ROMA TRE
UNIVERSITÀ DEGLI STUDI

Doctoral Thesis in
Computer Science and Automation
Department of Engineering

XXVI Cycle

A Memetic NSGA-II for bi-objective Mixed Capacitated General Routing Problem

Student: Santosh Kumar Mandal

Advisor: Dario Pacciarelli

Course Coordinator: Stefano Panzieri

A Memetic NSGA-II for bi-objective Mixed Capacitated General Routing Problem

A thesis presented by
Santosh Kumar Mandal
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Computer Science and Automation

Roma Tre University
Department of Engineering

2014

ADVISORS:
Prof. Dario Pacciarelli
Late Prof. Arne Løkketangen

REVIEWERS:
Prof. Franceso Viti
Prof. Stefano Giordani

Dedicated to my Parents.

Abstract

The Mixed Capacitated General Routing Problem (MCGRP) is concerned with the determination of the optimal vehicle routes to service a set of customers located at nodes and along edges/arcs on a mixed weighted graph representing a complete distribution network. Although using nodes, edges and arcs simultaneously yields better models for many real-life vehicle routing problems such as *newspaper delivery* and *urban waste collection*, very few research works have been dedicated since the MCGRP was defined. Furthermore, most of the studies have focused on the optimization of just one objective, that is, cost minimization. Keeping in mind the requirements of industries nowadays, MCGRP has been addressed in this thesis to concurrently optimize two crucial objectives, namely, minimization of routing cost and route balance (the difference between the largest route and the smallest route with respect to duration). To solve this bi-objective form of the MCGRP, a multi-objective evolutionary algorithm (MOEA), coined as Memetic NSGA-II, has been designed. It is a hybrid of non-dominated sorting genetic algorithm-II (NSGA-II), a dominance based local search procedure (DBLSP) and a clone management principle (CMP). The DBLSP and CMP have been incorporated into the framework of NSGA-II with a view to empowering its capability to converge at/or near the true Pareto front and boosting diversity among the trade-off solutions. In addition, the algorithm also contains a set of three well-known crossover operators (X-set) that are employed to explore different parts of the search space. It was tested on twenty three instances simulating real-life situations and of varying complexity. The computational experiments verify the effectiveness of the Memetic NSGA-II and also show the energetic effects of using DBLSP, CMP and X-set together while finding the set of potentially Pareto optimal solutions.

Acknowledgements

First of all, I would like to thank my supervisor Prof. Dario Pacciarelli for his guidance, support and kind advices throughout my Ph.D studies. This thesis work would not have been possible without my second advisor Late Prof. Arne Løkketangen, who passed away on June 10, 2013. A major portion of this research was accomplished under his constructive guidance of eleven months at Molde University College, Norway. I also would like to thank Dr. Geir Hasle, chief research scientist at SINTEF (an independent research organization in Norway), for providing valuable suggestions that helped improve the quality of this work.

I wish to thank Prof. Stefano Giordani (Università di Roma "Tor Vergata") and Prof. Franceso Viti (Université Du Luxembourg) for agreeing to be reviewers of my thesis. Their suggestions helped me shape this thesis in a better way.

Last but not the least, I wish to thank my parents, Dwarika Mandal and Laxmi Devi and my sister Renu Kumari, for their supports in achieving my personal and academic goals. I owe them everything and I hope that this work makes them proud.

Contents

List of Tables

List of Figures

Introduction

In today's fast growing and highly competitive goods distribution business, firms are giving the top priority to route planning in order to acquire a dominant position in the market. A more efficient routing plan not only brings economical advantages for a company but also reduces several private and public concerns, such as traffic congestion, air pollution and energy consumption, to name a few. Moreover, modern road infrastructures, especially in the urban areas, are much more complex and pose new challenges in the designing of vehicle routes. For these reasons, vehicle routing problem (VRP)– which was defined several decades ago and studied extensively – is still drawing considerable attention of researchers around the world. In its simplest form, the VRP/Capacitated VRP (CVRP) is defined on an undirected graph in which nodes represent customers and edges connecting them have a traversal cost. A fleet of identical vehicles with limited carrying capacity is stationed at a central depot node. The goal is to construct a set of tours for the vehicles to service geographically scattered customers. Some of the most commonly used objectives and constraints in VRPs have been shown below in the bulleted lists:

- Objectives:
  - Minimization of travel cost
  - Minimization of tour length
  - Minimization of the number of vehicles
  - Maximization of the route compactness
- Constraints:
  - Each tour should start and end at the depot node.
  - Vehicle capacity should not violate on any tour.
  - Each customer must be served only once by a single vehicle.

Several approaches have been suggested to solve the classical VRP and its different variants; for example, VRP with time windows, heterogeneous fleet VRP, multi-depot VRP, VRP with pickup & delivering and VRP with satellite facilities. In accordance with solution techniques, the available literatures on VRPs can be divided into three groups: exact algorithms, heuristics and meta-heuristics. It is a well-known fact that owing to NP-hard complexity of the VRP, exact algorithms can not accomplish the optimal solutions for large real-life instances. Heuristics and meta-heuristics approximation methods, on the other hand, prove to be viable techniques to find near optimal solutions for all kinds of VRPs within a reasonable amount of computational time.

One of the best and widely known heuristics for VRPs is the saving-based algorithm of Clarke and Wright (1964). It is simple in structure, easy to implement and flexible as well. The sweep algorithm of Gillett and Miller (1974) was also proved to be efficient for solving the medium and large scale VRPs. Beasley (1983) described the route-first cluster-second method for VRPs. This is a two-phase method in which a giant tour is constructed first, ignoring the vehicle capacity constraint and then this tour is partitioned into feasible vehicle routes. Due to the dependency of heuristic methods on the specific characteristics the problem, they can not be applied to a wide range of VRP variants. This is why meta-heuristic approaches have been the top choice to tackle VRPs. Osman (1993) implemented Simulated Annealing (SA) algorithm to solve a VRP subjected to the distance and capacity constraints. Gendreau et al. (1994) introduced the Taburoute algorithm, a variant of Tabu search meta-heuristic, to obtain near optimal solutions of constrained VRPs. Rochat and Taillard (1995) proposed an adaptive memory procedure within Tabu Search for the CVRP. An adaptive memory is basically a pool of routes belonging to the elite solutions discovered during the search. It is dynamically updated and used to provide new starting solutions for Tabu Search. Later, Toth and Vigo (2003) put forward another interesting variant of Tabu Search, so called Granular Tabu Search (GTS), for the CVRP. The GTS is based on the concept of drastically restricted neighbourhoods; it discards moves that insert only long edges into the solution. Berger and Barkaoui (2003) proposed a hybrid genetic algorithm to address the CVRP. It concurrently works on two populations of solutions with the periodic exchange of some local best individuals, and also uses well-known heuristics in the genetic operators.

Recently, Nagata and Bräysy (2009) suggested a Memetic Algorithm (MA), enhanced by a novel Edge Assembly Crossover (EAX) operator and efficient local search procedures, for solving the CVRP. This EAX based MA was found to be robust and very competitive. It yielded new best solutions to 10 large-scale benchmark instances (out of 12) of Golden et al. (1998) in a reasonable computational time. Some other notable approaches using meta-heuristic for VRPs include the work of Pisinger and Ropke (2006b)[Adaptive Large Neighbourhood Search (ALNS)], Prins (2004)[Hybrid Genetic Algorithm], Alba and Dorronsoro (2004) [Cellular Genetic Algorithm] and Reimann

et al. (2004) [Ant Colony Optimization].

All of the above cited and most existing works on VRPs were solved on an undirected network graph and assuming the node-based routing in which demand locations on street segments are clearly represented by nodes. Nevertheless, for many real world VRPs, such as *newspaper delivery* and *urban waste collection*, this assumption seems to be unrealistic. In such problems, sometimes vehicles have to serve full streets besides specific spots. Theses circumstances can not be modelled by even pure CARP (Capacitated arc routing problem), a class of VRP in which vehicles are constrained to move on arcs and demand is also assigned to only arcs. Furthermore, an undirected graph can only model a 2-way street whose both sides are serviced in parallel and in any direction. In the real road network, a street can be a 2-way with bilateral service (an edge in the modelled network), a 2-way street with two sides serviced independently (two opposite arcs) or even 1-way street (one arc)[Prins and Bouchenoua (2004)].

The MCGRP, formally defined in the Section 1.1, allows to tackle such real-world scenarios in a natural way. The MCGRP has been studied by several researchers in the last two decades using different terminologies. Pandit and Muralidharan (1995) solved the problem with a heterogeneous set of vehicles, denoting it as the capacitated general routing problem (CGRP). Gutiérrez et al. (2002) investigated the homogeneous fleet version of the CGRP. Prins and Bouchenoua (2004) proposed a Memetic Algorithm for the MCGRP, calling it as the MCGRP (node, edge and arc routing problem). Kokubugata et al. (2007) solved the MCGRP/MCGRP using a Simulated Annealing algorithm with competing neighborhood operators. More recently, Bosco et al. (2013) proposed an integer programming formulation for the MCGRP.

The common element in the above works on the MCGRP is the consideration of single objective of the minimization of the tour cost. The minimization of the cost function, without any doubt, is essential to survive in the competitive logistic market. However, most companies involved in the waste management, newspaper or other distribution business have also realized the need of well-balanced (in terms of cost or time or length) routes. As for instance, Distribution Innovation AS – an industrial partner of a research organization in Scandinavia, called SINTEF (Stiftelsen for industriell og teknisk forskning) – which manages the delivery of newspapers almost in the whole Norwegian market, has estimated that a maximum imbalance (duration-wise) between routes of about 20% is tolerable for the efficient operation [Hasle (2012)]. A routing plan with balanced tours can help achieve several other goals. For example, shift duration for drivers and workloads among vehicles are related to the route balancing objective. Hence, in this thesis, MCGRP is being resolved to concurrently optimize these two important objectives (minimization of routing cost and route balance). Moreover, route balance in this thesis has been defined as the difference between the largest route and the smallest route in terms of the routing cost (proportional to the travel time).

Since the targeted objectives are conflicting in nature; therefore, providing a set of compromised solutions will help decision makers to choose the best one according to the requirements. Thus, a multi-objective evolutionary algorithm (MOEA), named Memetic NSGA-II, is being proposed to obtain an efficient Pareto set for the present bi-objective MCGRP. The NSGA-II (Non-dominated sorting genetic algorithm-II), propounded by Deb et al. (2002), is a very popular MOEA and so far has had phenomenal success in solving multi-objective problems from different research domains. Similarly, local search based meta-heuristics (such as Tabu search, Simulated annealing and Memetic algorithm) have performed very well in the VRP area. Moreover, hybrid algorithms more often perform better than the individual algorithms from which they are designed. Motivated by these facts, in the proposed algorithm, NSGA-II has been hybridized with a dominance based local search procedure (DBLSP) for better approximation of the true Pareto set. Furthermore, even though NSGA-II is equipped with a so called *crowded comparison operator* to promote diversity (and convergence as well), it may become susceptible to the phenomena of *genetic drift*, which often occurs in real-world combinatorial optimization problems. Hence, it has also been amalgamated with a clone management principle (CMP) to increase its ability to generate uniformly distributed Pareto sets. In addition, Memetic NSGA-II is also equipped with a set of three different crossover operators (X-set) that are used to explore different promising regions of the search space. With the aid of these three components (DBLSP, CMP and X-set), it produced good compromised solutions on the MCGRP instances, maintaining a fine balance between exploration & exploitation and avoiding premature convergence.

The rest of the thesis has been organized into four parts as follows. The *first part* lays down the basic foundations for this thesis. It provides a detailed description of the MCGRP along with its two real life examples. Specifically, the key concepts of local search method, evolutionary algorithm, multi-objective optimization problem and multi-objective evolutionary algorithm are presented. In the *second part*, an extensive literature survey on three types of routing problems: node, arc and mixed, has been conducted. The *third and fourth parts* present the complete research work accomplished in this thesis. The third part mainly contains the mathematical formulation of objectives & constraints and the working process of Memetic NSGA-II. The details of computational experiments have been provided in the fourth part.

At last, conclusions and possible extensions of the present research work are discussed. An Appendix has also been provided which contains pseudocodes of some algorithms reported in this thesis.

* * * * *

Part 1

Foundations

Foundations

The purpose of this part is to explain the MCGRP in detail and study basic foundations of the MOEA. This part of thesis begins with the description of the problem. Next, two real-life cases of the MCGRP have been discussed. The basics of local search techniques and evolutionary algorithms, which are generally used to solve hard combinatorial optimization problems, have been presented then for better understanding of the whole algorithmic idea later in the third part. Following this, basic concepts of the multi-objective optimization problem are introduced. In particular, definitions of Pareto dominance, Pareto optimality and ideal & nadir points have been provided. Next, various design issues of MOEAs are discussed. This part ends with an overview of some available performance metrics for multi-objective optimizers.

1.1 Mixed Capacitated General Routing Problem

The Mixed Capacitated General Routing Problem (MCGRP) was first defined by Pandit and Muralidharan (1995) in order to assimilate peculiarities of real world vehicle routing problems. The classical Vehicle Routing Problem (VRP), introduced by Dantzig and Ramser (1959), is concerned with the determination of the optimal vehicle routes to service a given set of geographically dispersed customers. The exact locations of customers are known in advance and can be clearly represented by *nodes/points* on a network graph. A variant of the VRP, called Arc Routing Problem (ARP), seeks to service a set of *streets* on a street network. However, most of real world and/or industrial routing problems naturally involve in servicing streets as well as specific spots. Such circumstances can not be tackled by either the VRP or the ARP alone. The MCGRP system helps handle such complex routing situations as it generalizes the classical VRP, the ARP and the general routing problem.

A distribution network for the MCGRP can be represented by a mixed weighted graph (G) consisting of nodes, edges and arcs, i.e, $G = (V, E \cup A)$. Where, V represents the set of nodes, E is the set of edges and A stands for the set of arcs. Furthermore,

an edge on the G represents a *two-way* street whose both sides can be traversed in any direction. A street with uni-directional sides is represented by a pair of two opposite arcs. A single arc stands for the *one-way* street. The entities (nodes, edges and arcs) on the G are of two kinds: *required* and *non-required*. The required items (except nodes) have a traversal cost and have to be processed. The non-required items do not require any service. However, they are needed to determine the minimum *deadheading* costs (traversal costs without service) between all pairs of nodes to move between required items. More importantly, G may not obey the triangle inequality as there can be several edge/arcs between two given nodes. Thus, deadheading costs between nodes may not be symmetric. The system also contains a fleet of homogeneous (or non-homogeneous) vehicles. The vehicles are having limited carrying capacity and initially based at a central depot.

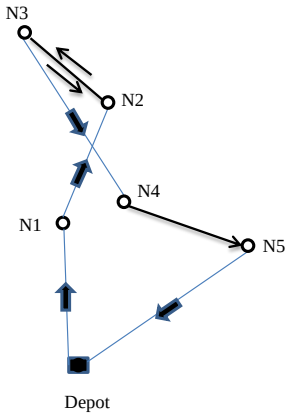

Figure 1: A vehicle tour in MCGRP

The problem seeks to design the set of optimal tours for the vehicles to process all *required* tasks. Each constructed vehicle tour, as shown in the Figure 1, contains a subset of tasks and also shows the order in which they will be processed. As for instance, the processing order of three tasks (node $N1$, edge $N2 \leftrightarrow N3$ and arc $N4 \rightarrow N4$) in the Figure 1 is $(N1) \rightarrow (N2 \leftrightarrow N3) \rightarrow (N4 \rightarrow N5)$. As stated earlier in the *Introduction*, the vehicle tours have to be constructed in such a way that some objectives associated with efficiency of the routing operation could be optimized, while meeting some constraints.

1.1.1 Real-life cases

The model of MCGRP brings node and edge/arc routings together on a network graph simulating the *real-world road topography*. It is flexible and can be used to tackle routing problems seeking to serve specific locations or full streets or both. In the following sub-sections, two practical cases - newspaper delivery and *urban waste collection* - have been described, where the MCGRP seems to be a better model than the VRP and the ARP.

1.1.1.1 Newspaper delivery problem

One of the world's largest newspaper publishers is Japan's "The Yomiuri Shimbun". As of mid-year 2011, it had a combined morning-evening circulation of whopping 13.5 millions for its national edition. The "Aftenposten", which is the Norway's largest newspaper, had a circulation of 239,831 in 2010 of its morning edition. Similarly, "La Republica", which is one of the Italy's largest daily-interest newspaper, had an average daily circulation of 504,098 in 2009. The "Dainik Jagran" of India has daily readership of about 16.370 millions, according to Indian Readership Survey 2012 Q4. These statistics [source: Wikipedia] certainly prove that the newspaper delivery problem (NDP) is one of the largest vehicle routing problems in terms of the number of units distributed.

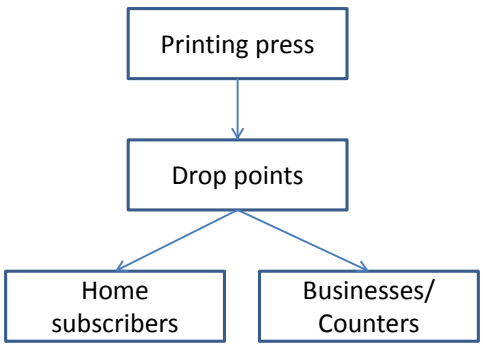

Figure 2: Newspaper distribution process

As shown in the Figure 2, the newspaper distribution process involves the downstream movement of newspapers from the printing press to the readers via a two-echelon process. In the first echelon, newspapers are transferred to the drop points. The final distribution takes place from these drop points to the ultimate customers in the second

echelon with the help of some transportation facilities, such as vans, cars and carriers. Some general features of the NDP in Norway (or similar countries) observed by Hasle et al. (2011) have been listed below.

- There are thousands of geographically scattered customers that are served from one drop point.
- Newspapers are delivered by either cars or pedestrians.
- For each subscriber, there is a service time.
- Route duration is the main constraint.
- Car routes are usually open with return to the drop point.
- Pedestrian routes are closed as trolleys are returned to the drop point.
- Travel times between subscribers & between subscribers and the drop point may not be symmetric due to road topography and traffic issues.

See Golden et al. (2002) for a detailed survey of the early (until 1996) papers on the newspaper delivery vehicle routing problem (NDVRP). It contains nine papers that focused on the first part of the distribution process, modelling the problem as node routing. Some other papers on the NDVRP since 1996 include the work of VanBuer et al. (1999), Song et al. (2002), Cunha and Mutarelli (2007), Russel et al. (2008), Boonkleaw et al. (2009) and Eraslan and Derya (2010). These papers also focused on the first echelon of the supply chain, using node routing formulation. There are, however, several cases where the NDVRP can not be considered as a node routing problem. As for instance, the distribution area of Euro Press (EP), a publisher/distributor of newspapers and other publications in a European country, is so dense that the problem is seen as an arc routing problem with two-sided service of the street segments [Toth and Vigo (2002)].

[Hasle (2012)] clearly stated: "The assumption that all point-based demands can be aggregated into edges or arcs may be crude in practice. It may lead to solutions that are unnecessarily costly, as partial traversal of edges is not possible. An industrial route planning task may cover areas that have a mixture of urban, suburban, and rural parts where many demand points will be far apart and aggregation would impose unnecessary constraints on visit sequences. A more sophisticated type of abstraction is aggregation of demand based on the underlying transportation network topology. Such aggregation procedures must also take capacity, time, and travel restrictions into consideration to avoid aggregation that would lead to impractical or low quality plans. In general, such procedures will produce a MCGRP/NEARP instance with a combination of demands on arcs, edges, and nodes."

1.1.1.2 Urban waste collection

The *waste collection* process is one of the crucial parts of the waste management practice. It is concerned with the transfer of solid wastes or recyclable materials from the site of use and disposal to the point of treatment or landfill by garbage vehicles. The design of vehicle routes in a waste collection vehicle routing problem (WCVRP), especially in the urban areas, is considered as one of the most important and challenging VRPs as it is linked with the public health and several environmental issues. Other factors, such as collection frequency (daily or weekly), large volume of waste and complex road network, greatly increases complexity of the task. The WCVRP is mainly divided into following three classes.

- ***Roll-on roll-off***: The Roll-on roll-off WCVRP deals with pickup, transportation, unloading and drop-off of large containers, which are typically found at construction sites and large shopping centers [Toth and Vigo (2002)]. The size of containers may range from 20 - 40 loose yards. Due to the large size, only one container is generally serviced at a time. The trip of a vehicle can be of two kinds: *round and exchange*. In the round trip, the vehicle picks up a full-container, transports it to the landfill for emptying and returns the empty container to the site of use. In the latter case, on the other hand, the vehicle picks up an empty container at the landfill, carries it to the site for exchanging with a full-container and returns back to the landfill.

- ***Commercial***: The commercial WCVRP involves in collecting solid wastes and recycling materials from commercial establishments, such as small shopping malls, restaurants, offices and schools These organisations deposit garbage items in a comparatively smaller container (about 8 loose yards) placed at the side of the road. Furthermore, these commercial organizations are generally scattered through the metropolitan area. This facilitates point-to-point marking of sites, and modelling the problem as a node routing.

- ***Residential***: A residential WCVRP, on the other hand, is concerned with the design of garbage vehicle routes to service customers residing in the densely populated residential areas. The vehicles move along the streets to collect garbage bags placed in front of houses and therefore, it has mainly been formulated as the ARP in literatures. The exact locations of customers are not needed as in the case of commercial/Roll-on-roll-off WCVRP. Moreover, the number of customers served in the residential problem is much higher than that of the commercial and Roll-on-roll off WCVRPs.

In most of today's modern cities, the waste collection process requires the movement of garbage vehicles in the residential areas as well as to the commercial spots for more profitable and efficient operation. Moreover, real road networks are generally mixture of both kinds of links: *uni-directional* and *bi-directional* streets. Neither the VRP nor the ARP model are applicable for such cases. The MCGRP would be the most appropriate model to handle such integrated problems as it is compromised of all three required elements: *nodes, edges and arcs*.

1.2 Solution Methods

The MCGRP belongs to the class of combinatorial optimization problems, where the goal is to find an optimal solution from a finite set of many alternative solutions. It is an integrated model of the VRP and the ARP, which are NP-hard (non-deterministic polynomial-time hard) combinatorial problems. Hence, *exact algorithms* are applicable to only small size instances of the MCGRP. To solve large instances simulating real-life situations, *heuristics* and *meta-heuristics* seem to be pragmatic approaches. These methods can produce near optimal solutions in a reasonable computational time, but do not provide any upper limits on deviation from the optimality like *approximation* algorithms. The computationally efficient approaches for the MCGRP (or any combinatorial optimization problem) can be broadly classified into two groups:

- *Local search*
- *Evolutionary algorithms*

The local search based methods start with a single solution, whereas evolutionary algorithms begin the search with a set of solutions. However, there is a common thread between these optimization techniques. They iteratively improve a solution/set of solutions by applying some *operators* in each iteration. In the context of the local search, an operator is generally referred as *neighbourhood operator/or move*. In the upcoming subsections, the concept of neighbourhood and neighbourhood operator are discussed first. Then, an insight into the working principles of a local search technique and an evolutionary algorithm are presented.

