the tsp by policy gradient," in International conference on the integration of constraint programming, artificial intelligence, and operations research, pp. 170–181, Springer, 2018.

[90] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," arXiv preprint arXiv:1906.01227, 2019.

[91] X. Bresson and T. Laurent, "Residual gated graph convnets," arXiv preprint arXiv:1711.07553, 2017.

[92] X. Chen and Y. Tian, "Learning to perform local rewriting for combinatorial optimization," in Advances in Neural Information Processing Systems, pp. 6281–6292, 2019.

[93] T. Barrett, W. Clements, J. Foerster, and A. Lvovsky, "Exploratory combinatorial optimization with reinforcement learning," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 3243–3250, 2020.

[94] P. R. d O Costa, J. Rhuggenaath, Y. Zhang, and A. Akcay, "Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning," in Asian Conference on Machine Learning, pp. 465–480, PMLR, 2020.

[95] Y. Wu, W. Song, Z. Cao, J. Zhang, and A. Lim, "Learning improvement heuristics for solving routing problems..," IEEE transactions on neural networks and learning systems, 2021.

[96] L. Gao, M. Chen, Q. Chen, G. Luo, N. Zhu, and Z. Liu, "Learn to design the heuristics for vehicle routing problem," arXiv preprint arXiv:2002.08539, 2020.

[97] E. J. Willemse, HEURISTICS FOR LARGE-SCALE CAPACITATED ARC ROUTING PROBLEMS ON MIXED NETWORKS. PhD thesis, University of Pretoria, 2016.

[98] N. V. Karadimas, K. Papatzelou, and V. G. Loumos, "Optimal solid waste collection routes identified by the ant colony system algorithm," Waste management & research, vol. 25, no. 2, pp. 139–147, 2007.

[99] P. Viotti, A. Polettini, R. Pomi, and C. Innocenti, "Genetic algorithms as a promising tool for optimisation of the msw collection routes," Waste management & research, vol. 21, no. 4, pp. 292–298, 2003.

[100] RouteSmart Technologies Inc., "Routesmart," [Online; accessed 12-April-2019]. [Online; accessed 12-April-2019].

[101] J. Evans and E. Minieka, "Optimization algorithms for networks and graphs. 1992."

[102] G. Ghiani, G. Improta, and G. Laporte, "The capacitated arc routing problem with intermediate facilities," Networks: An International Journal, vol. 37, no. 3, pp. 134–143, 2001.

[103] G. Ghiani, F. Guerriero, G. Laporte, and R. Musmanno, "Tabu search heuristics for the arc routing problem with intermediate facilities under capacity and length restrictions," Journal of Mathematical Modelling and Algorithms, vol. 3, no. 3, pp. 209–223, 2004.

[104] J. S. DeArmon, A comparison of heuristics for the capacitated Chinese postman problem. PhD thesis, University of Maryland, 1981.

[105] E. Benavent, V. Campos, A. Corberán, and E. Mota, "The capacitated arc routing problem: lower bounds," Networks, vol. 22, no. 7, pp. 669–690, 1992.

[106] R. W. Eglese, "Routeing winter gritting vehicles," Discrete applied mathematics, vol. 48, no. 3, pp. 231–244, 1994.

[107] M. Ramamoorthy and V. R. Syrotiuk, "Online re-routing for vehicle breakdown in residential waste collection," tech. rep., Arizona State University, 2020.

[108] A. Corberán and C. Prins, "Recent results on arc routing problems: An annotated bibliography," Networks, vol. 56, no. 1, pp. 50–69, 2010.

[109] H. A. Eiselt, M. Gendreau, and G. Laporte, "Arc routing problems, part i: The chinese postman problem," Operations Research, vol. 43, no. 2, pp. 231–242, 1995.

[110] M. Dror, Arc routing: theory, solutions and applications. Springer Science & Business Media, 2012.

[111] ORTEC, 2020.

[112] O. Lum, C. Cerrone, B. L. Golden, and E. Wasil, "Partitioning a street network into compact, balanced, and visually appealing routes," Networks, vol. 69, no. 3, pp. 290–303, 2017.

[113] Á. Corberán, B. L. Golden, O. Lum, I. Plana, and J. M. Sanchis, "Aesthetic considerations for the min-max k-windy rural postman problem," Networks, vol. 70, no. 3, pp. 216–232, 2017.