1.2.1 Neighborhood and moves

A move (or neighborhood operator) can be defined as a systematic mechanism of changing the structure of a solution to generate a new one in its vicinity. In order to describe

the notion of a neighborhood N, let's consider a solution s and an operator μ. Suppose that, operation of μ on s produced a neighboring solution s', that is, $s' \in N(s)$. Let $\beth(s)$ is the set of all possible operators that can be implemented on s. Then, the neighborhood of s can be represented as: $\{N(s) = s \oplus \mu; \mu \in \beth(s)\}$, where $s \oplus \mu$ stands for the transition of s to the neighboring solution s' [Gómez-Villouta et al. (2010)]. For an effective implementation of neighborhood operators, a solution of the problem must be represented in such a way that it can include problem specific features.

0	6	4	5	9	0	3	1	10	2	8	7	0

Figure 3: A sample solution string for MCGRP

A possible solution representation for the MCGRP could be as shown in the Figure 3. It has been coded as a *permutation* of 10 tasks (represented by positive integers) with the trip delimiter (0: depot node). The string contains two vehicle routes (sequences of tasks between the depot node). The sequence of tasks also represents their processing order by a vehicle. In the Memetic NSGA-II, detailed in the third part of this thesis, inter-route (between two different routes) version of following three commonly used VRP operators have been utilized.

- ***2-opt***: This neighborhood operator was first proposed by Croes (1958), although the basic version had already been suggested by Flood (1956). It involves in eliminating two existing connections between tasks and reconnecting the broken paths in some other way by inserting two new connections. The Figure 4 illustrates the working process of the standard 2-opt method on an example MCGRP solution (depicted in the Figure 3). In Figure 4, j is a randomly chosen task and j' is its successor task. The task denoted by j'' is another randomly selected task in the different tour and j''' is the task right after it. Two connections - between tasks 4 & 5 and tasks 10 & 2 - have been deleted. And, two new connections have been inserted between tasks 4 & 10 and tasks 5 & 2. The two new tours can be constructed as:

  1. 0–j–j''–0 (via task 1)
  2. 0–j'–j''' –0 (via task 8)

- λ-***interchange***: This method was brought by Osman and Christofides (1994) for the capacitated clustering problem. Let's Consider a MCGRP/VRP solution consisting of a set of routes $S = (R_1, \ldots, R_p, \ldots, R_q, \ldots, R_n)$. The λ-interchange

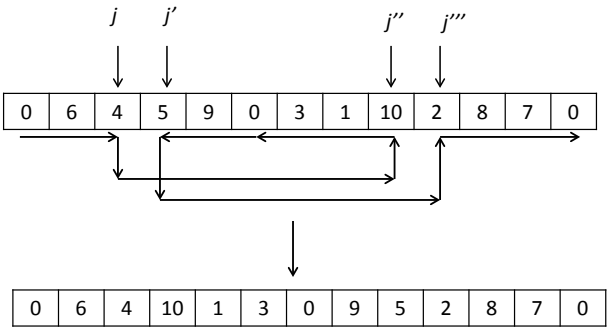

Figure 4: 2-opt illustration

between any two routes (R_p & R_q) is performed by replacing a subset of tasks (S_p) of R_p by another subset of tasks (S_q) of R_q. The cardinality of S_p & S_q should be less than or equal to λ. In the present research work, this operator has been used with $\lambda = 1$. If it is applied to the two routes displayed in the Figure 5 with $S_p = j$ and $S_q = j''$, then the resulting tours will be as shown in the bottom string in the Figure 5. The positions of tasks 4 & 10 have been exchanged.

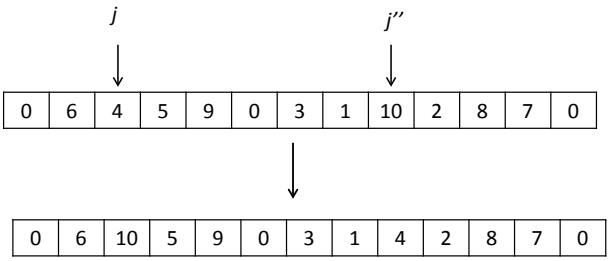

Figure 5: λ-interchange illustration

- ***Re-insert***: The reinsert neighborhood operator was conceived by Savelsbergh (1992) for the VRP with time windows. As the name suggests, it removes a task from its current position and inserts right after another task. The process has been graphically explained in the Figure 6. As it can be seen, task j has been moved after j''.

Some other neighborhood operators/moves that are frequently used to solve VRPs include *Or-opt*, *Cross exchange* and *3-opt*. In the Or-opt method, introduced by Or (1976), a chain of l consecutive tasks is relocated to another place. On the other hand, cross-exchange move, designed by Taillard et al. (1997), involves in exchanging two segments

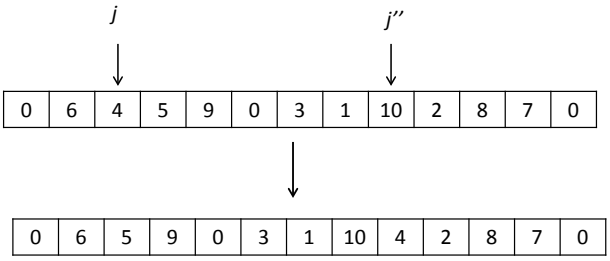

Figure 6: Re-insert illustration

of any length (or less than a prefixed length) from two different routes. The 3-opt [Lin (1965)] move is an extension of the 2-opt procedure. It works exactly in the same fashion, but deals with the removal and insertion of three connections.

1.2.2 Local Search

In computer science, *local search* is defined as an iterative algorithm, which is used to solve hard combinatorial optimization problems [Wikipedia]. It involves in transiting from one solution to another in the search of the optimal solution, using a *neighborhood operator*. A template of the basic local search has been provided in the Table 1. Starting with an initial solution, the current solution is repeatedly replaced by a better neighboring solution. The process is continued until a stopping criteria is met (e.g., the maximum number of iterations, an upper limit on computation time and the number of iterations without improvement). The final output is the best solution that minimizes (or maximizes) an objective function most. A local search algorithm mainly consists of four elements: a starting solution, a neighborhood operator, a solution updating rule and a termination criteria.

step 1:	Generate an initial solution (s)
step 2:	Find $s'(s' \in N(s))$ using a neighborhood operator μ
step 3:	$s \leftarrow s'$ (if s' is better)
step 4:	Repeat *step* 2 - *step* 3 until (*stopping criteria*)

Table 1: Template of local search

Initial solution: An initial solution for beginning the local search is a pivotal component in the algorithm. It can be generated uniformly at random or by means of any fast greedy construction heuristics. As for instance, the saving algorithm of Clarke and Wright (1964) or the sweep algorithm proposed by Gillett and Miller (1974) can be

utilized to build a reasonable good starting solution for the MCGRP. It has been experimentally observed that starting the search with a good quality solution brings several advantages. The algorithm takes comparatively lesser computational time to reach at the local optima. Furthermore, the quality of the final solution also gets better.

Acceptance rule: Another important ingredient in a local search procedure is the solution acceptance rule. In fact, it is the only component which can maintain the balance between intensification and diversification in the local search based meta-heuristics. A straight forward approach is to accept only better moves. This strategy promotes intensification, but leads the search towards the nearest local optimum solution if the landscape has several *"valleys of attraction"* as shown in the Figure 7. The local search based meta-heuristics use different strategies to jump out of the local optima. For example, Simulated Annealing [Kirkpatrick et al. (1983)] exerts a little randomness in the selection process by accepting non-improving moves with some probability. In Tabu Search [Glover (1986)], a *tabu list* is maintained that stores the recently visited solutions and the search is prevented from returning to these solutions for a fixed number of iterations. This is how the search is driven into different directions in Tabu search to avoid a local optima. One can also restart the search from a random position, which is indeed an effective way to explore different parts of the search space.

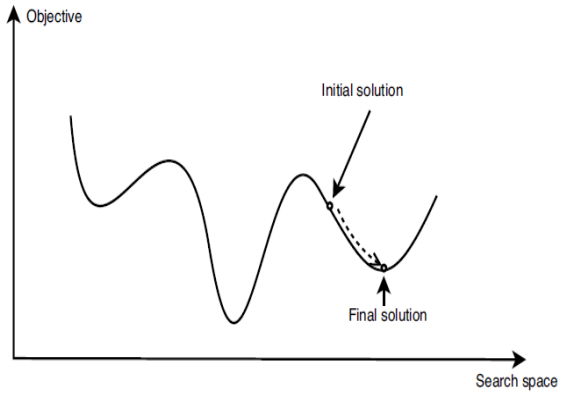

Figure 7: Search space

Neighborhood operator: The performance of a local search algorithm heavily depends on the design of a neighborhood operator. Its strength (i.e, the degree of perturbation) should be moderate. If it is too high, a neighborhood function will behave like a random solution generator. In case it is too low, the structural changes will not be high enough to create a neighbor solution possessing different properties. A good operator is thus the one which can act in between random and deterministic perturbations. More importantly, if more than one operators are used, then they should be applied in such a way that one can not undo the positive contributions of other(s).

1.2.3 Evolutionary Algorithm

Darwin (1859) in his seminal work "*On the origin of species by means of natural selection*" incepted an idea about the evolution of species. Darwin stated: "it follows one general law, leading to the advancement of all organic beings, namely, multiply, vary, let the strongest live and the weakest die". Later, based on this principle of natural selection/survival-of-the-fittest, three main forms of the Evolutionary Algorithm (EA henceforth) were developed:

- Evolutionary programming [Fogel et al. (1966)]
- Genetic Algorithm [Holland (1975)]
- Evolution strategies [Rechenberg (1973) and Schwefel (1977)]

The term EA, in fact, stands for all population based meta-heuristic optimization techniques whose functioning mimics the Darwinian's evolution principle. EAs have been capable of producing optimal/near-optimal solutions on varieties of research problems with complex and multi-modal search spaces. A general framework of an EA has been presented in the Table 2.

step 1:	Generate an initial population of solutions (P_t)
step 2:	Evaluate fitness values of each solution in P_t
step 3:	Mating selection
step 4:	Recombination
step 5:	Mutation
step 6:	Form a child population (Q_t)
step 7:	Create a new population (P_{t+1}) from $P_t \cup Q_t$
step 8:	Set $P_t = P_{t+1}$
step 9:	Repeat *step* 3 - *step* 8 until (stopping criteria)

Table 2: Template of an EA

As shown in the Table 2, an EA begins with the initialization of a population of candidate solutions. The quality of each of these solutions is evaluated with respect to a fitness function. Following this, the algorithm enters in an evolutionary cycle. Firstly, some promising members of the current population (P_t) are selected to create a pool, so-called mating pool. The solutions of the mating pool undergo recombination and mutation to produce a population of child solutions/off-springs (Q_t). A new population (P_{t+1}) of fittest individuals is then created from solutions of the current population and the child population. The new population becomes the current population and is passed to the

next cycle to continue the search process. It is quite clear that following components must be specified in order develop an architecture of the EA.

- Solution representation of the optimization problem
- Population initialization
- Fitness function
- Parent selection mechanism (mating selection)
- Recombination and mutation operators
- Survivor selection (or environmental selection) method

Representation of a solution: This is the first step in the designing of any optimization algorithms (local searches or EAs), where a solution of the problem is coded into a data structure that can be decoded by a computer program. In the context of an EA, a legal solution of the problem is called *chromosome* and each member of a chromosome is referred as *gene*. The design of a chromosome is an extremely crucial task in EAs as it is directly linked with other issues, such as the choice of crossover & mutation operators. Moreover, it also governs the size and shape of the search space. An inappropriate representation of the solution will eventually result failure of the whole algorithmic model. The most suitable form of representation for ordering (or queuing) problems is the *permutation* encoding. As for instance, for the Travelling Salesman Problem (TSP), a good representation is a sequence of integers showing the visiting order of cities. For VRPs, following two representation schemes have been widely used in literatures.

- ***Fixed length***: In this method, a VRP solution is coded as a fixed-length string of integers representing customers/tasks, as shown in the Figure 8. The string can be transformed into a complete VRP solution by simply dividing it according to the vehicle capacity or by any heuristic methods (such as Ulusoy's heuristic [Ulusoy (1985)] and *Split* [Prins et al. (2009)]). In the Figure 8, the bottom string contains three tours, each begins and ends at the depot node (represented by zero). The sequences of positive integers in the tours show the visiting order of tasks.
- ***Variable length***: As the name suggests, the length of the solution data structure varies during the search process. A variable length chromosome is a more precise model for the VRP as it gives the complete information about solution of the problem. An example of the variable length representation for the VRP has been shown in the Figure 9 [Tan et al. (2006)]. It contains routes with customers arranged in the order they have to be served. In the Figure 9, 10 customers have been assigned to three vehicles.

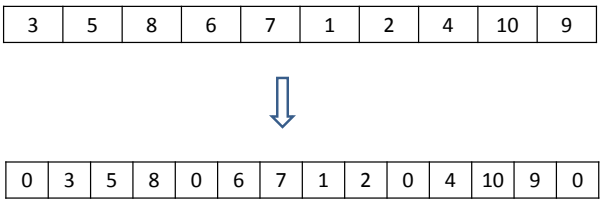

Figure 8: Fixed length chromosome

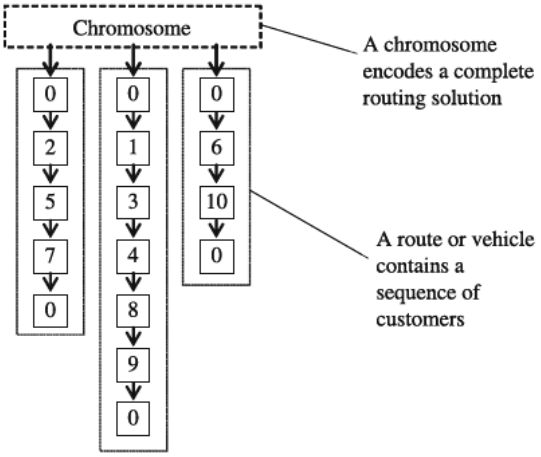

Figure 9: Variable length chromosome

Population initialization: With the representation of solution on hand, the next crucial task in EAs is to create an initial population of candidate solutions to start the search. A general approach is filling the initial population by randomly generated solutions. This method favours *unbiased* exploration of different parts of the search space. However, poor random solutions require more computational efforts and may also cause premature convergence to a suboptimal point. Experiments have demonstrated that placing initial solutions in the fruitful regions of the search space helps improve the online per-

formance (i.e., time-to-solution) and quality of the final solution. Several other methods have also been tried. For example, to solve the single-objective MCGRP, Prins and Bouchenoua (2004) created a mixed initial population, which contains randomly generated chromosomes and also a few good solutions created by fast greedy heuristics. Furthermore, the size of the population is generally fixed constant throughout the search. In fact, it is an important parameter of an EA that must be set appropriately according to the size of the search space. A higher population size will consume a large amount of computation time. Whereas, if the size is too small, the EA will not be able to find (or reach near) the optimal solution. The optimal population size is therefore the point at which a satisfactory balance could be achieved between solution quality and computational time.

Fitness Evaluation: The fitness value of a solution indicates how promising it is in comparison to others. The fitness of a chromosome is evaluated with respect to an objective function, which represents the ultimate goal of the problem solving. For a single-objective optimization problem, the fittest solution will optimize the objective function most. Note that, the fitness function and the objective function can be different for the same problem. For example, in a single-objective minimization problem, a solution with the lowest objective function value will have the highest fitness value. Therefore, a fitness function must be derived from the objective function to apply an EA on mono-objective minimization problems. A simple mapping method is shown below.

$$F(x) = \frac{1}{1 + f(x)} \tag{1.2.1}$$

In the equation (1.2.1), $F(x)$ and $f(x)$ are the fitness and objective values of a solution x. For mono-objective maximization problems, the fitness function can be considered equivalent to the objective function. The section 1.4 will discuss various fitness assignment strategies for multi-objective optimization problems.

Mating selection: As the name suggests, mating selection is the process of choosing solutions from a population that mate with each other to produce off-springs. All these selected solutions are separately stored in a *mating pool*, whose size is set equal to the size of the population. A selection operator should function in such a way that it could maintain an appropriate amount of the selection pressure (the degree of being biased towards the better individuals). The selection pressure largely controls the convergence rate of an EA. More is the selection pressure, better will be the convergence rate. However, if the selection pressure is too high, only fittest solutions will get selected and premature convergence to a suboptimal point may occur. On the other hand, if it is set too low, then the rate of convergence towards the optimum will be slower. The most commonly used selection methods are *roulette wheel* and *tournament selection*. Both of

these methods favour the fittest individuals. Their working processes have been detailed below.

- ***Roulette wheel selection***: This method was developed by Holland (1975). It is also known as the *fitness proportionate selection* as the probability of selection of an individual depends on its fitness value. The most fit members of the population get highest selection probability. To understand the underlying concept, imagine a roulette wheel of area 1 $unit^2$. Each individual i $(i = 1, 2, 3, \ldots, n)$ of the population is assigned some portion of the wheel. The area of the wheel allocated to a solution i is directly proportional to its fitness value $\left(\frac{F_i}{\sum_{i=1}^{n} F_i}\right)$. Thus, fittest individuals will get larger part of the wheel. The wheel is rotated and the solution corresponding to the segment on which it stops is selected. The pseudocode shown in the Table 3 transforms this principle into a valid computer program. First, solutions are sorted in the ascending order of their fitness values. Then, the probability of selection of each solution is determined by dividing its fitness value by the sum of fitness values of all solutions. Next, the cumulative probability of selection for each solution is calculated. Subsequently, the method enters in a loop, which stops once the mating pool is full. In each iteration of this loop, a random number (r) is generated in the range [0, 1]. If r is less than the cumulative probability of selection of the first solution (in the arranged sequence), then it is selected. In case, r lies between cumulative probabilities of two consecutive solutions, then the solution having higher cumulative probability of selection is inserted into the mating pool.

- ***Tournament selection***: In the tournament selection method, t (tournament size) solutions are randomly drawn from the population and the fittest one among them is selected with some probability. The t individuals are then put back into the population. A general template of the tournament selection has been shown in the Table 4. In the binary tournament selection, only two players are involved. Furthermore, in the deterministic version of the tournament selection, the winner of tournament is always the fittest one.

With regards to maintaining the selection pressure, the tournament scheme is more robust than the roulette wheel procedure. The intensity of the selection pressure is directly proportional to the tournament size. As the number of competitors increases, the selection pressure rises and weak individuals get smaller chance to get selected. The effects of different selection operators on the convergence speed have been studied by Goldberg and Deb (1991).

Recombination/Crossover: This is the process of generating off-springs by mixing genetic properties of two or more solutions. The solutions to be recombined are selected

Arrange solutions in the ascending order of their fitness values;
Calculate the probability of selection of each solution i as $P_i = \frac{F_i}{\sum_{i=1}^{n} F_i}$;
Set $C_1 = P_1$ C_1: cumulative probability of selection of the first solution;

For each $i(> 1)$
set $C_i = 0$;
End For

For ($i = 2$ *to* n)
$C_i = P_i + P_{i-1}$;
End For

Repeat
Generate a random number $r \in (0, 1)$;
If ($r < C_1$), then select the first solution ;
Else
For ($i = 1$ *to* $n - 1$)
if ($C_i \leq r < C_{i+1}$), then select solution $i + 1$;
End For
End If
Until (the size of the mating pool)

Table 3: Roulette wheel selection

set: p (probability of selection)
Repeat
Draw t solutions from the population t: tournament size ;
Generate a random number $r \in (0, 1)$;
If ($r < p$)
select the fittest one;
Else
select the least fittest one;
End If
Put drawn solutions back into the population ;
Until (the size of the mating pool)

Table 4: Tournament selection

from the mating pool and the operation is executed with some probability. A crossover probability in fact indicates that how many solutions will be given chance to produce child solutions. For example, a crossover probability of 0.8 suggests that 80% solutions of the mating pool will participate in the off-springs generation process. The crossover operation in EAs can help in both exploring and exploiting the search space. Some of

the well-known crossover operators for *permutation chromosomes* are being described below.

- *Partially Mapped Crossover* (**PMX**): The PMX, which was designed by Goldberg and Robert Lingle (1985), is a two point crossover operator with an additional repairing procedure. Given two parent solutions, an offspring is created as follows. First, two crossover points are randomly selected and the substring defined by these two points from one parent is exactly copied to the child solution at the same position. Then, the remaining positions in the child solution are filled by elements from the alternative parent. An infeasibility is tackled with the help of a mapping relation, which is defined by the previously selected sub-strings in the parent solutions. This whole procedure has been graphically explained in the Figure 10.

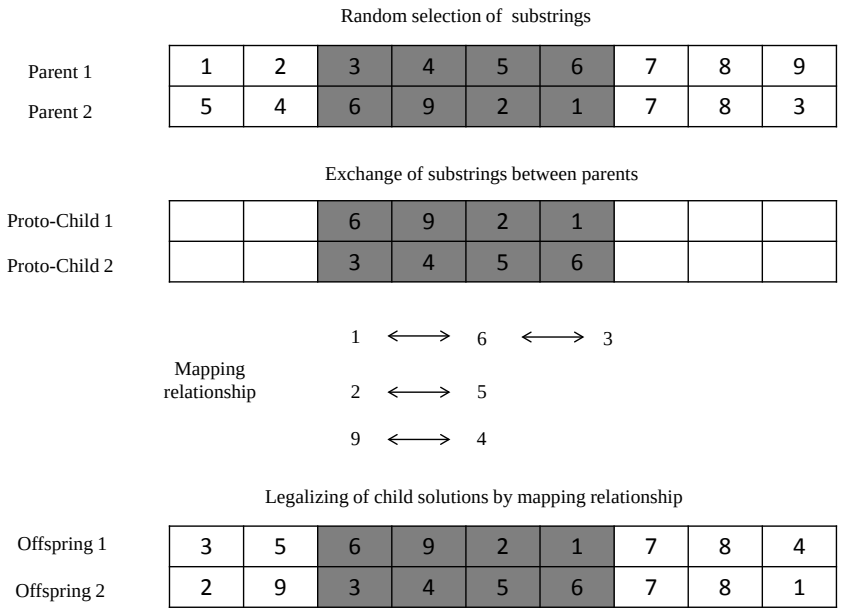

Figure 10: An illustration of PMX

- *Order Crossover* (**OX**): The OX operator is also a two point crossover like the PMX, but works in a slightly different fashion. The Figure 11 has been supplied to help visualize the working process of OX developed by Oliver et al. (1987). As in the case of PMX, first a randomly selected sub-string from one parent is copied to the offspring without disturbing the order and positions of elements. Then, the other parent is scanned (from right after the second cut-point to the end and then from the beginning to the second cut-point), while inserting the missing elements

in the offspring, starting at the the position just after the second crossover point. If the end of offspring-string is reached, then the insertion process is resumed from the first position.

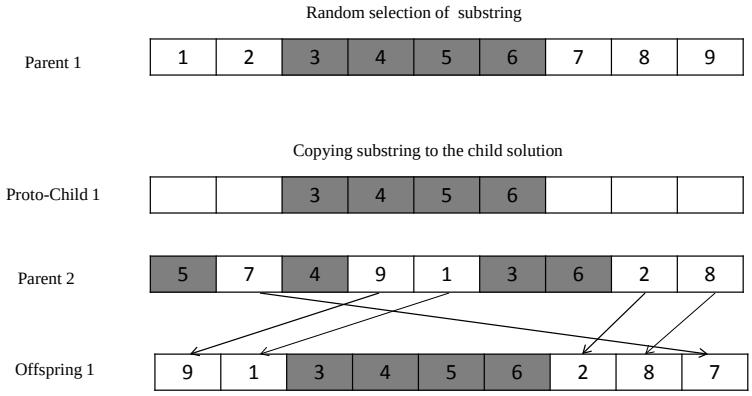

Figure 11: An illustration of OX

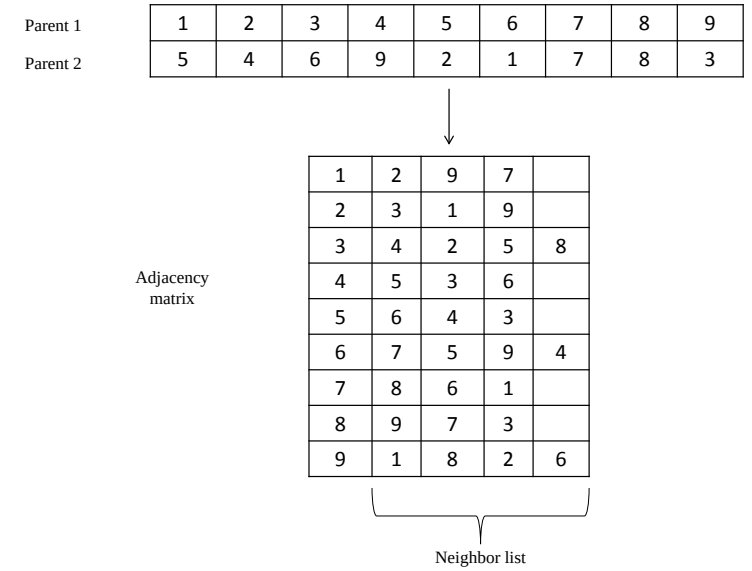

Figure 12: Formation of adjacency matrix

- *Edge Recombination* (**ERX**): The ERX operator was conceptualized by Whitley et al. (1989). It focuses on the *adjacency relation* instead of the order of elements

in a sequence. It uses an adjacency matrix - which stores immediate neighbors of all elements from parent solutions - to create an offspring with the minimum number of new relations. An example of the adjacency matrix has been shown in the Figure 12. The neighbor lists give informations about edges/links of the network. After forming an adjacency matrix, a child solution is created as follows. First, an element from the parent starting points is randomly selected. This element is deleted from neighbor lists. Then, among the neighbor members of this previously selected element, a new element with the smallest neighbor set is selected. The ties are broken arbitrarily. The process is repeated until all elements have been selected.

Mutation: Once an offspring is created through recombination, its structure is further changed slightly by altering genes (or their positions) in the mutation/education process. The mutation is also done with some probability and helps in exploring the search space. The design of mutation operators is a little tricky, especially in the problems where a chromosome is represented as a sequence of elements without duplication. The two widely used mutation operators for a permutation chromosome are:

- *Swap*: In this method, two elements on the chromosome are randomly selected and their positions are exchanged.
- *Inversion*: The Inversion operator, on the other hand, takes a segment of the chromosome out and puts back at the same site but with inverted direction. An illustration of the inversion mutation has been shown in the Figure 13. The substring defined by the two randomly selected points on the chromosome has been reversed.

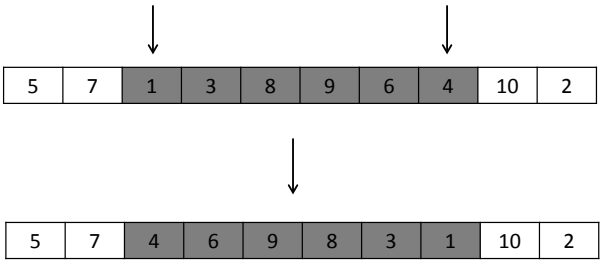

Figure 13: An illustration of Inversion mutation

Environmental/Survivor Selection: This is the final stage of an EA cycle in which a new population is formed for the next generation of search. Since the size of the population

is kept constant; therefore, a selection is made among the parent and offspring solutions. Following the principle of survival-of-the-fittest, basic EAs directly select the top best individuals to fill the new population.

Stopping criteria: The EAs are stochastic search techniques; therefore, the chances of finding the optimal solution is low, especially in high dimensional problems. The stopping criteria should be set in such a way that high quality solutions can be obtained within a reasonable computational time. An EA is generally allowed to run over a number of generations until a termination criteria is met. The most commonly used termination conditions are as follows:

- The maximum allowable CPU time.
- A limitation on the number of function evaluations.
- The maximum number of iterations with fitness improvement under a threshold value.
- A satisfactory fitness level has been achieved.

Parameters: EAs require the right settings of several parameters (shown below in the bulleted list) to find a near-optimum solution. The optimal parameter values set does not exist for any meta-heuristic [Talbi (2009)]. As discussed above, the best setting is the one with which an EA can output a high quality solution in a reasonable computational time.

- Population size
- Tournament size (for tournament selection)
- Crossover probability
- Mutation rate
- Termination criteria

The above parameters can be adjusted in two ways: *tuning* and *control*. In the parameter tuning, experiments are performed with several possible values. The best ones are selected and kept constant throughout the run of the algorithm. Whereas, in the control method, the algorithm begins with initial parameter values that are changed dynamically or adaptively during the evolutionary process.

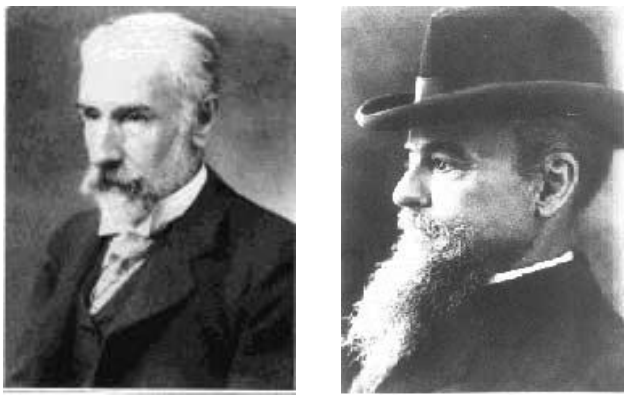

Figure 14: (left) Francis Y. Edgeworth (1845-1926) and (right) Vilfredo Pareto (1848-1923)

1.3 Multi-objective Optimization Problem

The multi-objective optimization problem (MOP), which is sometimes also called as the vector optimization problem, deals with the simultaneous optimization of two or more objectives (normally conflicting with each other). The roots of multi-objective optimization originate from the original work of Edgeworth (1881). In fact, Edgeworth first proposed the notion of *optimum* for multi-criteria economic decision making. For the multi-utility problem with two hypothetical consumer criteria P and Π, Edgeworth stated: "It is required to find a point (x, y) such that in whatever direction we take an infinitely small step, P and Π do not increase together but that, while one increases, the other decreases." Later, Pareto (1906) generalized this notion as (English version): "The optimum allocation of the resources of a society is not attained so long as it is possible to make at least one individual better off in his own estimation while keeping others as well off as before in their own estimation." Afterwards, this theory has intensively been used in mathematics and engineering. And, in the last few decades, the MOP has become a prominent research area in various scientific fields. See Stadler and Dauer (1992) for more detailed history of MOPs.

1.3.1 MOP basics

Definition 1. *Consider an optimization problem with $n(\geq 2)$ objectives, which are, without loss of generality, all to be minimized. A MOP can be defined as follows:*

$$MOP = \begin{cases} min & F(x) = (f_1(x), f_2(x), ..., f_n(x)) \\ s.c. & x \in S \end{cases} \tag{1.3.1}$$

Where, $x = (x_1, x_2, \ldots, x_l)$ represents a solution vector of length 'l' in the *decision space*, and S is a feasible decision-variable space. The vector $F(x) = (f_1(x), f_2(x), \ldots, f_n(x))$ contains objectives to be optimized at the same time. The space to which $F(x)$ belongs is called *objective space*.

Definition 2. *Pareto dominance: A solution x is said to dominate another solution y (denoted by $x \prec y$) if and only if $\forall i \in (1, 2, 3, ..., n)$, $F(x) \leq F(y) \wedge \exists\, i \in (1, 2, 3, ..., n)$: $F(x) < F(y)$.*

Definition 3. *Pareto optimality: A solution x^* is Pareto optimal if for every $x \in S$, $F(x)$ does not dominate $F(x^*)$.*

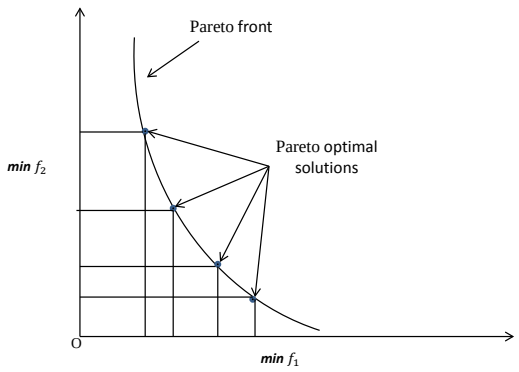

Figure 15: Pareto front

The definition of Pareto optimality directly comes from the seminal work of Edgeworth (1881) and the theory given by Pareto (1906). A Pareto optimal solution denotes that an objective can not be improved without deteriorating atleast one other objective. There exists no solutions in the dominance cone of a Pareto optimal solution (see Figure 15). The definition of Pareto optimality leads to the possibility of several alternative Pareto optimal solutions as can be seen in the Figure 15. These solutions are also called as non-dominated solutions, trade-off solutions and compromised solutions. The whole set of non-dominated solutions is referred as the *Pareto set*. The image of this set in the objective space is termed as the *Pareto front*.

Definition 4. *Ideal vector: The objective vector $F(x) = (f_1(x), f_2(x), ..., f_n(x))$ is said to be an ideal vector if $\forall i \in (1, 2, 3 \ldots, n)$ $f_i(x)$ is the minimum possible value in the entire Pareto-optimal set.*

Definition 5. *Nadir point: The objective vector* $F(x) = (f_1(x), f_2(x), ..., f_n(x))$ *is said to be a nadir point if* $\forall i \in (1, 2, 3 \ldots, n)$ $f_i(x)$ *is the maximum possible value in the entire Pareto-optimal set.*

The ideal and nadir points are essential in many ways. For example, these two extreme points are used for determining the range of a Pareto front along each objective and visualizing the trade-off informations through bar charts, petal diagrams, etc. [Miettinen (2003)]. They are also used for scaling objectives of different magnitudes in the same range [Miettinen (1999)], which is indeed necessary for multi-objective optimization algorithms to perform an unbiased search.

1.4 Design issues: Multi-objective Evolutionary Algorithm

A MOP seeks to find the optimal Pareto set constituting of non-dominated solutions of equivalent quality. The Pareto set should be uniformly distributed and diverse so that a decision maker can easily select the one best compromised solution. EAs have had the ability to approximate a Pareto set in just a single simulation run as they operate on several solutions simultaneously. However, single-objective EAs are not applicable on MOPs due to the need of a set of compromised solutions. To design a multi-objective evolutionary algorithm (MOEA), three major components are required. First, a fitness assignment strategy to select non-dominated solutions. Second, a diversity preservation method to avoid premature convergence at a suboptimal Pareto front and obtain an evenly spaced Pareto set. Last but not the least, an elitist mechanism for preventing the loss of global non-dominated solutions found during the search. In addition, one also has to define a performance indicator to check the quality of an approximate Pareto set generated by a MOEA. In the following subsections, existing ideas to tackle these issues have been presented.

1.4.1 Fitness assignment

The fitness assignment strategy in MOPs is not as straightforward as in single objective optimization problems (SOPs). In a single objective minimization (maximization) problem, the fitness of a solution is inversely (directly) proportional to the objective function value. A MOP deals with several conflicting objectives and therefore, the fitness value must be derived from all objectives. More importantly, it should drive the evolutionary

search towards the optimal Pareto set. Some of the commonly used fitness assignment methods are being discussed below.

Weighted linear aggregation: This is one of the first methods developed to formulate a fitness function for MOPs. It actually transforms a MOP into a SOP by combining various objective functions into a single form, mostly in a linear way as shown in the equation (1.4.1). The traditional single-objective local searches and EAs can be applied on the equation (1.4.1).

$$f(x) = \sum_{i=1}^{n} c_i \times w_i \times f_i(x) \tag{1.4.1}$$

Where, $w_i \in (0, 1)$ $[\sum_{i=1}^{n} w_i = 1]$ is the weight assigned to the objective f_i. The weight vector is determined by some prior knowledge of the problem and represents a hyperplane (a line in the 2-dimension case) in the objective space. A Pareto optimal solution lies on the border of the Pareto curve, and is defined as the point of contact of this hyperplane and the curve. The weighted linear aggregation concept is simple and computationally efficient too, but fails to detect non-dominated solutions lying on the concave portion of the Pareto curve.

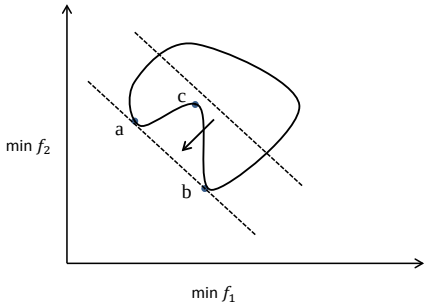

Figure 16: Concave Pareto curve

To demonstrate what has been stated above, consider a Pareto curve as shown in the Figure 16. The point 'c' lies on the concave portion, whereas 'a' & 'b' are on the convex part of the curve. The objectives are to be minimized and the hyperplane has been shown by the dotted line. As the combined objective function is minimized, the hyperplane shifts downwards. It can be seen that point 'c' has been missed and the final contact points are 'a' & 'b'. No matter how the weights are initialized, all other non-dominated solutions between the points 'a' and 'b' can not be found [Talbi (2009)]. Another disadvantage of this method is that it generates only one solution in a single simulation run.

$\in$-***constraint method***: The $\in$-constraint method, conceived by Haimes et al. (1971), is another traditional scalar approach for fitness assignment in MOPs. It aims to optimize one of the objectives, while considering other(s) as inequality constraints of the optimization problem (see equation 1.4.2).

$$MOP_k(\in) = \begin{cases} min f_k(x) \\ x \in S \\ s.c. f_j(x) \leq e_j \ \forall j = 1, 2, \ldots, n \ \& \ j \neq k \end{cases} \tag{1.4.2}$$

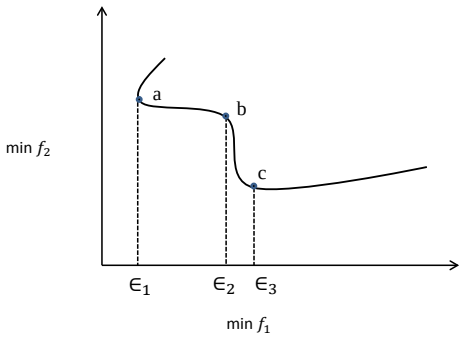

Figure 17: Epsilon constraint method

In the equation (1.4.2), the vector $e = (e_1, e_2, \ldots, e_n)$ represents an upper bound for the objectives. For each e-vector, the $\in$-constraint method will produce one Pareto solution. For exemplification, consider a bi-objective case shown in the Figure 17. The objective f_2 is being minimized while setting upper limits on the objective f_1. As it can be seen, Pareto solutions 'a', 'b' and 'c' have been obtained under the constraints $f_1 \leq \in_1$, $f_1 \leq \in_2$ and $f_1 \leq \in_3$, respectively. Furthermore, unlike the weighted linear aggregation method, this approach is able to find points on the non-convex boundary of the Pareto front. Nonetheless, a major disadvantage of this method is that determining an appropriate e-vector can be a difficult task. Also, it increases complexity of the optimization problem by introducing extra constraints.

Dominance-based ranking methods: In these approaches, a MOP is not converted into a mono-objective one like the weighted linear aggregation and $\in$-constraint methods. Instead, solutions are ranked using the concept of Pareto dominance [Definition 2, section 1.3.1]. As a general rule, lower is the rank, better is the fitness of a solution. The most commonly used dominance-based ranking procedures for multi-objective optimizers are described below.

- ***Dominance rank*** (r_p): The dominance rank of a solution is calculated as the

number of solutions in the population P that dominate it [Fonseca and Fleming (1993)]. The rank of a solution $p \in P$ is calculated as follows:

$$r_p = 1 + |s| \qquad s \prec p,\ s \in P\ and\ s \neq p \tag{1.4.3}$$

- ***Dominance depth***: In this method, population of solutions is decomposed into several fronts of dominance. The non-dominated solutions of the whole population create the first front (F_1) and are awarded rank 1. Among the remaining solutions, those that are non-dominated except by the solutions of F_1 receive rank 2 and construct the second front (F_2). The members of subsequent fronts are extracted in the similar way. In general, members of front $F_f(f > 1)$ are dominated by solutions of the population belonging to the front $F_1 \cup F_2, \cup \cdots \cup, F_{f-1}$. This procedure has been illustrated in the Figure 18.

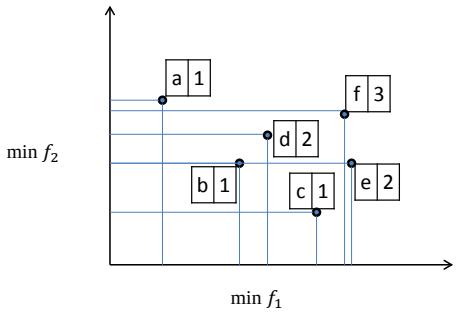

Figure 18: Dominance depth

The solutions 'a', 'b' and 'c' have been given rank 1 as they are non-dominated solutions. The solutions 'd' and 'e' are dominated by only solutions having rank 1 and therefore receive rank 2. The solution 'f' is dominated by solution 'd' (rank 2) and solutions 'b' & 'c' (rank 1) and hence, gets rank 3.

- ***Dominance count*** (c_p): The dominance count of a solution is defined as the number of solutions in the population P that are dominated by it. The fitness of an individual $p \in P$ can be directly obtained by using its dominance count [Zitzler and Thiele (1999)].

$$fitness(p) = c_p = |s| \qquad p \prec s,\ s \in P\ and\ s \neq p \tag{1.4.4}$$

1.4.2 Diversity Preservation

The fitness assignment strategies just described in the previous section drive the search towards the optimal Pareto set, but in no way helps in maintaining diversity among the trade-off solutions. One of the most successful ways to obtain an uniformly distributed Pareto set in MOEAs is by giving higher selection probability to lesser dense/crowded solutions at various selection stages of the algorithm. The density of a solution in MOEAs can be measured by one of the following methods.

- ***Kernel methods***: In these methods, in order to estimate the density i_d of a solution i, the distances d_{ij} between i and all other solutions in the population j are calculated. Then, a kernel function (K) is applied over all the distances. The sum of all kernel function values gives the density of solution i. The kernel methods have been used in NSGA[Srinivas and Ded (1995)] and NPGA[Horn and Nafpliotis (1993)].

$$i_d = \sum_j K(d_{ij}) \qquad j \neq i \tag{1.4.5}$$

- ***Nearest neighbor methods***: Contrary to Kernel methods, this principle does not consider all of the neighbors to determine the density of a solution. Only a set of k^{th} nearest neighbors is used. This technique has been used in the SPEA2 (Strength Pareto Evolutionary Approach 2) algorithm [Zitzler et al. (2002)].

- ***Histogram***: In this approach, the search space is divided into several hyper-grids that define neighborhood of a solution, as shown in the Figure 21. The density of a solution i is computed as the number of solutions in the same box of the grid. This idea has been used in the PAES (Pareto Archived Evolution Strategy) algorithm [Knowles and Corne (1999)].

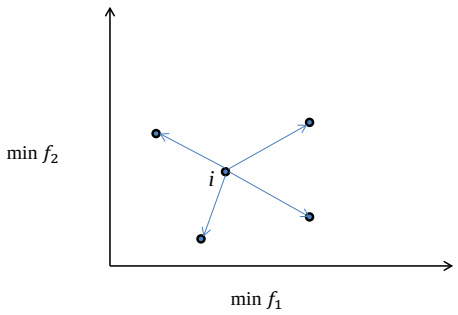

Figure 19: Kernel

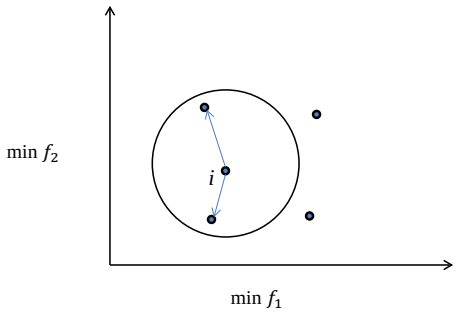

Figure 20: Nearest neighbor

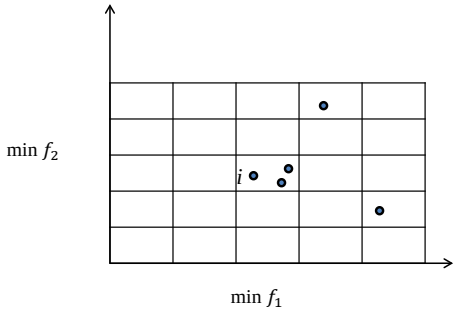

Figure 21: Histogram

In MOPs, diversity is generally desired in the objective space. But, it can also be computed in the decision space. To calculate the density of a solution in the decision space, the Levenshtein edit distance method (or an encoding specific solution similarity measure) can be used for defining the distance between two strings. Løkketangen et al. (2012) discussed various solution similarity measures for VRPs.

1.4.3 Elitism

Elitism is the process of preventing the loss of promising solutions discovered during the search. One of the simple ways to apply elitism in EAs is to select the top best individuals from parent and offspring solutions for the next generation of search. Another technique is to store elite solutions in a secondary population, which is called *archive* in EAs. An archive can be of two kinds:

- ***Passive archive***: The passive archive stays out of the main search engine and is only used to store elite solutions. With the passive mechanism, it is sure that the algorithm will have a monotonically non-degrading performance in terms of the

approximated Pareto set [Talbi (2009)]

- ***Active archive***: The solutions of the active archive participate in the different selection processes (mating/survivor selection) during the evolutionary search. The active mechanism can boost the convergence rate, but may cause premature convergence if the adequate elitist-selection pressure is not maintained.

In MOEAs, an archive is used to store only elite non-dominated solutions and its size is generally bounded. Thus, all trade-off solutions obtained during the search can not be stored. Several strategies were developed to cope with the size restriction of the archive. For example, if the number of non-dominated solutions exceeds the size of the archive, then NSGA-II takes the density information into account to reject some solutions. The SPEA algorithm [Zitzler and Thiele (1999)] uses the average linking clustering method [Morse (1980)] to reduce the size of the non-dominated set to a desired level.

1.5 Performance metrics

The performance of a mono-objective optimizer is judged on the basis of quality of the final single solution and utilized computational resources. In MOPs, the outcome is a set of solutions possessing multiple criteria. This raises difficulty in comparing the effectiveness of multi-objective optimizers. Usually, a Pareto set is assessed on the basis of following three criteria:

- The number of non-dominated solutions in the set.
- Its closeness to the true Pareto set.
- The distribution and spread of the non-dominated solutions.

Several metrics have been proposed to determine the quality of an approximate Pareto set. However, truly speaking, there is still need of a *standard indicator* for the performance evaluation of multi-objective algorithms. The available metrics can be divided into two categories: *unary indicators* and *binary indicators*. The former assigns an approximate Pareto set a scalar value, which signifies its quality in terms of convergence/diversity. A binary performance measure outputs a scalar value by comparing two approximations. Some quality indicators require additional informations, for example, a reference point/set, the ideal vector, the nadir point, etc.