[114] N. Jozefowiez, F. Semet, and E.-G. Talbi, From single-objective to multi-objective vehicle routing problems: Motivations, case studies, and methods, pp. 445–471. Springer US, 2008.

[115] C. A. Coello, S. G. Brambila, J. F. Gamboa, M. Tapia, C. Guadalupe, and R. H. Gómez, "Evolutionary multiobjective optimization: open research areas and some challenges lying ahead," Complex & Intelligent Systems, vol. 6, no. 2, pp. 221–236, 2020.

[116] P. Lacomme, C. Prins, and M. Sevaux, "Multiobjective capacitated arc routing problem," in International Conference on Evolutionary Multi-Criterion Optimization, pp. 550–564, Springer, 2003.

[117] P. Lacomme, C. Prins, and M. Sevaux, "A genetic algorithm for a bi-objective capacitated arc routing problem," Computers & Operations Research, vol. 33, no. 12, pp. 3473–3493, 2006.

[118] C. Prins, P. Lacomme, and C. Prodhon, "Order-first split-second methods for vehicle routing problems: A review," Transportation Research Part C: Emerging Technologies, vol. 40, pp. 179–200, 2014.

[119] Y. Mei, K. Tang, and X. Yao, "Decomposition-based memetic algorithm for multiobjective capacitated arc routing problem," IEEE Transactions on Evolutionary Computation, vol. 15, no. 2, pp. 151–165, 2011.

[120] L. Grandinetti, F. Guerriero, D. Laganà, and O. Pisacane, "An optimization-based heuristic for the multi-objective undirected capacitated arc routing problem," Computers & Operations Research, vol. 39, no. 10, pp. 2300–2309, 2012.

[121] K. Tang, Y. Mei, and X. Yao, "Memetic algorithm with extended neighborhood search for capacitated arc routing problems," IEEE Transactions on Evolutionary Computation, vol. 13, no. 5, pp. 1151–1166, 2009.

[122] R. K. Arakaki and F. L. Usberti, "Hybrid genetic algorithm for the open capacitated arc routing problem," Computers & Operations Research, vol. 90, pp. 221–231, 2018.

[123] C. Martinez, I. Loiseau, M. G. Resende, and S. Rodriguez, "Brkga algorithm for the capacitated arc routing problem," Electronic Notes in Theoretical Computer Science, vol. 281, pp. 69–83, 2011.

[124] Y. Mei, X. Li, and X. Yao, "Cooperative coevolution with route distance grouping for large-scale capacitated arc routing problems," IEEE Transactions on Evolutionary Computation, vol. 18, no. 3, pp. 435–449, 2013.

[125] G. Fleury, P. Lacomme, and C. Prins, "Evolutionary algorithms for stochastic arc routing problems," in Applications of Evolutionary Computing, pp. 501–512, Springer Berlin Heidelberg, 2004.

[126] Y. Mei, K. Tang, and X. Yao, "Capacitated arc routing problem in uncertain environments," in IEEE Congress on Evolutionary Computation, pp. 1–8, IEEE, 2010.

[127] J. Wang, K. Tang, and X. Yao, "A memetic algorithm for uncertain capacitated arc routing problems," in 2013 IEEE Workshop on Memetic Computing (MC), pp. 72–79, IEEE, 2013.

[128] J. Wang, K. Tang, J. A. Lozano, and X. Yao, "Estimation of the distribution algorithm with a stochastic local search for uncertain capacitated arc routing problems," IEEE Transactions on Evolutionary Computation, vol. 20, no. 1, pp. 96–109, 2015.

[129] H. Handa, L. Chapman, and X. Yao, "Dynamic salting route optimisation using evolutionary computation," in 2005 IEEE Congress on Evolutionary Computation, vol. 1, pp. 158–165, IEEE, 2005.

[130] H. Handa, L. Chapman, and X. Yao, "Robust salting route optimization using evolutionary algorithms," in Evolutionary Computation in Dynamic and Uncertain Environments, pp. 497–517, Springer, 2007.

[131] M. Liu, H. K. Singh, and T. Ray, "A memetic algorithm with a new split scheme for solving dynamic capacitated arc routing problems," in 2014 IEEE Congress on Evolutionary Computation (CEC), pp. 595–602, IEEE, 2014.