- ***Contribution***: It is a convergence-based binary metric, which measures the percentage of solutions of the combined non-dominated set of two approximations ($PS^* = PS_1 \cup PS_2$) that are provided by individual sets (PS_1 & PS_2). For example, $Cont\,(PS_1/PS^*)$ (equations 1.5.1 – 1.5.2) value of 0.7 indicates that 70 % of the solutions of PS^* are from PS_1 and 30 % are provided by PS_2.

$$Cont\,(PS_1/PS^*) = \frac{(||PS||/2) + ||W_1|| + ||N_1||}{||PS^*||} \tag{1.5.1}$$

$$N_1 = \frac{PS_1}{PS \cup W_1 \cup L_1} \tag{1.5.2}$$

$$Cont\,(PS_1/PS^*) + Cont\,(PS_2/PS^*) = 1 \tag{1.5.3}$$

Where, PS is the set of solutions in $PS_1 \cap PS_2$. W_1 is the set of solutions in PS_1 that dominates some solutions of PS_2. L_1 is the set of solutions in PS_1 that are dominated by some solutions of PS_2.

- ***Generational distance***: The generational distance (GD), suggested by Veldhuizen and Lamont (1998), is also a convergence-based metric. It measures the average distance from an approximate Pareto set (PS) to the true Pareto set (PS^*).

$$GD(PS, PS^*) = \frac{\left(\sum_{i=1}^{|PS|} d_i^q\right)^{1/q}}{|PS|} \tag{1.5.4}$$

$$d_i = \min_{p \in PS^*} \left\{ \sqrt{\sum_{k=1}^{M} \left(f_k(s_i) - f_k(p)\right)^2} \right\} \tag{1.5.5}$$

Where M is the number of objectives. When q is set equal to 2, d_i becomes the Euclidean distance.

- ***Distribution***: Deb et al. (2000) proposed a distribution performance indicator ($\triangle$) for measuring the distribution of non-dominated solutions in an approximate Pareto set(PS). Firstly, the Euclidean distance d_i between consecutive solutions in PS is calculated. Then, the average distance $\bar{d}$ is determined. Finally, $\triangle$ is calculated by the equation 1.5.6.

$$\triangle(PS) = \sum_{i=1}^{|PS|-1} \frac{|d_i - \bar{d}|}{|PS| - 1} \tag{1.5.6}$$

- ***Spread***: The spread indicator (I_s) measures the extent of the spread of the obtained Pareto front. It can be evaluated as shown below.

$$I_s = \frac{\sum_{u \in PS} |\{u' \in PS : ||F(u) - F(u')|| > \sigma\}|}{|PS| - 1} \tag{1.5.7}$$

  In the equation (1.5.7), σ is a neighborhood parameter. I_s gives a value in the range [0-1]. The closer it is to 1, better will be the dispersion of non-dominated solutions in the Pareto set.

- ***Hypervolume***: It is an unary indicator, which can assess the quality of a potentially optimal Pareto set in terms of both criteria (convergence towards the true one and diversity among trade-off solutions). It is measured as the volume of the objective space portion that is dominated by a set of non-dominated solutions. It is calculated with respect to a fixed anti-optimal reference point. With all objectives ($Z_1, Z_2, \ldots, Z_n$) are of minimization type, the reference point can be set as ($1.05 \times Z_1^{max}, 1.05 \times Z_2^{max}, \ldots, Z_n^{max}$) [Talbi (2009)]. Fleischer (2003) proved that hypervolume is maximized if and only if the set of solutions contains only Pareto optima. The Figure 22 has been drawn to demonstrate the calculation of the hypervolume metric in a two-dimensional objective space. The Pareto front contains four non-dominated solutions. The area of the shaded region will give the hypervolume measure. This whole area can be determined by summing up the areas of rectangles drawn using non-dominated points and the reference point, counting dominated space only once.

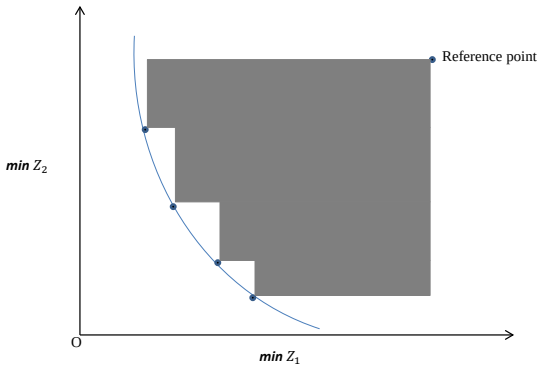

Figure 22: Hypervolume

* * * * *

Part 2

Literature Survey

Literature survey

This part of the thesis presents an in-depth literature review of some *notable* approaches developed to solve single as well as multi-objective models of the VRP, considering different routing mechanisms. However, due to the excessive abundance of published papers for VRPs, this survey has mostly been restricted to research works that used heuristic and meta-heuristic algorithms to tackle the most basic form (i.e., routing problems under the capacity constraint). The whole survey has been divided into three parts: *node-based routing*, *arc-based routing* and *mixed general routing*. Each part has further been broken into different sections based on the number of considered objectives. The main concepts and key features of the reported algorithms have been highlighted.

2.1 Notations

Notation	*Description*
ILS	Iterated Local Search
TS	Tabu Search
GA	Genetic Algorithm
MA	Memetic Algorithm
ACO	Ant Colony Optimization
SA	Simulated Annealing
LSP	Local Search Procedure
HA	Heuristic Algorithm
GRASP	Greedy Randomized Adaptive Search Procedure
ALNS	Adaptive Large Neighborhood Search
NPM	Non-Pareto Method
VND	Variable Neighborhood Decent
VNS	Variable Neighborhood Search

2.2 Node-based Vehicle Routing Problems

A VRP in which locations of customers/tasks on the streets are exactly known and sparse enough to be clearly represented by points, as shown in the Figure 23, belongs to the class of node routing problems. Under the three basic VRP constraints discussed earlier, a node-based VRP is popularly known as the Capacitated Vehicle Routing Problem (CVRP). It is generally represented on an undirected graph, which consists of nodes representing service points and edges connecting them.

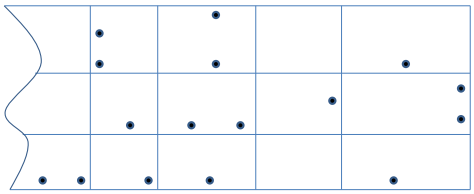

Figure 23: Node routing

2.2.1 Single-objective CVRP

HA1: One of the mostly discussed heuristics in the VRP literature is the saving-based algorithm of Clarke and Wright (1964). Given two customers i and j, saving $s(i, j)$ is calculated as: $s(i, j) = d(i, D) + d(j, D) - d(i, j)$, where D stands for the depot node. The main algorithm starts with an initial solution in which each customer is served alone on a route. Then, savings are calculated for all customer pairs and arranged in a non-increasing fashion. The customer pair (i, j) with the highest saving is considered first. While allocating these customers, link (i, j) is created if constraints do not violate and one of the following two situations emerges.

1. Either i or j has already been assigned to a route and its position is adjacent to the depot. In this case, link (i, j) is added to that same route.
2. Both i and j are already present in two different existing routes and their positions are adjacent to the depot. The routes are merged in this case.

Likewise, the next customer pair with the second highest saving is processed and so on. The key features of this algorithm are: simple structure, easy to implement and flexibility. However, due to its greedy nature, a major shortcoming of this method is that

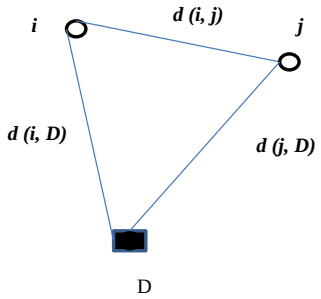

Figure 24: Saving calculation

it does not produce good routes towards the end. To eliminate this drawback, Gaskell (1967) and Yellow (1970) introduced a shape parameter (λ), a measure of relative importance of the direct arc between two customers, in the calculation of saving equation: $s(i,j) = d(i,D) + d(j,D) - \lambda \times d(i,j)$. Recently, Segerstedt (2013) proposed another variant, which uses only the first pair of calculated savings and utilizes these for searching customers to an already decided route. First priority is given to the position before or after the distribution point that presents the highest saving to the new point or that has the shortest distance to it. If both conditions satisfy, then the distribution sequence which gives the total shortest distance is chosen. The author retrieved the best known solution to the Dantzig and Ramser (1959)-problem.

HA2: Gillett and Miller (1974) introduced the *Sweep Algorithm* for efficiently solving medium and large scale vehicle-dispatch problems. In order to implement this heuristic, the polar co-ordinates of all locations are first determined, fixing depot as the pole. The vertices are then re-arranged in the increasing polar angle by sweeping (clockwise or counter-clockwise) a ray joining the depot to an arbitrary point, called seed point. From this seed point, formation of the first cluster begins. The vertices are included satisfying vehicle's loading capacity as they are swept. The next cluster is created by resuming the sweeping operation from the next available vertex and the process is stopped when all vertices have been swept. Once the assignment of vertices to the vehicles is done, *Travelling Salesman Problem* (TSP) is solved for each cluster independently to form the optimum routing sequence of vertices. Although this method is simpler than the Clarke and Wright's saving heuristic, the latter dominates the former in both accuracy and speed. Another major disadvantage of Sweep Algorithm is that it is not suitable for city logistic problems.

HA3: The algorithm of Fisher et al. (1982) is also widely used to solve the VRP. It is a two-phase heuristic like the Sweep Algorithm. The first phase (clusters formation

phase) itself consists of three steps:

step 1 : The selection of initial dummy seed points (one for each cluster)

step 2 : The determination of insertion costs of all vertices (with respect to the seed point) for each cluster.

step 3 : Solving a General Assignment Problem (GAP), by the Lagrangian Relaxation Technique.

While in the second phase, TSP is solved (optimally or approximately) to optimize the delivery routes for vehicles. This algorithm was found to be good for the single-day VRP; nevertheless, its computing speed is a matter of concern. Furthermore, it is neither flexible nor easy to implement like Clarke and Wright's saving-based algorithm. Contrary to Fisher's algorithm, Bramel et al. (1991) proposed a two-phase heuristic in which the initial seed points are determined by solving a capacitated location problem. After this, vertices are gradually included, following the least cost insertion rule and satisfying the vehicle loading capacity. The insertion cost of an unrouted customer i into a tour T_k is calculated as: $t\left(T_k \cup \{i\}\right) - t\left(T_k\right)$, where $t\left(T_k\right)$ is the length of an optimal TSP tour on T_k.

HA4: Beasley (1983) described another two phase heuristic method for VRPs, called route-first cluster-second. In this approach, a giant TSP tour is constructed first, ignoring the vehicle capacity constraint and then this tour is partitioned into feasible routes. To decompose a giant tour into feasible tours, a standard shortest path problem is solved on an acyclic graph. While solving the shortest path problem, the travel cost between two nodes i and j is calculated as: $d(i,j) = d(i,D) + d(j,D) + l_{i,j}$, where D is the depot and $l_{i,j}$ is the travel cost between nodes i and j on the TSP tour. Although this principle is not as effective as the cluster-first route-second method, it has had some attractive features. As for instance, the partition of a giant tour is a computationally fast procedure ($o(n^2)$ operations by Dijkstra's Algorithm), where n is the total number of nodes. Thus, several giant tours can be created to obtain a diverse set of solutions. Moreover, this method is naturally well suited to problems with the free number of vehicles.

TS1: Gendreau et al. (1994) introduced the Taburoute algorithm, a variant of TS, to obtain near optimal solutions for constrained VRPs. Some of the salient features of Taburoute are:

- *Neighborhood operator*: The neighbor solutions are generated by removing a vertex from its current route and inserting to another route containing one of its closest neighbours, using the GENI method developed by Gendreau et al. (1992) for the TSP.

- *Random tabu tags*: After removing a vertex from a route, its insertion back into that route is forbidden till $t + \theta$ iterations, where t is the current iteration and θ is a random number in the range [5, 10].
- *False start*: In this strategy, several solutions are generated initially and a limited search is performed on each of them. The best one is selected as a starting solution for the main search.
- Periodic post-optimization by US (unstringing and stringing) procedure during the search.
- *Intense search*: If the current solution does not improve after some number of iterations, Taburoute performs an extensive search by allowing the removal of more vertices.
- *Continuous diversification*: It is produced by adding a term in the objective function that is proportional to the movement frequency of vertices.

With the aid of these effective features, Taburoute had outperformed the best heuristics available at that time, producing high quality solutions and often the best knowns.

TS2: The adaptive memory procedure (AMP) utilized by Rochat and Taillard (1995) within the framework of TS for the VRP is an interesting idea. An adaptive memory is basically a pool of routes belonging to the elite solutions discovered during the search. It is dynamically updated and used to provide new starting solutions for TS. Initially, several solutions are generated and routes belonging to them are stored in the adaptive memory. In order to create a new solution, routes of the adaptive memory are probabilistically selected (giving larger weight to the routes of fittest solutions) and combined differently. Moreover, once the first route is selected, the next route to be included must not contain customers which are already present in the previously extracted route(s). The selection process of routes from the adaptive memory is continued until all customers have been served or no more feasible extraction is possible. If latter is the case, an insertion algorithm developed by Solomon (1987) is invoked to insert all the unrouted customers. This AMP empowered TS obtained two new best solutions on the 14 standard VRP benchmark instances.

TS3: Rego and Roucairol (1996) also used the framework of TS algorithm for a VRP under the capacity and distance restrictions. They introduced a novel feature, so called node-ejection chains for defining neighbor solutions. This method ejects a vertex from its current position and inserts at the place of another *neighbor* vertex, thereby triggering a chain reaction that might end up producing a cycle. An infeasibility condition is defined to ensure the feasibility of the resulting solution. To reduce computational time, a set of vertices was chosen for the ejection together with their nearest neighbours.

Although the ejection chain method increased the capability of TS to produce quality solutions, it failed to reach the level of the best known algorithms of that time period. Hence, Rego (1998) further proposed a variant of this scheme, called Flower algorithm. In this approach, a route is seen as a blossom, a path as a stem and a complete VRP solution (consisting of several routes) is called flower. An ejection move is performed by deleting and inserting suitable edges so that the flower structure can be maintained. The computational results on a set of real world and academic problems suggested that the flower algorithm might be a good alternative to the best known VRP heuristics.

Another well-known TS that exploits the concept of ejection chain was proposed by Xu and Kelly (1996). A neighbour solution in this work is defined by oscillating between ejection chains and vertex swaps between two routes. The ejection chains are determined by solving an auxiliary network flow problem. A pool of best solutions is maintained that are periodically used to re-initiate the search. Furthermore, individual routes are also periodically re-optimized by means of 2-opt [Croes (1958)] and 3-opt [Bock (1998), Lin (1965)] tour improvement heuristics. The Xu and kelly's TS obtained several best known solutions on the CVRP benchmark instances. But, the major disadvantage of this method is that it needs high computational effort and the right settings of several parameters.

TS4: Toth and Vigo (2003) put forward another variant of TS, so called Granular Tabu Search (GTS), for capacitated VRPs. The GTS is based on the concept of drastically restricted neighbourhoods. The moves that insert only long edges into the solution are discarded. This idea originates from the observation that longer edges have a small likelihood of belonging to the final optimum solution. In order to implement GTS, first the problem is solved by means of any fast heuristic (e.g., Clarke and Wright's saving algorithm) and the average edge cost is determined. Then, a sparse graph is obtained from the original network graph by eliminating edges whose cost is greater than the *granularity threshold* (ϑ)[see equation 2.2.1], but except those incident to the depot and belonging to the high quality solutions. A move is applied only if it inserts at least one edge of the sparse graph. The GTS was implemented with inter & intra neighbourhood exchange operators and several features of the Taburoute algorithm [TS1]. It obtained excellent results, that too within very short computing times. The authors also showed that with sparsification parameter between 1.0-2.0, 80-90% of the unpromising edges could be removed.

$$\vartheta = \beta \times \frac{z'}{(n+k)} \tag{2.2.1}$$

In the equation (2.2.1), n & k are the number of customers and vehicles, respectively. The β is a positive sparsification parameter and z' is the solution value found by a heuristic algorithm.

GA1: Berger and Barkaoui (2003) proposed a competitive hybrid GA to address the classical VRP. It concurrently works on two populations of solutions with the periodic exchange of some local best individuals. The algorithm also uses well-known heuristics within the genetic operators in order to minimize the total travelled distance. A chromosome was simply represented as an ordered list of customers, as depicted in the Figure 25. The initial populations were generated by a sequential insertion heuristic proposed by Solomon (1987), which adds customers in random fashion at randomly chosen positions. The selection procedure was carried out by the roulette-wheel scheme. To perform the crossover operation, an insertion based operator was designed with the idea to improve some routes of one parent ($P1$) using good genes of other parent ($P2$). The routes $(1, \ldots, k, \ldots, n)$ of $P1$ to be improved are probabilistically selected (by uniform probability distribution or based on the number of customers defining a tour/the average distance between consecutive route members). The gene set ($g - set$) from $P2$ includes all members of the routes whose centroid is located within a certain range from the centroid of route k of $P1$. To create a child route, some of the customers (selected randomly or based on the large waiting times/distance separating adjacent members) are removed. Next, Solomon's insertion heuristic with stochastic features (random selection of the cost function parameters and the consideration of three best candidates) is applied to add customers from a combined set containing genes of the $g - set$ and the unrouted but already visited customers. The feasibility is maintained by removing inserted customers from the remaining unvisited routes chosen for the improvement in $P1$. Finally, an offspring is created by inheriting other routes of $P1$ while discarding the already routed customers. If some customers have not been routed, then a new tour is built using the nearest neighbor principle. The mutation operator, on the other hand, consists of three heuristics:

1. Large Neighborhood Search (LNS) [Pisinger and Ropke (2006a)]
2. Edge Exchange (EE)
3. Reorder Customers (RC)

The LNS and EE are applied with some probability, whereas RC is employed only when a new best feasible solution is discovered. In the EE operation, each customer is checked for reinsertion in its neighbor routes. While, RC tries to reconstruct a new tour by reordering customers within a route, using the non-deterministic form of Solomon's insertion heuristic. This synergy of GA and heuristic methods was tested on the well-known VRP benchmarks created by Christofides et al. (1979) and proved to be very

5	7	1	3	8	9	6	4	10	2

Figure 25: Representation of a solution in GA1 (10 customers)

cost-effective. It reached at the best known solutions for 6 out of 14 instances, giving an overall average deviation of 0.48% from the best knowns and consuming on average 21.25 minutes of CPU time.

GA2: Prins (2004) developed a simple and effective hybrid GA to solve the CVRP. The key components of this algorithm are: chromosome representation, clone management and a LSP for mutation. A chromosome was coded as a sequence of clients without trip delimiters (see Figure 25), and the procedure *Split* [see Prins et al. (2009) for details] was used to partition it into feasible vehicle routes. The initial population was made up of random chromosomes and three good solutions created by the heuristics of Clarke and Wright (1964), Mole and Jameson (1976) and Gillett and Miller (1974). The clones (solutions whose fitness values differ by a pre-defined positive constant) were not allowed in the main population for better dispersion of solutions. For improvements, LSP consisting of nine neighborhood operators (see Table 5) was applied instead of using any conventional mutation operator. Each iteration of the LSP scans all possible pairs of two distinct tasks (nodes), successively implements moves and stops with the first improvement. The current solution is updated and the next iteration is continued. The LSP stops when no further improvement can be made. They compared the performance of this hybrid GA on the instances of Christofides et al. (1979) against 10 TS methods and the best SA algorithm published for the CVRP at that time. The results were outstanding indeed; this hybrid GA surpassed most TS algorithms in terms of the average solution cost and became the best solution method for the 20 large instances created by Golden et al. (1983).

GA3: Alba and Dorronsoro (2004) utilized the cellular Genetic Algorithm (cGA), a subclass of GA in which individuals interact only with their neighbors, to solve the VRP. A solution was represented as a permutation of $c + k - 1$ integers $[0 \ldots (c + k - 1)]$, where c and k are the number of customers and vehicles, respectively (see Figure 26). For crossover operation, they used Edge Recombination operator (ERX) [Whitley et al. (1989)], which builds a child solution by preserving edges from both parents. The mutation operator consists of three neighborhood operators: insertion, swap and inversion. These operators were applied to each gene with equal probability. In this work, they also added a local post optimization stage in which 2-opt and λ-interchange were applied to all members of the population after each generation. These ideas were tested on the problems taken from an online OR library (http://people.brunel.ac.uk/ mas-

1	If u is a client node, remove u then insert it after v.
2	If u and x are clients, remove them then insert (u, x) after v.
3	If u and x are clients, remove them then insert (x, u) after v.
4	If u and v are clients, swap u and v.
5	If u, x and v are clients, swap (u, x) and v.
6	If (u, x) and (v, y) are clients, swap (u, x) and (v, y).
7	If $T(u) = T(v)$, replace (u, x) and (v, y) by (u, v) and (x, y).
8	If $T(u) \neq T(v)$, replace (u, x) and (v, y) by (u, v) and (x, y).
9	If $T(u) \neq T(v)$, replace (u, x) and (v, y) by (u, y) and (x, v).

Note: u & v are distinct tasks (nodes) and x & y are successor nodes of u and v, respectively. $T(u)$ and $T(v)$ are trips of u and v, respectively.

Table 5: Neighborhood operators in GA2

tjjb/jeb/info.html). The results were very clear; local search operators greatly improved the performance of cGA. The cGA with only 2-opt significantly reduced the overall cost on all instances, but could not find the best known solutions. It was λ-interchange that helped cGA reach at the optimum solutions. The experiments also indicated that CGA2o1i (cGA with 2-opt and 1-interchange) was better than cGA2o2i as it achieved the similar results in lesser computational time. They also compared the performance of cGA2o1i with some classical VRP algorithms: saving, sweep, 1-petal, 2-petal, TS [Rochat and Taillard (1995)], GA [Prins (2004) and Berger and Barkaoui (2003)] and Ant Colony Optimization [Bullnheimer et al. (1999) and Reimann et al. (2004)]. It was seen that TS, Prins' GA and cGA2o1i outperformed other algorithms on all the studied instances.

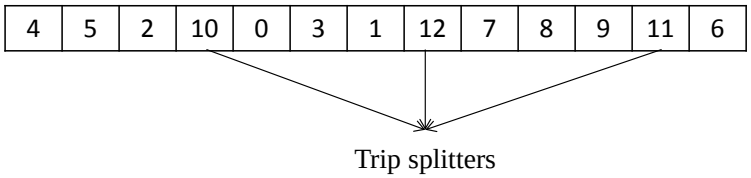

Figure 26: Representation of solution in GA3 (10 customers)

The Figure 26 shows a sample solution string with $c = 10$ and $k = 4$ as coded in (GA3). The customers have been represented by numbers [0 ... $c - 1$]. The integers greater than $(c - 1)$ represent trip delimiters that belong to the range [c ... $c + k - 2$].

GA4: Nazif and Lee (2012) proposed a GA, which uses an optimized crossover operator designed by a complete undirected bipartite graph, to solve the CVRP. The solution was represented as an integer string. The optimized crossover operation, originally proposed by Aggarwal et al. (1997) for the independent set problem, works as follows:

step 1 : Given two parents P_1 and P_2, an undirected bipartite graph $G = (U \cup V, E_1 \cup E_2)$ is constructed, where $U = (u_1, u_2, \ldots, u_n)$ represents customers, $V = (v_1, v_2, \ldots, v_n)$ represents nodes, E_1 & E_2 represent the arc sets in which, $(u_j, v_i) \in E_1$ if customer j of P_1 is located at node i and $(u_j, v_i) \in E_2$ if customer j of P_2 is located at node i.

step 2 : All perfect matchings representing temporary off-springs are determined in G (several efficient matching algorithms can be found in the work of Ahuja et al. (1993)).

step 3 : A temporary offspring having the least objective function value is selected as a O-child (Optimum child).

step 4 : An E-child (Exploratory child) is created as $((P_1 \cup P_2) - child) \cup (P_1 \cap P_2)$.

For mutation, inversion and swap operators were applied with equal probability. In addition, clones were replaced by uniformly randomly generated solutions while filling the new population. The algorithm was tested on problem instances from the benchmark of Christofides et al. (1979) and Taillard (1993). On Christofides' instances, the algorithm retrieved the best known solutions for most cases. The deviation from the best known was always under 0.7435 %. On 5 cases of Taillard's instances, the algorithm found the best known solution available at that time. In addition, the algorithm improved the best known solution by 0.0018% on one instance.

MA1: Nagata and Bräysy (2009) suggested a MA, enhanced by Edge Assembly Crossover (EAX) operator and well-known local searches, for the CVRP. They also designed an efficient modification algorithm to convert an infeasible solution into a feasible one. The main ingredient in their approach was the EAX operator. In the first step of EAX, two parents are selected and a graph is created that consists of uncommon edges between them. In the second step, cycles are formed on this graph by randomly selecting a starting point and then alternately linking edges from the parent solutions. All edges belonging to these cycles are deleted from the graph. This process is repeated until all edges on the graph have been removed. The third step constructs, so called E-sets, which are any combination of cycles formed in the previous step. Two strategies, called single and block are applied. In the single strategy, one cycle is randomly selected that form the current E-set. Whereas, in the block method, cycles sharing nodes with the

randomly selected cycle are also included in the E-set. Finally, in the fourth step, intermediate child solutions are generated as follows. A parent (as the base solution) and an E-set are selected. Then, all edges from the base solution that are also member of the selected E-set are removed. Next, all common edges between the E-set and the other parent solution are added to the base solution. In the fifth and the last step, disjoint sub-tours are eliminated by merging them in random order with the routes connected to the depot, using the 2-opt heuristic. In the local search phase of MA, 2-opt and (λ, μ)-interchange neighborhood operators were employed. This Edge Assembly based MA was found to be robust and competitive. It yielded new best solutions to 10 large-scale benchmark instances (out of 12) of Golden et al. (1998) in reasonable computational time.

SA1: Osman (1993) implemented SA algorithm to solve a VRP subjected to the distance and capacity constraints. The SA structured by him was more involved and successful, mainly due to the combination of following characteristics:

- Use of λ-*interchange move*: This operator first selects two routes together with two subsets of customers (one from each route and of size less than or equal to λ). A neighbour solution is then produced by exchanging the selected subsets.
- Beginning the search with a high quality solution found by means of Clarke and Wright's saving algorithm.
- Adjustment of the algorithmic parameters in the trial phase itself.

In addition, the cooling schedule was used in a more sophisticated way. It is decreased only when the current solution is modified. Otherwise, the current temperature is either halved or replaced by the temperature at which the current solution was obtained. The algorithm yielded good solutions, but failed to identify the best known solutions available at the same time period.

ALNS1: Pisinger and Ropke (2006a) presented a robust ALNS meta-heuristic, an improved version of the Large Neighborhood Search proposed by Shaw (1998), to solve VRPs and pickup & delivery problems with time windows. The ALNS itself consists of several fast removal and insertion heuristics and was embedded into the framework of SA algorithm. The main search begins with a solution found by any simple construction heuristic (e.g., sequential insertion). In each iteration of the main loop, a destroy (or removal) heuristic and a repair (or insertion) heuristic are applied to the current solution for building a new one. Furthermore, the selection of destroy and repair heuristics depends on their past performances. In fact, based on the quality of the new solution, some weights are given to the heuristics that determine their selection probabilities (see equation 2.2.2). They used the roulette-wheel selection method to choose heuristics.

The ALNS was applied on more than 350 benchmark problems containing up to 500 customers. It delivered a superb performance, improving the best known solution for more than 50% of the problems. They also observed that the use of several competing heuristics produced better results than the just one.

$$w_{i,j+1} = w_{i,j} \times (1 - r) + r \times \frac{\pi_i}{\theta_i} \tag{2.2.2}$$

In the equation (2.2.2), $w_{i,j}$ is the weight associated with heuristic i at j^{th} segment of the search. The π_i is score of heuristic i obtained during the last segment of the search, r is a reaction factor and θ_i counts the number of times heuristic i was used.

VND1: Chen et al. (2010) designed Iterated Variable Neighbourhood Decent (IVND) algorithm for solving the CVRP. Firstly, an initial solution s is built by the saving algorithm of Clarke and Wright (1964). Then, the solution s is improved by a VND procedure. Once a local optimum solution s^* is found, the VND procedure is stopped and s^* is perturbed to obtain a new solution s'. From the solution s', a new VND procedure is started to obtain a new local optimum $s^{'*}$. The acceptance of $s'*$ as s^* depends on its quality and the search history. If the best solution does not improve after a predetermined number of iterations, then the best found solution is set as s^* and the above procedure is repeated. Otherwise, reminiscent SA criteria is used to update s^*. In the VND procedure, four operators: *relocate, swap,* $2 - opt$ *and* $2 - opt^*$, were used. To restrict the neighbourhood size, the concept of *Granular neighbourhood* [Toth and Vigo (2002)] was applied. For perturbation, cross-exchange move [Taillard et al. (1997)] was used. The IVND algorithm was applied on 14 VRP instances of (Christofides et al., 1979) and 20 large scale problems by Golden et al. (1998). On Chhristofides' instances, IVND found more best known solutions than the general heuristic of Pisinger and Ropke (2006a). The algorithm produced an average relative percentage deviation (RPD) of 0.12 from the best knowns. For large scale problems, the proposed IVND found the best known solutions to 3 problem, giving an average RPD of 0.67.

ACO1: Bell and McMullen (2004) tackled the VRP, using ACO algorithm in which an ant simulates a vehicle and its route is constructed by incrementally selecting customers. To select the next available customer j, an ant uses the equation 2.2.3.

$$j = argmax \left\{ (\tau_{iu}) (\eta_{iu})^{\beta} \right\} \quad u \notin M_k, \quad q \leq q_o \tag{2.2.3}$$

Where, τ_{iu} is equal to the amount of pheromone on the path between the current location i and possible locations u. The value η_{iu} is defined as the inverse of the distance between two customer locations and the parameter β establishes the importance of distance in comparison to pheromone quantity in the selection algorithm ($\beta > 0$).

Locations already visited by an ant are stored in the ants working memory M_k and are not considered for selection. The value q is a random uniform variable in [0,1] and the value q_o is a parameter. When each selection decision is made, the ant selects the arc with the highest value from equation 2.2.3 unless q is greater than q_o. In this case, the ant selects a random variable (S) to be the next customer to visit based on the probability distribution of p_{ij} (equation 2.2.4), which favors short paths with high levels of pheromone:

$$p_{ij} = \begin{cases} \frac{(\tau_{ij})(\eta_{ij})^{\beta}}{\sum_{u \notin M_k}(\tau ij)(\eta_{ij})^{\beta}} & j \notin M_k \\ 0 & \text{otherwise} \end{cases} \tag{2.2.4}$$

Once the capacity constraint is satisfied, the ant returns back to the depot. This selection process is continued until all customers have been visited. For further improvements, pheromone trails are updated by the equation 2.2.5.

$$\tau_{ij} = (1 - \alpha) \times \tau_{ij} + \alpha \times \tau_o \tag{2.2.5}$$

Where, α is a parameter that controls the speed of evaporation and τ_o is equal to an initial pheromone value assigned to all arcs in network graph. In this work, τ_o is set equal to L^{-1} (L is the best known route distance found for the particular problem). Furthermore, 2-opt heuristic is also applied to each vehicle route for attainment of better solutions. Another improvement strategy applied in this work is the use of candidate list for determining the next location to be added in a vehicle route. Only a set of closet locations are made available for selection. The algorithm was applied to 3 problems [Christofides et al. (1979)], and was found to be able to generate solutions within 1% of the optimum solution.

2.2.2 Multi-objective CVRP

MOEA1: Murata and Itai (2005) conceived an idea of Two-fold NSGA-II for optimizing two conflicting VRP objectives: minimization of the maximum driving duration and the number of vehicles, in periods when demands suddenly surge up (e.g., festive seasons). Firstly, trade-off solutions were obtained for the normal demand problem (NDP). Then, the high demand problem (HDP) was solved, using trade-off solutions of the NDP in the initial population. In NSGA-II, they applied cyclic crossover[Oliver et al. (1987)] and following two kinds of mutation operators:

- *Split*: In the split mutation, an offspring is sub-divided into new vehicle routes by randomly changing positions of the splits (trip delimiters).

- *Order*: In the order mutation, a route is selected and the order of customers is reversed.

They also defined a solution similarity measure to compare the non-dominated solutions of HDP and NDP. The two-fold approach produced higher similarity on a data set consisting of 5 customers for the NDP and 10 customers for the HDP, thereby reduced the efforts to make changes in the normal routing plan.

MOEA2: Tan et al. (2006) solved a multi-objective truck-trailer CVRP for a logistic company in Singapore. The aim was to design an effective routing schedule so that the overall distance and the number of trucks could be minimized while satisfying some constraints, such as time window and availability of the trailers. To solve the problem, they hybridized the standard MOEA with a LSP. A solution string was coded as a variable-length chromosome representing the complete routing plan (Figure 9). It includes tasks placed in their processing order and the number of routes. In the LSP, all routes in which the number of tasks is below a pre-defined threshold value are randomly grouped into pairs. After this, all tasks in each pair are combined to form a new route that is sorted in ascending order by the earliest service time. The infeasible tasks are stored in an outsourced list. Another novelty in their approach was the use of specialized genetic operators:

- *Route exchange crossover*: It consists of two simple and independent steps as shown in the Figure 27: (1) two randomly selected routes (one from each parent) are swapped and (2) routes with the highest number of tasks from each parent are swapped. In the Figure 27 showing route exchange crossover process, lets say randomly selected routes are 1 & 2 and routes 3 & 4 contain the maximum number of tasks in the chromosomes 1 & 2, respectively.

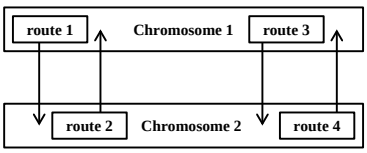

Figure 27: Route exchange crossover in MOEA2

- *Multi-mode mutation*: Two modes of operations were used and one of them was selected (with equal probability) to mutate an offspring. The first operation randomly selects two routes and concatenates them. In the second method of mutation, the sequence containing all the outsourced tasks is considered as a new route.

This hybrid multi-objective evolutionary algorithm (HMOEA) performed better than two of its variants, i.e., STD_MOEA (MOEA with standard genetic operators: cycle crossover and remove & insert mutation operation) and NH_MOEA (MOEA without local search) on various test cases with the different number of tasks. The HMOEA was also found to be computationally efficient in finding good Pareto solutions for the truck-trailer VRP.

MOEA3: Jozefowiez et al. (2006) solved the CVRP on an undirected graph, minimizing the overall tour length and the route balance (length difference between the largest route and the smallest route). To jointly optimize these two objectives, they proposed an enhanced version of the NSGA-II by adding two mechanisms into its framework:

- *Elitist diversification*: It maintains an additional archive, which contains potentially Pareto optimal solutions when one objective is maximized. These solutions are included into the main population at each generation to improve the exploration capability of the algorithm.
- *Parallelization*: The parallelization was done by means of a ring-network based island model, a network topology in which each node is connected with exactly two other nodes. The aim of parallelization was not to reduce computational time, but explore larger part of the search space in a given time.

Other features include the use of Route Based [Potvin and Bengio (1996)] & Split [Prins (2004)] crossover operators and Or-opt/2-opt local search methods as mutation operators. Nevertheless, when a generation corresponds to the communication phase, crossover and mutation are not performed. Instead, an island receives the top 50 % of solutions (according to the ranking and crowding distance sort, of the population after the selection phase) from each neighbor. These solutions form the child population. The computational experiments carried out on the benchmark problems of Christofides et al. (1979) showed that the elitist diversification significantly improved the performance of NSGA-II when one processor was used. In the case of more processors, the algorithm performed quite well without the elitist diversification as parallelization alone was capable of approximating a better Pareto set.

MOEA4: Jozefowiez et al. (2007) designed a Unified Tabu Search [Cordeau et al. (2001)] based multi-directional Pareto local search, named Target Aiming Pareto Search (TAPaS), for the capacitated VRP with route balance (CVRPRB). The TAPaS was employed for improving the quality of the approximate Pareto set generated by NSGA-II. For each local search, a different goal point is set to explore different parts of the search space. The local search loop is terminated when it either reaches at the goal point or finds a solution that dominates the goal point. To establish the effectiveness of this idea,

they first compared the performance of TAPaS against NSGA-II on the benchmarks of Christofides et al. (1979) using the S-metric [Zitzler and Thiele (1999)]. To do so, the initial population for NSGA-II was generated by a greedy algorithm and the non-dominated solutions of this population were used as a starting Pareto set for TAPaS. It was seen that TAPaS-std (with standard UTS) and TAPaS-dp (UTS with two diversity penalties: length and balance) outperformed NSGA-II on average three and four times, respectively. On the other hand, observations on the C-metric [Zitzler and Thiele (1999)] confirmed that TAPaS improved the convergence ability of NSGA-II towards the optimal Pareto set. Whereas, NSGA-II alone was found to be better in generating well-diversified sets.

MOEA5: Pasia et al. (2007) also utilized the idea of *Pareto local search* to address the CVRPRB. The algorithm constitutes of just two phases: the initialization of a set of trade-off solutions and the Pareto local search. In the initialization phase, a pool of solutions is created by a randomized saving algorithm. Thereafter, the 2-opt heuristic is applied to each member of this pool in order to balance the artificially bounded solutions. Before beginning the local search phase, a Pareto set is constructed from this pool by removing all the dominated solutions. A Pareto local search (P-LS) designed by Basseur et al. (2005) is then implemented in the second phase on each non-dominated solution. The P-LS utilizes three neighborhood structures, namely, *move, swap and 2-opt*. Moreover, 2-opt is always applied after performing either move or swap to the affected partial tours. A set of seven CVRP benchmark instances of Christofides et al. (1979) was used to compare the performance of P-LS algorithm with an enhanced version of the NSGA-II developed by Jozefowiez et al. (2006). The box plots of three unary quantitative measures – hypervolume, unary epsilon and R3 indicator – demonstrated that the median values found by the P-LS were better than those obtained by the NSGA-II.

MOEA6: Jozefowiez et al. (2009) again employed the same elitist diversification and parallelization methods to propose a standard MOEA for the CVRPRB. However, the co-operative model in this work was a two-dimensional toroidal grid instead of a ring, as depicted in the Figure 28. An island has been denoted by I_j^i, which means that it belongs to the i^{th} brick and its additional archive is of A_j type. The main loop of an island algorithm consists of four steps: communication, selection, recombination and archives update. In the communication phase, an island I_j^i sends its standard archive (A_o) to all neighbors: I_{j-1}^i, I_{j+1}^i, I_j^{i-1} and I_j^{i+1}. But, it communicates A_j archive to only I_j^{i-1} and I_j^{i+1}. The selection phase itself consists of several steps: ranking [Deb et al. (2002)], sharing [Goldberg and Richardson (1987)], fitness calculation and sorting according to the fitness values. After sorting solutions, half of the mating pool is filled by solutions of the current population and rest half by the solutions belonging to A_o and A_j. Following this, recombination (with route-based and Split crossover operators) and

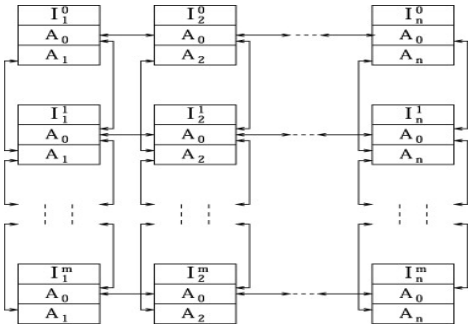

Figure 28: Co-operative model in MOEA6

mutation (by 2-opt) are performed to yield child solutions and finally archives ($A_o \& A_j$) are updated. In case, the size of an archive exceeds from a predetermined limit, it is reduced by a clustering method, so called average linking method [Morse (1980)]. Similar to the prior work, the S-metric was used to asses the contribution of the elitist mechanism on the CVRP benchmarks of Christofides et al. (1979). It was noticed that the elitist diversification technique was always able to improve the quality of the final Pareto set.

NPM1: Corberán et al. (2002) addressed a school bus routing problem with the aim to optimize two conflicting objectives, namely, minimization of the number of buses and minimization of the maximum time a student spends in the bus. They modelled the problem as a *node routing*, created a composite objective function (weighted combination of the individual objectives) and applied the Scatter Search (SS) meta-heuristic. The SS was equipped with four effective features:

First *Clustering & sector based heuristics* H1 & H2: The H1 is based on a clustering mechanism, which keeps the nodes assigned to each cluster ordered. Initially, all nodes are served alone on separate routes. Then, at each iteration, the best pair of routes (one with the minimum traveling time between the two routes) are merged. Instead of considering all possible pairs of routes, an ordered candidate list is maintained that stores the top pairs of routes. On the other hand, H2 works on building sectors around nodes/locations (see Figure 29). The size of a sector is predefined by an input parameter (Angle). First, the location with largest traveling time (tm) to school is selected and a sector is built around it. The next location to be assigned is randomly selected from a pool of unassigned locations with traveling time to school larger than or equal to $tm \times \alpha$, where α is a user defined parameter. A chosen node is assigned to the current sector if the assignment is feasible; otherwise, a new sector is created around it.

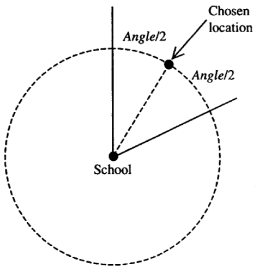

Figure 29: Definition of a sector

Second *Swap exchange*: It tries to locally optimize the length of a route by swapping positions of two vertices. If the length is reduced, then positions are exchanged. The procedure stops when no more reduction is possible.

Third *Insert exchange*: This mechanism is applied to the longest route only. It removes a node from its current route and feasibly inserts into another route, which contains one of its nearest neighbors. The node to be removed is carefully chosen such that its removal could bring the maximum decrease in the length of the route.

Fourth *Combine*: This procedure generates new solutions from the combination of two existing solutions. It begins with building a *match* matrix, which contains the number of common elements between routes of the solutions. Afterwards, a voting mechanism is used to create a new combined solution.

A real data set of 16 middle schools (Burgos, spain) was used to check the performance of this Scatter Search. It produced an average improvement of 23.4% over the existing solution at that time.

2.3 Arc-based Vehicle Routing Problems

In arc-based VRPs, locations of customers along streets are so dense that they can not be clearly marked by points, as shown in the Figure 30. Therefore, the problem seeks to design vehicle routes to service full streets rather than individual nodes. The network graph can be of all three kinds: *undirected, pure-directed and mixed*. Some of the practical examples of such problems are: mail delivery, school bus routing, meter reading and snow plowing. Among its several variants, this survey has been focused on its basic model, that is, the capacitated arc routing problem (CARP).

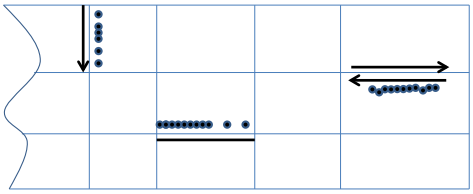

Figure 30: Arc routing

2.3.1 Single-objective CARP

TS5: Hertz et al. (2000) suggested the first TS based algorithm, called CARPET, for an undirected CARP. The CARPET algorithm contains seven efficient procedures – *Shorten, Drop, Add, Paste, Cut, Switch and Postopt* – that are applied to each route of a solution one at a time. It is advised to see Hertz et al. (1999) and Hertz et al. (1997) for the implementation details of Shorten, Drop, Add and Cut methods. The method Paste first builds a single route by relaxing the capacity constraint and then calls the Shorten procedure. The function Switch reverses the order of vertices present between the two consecutive copies of another vertex that has appeared more than once in a route. The Postopt method attempts to find a better solution by sequentially applying Paste, Switch, Cut and Shorten methods. An initial solution for TS is constructed by solving a Rural Postman Problem (RPP), using the heuristic proposed by Frederickson (1979). The feasibility is maintained by the Cut method and neighborhood search is performed by successively applying Drop & Add to routes of the current solution. The CARPET was tested on 23 instances of gdb set [DeArmon (1981)] and 34 instances of val set [Benavent et al. (1992)]. On DeArmon's instances, it generated the best known solutions 20 times, giving an average deviation of 0.17 %. Whereas, on Benavent's instances, it obtained 17 best known solutions while giving an average and the worst deviations of 1.13 % and 8.110 %, respectively.

TS6: Brandáo and Eglese (2008) described a completely deterministic version of the TS algorithm for an undirected CARP. Since the starting point in TS affects the quality of the final solution; therefore, they designed five different types of fast methods – *Cheapest edge, Dearest edge, Insert, Connected component and Path scanning* – to produce varieties of feasible initial solutions. In the Cheapest edge method, the construction of a vehicle route begins with the required edge that is nearest to the depot. The next edge to be added is the unassigned required edge, which is nearest to the end vertex of the last inserted edge. In case there is more than one candidate, the method selects the cheapest cost edge while breaking ties arbitrarily. The Dearest edge method works exactly in

the same way, but chooses the highest cost edge whenever a selection has to be made. In the Insert procedure, the next edge to be inserted is the one that increases the route cost by the least amount. Moreover, an edge can be inserted between any pair of edges connected together by a dead-heading path. The Connected component procedure uses a version of Frederickson's heuristic [Frederickson (1979)] described in the work of Pearn and Wu (1995) to build a solution. The path scanning is a well known heuristic proposed by Golden et al. (1983) for the CARP. It uses five selection rules to determine the next required edge:

rule 1 : Maximize the distance to the depot.

rule 2 : Minimize the distance to the depot.

rule 3 : Rule 1 if the vehicle is less than half-full, otherwise Rule 2.

rule 4 : Maximize the ratio $d(e)/c(e)$, where $d(e)$ and $c(e)$ are the demand and the cost of edge e, respectively.

rule 5 : Minimize $d(e)/c(e)$.

The main search utilizes three neighborhood operators: single insertion, double insertion and swap. In addition, at the end of each iteration, a route improvement procedure is applied to further reduce the cost of the two individual routes that have been modified during the main search to obtain a new solution. For each route, the least-cost visiting order of required edges belonging to the route is formed using Frederickson's heuristic. With the assistance of these heuristics and the absence of random parameters, the algorithm performed quite good against the CARPET on instances of gdb & val sets and MA [Lacomme et al. (2004a)] on instances of gdb, val as well as egl set [Eglese (1994), Li and Eglese (1996b) and Li and Eglese (1996a)].