[132] R. Shang, J. Wang, L. Jiao, and Y. Wang, "An improved decomposition-based memetic algorithm for multi-objective capacitated arc routing problem," Applied Soft Computing, vol. 19, pp. 343–361, 2014.

[133] R. Shang, Y. Wang, J. Wang, L. Jiao, S. Wang, and L. Qi, "A multi-population cooperative coevolutionary algorithm for multi-objective capacitated arc routing problem," Information Sciences, vol. 277, pp. 609–642, 2014.

[134] "Supplementary materials and reproducibility information for ma-abc." http://www.public.asu.edu/ syrotiuk/gecco21.html, 2021.

[135] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," Journal of Machine Learning Research, vol. 13, pp. 2171–2175, jul 2012.

[136] M. Ramamoorthy and V. R. Syrotiuk, "Online re-routing for vehicle breakdown in residential waste collection," in 2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall), pp. 1–5, IEEE, 2020.

[137] L. Juan, C. Zixing, and L. Jianqin, "Premature convergence in genetic algorithm: Analysis and prevention based on chaos operator," in Proceedings of the 3rd World Congress on Intelligent Control and Automation (Cat. No. 00EX393), vol. 1, pp. 495–499, IEEE, 2000.

[138] E. B. Khalil, P. Le Bodic, L. Song, G. Nemhauser, and B. Dilkina, "Learning to branch in mixed integer programming," in Thirtieth AAAI Conference on Artificial Intelligence, 2016.

[139] M.-F. Balcan, T. Dick, T. Sandholm, and E. Vitercik, "Learning to branch," in Proceedings of the 35th International Conference on Machine Learning (J. Dy and A. Krause, eds.), vol. 80 of Proceedings of Machine Learning Research, pp. 344–353, PMLR, 2018.

[140] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in International conference on machine learning, pp. 1263–1272, PMLR, 2017.

[141] Y. Wu, W. Song, Z. Cao, J. Zhang, and A. Lim, "Learning improvement heuristics for solving the travelling salesman problem," arXiv preprint arXiv:1912.05784, 2019.

[142] H. Li and G. Li, "Learning to solve capacitated arc routing problems by policy gradient," in 2019 IEEE Congress on Evolutionary Computation (CEC), pp. 1291–1298, IEEE, 2019.

[143] W. Hong and T. Liu, "Faster capacitated arc routing: A sequence-to-sequence approach," IEEE Access, vol. 10, pp. 4777–4785, 2022.

[144] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in Proceedings of the 30th International Conference on Neural Information Processing Systems, NIPS'16, p. 3844–3852, 2016.

[145] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 855–864, 2016.

[146] L. Lovász, "Random walks on graphs," Combinatorics, Paul erdos is eighty, vol. 2, no. 1-46, p. 4, 1993.

[147] J. L. Gross, J. Yellen, and M. Anderson, Graph theory and its applications. Chapman and Hall/CRC, 2018.

[148] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in International Conference on Learning Representations, 2019.

[149] W. L. Hamilton, "Graph representation learning," Synthesis Lectures on Artifical Intelligence and Machine Learning, vol. 14, no. 3, pp. 1–159, 2020.

[150] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML'10, p. 807–814, 2010.

[151] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.

[152] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," arXiv preprint arXiv:1312.5602, 2013.

[153] C. Watkins, Learning form delayed rewards. PhD thesis, King's College, University of Cambridge, 1989.

[154] M. Riedmiller, "Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method," in European conference on machine learning, pp. 317–328, Springer, 2005.

[155] S. Zhang and R. S. Sutton, "A deeper look at experience replay," arXiv preprint arXiv:1712.01275, 2017.

[156] W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, and W. Dabney, "Revisiting fundamentals of experience replay," in International Conference on Machine Learning, pp. 3061–3071, PMLR, 2020.

[157] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (Y. Bengio and Y. LeCun, eds.), 2015.

[158] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in Advances in Neural Information Processing Systems 32 (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[159] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[160] M. C. Mourão and L. S. Pinto, "An updated annotated bibliography on arc routing problems," Networks, vol. 70, no. 3, pp. 144–194, 2017.