GA5: Deng et al. (2007) solved a real-life directed CARP for designing routes of sprinkler-cars for the sanitation department of Chongquing, China. They proposed a GA to minimize the total travelled distance. Similar to most of the approaches of GA to routing problems, a chromosome was encrypted as a sequence of tasks and the procedure *Split* was used to subdivide it into feasible car routes. The initial population was completely filled by randomly generated solutions. The crossover operation was performed by order crossover operator (OX) [Davis (1985)] and the mutation was done by simply exchanging positions of two randomly selected tasks. In addition, they also implemented a LSP consisting of five moves and a function restart, which restarts the system from its beginning if the population does not change after a large number of iterations. On a real data consisting of 37 vertices and 8 sprinkler cars having equal capacity of 8 tons, computational experiments illustrated that this approach reduced the

per day travel distance by 33 % (from 95 Km to 64 Km), thereby resulted the overall saving of 40,000 Yuan per year. They also compared the performance of this hybrid GA with the MA proposed by Lacomme et al. (2004a). The MA obtained the lowest objective value of 67 Km, consuming on average 29m 18s. On the other hand, GA took only 11m 13s to obtain the best result of 64 Km.

GA6: A biased random key genetic algorithm (BRKGA) was proposed by Martinez et al. (2011) for solving the CARP with the fixed vehicle fleet size. See Goncalaves and Almeida (2002) and Ericsson et al. (2002) for details of BRKGA. A chromosome was coded as a vector of random keys, each having a value in the interval [50, 200]. The length of each random key vector is equal to the number of required arcs. A random key vector is decoded as follows. First, it is sorted in an increasing order of the keys. Mapping the i^{th}-position in the chain with the position of the key in the ordered chain, a sequence of positive integers is then obtained that represents the processing order of required arcs. Finally, the tour partitioning procedure of Haimovich and Kan (1985) is applied to obtain a complete CARP solution. Before applying genetic operators, all random key vectors are sorted according to their fitness values and classified as Elite individuals, Non-elite individuals and random key vectors that will be mutated. The crossover operation is performed by parametrized uniform crossover method [Spears and DeJong (1991)] between an Elite individual and a Non-elite one. In mutation, some keys are replaced by new random keys, thereby producing a new solution. A local search procedure is further applied to improve an offspring, following ideas of Beullens et al. (2003). Furthermore, if the best found solution does not change after a certain number of iterations, a new population is generated. Elite solutions are kept and new solutions are generated to fill the new population. The BRKGA was tested on several instances taken from Hirabayashi et al. (1992), Golden et al. (1983) and Benavent et al. (1992). In 21 (out of 25) instances, the algorithm found the best known results. On other 4 instances, the gap was less than 4%.

MA2: Lacomme et al. (2004a) combined basic heuristic procedures with MA for solving an extended version of the CARP. The extension parts include the consideration of: (a) mixed multi-graph with two kinds of links (edges and arcs) and parallel links, (b) two distinct costs per link (dead-heading and collecting), (c) prohibited turns (e.g., U-turns) & turn penalties (e.g., to penalize left turn) and (d) maximum trip length (an upper limit on the cost of any trip). The MA search was started with a population consisting of random chromosomes and the two good quality solutions generated by Path Scanning and Augment Merge heuristic [Golden and Wong (1981)]. A solution was directly coded as a sequence of tasks and Ulusoy's tour splitting method [Ulusoy (1985)] was used to obtain feasible trips. Other main features incorporate the use of order crossover (OX)[Davis (1985)] and a LSP in the place of a simple mutation operation. The LSP utilizes five different moves (see Table 6) and its working process is same as in Prins

(2004). The resulting best MA (with the optimum parameter setting) outperformed heuristic techniques on the benchmark instances of gdb, val and egl sets. It found 26 new best solutions and retrieved the best known solutions on 55 problems. The obtained results by the MA were also compared against those found by the CARPET algorithm. This MA produced better results than the CARPET. For example, CARPET found 17 best solutions in an average running time of 63.87 seconds on val set. Whereas, MA produced 32 best solutions in only 38.35 seconds.

1	Invert task u in $T(u)$ if it is an edge task.
2	Move adjacent tasks (u, x) after task v, or before v if v is the first task in $T(v)$.
3	Swap tasks u and v.
4	2-opt moves

Note: u & v are distinct tasks and x & y are successor tasks of u and v, respectively. $T(u)$ and $T(v)$ are trips of u and v, respectively.

Table 6: Neighborhood operators in MA2

MA3: Tang et al. (2009) proposed a variant of the MA, named Memetic Algorithm with Extended Neighborhood Search (MAENS), for solving a mixed CARP. As the name itself indicates, MAENS is equipped with a special local search operator, called Merge-split (MS), for exploring a large number of neighbor solutions. The MS operator chooses some routes of a solution and all tasks belonging to them are merged to form an unordered list. After this, Path Scanning heuristic is applied to obtain an ordered list of tasks and finally it is decomposed into feasible trips by Ulusoy's heuristic. The MS operator was used in a two phase local search procedure within MA. In the first phase, the most commonly used moves for VRPs were employed (single & double insertion, swap and 2-opt). Whereas, MS operator was applied in the second phase to the local optima obtained in the previous stage. Other simple features of MAENS include the use of sequence based crossover (SBX) [Potvin and Bengio (1996)] and the prevention of clones in the population. The performance of MAENS was tested against CARPET [Hertz et al. (2000)], VND [Hertz and Mittaz (2001)], Guided Local Search [Beullens et al. (2003)], MA [Lacomme et al. (2004a)] and TS [Brandáo and Eglese (2008)] on the CARP benchmark instances taken from gdb, val, egl and Beullens' [Beullens et al. (2003)] sets. It retrieved the best known solutions on 175 problems and discovered new best solutions on 16 problems. Overall, MAENS was found to be superior in terms of solution quality, but suffered from heavy computational burden, mainly due to the implementation of MS operator.

MA4(EA+LSP): Xing et al. (2010) solved the extended capacitated arc routing problem, defined by Lacomme et al. (2004a), but considering both the multiple depot and

1	Invert arc u in $T(u)$ if it is a bi-directional arc.
2	Move adjacent arcs (u, x) after arc v.
3	Move u after v.
4	Swap arcs u and v.
5	2-opt moves on one trip.
6	2-opt moves on two trips.

Note: u & v are distinct arcs and x & y are successor arcs of u and v, respectively. $T(u)$ and $T(v)$ are trips of u and v, respectively.

Table 7: Neighborhood operators in MA4

the maximum service time. They named the problem Multidepot CARP(MCARP). An individual solution is defined simply as a sequence of required arcs, without trip delimiters, and the *Split* procedure is applied to generate a valid MCARP solution (i.e, list of required arcs with trip delimiters). The initial population is generated by Extended Random Path-Scanning (ERPS) heuristic, Extended Random Ulusoy's heuristic (ERUH) and uniformly at random. The mating pool is formed by either binary tournament selection or rank selection method. The crossover operation is performed by OX and Linear order crossover (LOX) operators. With some probability, an offspring undergoes a LSP consisting of six moves (see Table 7). A move is probabilistically selected based on its successful past performances and a fixed number of optional moves are produced. Finally, the best move (in terms of the objective function value) is performed. After this, Partial Replacement Procedure (PRP) [Lacomme et al. (2004a) and Cheung et al. (2001)] is applied with five restarts, each having length 20, to all instances.

Phases	Version 1	Version 2	Version 3
selection	S1	S2	S2
Crossover	S1+S3	S2+S3	S2+S4
Mutation	S1+S3	S2+S3	S2+S4

Table 8: Operator selection strategies in MA4

The authors developed three different versions of the algorithm based on different strategies for selection, crossover and mutation. See Table 8, where S1 is *uniform selection of different operators*, S2 is *empiristic selection of different operators adopting the obtained heuristic data*, S3 stands for *uniform selection of one broken position* and S4 represents *empiristic selection of a broken position adopting the obtained heuristic data*. The algorithms were tested on 107 instances containing up to 140 nodes and 380 arcs against the two extended heuristics (ERPS and ERUH). The computational experiments showed that the performances of algorithms were greatly improved by integrating clas-

sical heuristics and heuristic information for choosing selection, crossover and mutation operators. It was found that the Version 2 selects more appropriate operators than the Version 1. The Version 3 outperformed others, giving a confidence degree of 0.95. The Version 3 was also compared with the MA of Lacomme et al. (2004a) on 23 MCARP instances. The Version 3 was significantly better with a confidence degree of 0.95.

MA5: Liu et al. (2013) proposed MA with ILS (MAILS) to solve the CARP on an undirected graph. In addition to ILS, MAILS also incorporates a new crossover operator (LCSX: longest common substring crossover) and a perturbation mechanism. A solution was represented as a permutation of required edge tasks and Ulusoy's partition procedure was used to convert it into a CARP solution (a set of feasible vehicle trips). The LCSX operates on two parent solutions as follows. Firstly, the longest common substring from one parent is copied to the offspring solution at the same position. Then, the other parent solution is scanned from the beginning to the end to fill vacant slots in the offspring with the missing tasks. The initial population consists of ps chromosomes. Among them, two chromosomes were constructed by Frederickson's heuristic & Path scanning method and $(ps - 2)$ solutions were created randomly. All chromosomes are stored in increasing order of cost. Each iteration of the main loop compromises of following three major steps:

step 1 : Two off-springs are created using either LCSX or OX and one of them is randomly selected as a child solution.

step 2 : With some probability, this child solution undergoes a LSP containing six neighborhood moves (see Table 9). The LSP works in the similar fashion as in Prins (2004) and also explores capacity infeasible solutions by operating on a penalized cost function. The obtained solution after LSP is further improved by the ILS if its cost is close enough to the best found solution.

1	Move task u after task v.
2	Move two adjacent tasks (u, x) after task v.
3	Swap tasks u and v.
4	Swap task u and (v, y)
5	Swap task (u, x) and (v, y)
6	2-opt move

Note: u & v are distinct tasks and x & y are successor tasks of u and v, respectively. $T(u)$ and $T(v)$ are trips of u and v, respectively.

Table 9: Neighborhood operators in MA5

step 3 : Two solutions are selected from the population and the worst one is replaced by the child solution. If the child solution is a clone of any other solution in the population, then double swap perturbation is applied to it.

The main phase is restarted with a new population in which $first, third, fifth, ..., (ps-1)^th$ old chromosomes are kept and others are replaced by randomly generated solutions. This whole process is repeated for a predetermined number of times. The MAILS was tested on gdb, val and egl instances. It found all 23 optimal solutions on gdb set, best solutions on 30 instances (out of 34) of val set and 17 (out of 24) large instances of egl set. It also discovered two new best solutions on the egl set.

VND2: Hertz and Mittaz (2001) reported the use of VND algorithm for the undirected CARP. In the first neighborhood, a solution is generated by moving a required edge (u, v) from its current route (T_1) to another one (T_2). Route T_2 contains either only the depot or a required edge whose one of the end points lies within a predetermined limiting value from u or v. The insertion of (u, v) into T_2 is performed only if the capacity constraint does not violate. The removal of (u, v) from T_1 and insertion into T_2 are performed using *Drop* and *Add* procedures described in the work of Hertz et al. (1999). Furthermore, for obtaining a neighbor solution in other neighborhoods, a set of routes of the current solution is merged into a single tour and the procedure *Switch* is applied to it for making an ordered list of required edges. Then, the method *Cut* divides this tour into feasible routes that further undergo the *Shorten* process. The algorithm was tested against CARPET on the three sets of CARP instances: gdb, val and the set created by Hertz et al. (2000). On gdb instances, both algorithms were able to generate the proven optima for 18 out of 23 instances. On val set, both algorithms produced almost similar results, but VND was faster. On the instances proposed by Hertz et al. (2000), VND surpassed CARPET in terms of solution quality and computing time as well. The average deviations from the lower bound (of optimum value) and computational times on 270 instances were 0.71% & 349 seconds for the CARPET and 0.54% & 42 seconds for VND algorithm.

VNS1: Polacek et al. (2008) solved a CARP with intermediate facilities (IFs), which serve as the loading/unloading points for vehicles. An example of the IF that has also been cited by them is the rivers at special dump sites where vehicles dump snow during the plowing operations. They applied a basic version of the VNS algorithm, proposed by Hansen and Mladenovi (2001), to minimize the travel cost of vehicles. To provide a feasible initial solution for the VNS, a giant tour was created and Ulusoy's heuristic was implemented to obtain vehicle trips. Furthermore, cross-exchange operator [Taillard et al. (1997)] was used to define a neighborhood of the current solution in the shaking phase. It takes two segments of different routes and exchanges them while preserving the orientation of the selected sequences. A solution obtained through the shaking is

sent to a LSP, which re-optimizes the two routes that have changed. A simple *inversion* operator was adopted for the local search; however, inverting was done for all combinations of successive serviced edges of a tour. The obtained candidate solution after the shaking and the local search is straightforward accepted if it is better. Otherwise, it is accepted only if its objective function value deviates from the best found solution value by a pre-determined fixed threshold. The VNS yielded excellent results on four data set (two of them include the extension of IFs) adopted from literatures. On egl and val sets, it outperformed MA of Lacomme et al. (2004a). In fact, 17 new best solutions were found on the egl set. On the instances of Belenguer and Benavent (2003), the VNS found 4 new best solutions and reproduced all other 30 best known solutions. For the CARPIF instances, the VNS found 23 new best solutions. Furthermore, it improved all 28 solutions for the CLARPIF (CARPIF with route length constraint) of Ghiani et al. (2004). Overall, it could find the best known solutions for 120 instances and discovered new best solutions for 72 cases.

GRASP1: The GRASP meta-heuristic was used by Usberti et al. (2011) in conjunction with the Path-Relinking (PR) algorithm to solve the undirected CARP. The objective was to minimize the overall tour cost. Some impressive features of this integrated approach are:

- *reactive parameters tuning*, where parameters are stochastically selected while being biased towards those values, which produced the best solutions in average.
- *statistical filter*, which allows a solution to undergo the local search only if the confidence probability (see equation 2.3.1) of obtaining a better solution than the best one is minimum 95 %. In the equation (2.3.1), c_{ini} and c_{best} are costs of the candidate solution for the local search and the best found so far, respectively. The μ equals to the average ratio of c_{ini}/c_{ls} of the first 100^{th} iterations, where c_{ls} is the local search solution cost. Similarly, σ is the standard deviation of the ratio c_{ini}/c_{ls} of the first 100^{th} iterations. The right hand side of the equation (2.3.1) gives a confidence interval of slightly more than 95 %.

$$\frac{c_{ini}}{c_{best}} \leq \mu + (2 \times \sigma) \tag{2.3.1}$$

- *infeasible local search*, where high quality infeasible solutions are used to explore the feasible/infeasible boundaries of the solution space.
- *evolutionary PR*, where the pool of elite solutions is progressively improved by successive re-linking of pairs of elite solutions.

For the local search, three operators: swap+reversal, single insertion + reversal, double insertion + reversal, were employed. A block-insertion operator, which removes a block

of adjacent required edges and inserts at another position, was used to generate solutions in PR. This whole idea was tested on 81 instances adopted from gdb, val and egl sets. On comparing results with TS [Brandáo and Eglese (2008)], VNS [Polacek et al. (2008)] and ACO [Santos et al. (2010)], it was found that this hybrid algorithm achieved the best overall deviation from the lower bounds and also obtained the highest number of best solutions.

ACO2: Santos et al. (2010) presented an improved ACO algorithm for the CARP. It has five basic steps: (1) *Generation of initial population*: the initial population contains complete CARP solutions. Three heuristics, namely, random arc selection, path scanning with ellipse rule and path scanning with ellipse rule & a local search heuristic, were tested to generate the initial population. The initial population is used to determine the initial pheromone values on paths joining required arcs., (2) *Generation of single network tours*: each ant generates a single network tour servicing all required arcs (using pheromone update strategies of Bullnheimer et al. (1999) & Lacomme et al. (2004b)) and decision rules. Two decision rules were tested. The first one is a pseudo-random-proportional-rule, in which the attractiveness of moving between required arcs depends on the amount of pheromones determined by local heuristic information which is usually a function of distance between arcs. In the second rule, some ants make their decisions based on only pheromone amounts and others consider both pheromone amounts and local heuristic information. Which strategy to use is decided randomly., (3) *Generation of feasible routes*: Ulusoy's heuristic was used to decompose a single network tour into feasible routes., (4) *Improvement of feasible routes*: to improve feasible routes identified in the previous stage, a local search procedure consisting of 12 moves was applied., (5) *termination of algorithm*: steps 2-4 are repeated until either the maximum number of iterations is reached or the lower bound has been attained.

The algorithm was tested on 181 CARP instances taken from DeArmon (1981), Benavent et al. (1992), Belenguer and Benavent (2003) and Beullens et al. (2003) against the best five CARP algorithms available at that time: GLS [Beullens et al. (2003)], MA [Lacomme et al. (2004a)], BACO [Lacomme et al. (2004b)], VNS [Polacek et al. (2008)] and TS [Brandáo and Eglese (2008)]. Overall, this ACO algorithm identified 95% of the best known solutions. Whereas, GLS, VNS, TS, MA and BACO retrieved 90%, 86%, 80%, 74% and 58% best known solutions, respectively. Furthermore, this ACO algorithm could also generate 14 new best known solutions for the 181 problems.

2.3.2 Multi-objective CARP

MOEA7: Lacomme et al. (2006) solved a bi-objective CARP, using the framework of NSGA-II. One of the objectives was the minimization of the overall tour cost. While,

the other was the minimization of the makespan, which is defined as the cost of the longest trip. A chromosome was represented as a list of tasks without trip delimiters and the procedure *Split* was implemented for deriving a set of least-cost vehicle tours. Furthermore, solutions computed by the well-known heuristics, namely, Path Scanning, Augment-Merge and Ulusoy's heuristic, were included in the initial population to speed up the convergence of NSGA-II. Taking advantage of permutation chromosomes, OX operator was utilized for recombination. The mutation operation was performed by a LSP, which operates in the same way as in Prins (2004), but employs only two moves (remove & insert and 2-opt). In addition, four different acceptance criterion: based on the cost (LS_1), makespan (LS_2), Pareto dominance (LS_3) and weighted sum method (LS_4), were suggested to update the current solution while performing the local search. They prepared 8 versions of the algorithm, called MO2-MO9, based on the number of iterations and types of local search (see Table 10). A basic version, called MO1 (NSGA-II without local search), was selected as a reference algorithm. The performances of MO2-MO9 were tested on three sets (*gdb, val and egl*) of the classical CARP instances against the MO1, using $\mu(F, R)$ measure proposed by Riise (2002) for comparing a Pareto front F with a reference Pareto front R. On the *gdb set*, MO9 gave the minimum normalized μ value of -6.53 unit (sum of the signed distances between the solutions of the Pareto front F generated by MO9 and their projections onto the etrapolated reference front R created by MO1). On the *val set*, MO7 was the best in terms of the normalized μ value. Whereas, on the *egl set*, MO5 had a slightly better normalized μ value than MO9. The algorithms were also found to be competitive vis-à-vis the state-of-the-art meta-heuristic CARPET. The results on the *gdb set* showed that MO9 was very robust as its worst deviation to the lower bound on the cost (LB_1) was 2.23 % vs. 4.62 % for CARPET. No total cost established by MO9 was improved by other versions. On the *val set*, MO4 -MO9 outperformed CARPET in terms of LB_1. On the set, all versions except MO1 and MO3 outperformed CARPET with regard to LB_1. MO9 was even able to improve two best known solutions.

NPM2: Mei et al. (2011) put forward D-MAENS (Decomposition-based Memetic Algorithm with Extended Neighbourhood Search) for a bi-objective CARP. The targeted objectives were minimization of the total cost of all the routes and the makespan. In the D-MAENS approach, the original multi-objective problem is decomposed into several single-objective problems, using the weighted sum method with a set of uniformly distributed weight vectors. A population X of solutions whose size equals the number of sub-problems is maintained throughout the search. Each solution in the population X represents a unique sub-problem. For each sub-problem, a sub-population is created. Two solutions are randomly selected and the crossover and local search operators of MAENS [Tang et al. (2009)] are applied to produce one child solution. A child population Y is formed by generating child solutions for all sub-problems. After this, X and Y are combined and decomposed into several fronts of dominance as in NSGA-II. A

Version	Iterations	LS type	LS location
MO1	100	None	Irrelevant
MO2	100	LS_1	On children, rate 10%
MO3	100	LS_2	On children, rate 10%
MO4	100	LS_3	On children, rate 10%
MO5	200	LS_3	On children, rate 10%
MO6	100	LS_4	On children, rate 10%
MO7	200	LS_4	On children, rate 10%
MO8	100	LS_4	Periodic, every 10 iterations
MO9	200	LS_4	Periodic, every 10 iterations

Table 10: Different versions of NSGA-II in MOEA7

new population is then formed for the next generation. The D-MAENS was tested on three well-known CARP benchmark sets: *gdb, val and egl sets*, against NSGA-II and the MOEA-9 version (LMOGA henceforth) of Lacomme et al. (2006). On the I_D metric (distance from reference Pareto set), D-MAENS produced significantly better results than the others on 71 out of the total 81 instances. On $gdb1$ and $gdb19$, D-MAENS and LMOGA reached the minimal value 0 of I_D. In terms of $\triangle$ [Deb et al. (2000)] metric, D-MAENS was better than the NSGA-II and LMOGA on 10 gdb instances, 25 val instances and 5 egl instances. It was also found that D-MAENS achieved larger value of the hyper-volume measure on 18 gdb instances, 32 val instances and 22 egl instances. The LMOGA was superior on gdb instances, but the NSGA-II could not surpass others on any instance.

NPM3: Grandinetti et al. (2012) tackled a multi-objective undirected CARP, optimizing three conflicting objectives at the same time. The considered objectives were minimization of transportation cost, makespan (cost of the longest route) and the number of vehicles. They approached the problem by ϵ-constraint method, considering the third objective (minimization of the number of vehicles) as a parameter of the optimization procedure. The proposed optimization algorithm consists of following four stages:

stage 1 *Populate* Ω: The generation of a set of feasible solutions (Ω) in the first stage was accomplished with the aid of three heuristics: path scanning with an ellipse rule [Santos et al. (2010)], Deterministic TS [Brandáo and Eglese (2008)] and ILS. In the ILS, the initial solution is created randomly and the local search contains three inter-route moves: Add, Switch and Remove. The method *Add* removes a required edge from its current route and inserts into any other route. Whereas, the function *Remove* inserts into the least-cost route. The *Switch* is the swapping mechanism of two randomly selected required tasks.

stage 2 $Optimize_{first}$: In the second stage, $Optimize_{first}$ solves a mixed integer linear model in which the transportation cost is minimized while considering the makespan as a constraint.

stage 3 $Optimize_{second}$: Likewise, $Optimize_{second}$ minimizes the makespan while fixing an upper limit on the cost function.

stage 4 Dominance (Ω): Finally, Dominance(Ω) returns all the non-dominated solutions by removing the dominated individuals from Ω.

The algorithm was run for each possible value of the number of vehicles and the two optimization models were solved by CPLEX ILOG 10.1. They tested the performance of this ϵ-constraint method against the NSGA-II of Lacomme et al. (2006). On gdb set, in about 57% of the problems, the number of non-dominated solutions was higher. The average improvements of the makespan and routing cost objectives were equal to 9.75% and 1.21%, respectively. On val set, in 10 (out of 33) problems, the minimum routing cost established by the NSGA-II was improved. However, obtained values of the makespan on this dataset were mostly worse. On egl set, in majority of the instances, the number of trade-off solutions was higher and the lowest value of the routing cost on the approximated Pareto front was better.

2.4 Mixed General Routing Problems

A general VRP that seeks to design vehicle routes on a mixed graph for servicing full-streets and specific spots both falls into the class of mixed general routing problems (MGRPs). Under the capacity constraint, the MGRP can be termed as the Mixed Capacitated General Routing Problem (MCGRP). It is an integrated form of node and edge/arc routing problems, as represented in the Figure 31. The *newspaper delivery* and *urban waste collection* are typical examples of such problems. The network graph consists of all three entities: *nodes, edges and arcs*.

2.4.1 Single-objective MCGRP

HA5: Pandit and Muralidharan (1995) first addressed the MCGRP, routing a heterogeneous set of vehicles over specified segments and nodes of a street network under the capacity and route duration constraints. The problem was denoted as the Capacitated General Routing Problem (CGRP). They formally defined the CGRP and designed a route-first partition-second based heuristic algorithm for solving it. Firstly, a condensed

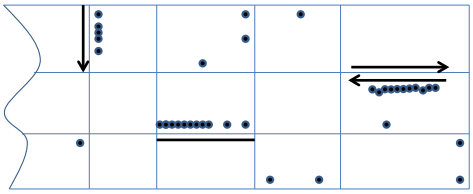

Figure 31: Mixed routing

network graph G_1 is formed using nodes, edges and arcs that need to be visited. Next, the disconnected components of the sub-graph G_1 are joined together by finding a minimum spanning tree (MST) between these components. Once the MST is formed, the direction of arcs are restored and the graph G_1 is augmented with the links of MST. The restoration of direction on arcs may create zero in-degree or zero out-degree nodes on the connected graph G_2. To make G_2 a strongly connected graph, additional links are added to it such that there is a path between each pair of vertices. They devised a fast heuristic procedure that uses Tarjan's algorithm [Eswaran and Tarjan (1987)] to find strongly connected components. Finally, the strongly connected sub-graph is converted into an Eulerian graph which produces a tour. This Eulerian tour is partitioned into vehicle routes. They tested the performance of this heuristic algorithm on randomly created test instances from curb-side waste collection in residential areas on a network with 50 nodes and 100 links. They also investigated the robustness of proposed method on random instances of the Capacitated Chinese Postman Problem for which they had two lower bound procedures. From computational experiments, they observed: (1) As the percentage of directed arcs increases, the ratio of the solution to the lower bound increases and (2) the solution value also increases as the number of directed arcs increases because the distance to reach some of the nodes/arcs from the depot increases. They integrated their solver in a micro-computer based interactive route planning system.

HA6: Gutiérrez et al. (2002) solved a homogeneous fleet version of the CGRP studied by Pandit and Muralidharan (1995) under the given number of vehicles. However, the problem was titled as CGRP-m (CGRP on mixed graphs). They devised a heuristic procedure, which constructs vehicle routes sequentially. Firstly, for each required task, the algorithm finds the minimum cost route joining the required task with the depot. A largest route is selected and demands are added to this route according to the following preferences: (1) the nearest required element without violating the capacity constraint, (2) demands corresponding to required elements traversed by the route but with demands not inserted yet and (3) if the loaded demand is not greater than 0.9W (W: vehicle capacity), it tries to insert demands corresponding to required elements that are

very close to the route. The next largest route is selected and demands are added in the same fashion. This process is repeated until all required elements have been assigned to vehicles. At last, the heuristic proposed by López (1998) is applied to each route of the obtained solution for further improvement. They tested this heuristic algorithm on a set of 28 instances containing 20 - 50 vertices, 25 - 97 edges, and 0 - 16 arcs. They also compared it against the heuristic proposed by Pandit and Muralidharan (1995). On average, the heuristic of Pandit and Muralidharan (1995) produced a cost increment in the solution of 11.05 % and produced better solution in only one instance. In 2 instances (out of 28), this CGRP-m heuristic gave a solution with $k + 1$ vehicles (k is the maximum number of vehicles), whereas the heuristic of Pandit and Muralidharan (1995) produced a solution with $k + 1$ vehicles in 6 instances.

MA6: Prins and Bouchenoua (2004) proposed a MA to address the MCGRP, calling it as the Node, Edge and Arc Routing Problem (NEARP). The objective was to minimize the cost under the vehicle capacity constraint. First of all, they created an internal network in which all tasks were coded with the same attributes and stored together in a list. The attributes assigned to each task were: *a begin node, an end node, a traversal cost, a demand and a processing cost*. A chromosome was represented as a sequence of tasks without trip delimiters. Furthermore, the initial population was filled with the solutions generated by well-known heuristics: Nearest neighbor with path scanning, Augment-Merge with Clarke & Wright saving algorithm and the first method with *Split*, and random chromosomes. The clones were managed in the same way as in Prins (2004). For recombination, they used order crossover operator developed by Oliver et al. (1987) to quickly produce good solutions. The mutation was performed by a LSP (see Prins (2004)) consisting of following moves: task flipping, movement of one task after another task or after the depot, movement of two consecutive tasks after another task or after the depot, swapping and 2-opt. An offspring created by recombination is first converted into a MCGRP solution (a list of tasks with trip delimiters) using the *Split* procedure and then LSP is applied to it. The resulting solution is structured back into a standard chromosome by concatenating all trips. They established the first upper bounds on 23 randomly generated instances. However, they tested the performance of this MA on 25 CARP instances [Golden et al. (1983)] and 14 VRP instances created by Christofides et al. (1979). For CARPs, this MA solved 17 out of 23 instances to optimality, consuming on average 42 seconds and giving an average deviation of 0.43 % from the best knowns. While in the case of VRP instances, MA retrieved 3 best known solutions, produced a small average deviation of 0.39 % from the best known solutions and utilized an average CPU time of 10 minutes.

SA2: Kokubugata et al. (2007) resolved the NEARP/MCGRP, proposing a simpler structure of the internal network for representation of the problem data and using SA algorithm. The internal network in this work is a three dimensional array in which the

first component expresses the head nodes of entities and the second expresses the tail nodes. The third is a Boolean value that attains 1 if and only if the entity is an arc. A solution for the NEARP was expressed as a sequence of integers representing required tasks with trip delimiters. The algorithm starts with an initial solution, which is generated as follows. First, all tasks are sorted in the descending order of their quantities of demand. Afterward, the sorted tasks are assigned to the vehicles without violating their loading capacity. Subsequently, a solution string is formed in accordance with the assignment. Furthermore, random changes are applied according to one of the three transformation rules (one to one exchange, delete & re-insert and partial reversal) for 1000 times to the initial solution. Then, for each transformation rule, the feasibility rate is calculated as the ratio of the number of feasible solutions generated to the number of total generated solutions. The feasibility rates are used in the main loop of the algorithm for probabilistically choosing a transformation rule to generate a new state of solution. With theses features, the algorithm produced some new best known solutions on the CBMix instances. Out of the 23 instances, SA with standard parameter settings obtained better results on 10 instances and similar results on 5 instances.

2.5 Summary

The large volume of published research papers on VRPs shows its importance in the distribution management. In this part of the thesis, a survey was conducted on the single and multi-objective approaches to routing problems under the vehicle capacity constraint. This survey included different routing mechanisms (node, edge/arc and their combinations), but focused on only applications of heuristic and meta-heuristic techniques. It was found that local search based methods (such as TS, SA, MA and GA with a LSP) yielded excellent results on the standard benchmark instances for mono-objective VRPs. The NSGA-II, on the other hand, has been the top choice for solving multi-objective VRPs. It is clearly noticeable from this survey that there is a dearth of research on the single and multi-objective MCGRP despite the fact that it possesses the peculiarities of real-life routing problems. Hence, the VRP research community should pay more attention on the MCGRP, especially on the multi-objective model. In this thesis, the work of Prins and Bouchenoua (2004) has been extended by including the route balance objective. Seeing the effective use of NSGA-II for multi-objective VRPs in literatures, a Memetic version of it has been designed to solve the problem. This research, detailed in the next two parts, is expected to make a significant contribution in promoting research on the multi-objective MCGRP.

* * * * *

Part 3

Application of Memetic NSGA-II

Application of Memetic NSGA-II on bi-objective MCGRP

This part of the thesis begins with the description of the vehicle tour construction rule. Subsequently, the considered model of bi-objective MCGRP is presented along with the mathematical formulation of objectives and constraints. The importance of considered objectives: minimization of the overall routing cost and the route balance, in the vehicle route planning has also been briefly highlighted. Next, the working process of Memetic NSGA-II is described, while detailing its three major components (*Dominance based local search procedure, X-set and Clone management principle*). Lastly, some details of the CBMix dataset have been provided on which the effectiveness of the Memetic NSGA-II was examined.

3.1 Problem background

In the MCGRP model considered in this research work, the unlimited number of homogeneous vehicles are available, all having an equal carrying capacity. The routes must be closed; therefore, vehicles start their journey from the depot node, process some assigned tasks in the given order and return back to the same depot. In order to have a clearer picture, let us consider a very small hypothetical instance containing five nodes $(N1, N2, ..., N5)$, one depot node, one vehicle with the infinite loading capacity and three tasks. These three tasks, shown in Figure 1, are as follows:

1. node task $(N1)$.
2. task along edge $(N2 \leftrightarrow N3)$ having uni-directional sides.
3. arc task $(N4 \rightarrow N5)$.

$\ll See\ Figure\ 1 \gg$

Let's also assume that the processing/visiting order of these three tasks defined for this vehicle is $1 \rightarrow 2 \rightarrow 3$. As stated and also shown in the Figure 1, a route originates from the depot node. The first task to be processed is a node task $(N1)$; therefore, vehicle follows the least-cost path to arrive at the node $N1$ from the depot and processes the associated task. Now, the next is an edge task, processing of which can be started from either $N2$ or $N3$. Since node $N2$ is *closer* to $N1$ than $N3$, vehicle goes to $N2$, begins service and finishes at node $N3$. The last task to be serviced is an arc task, processing of which can only be started from its beginning node $N4$. So, vehicle moves to $N4$ and traverses in the direction of the arc while processing the assigned task and eventually returns back to the same depot. This is how a vehicle tour is constructed in this work.

In the bi-objective MCGRP addressed in this thesis, vehicles have limited loading capacity and therefore a solution may contain several tours, each constructed by a different vehicle. The aim is to design an efficient delivery routes for the vehicles so that the objectives (minimization of route balance and routing/traversal cost of vehicles) could be jointly optimized. And for this purpose, Memetic NSGA-II - a Pareto-dominance based MOEA - has been conceptualized which determines the visiting order of required tasks for the vehicles.

Furthermore, among the several existing performance indices for MOEAs, the Hypervolume metric [Zitzler and Thiele (1999)] has been considered in this research for a fair assessment of the Pareto sets retuned by the Memetic NSGA-II. It can compare two sets in terms of both criteria: *proximity to the true Pareto set and diversity among the trade-off solutions.*

$\ll See\ Figure\ 22 \gg$

It can be observed in the Figure 22 that lateral diversity (convergence towards the true Pareto curve) and longitudinal diversity (the spread of solutions along the Pareto curve) will improve if the magnitude of hyper-volume surges. Nonetheless, it is not guaranteed that a front closer to the true Pareto front than another one will have higher hypervolume as it is also governed by the number of solutions in the Pareto set and the shape of the Pareto front.

3.2 Model of bi-objective MCGRP

The bi-objective MCGRP considered in this study can be defined on a mixed weighted graph, $G = (V, A \cup E)$, where $V = (V_R \cup V_{NR})$, $E = (E_R \cup E_{NR})$ and $A = (A_R \cup$

A_{NR}). The V stands for the set of nodes, E is the set of edges and A represents the set of arcs. The symbols (V_R, E_R, A_R) and (V_{NR}, E_{NR}, A_{NR}) stand for the set of (nodes, edges, arcs) with required and non-required tasks, respectively. Each required task j ($j = 1, 2, 3, \ldots, \mid J \mid$), with $J = V_R \cup E_R \cup A_R$, is associated with a non-negative demand d_j of some goods/services. A homogeneous fleet $|M|$ of vehicles, each having a maximum loading capacity of Q units, are based at the depot node. The size of the vehicle fleet ($|M|$) is assumed to be large enough to ensure the existence of a feasible solution.

The problem seeks to design the set of optimal vehicle routes to process all required tasks. Each route, constructed by a different vehicle m ($m = 1, 2, 3, \ldots, |M|$), consists of *permutation* of some required tasks which represents their processing/visiting order. The routes have to be constructed in such a way that two objectives, namely, minimization of routing cost and route balance, could be optimized concurrently while satisfying some constraints. The following three basic VRP constraints have been considered in this research work.

- The route of a vehicle should originate and end at the depot node.
- Each required task must be visited exactly once by exactly one vehicle.
- Total demands of tasks on each tour should not exceed the vehicle capacity.

3.2.1 Mathematical formulation

In this section, importance of the two aforementioned objectives (minimization of routing cost and route balance) will be briefly highlighted. Subsequently, the objectives and constraints will be mathematically represented using the following 0/1 variable. The various notations used in this part have been provided in the Table 11. In addition, $j = j' = 0$ denotes the depot node.

$$\chi_{mjj'} = \begin{cases} 1 & \text{if vehicle } m \text{ visits task } j' \text{ immidiately after visiting task } j \\ 0 & \text{otherwise} \end{cases} \tag{3.2.1}$$

- ***Minimization of routing cost***: The routing cost has been the primary criterion to assess the effectiveness of a vehicle routing plan. It accounts for a major portion of the whole logistic cost and therefore directly impacts on the revenues/long term sustainability of the logistic company. Hence, this objective function has been

G	The network graph
$\|M\|$	Total number of vehicles used
$\|J\|$	Total number of required tasks
j, j'	Index for required tasks
m	Index for vehicles
$c_{jj'}$	Cost of deadheading from task j to task j'
t_j	Traversal cost of task j
c_m	Travel cost of vehicle m
c_{max}	Maximum route cost
c_{min}	Minimum route cost
O_1	Overall routing cost (RC)
O_2	Route balance (RB)
$\chi_{mjj'}$	A 0-1 variable

Table 11: Notations-I

considered. It can be measured in terms of any unit related to the economy of the plan, e.g., actual travel cost, distance travelled, time and the number of served customers [Jozefowiez et al. (2008)]. In this study, however, routing cost has been defined as the total traversal cost of vehicles to process all the given required tasks. It is represented by O_1, as shown in equations (3.2.2)–(3.2.3).

$$O_1 = \sum_{m=1}^{|M|} c_m \tag{3.2.2}$$

$$c_m = \sum_{j=0}^{|J|} \sum_{j'=0,\, j' \neq j}^{|J|} \chi_{mjj'} \times (c_{jj'} + t_{j'}) \tag{3.2.3}$$

The equation (3.2.3) computes the cost associated to vehicle m. The deadheading cost from j to j' ($c_{jj'}$) is calculated as the minimum cost of travel from the tail (node at which the vehicle stops processing) of j to the head (node at which the vehicle begins processing) of j'. In case of an edge task, as mentioned earlier, both ends are checked and the vehicle starts processing from the nearest end (routing cost is proportional to the travel time). Also, note that $t_{j'}$ (traversal cost) is zero for all nodes and so is for the depot node.

- ***Minimization of route balance***: Efficient utilization of resources (materials, machines or humans) is one of the important factors for success of any enterprise. Thus, it is of utmost importance to strike an appropriate balance among drivers'

working duration and vehicles' usage in the distribution business. The route balance objective fulfils this purpose as it helps to efficiently allocate customers/tasks among vehicles. In fact, it is another important criteria that greatly influences the vehicle route planning and decision making process. It can be defined as the difference between two tours with respect to some disparity measures (e.g., the number of served customers, total demands of goods and duration/length/cost). In the present research, two tours are discriminated with respect to their *routing cost* values. This objective function has been represented by O_2 and is computed as shown in the equations (3.2.4)–(3.2.6).

$$O_2 = c_{max} - c_{min} \tag{3.2.4}$$

$$c_{max} = max\left\{c_m; m = 1, 2, 3, ..., |M|\right\} \tag{3.2.5}$$

$$c_{min} = min\left\{c_m; m = 1, 2, 3, ..., |M|\right\} \tag{3.2.6}$$

Where, c_{max} and c_{min} are traversal costs of vehicles belonging to the largest and smallest routes (in terms of cost), respectively. Thus, route balance can simply be defined as the cost difference between the costliest tour and the cheapest tour.

3.2.2 Constraints

Both the objectives (minimization of routing cost and route balance), which are of equal importance for management and operational personnel (e.g., drivers), are jointly optimized under the following constraints.

$$\sum_{j=1}^{|J|}\left(d_j \sum_{j'=0,\, j\neq j'}^{|J|} \chi_{mjj'}\right) \leq Q \qquad \forall\, m \in M \tag{3.2.7}$$

$$\sum_{j=0,\, j\neq j'}^{|J|} \sum_{m=1}^{|M|} \chi_{mjj'} = 1 \qquad \forall\, j' \in J \tag{3.2.8}$$

$$\sum_{j=0,\, j\neq p}^{|J|} \chi_{mjp} - \sum_{j'=0,\, j'\neq p}^{|J|} \chi_{mpj'} = 0 \qquad \forall\, m \in M,\ p = 0, \ldots, |J| \tag{3.2.9}$$

$$\sum_{j'=1}^{|J|} \chi_{mjj'} = 1 \qquad j = 0,\ m \in M \tag{3.2.10}$$

$$y_j - y_{j'} + |J| \times \sum_{m=1}^{|M|} \gamma_{mjj'} \leq |J| - 1 \qquad \forall j, j' \in J,\ j \neq j' \qquad (3.2.11)$$

$$\chi_{mjj'} \in \{0, 1\} \qquad \forall\ j, j', m \qquad (3.2.12)$$

$$y_j \qquad arbitrary.$$

Constraints (3.2.7) ensure that the vehicle loading capacity will not be violated on any route. Constraints (3.2.8) state that each task is visited exactly once. Expression (3.2.9) states that if a vehicle arrives at a task, it also departs from it. Constraints (3.2.10) confirm that each vehicle $m \in M$ must be used exactly once. The subtour-elimination condition has been represented by equation (3.2.11) [Miller et al. (1960)]. Constraints (3.2.12) impose binary conditions on the variables.

It can be noticed that the problem has been formulated as an integer programming formulation of the CVRP [Christofides et al. (1981)]. Using the definition of the decision variable $\chi_{mjj'}$ (equation 3.2.1), it could be done nicely and the work was mainly focused on the development of the algorithmic model. In the next section, Memetic NSGA-II has been described which employs appropriate solution structures according to the definition of $\chi_{mjj'}$.

3.3 Solution Methodology

An Evolutionary Algorithm (EA) is seen as a potential optimization technique for solving real-world hard combinatorial optimization problems. EAs draw inspiration from *natural selection*/or *survival-of-the-fittest* of biological evolution process, and have already shown their strengths on a variety of hard optimization problems. It maintains a population of solutions, which undergo an iterative improvement process consisting of four successive phases:

1. Evaluation
2. Mating selection
3. Recombination and mutation
4. Environmental selection

The one characteristic that makes an EA most suitable for the MOP is its inherent ability to carry several solutions simultaneously during the search. This enables it to approximate an entire Pareto set in just a single simulation run. Thus, to solve the present

step 1:	Initialize a population of parent solutions (P_t)
step 2:	Evaluate objective values for each solution of P_t
step 3:	Improve solutions of P_t by DBLSP
step 4:	Apply *non-dominated sorting* procedure on P_t
step 5:	Calculate *crowding distance* of solutions on each front
step 6:	Form a mating pool using *crowded comparison operator*
step 7:	Perform *crossover operation* and make a child population (Q_t)
step 8:	Apply DBLSP on each solution of Q_t
step 9:	Create a combined population $R_t = P_t + Q_t$
step 10:	Remove clones from R_t by CMP
step 11:	Execute *step* 4 - *step* 5 on R_t
step 12:	Form a new population P_{t+1} (Environmental selection)
step 13:	Repeat *step* 6 - *step* 12 until (*stopping criteria (S.C.)*)

Table 12: Pseudocode of Memetic NSGA-II

bi-objective MCGRP, the framework of NSGA-II has been utilized. It was empowered by a dominance based local search procedure (DBLSP) and a clone management principle (CMP). A pseudocode illustrating the working process of proposed technique, coined as Memetic NSGA-II, is provided in the Table 12. The implementation details of its various stages have been discussed in the following subsections.

3.3.1 Initialization of population and Evaluation

Like any other EAs, Memetic NSGA-II also begins with a population of candidate solutions. Following the notion of stochastic search technique, the initial population is generated uniformly at random to favour unbiased investigation of the solution space. Each member of the initial population is further improved by DBLSP for better convergence and quality of the final non-dominated solutions. A solution has been coded as the *permutation* of integers $[1 \ldots |J|]$ representing required tasks. It can be seen as a giant tour for a vehicle of infinite carrying capacity. Whenever needed in any step of the algorithm, it is decomposed into a complete MCGRP solution (i.e., list of required tasks with trip delimiters) by simply dividing it according to the vehicle loading capacity. Before describing the further steps of the Memetic NSGA-II, the working process of DBLSP is being detailed below.

Dominance based local search procedure: The DBLSP is one of the main ingredients within the framework of Memetic NSGA-II, and consisting of three phases. Each phase contains a different *inter-route* neighborhood operator. The following three operators

have been used: 2-opt, Re-insert and $\lambda - interchange$ with $\lambda = 1$. A solution is successively passed through *all three* phases while updating the current solution as shown in the Figure 32. The order in which these phases are used is determined randomly. Each of these phases remains active until a new solution is discovered that *dominates* the current solution or the end of *full search*.

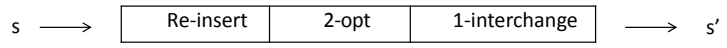

Figure 32: A random sequence of phases

To speed up the convergence, a restricted number of neighbours are explored as follows. Let $R(j)$ stands for the route containing task j and (j, j') stands for the partial route from task j to task j'. Now, assume that $N(j)$ is the set of nearest neighbours of task j and j'' is a task chosen in the set $N(j) \setminus R(j)$. The operators work in the following way:

- 2-opt: Let j' be the successor of j in $R(j)$ and j''' be the successor of j''. For each task j, for each task j'', replace (j, j') and (j'', j''') by (j, j'') and (j', j'''), respectively.
- Re-insertion: For each task j, for each j'', j is removed from its current position and inserted right after j''.
- $1 - interchange$: each task j, for each j'', positions of j and j'' are interchanged.

To implement these neighborhood operators in the above described manner, first a task j is randomly picked up. Next, the nearest neighbors of task j are determined. Subsequently, the neighborhood exploration is begun with the closest task j''. Once all combinations of j and j'' are checked, the process is repeated with a new randomly selected task j that has not been used before. Nevertheless, as mentioned above, neighborhood exploration in each phase is stopped once a better solution (in terms of dominance relation) has been found. The current solution is updated and passed to the next phase. The operators are applied to the complete MCGRP solution. The DBLSP may produce an infeasible solution; therefore, the resulting solution is changed into a giant tour by removing trip delimiters and converted back into a feasible MCGRP solution before evaluating the objective values. As described in the subsection 3.2.1, the routing cost is calculated as the sum of traversal cost of all vehicles (equations 3.2.2 – 3.2.3) running between nodes on the network. Whereas, route balance is equal to the cost difference between the most expensive tour and the least expensive tour (equations 3.2.4 – 3.2.6) in the obtained set of vehicle tours.

3.3.2 Formation of mating pool

As mentioned earlier in the first part of this thesis, the mating pool ($\uplus$) contains solutions which mate with each other to give birth to new solutions, called off-springs in the context of an EA. The size of $\uplus$ is set equal to that of population; however, only promising solutions are inserted into it. In fact, members of $\uplus$ in NSGA-II are selected by a special operator, called as *crowded comparison operator* ($\prec_t$). It merely uses two attributes: rank and crowding distance, associated with solutions to guide the selection process at various stages of the algorithm toward a uniformly spread-out Pareto-optimal front (PF^*)[Deb et al. (2002)]. The procedures to determine non-domination rank and crowding distance of solutions are explained below.

Non-dominated sorting: It is a method of classifying solutions into different fronts of dominance in such a way that non-dominated solutions lie on the nearest front to PF^*. In other words, closer a front to PF^*, better is the solution belonging to it. The rank of a solution is simply the front number (f) on which it lies. The implementation details of sorting procedure can be divided into three phases as shown in the form of a pseudocode in Table 13. Firstly, for each individual, domination count (the number of solutions that dominates it) and solutions dominated by it are determined. Secondly, all those solutions which have zero domination count are extracted. These are trade-off solutions and therefore assigned to the first front (PF_1). The subsequent fronts are formed in the third phase as follows. For each solution in the newly formed front, domination count of all solutions dominated by it is reduced by one. In doing so, if domination count of any solution becomes zero, it is stored in a separate list (H). These solutions together create the second non-dominated front (PF_2). This process is continued until all solutions have been decomposed into different fronts. It is worth noting here that a member of $PF_f(f > 1)$ is dominated by solutions of the population belonging to $PF_1 \cup PF_2 \cup \cdots \cup PF_{f-1}$.

Crowding distance ($\rho_{distance}$): It provides an estimate of the density of solutions surrounding that solution [Deb et al. (2002)], and more importantly, is calculated front wise. For a solution, it is defined as the average length of the cuboid formed by using its nearest neighbours as vertices. Therefore, a solution with smaller value of $\rho_{distance}$ will be more crowded. The computation of $\rho_{distance}$ requires the sorting of solutions along each objective. Once solutions are arranged, the extreme points are assigned an infinitely large value as their $\rho_{distance}$. For all intermediate points, the sum of distances between adjacent solutions over all the objectives gives $\rho_{distance}$. If this concept is applied to the bi-objective case shown in the Figure 33, $\rho_{distance}$ of solution 'b' will simply be equal to $d_1 + d_2$, which is indeed the *Manhattan* distance between its immediate neighbour solutions ('a' & 'c') on the front. Furthermore, since 'a' & 'c' are boundary solutions, $\rho_{distance}$ can be set a very large number for them. A pseudocode for com-

$Phase$ 1:
For each $p \in P$ $\quad P:$ $Population\ of\ solutions,\ p, q : solution\ counter$
$S_p = \emptyset$ $\quad S_p : Set\ of\ solutions\ dominated\ by\ p$
$n_p = 0$ $\quad n_p : no.\ of\ solutions\ that\ dominate\ solution\ p$
for each $q \in P$
if ($p \prec q$), then $S_p = S_p \cup q$
else if ($q \prec p$), then $n_p = n_p + 1$
End for
End For

$Phase$ 2:
For each $p \in P$
if ($n_p = 0$), then $PF_1 = PF_1 \cup p$ $PF_1 : First\ front$
End For

$Phase$ 3:
$f = 1$ $\quad f : front\ counter$
while ($|PF_f| \neq 0$)
$H = \emptyset$ $\quad H : list\ to\ store\ members\ for\ next\ front$
for each ($p \in PF_f$)
for each ($q \in S_p$)
$n_q = n_q - 1$
if ($n_q = 0$), then $H = H \cup q$
End for
End for
$PF_{f+1} = H$
$f = f + 1$
End while

Table 13: Non-dominated sorting

puting $\rho_{distance}$ in the objective space of any dimension is provided in Table 14. While calculating it, the distance between solutions along each objective is locally normalized as shown in Table 14. If not done so, the search may become biased towards the objective function with largest value.

Having calculated the non-domination rank and the crowding distance of solutions, $\prec_t$ in conjunction with the binary tournament selection (with replacement strategy) is used to create $\uplus$. Basically, two solutions are randomly drawn from the population and $\prec_t$ is applied to them. It selects a solution with lower rank; if they possess the same rank,

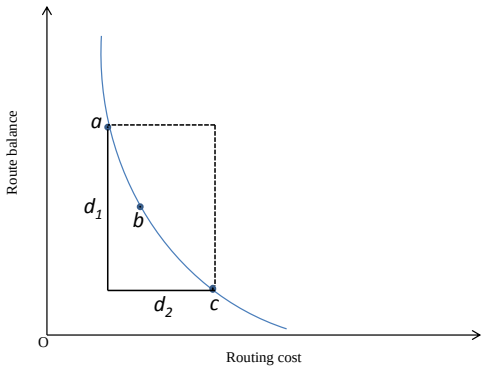

Figure 33: Crowding distance calculation (bi-objective case)

For each front PF_f
  Determine the total number of solutions on it (|l|)
  For each solution $p \in PF_f$
    set $\rho^p_{distance} = 0$
  End For
  For each objective O
    sort solutions along O (in the ascending order)
    set $\rho^1_{distance} = \rho^{|l|}_{distance} = \infty$
    For $p = 2$ *to* $|l| - 1$
      $\rho^p_{distance} = \rho^p_{distance} + \frac{PF_f[p+1].O - PF_f[p-1].O}{PF_f[O_{max}] - PF_f[O_{min}]}$
    End For
  End For
End for

Note: $PF_f[p].O$: value of objective O of solution p on the front PF_f
$PF_f[O_{max}]$ & $PF_f[O_{max}]$: maximum and minimum values of objective O on the front PF_f

Table 14: Crowding distance calculation

then the solution that resides in the lesser crowded region is preferred. This process is repeated until $\uplus$ is completely filled.

3.3.3 Recombination, Mutation & Environmental selection

During the recombination phase, off-springs are generated by mixing genetic properties of two or more parents chosen from $\uplus$. Following the scheme of binary tournament selection, two solutions are drawn randomly from $\uplus$, and a crossover operator is applied on them to produce child solutions. One child is generated via crossover in Memetic

NSGA-II; hence, this process is repeated $\lceil Pop_{size} * P_c \rceil$ times. The Memetic NSGA-II has been appended with a set containing three well-known crossover operators (X-set) for ordered chromosomes:

- Partially mapped crossover (PMX)[Goldberg and Robert Lingle (1985)]
- Order crossover (OX)[Oliver et al. (1987)]
- Edge recombination crossover (ERX)[Whitley et al. (1989)]

$$\ll see\ section\ 1.2.3\ for\ details\ of\ operators \gg$$

One of them is chosen (with equal probability) each time crossover is to be performed. All of these operators create a feasible solution string (i.e., each task appears only once in it). Furthermore, once an off-spring is created, it is sent to the DBLSP for its maturation/education. The finally obtained child solutions after crossover and DBLSP may be similar to one or more other solutions. The presence of such solutions, called clones, are extremely undesirable as they may cause premature convergence to a suboptimal Pareto front. To avoid this in Memetic NSGA-II, a clone management principle (CMP) has also been included into its framework. In CMP, two solutions are considered clone of each other if they are at the same position in the *objective space* AND Jaccard's index based similarity co-efficient (Ω) is *greater than zero* in the *decision space*.

$$\Omega = \frac{\sum_{j=0}^{|J|-1} \sum_{k=j+1}^{|J|} \psi_{jk}}{\sum_{j=0}^{|J|-1} \sum_{k=j+1}^{|J|} \Psi_{jk}} \tag{3.3.1}$$

$$\psi = \begin{cases} 1 & \text{if tasks } j \text{ and } k \text{ are adjacent to each other in both solutions} \\ 0 & \text{otherwise} \end{cases} \tag{3.3.2}$$

$$\Psi = \begin{cases} 1 & \text{if tasks } j \text{ and } k \text{ are adjacent to each other in one of the solutions} \\ 0 & \text{otherwise} \end{cases} \tag{3.3.3}$$

More importantly, similarity between two solutions is checked after converting their chromosome strings into a complete MCGRP solution (i.e., list of required tasks with trip delimiters). The trip delimiter (depot node) is represented by $j = 0$ in the equation (3.3.1). If solutions are 100 % similar, then Ω will be equal to 1. Once all clones

are detected, they are replaced by their mutated structures. To do so, two mutation operators, namely, SWAP and INVERSION have been utilized as follows:

$$\begin{cases} SWAP & 0 < \Omega \leq 0.5 \\ INVERSION & 0.5 < \Omega \leq 1 \end{cases} \tag{3.3.4}$$

In SWAP, two tasks are randomly chosen on the standard chromosome string of a clone and their positions are exchanged. The INVERSION operator, on the other hand, selects two points and the order of tasks between them is reversed. It is more likely that INVERSION will produce a new solution with higher degree of structural diversity, and this is why it has been used when Ω is quite large.

Once the population is clone free, non-dominated sorting and crowding distance calculation are done and then the new population is formed for the next generation. This stage is called Environmental selection (or survivor selection), and executed with the help of crowded comparison operator ($\prec_t$). Starting with the first non-dominated front (PF_1), solutions are inserted into the new population. While doing so, following two cases may emerge:

case 1 $|PF_f| \leq |rs|$: In this case, all solutions belonging to the front under consideration are inserted into the new population.

case 2 $|PF_f| > |rs|$: In this case, first solutions are sorted in the descending order of their crowding distance values with the largest one at the top and the smallest one at the bottom. Next, the top $|rs|$ solutions are inserted into the new population and rest of the solutions are rejected.

The symbols $|PF_f|$ and $|rs|$ stand for the total number of solutions on the Pareto front PF_f and the number of remaining slots in the population, respectively.

3.4 CBMix dataset

The Memetic NSGA-II has been implemented on the CBMix dataset. It was created by a random generator. The networks are mixed, planar, strongly connected and imitate the shape of real street networks [Prins and Bouchenoua (2004)]. The dataset consists of twenty-three large scale MCGRP (or NEARP) instances containing 11-150 nodes, 29-311 internal arcs and 20-212 tasks. The tasks comprise 3-93 node tasks, 0-94 edge tasks and 0-149 arc tasks. The vehicles are homogeneous with limited loading capacity

No.	Upper bound (U^b)	Author	Lower bound (U^l)	Author	GAP (%)
1	2589	PB	2409	BHW	7.2
2	12220	KMK	9742	BHW	22.3
3	3643	HKSG	3014	BHW	18.9
4	7583	PB	5302	BHW	35.4
5	4531	KMK	3789	BHW	17.8
6	7087	PB	5201	BHW	29.1
7	9607	HKSG	7296	BHW	27.3
8	10524	KMK	7956	BHW	27.8
9	4038	PB	3460	BHW	14.6
10	7582	PB	6432	BHW	16.4
11	4494	PB	3031	BHW	38.9
12	**3138**	BLMV	**3138**	BHW	0.0
13	9110	PB	6524	BHW	33.1
14	8566	PB	5731	BHW	39.7
15	8280	KMK	6318	BHW	26.9
16	8886	KMK	7416	BHW	18.0
17	4037	PB	3654	BHW	10.0
18	7098	KMK	6089	BHW	15.3
19	16347	KMK	11143	BHW	37.9
20	4844	PB	3452	BHW	33.6
21	18069	KMK	12474	BHW	36.6
22	1941	PB	1825	BHW	6.2
23	**780**	PB	**780**	BLMV	0.0

Table 15: Best known results on CBMix dataset

PB: Prins and Bouchenoua (2004), HKSG: Hasle et al. (2011), KMK: Kokubugata et al. (2007), BLMV: Bosco et al. (2013)

in all instances. The depot is *not* at the same node in all instances. See Prins and Bouchenoua (2004) for further details. The best known upper and lower bound values of the cost objective have been shown in the Table 15. As highlighted in boldface, the optimal solutions have already been found for two instances, that are, CBMix 12 and CBMix 13. The gaps from the best known upper bound (U^b) to the lower bound (U^l) vary between 0.0 % and 39.7 % with an average of 23.1 %. The % gaps have been calculated by the equation (3.4.1).

$$\% \; GAP = \frac{U^b - U^l}{(U^b + U^l)/2} \times 100 \tag{3.4.1}$$

Note: In addition to the CBMix dataset, which has mostly been used in the MCGRP literature, there are four more MCGRP dataset:

- BHW dataset [Bach et al. (2013)]
- DI-NEARP dataset [Bach et al. (2013)]
- MGGDB dataset [Bosco et al. (2013)]
- MGVAL dataset [Bosco et al. (2013)]

The 20 BHW, 114 MGGDB and 150 MGVAL instances were derived from the benchmark CARP instances. The 24 instances of the DI-NEARP benchmark were generated from six real life cases from the design of carrier routes for home delivery of subscription newspapers and other media products in the Nordic countries.

* * * * *

Part 4

Results and discussion

Results and discussion

In this segment of the thesis, the effectiveness of the whole algorithmic concept just described in the previous part will be tested on the CBMix dataset. The algorithm was coded in the Visual Studio C++ 2010, and run on windows server 2008 R2 Enterprise with 3.40 GHz and 16GB RAM. It was allowed to carry out till Max_{gen} iterations (see Table 17) no matter how much computational time is absorbed; however, it stops if *no change* in the hyper-volume measure (I_H) is observed in gen_{wc} consecutive iterations (see Table 17). Unless otherwise stated, the algorithm has been run *five times* and the presented results belong to the Pareto set *having the highest hypervolume* (not the minimum value of routing cost/route balance). See Table 16 for the definitions of various notations used in this part.

j	Index for required tasks
$\|J\|$	The total number of required tasks
I_H	Hyper volume of the obtained Pareto set
RNI	The ratio of non-dominated solutions
RC*	The lowest routing cost in the obtained Pareto set
RB	Associated route balance
RB*	The lowest route balance in the obtained Pareto set
RC	Associated routing cost
CPU(s)*	CPU time in seconds to obtain the whole Pareto set
Avg. RC	The average routing cost of the obtained Pareto set
Avg. RB	The average route balance of the obtained Pareto set
d_j	Demand associated with task j
t_j	Traversal cost of task j
I_j	A boolean parameter to identify an arc task

Table 16: Notations-II

The problem data has been modelled in the form of a two-dimensional array as shown in the Table 18. The second and third columns in the Table 18 show beginning node (b_n) and end node (e_n) of required tasks, respectively. The same value of these two

Notations	*Definitions*	*Values*
Pop_{size}	Population size	100
Max_{gen}	Maximum number of iterations	5000 * $\sqrt{\lvert J \rvert}$
gen_{wc}	Number of generations with *no change* in I_H	1000*$\sqrt{\lvert J \rvert}$
NNS	Nearest neighbour set size for local search	10 + $\sqrt{\lvert J \rvert}$
P_c	Crossover probability	0.95
L_s	Local search probability	1.0

Table 17: Algorithm parameters

j	b_n	e_n	d_j	t_j	I_j
1	6	6	10	0	0
2	4	5	12	8	0
3	7	9	13	11	1
-	-	-	-	-	-
-	-	-	-	-	-
-	-	-	-	-	-
$\lvert J \rvert$	11	13	20	40	1

Table 18: Problem data

attributes ($b_n = e_n$) verifies that j is a node task. Furthermore, a node task incurs the zero traversal cost as can be seen in the fifth column. The last column is assigned value 1 if and only if j is an arc task. The calculations of the objective functions require the determination of the minimum deadheading costs between all pairs of nodes in the mixed weighted graph. In this work, the Floyd-Warshall algorithm [Floyd (1962)] was implemented to find the cheapest paths between nodes beforehand. A pseudocode of the Floyd's algorithm has been provided in the Table 36.

4.1 Parameter tuning

To analyze the impact of parameters of the Memetic NSGA-II (population size and crossover & local search probabilities), initial computational experiments were conducted on the CBMix19, which is the largest instance in terms of the length of the solution string in the CBMix dataset. Following some existing guidelines, the algorithm was run with different values of these parameters and the obtained results were analysed with regard to the following criteria:

- Hypervolume metric

- RNI measure
- RC*/RB
- RB*/RC
- Avg. RC
- Avg. RB
- CPU(s)*

The details of the observations made while tuning the above mentioned parameters are provided below.

Effect of population size : The population size (Pop_{size}) was varied from 50 to 100 in a step of 25, keeping the crossover and local search probabilities at 0.95 and 1.0, respectively. As it can be seen in the Table 19, Memetic NSGA-II obtained the best RC* value of 18297 units with the population size of 100 solutions. As the size of the population increases, the values of RC* and Avg. RC get better. The algorithm produced better value of the average route balance (167.1 units) using 50 initial solutions than 75 and 100. However, the algorithm found solutions producing good route balance objective at the population sizes of 75 and 100 also. As tabulated, the algorithm gave RB* value of 31 and 89 units at the population sizes of 75 and 100, respectively. The value of the hypervolume measure (I_H) is slightly higher at the population size of 50 than 75 and 100, mainly because of comparatively larger range of the Pareto front along the cost objective (See Figures 34, 35 and 36). From these results, it can be concluded that a large population size (> 50) is not needed if the route balance objective is the main decision making factor. However, to obtain a large set of better compromised solutions, larger population sizes (≥ 75) seem to be more promising.

Effect of crossover probability : To check effects of the crossover probability (P_c), the algorithm was run with three different values (0.45, 0.70 and 0.95), fixing the size of the population and the local search probability at 100 and 1.0, respectively. The properties of the obtained Pareto sets and their images in the objective space have been shown in the Table 20 and Figures 37 – 39, respectively. Contrary to population size, the medium value of P_c produced better solutions in terms of the route balance objective. As presented, the algorithm gave Avg. RB of 171.696 units at $P_c = 0.7$. Whereas, it reached at the Avg. RB values of 206.509 and 268.295 units at P_c values of 0.45 and 0.95, respectively. With regard to the RNI measure, the algorithm performed better with P_c value of 0.95, discovering 95

different non-dominated solutions. Whereas, it could create only 57 and 69 non-dominated solutions with P_c of 0.45 and 0.70, respectively. The values of the hypervolume measure (I_H) are larger at $P_c = 0.45$ & 0.7 than at $P_c = 0.95$ due to higher values of the end points of the Pareto curves along the cost objective. From Pareto fronts portrayed in the Figures 37, 38 and 39, it is very clear that the distribution of non-dominated points is more uniform at high values of P_c (> 0.45). At $P_c = 0.95$, the algorithm obtained the best RC* value of 18297 units, while giving the associated route balance of 577 units. These observations certainly suggest the use high crossover probability (> 0.75) for better exploration/exploitation of the search space and create a more smooth & uniformly-spread-out Pareto front.

Effect of local search probability : The table 21 presents the summary of the obtained results on CBMix19 with different local search probability (P_{ls}) values. It was increased from 0.5 to 1.0 in a step of 0.25, while keeping the size of the population and the probability of crossover at 100 and 0.95, respectively. The behaviour of the algorithm is quite similar as in the above experiments on population size and crossover probability. Using high values of P_{ls}, the algorithm could generate a better approximation of the true Pareto set. As can be seen from the reported RNI values in the Table 21, the algorithm obtained more non-dominated solutions at $P_{ls} = 1.0$ than at 0.5 and 0.75. Despite this, the value of I_H is comparatively lower at $P_{ls} = 1.0$. The reason being the same, i.e, high end points of the cost objective. At $P_{ls} = 1.0$, the algorithm gave the lowest average routing cost (Avg. RC) of 20367.9 units, utilizing 49695.9 cpu seconds of computational time. The average route balance increases as P_{ls} value is raised. At the small value of P_{ls}, the algorithm obtained better value of the route balance objective, but the associated routing cost is much higher. On the other hand, at high values of P_{ls}, the algorithm produced very good solutions in terms of the routing cost. Also, the associated route balance objectives are not too bad. A high local search probability is therefore more effective for generating good compromised solutions. Nonetheless, computational time increases as more solutions undergo the dominance based local search procedure (DBLSP). As tabulated, when the local search probability was increased from 0.5 to 1.0, the algorithm took 16534.1 cpu seconds more to output the final Pareto set.

I_H	CPU(s)*	RC*	RB	RB*	RC	RNI	Avg. RC	Avg. RB	Pop_{size}
1.14894	22333.3	19619	342	22	30566	1.0	22562.9	167.1	50
1.09258	31972.8	18978	499	31	27619	0.9733	21813.2	222.274	75
0.902345	49695.9	18297	577	89	23209	0.95	20367.9	268.295	100

Table 19: Effect of population size

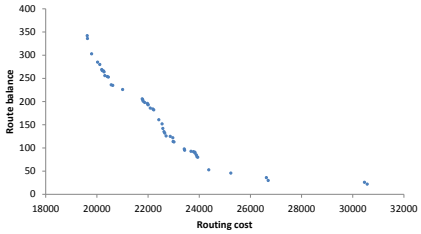

Figure 34: Pareto front of CBMix19 ($Pop_{size} = 50$)

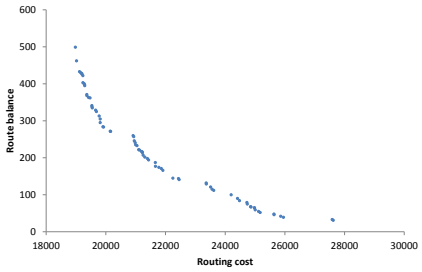

Figure 35: Pareto front of CBMix19 ($Pop_{size} = 75$)

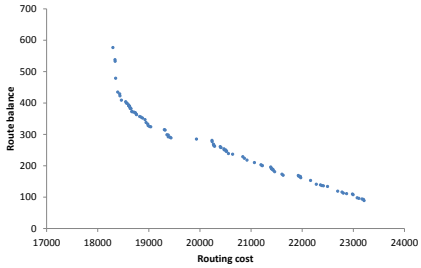

Figure 36: Pareto front of CBMix19 ($Pop_{size} = 100$)

I_H	CPU(s)*	RC*	RB	RB*	RC	RNI	Avg. RC	Avg. RB	P_c
0.925304	18183	19874	543	62	24719	0.57	21694.7	206.509	0.45
0.997011	33968.3	19515	390	30	26390	0.69	22632.9	171.696	0.70
0.902345	49695.9	18297	577	89	23209	0.95	20367.9	268.295	0.95

Table 20: Effect of crossover probability

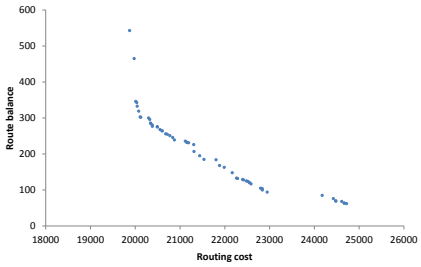

Figure 37: Pareto front of CBMix19 ($P_c = 0.45$)

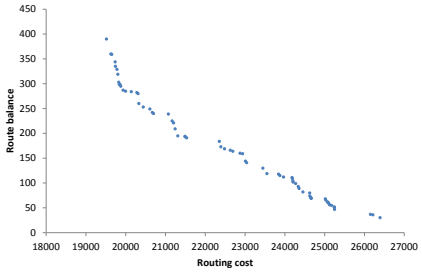

Figure 38: Pareto front of CBMix19 ($P_c = 0.7$)

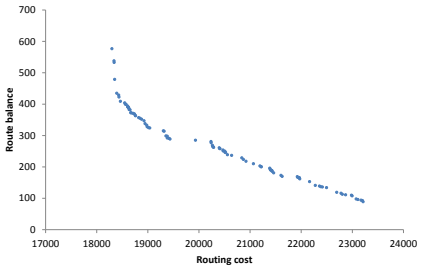

Figure 39: Pareto front of CBMix19 ($P_c = 0.95$)

I_H	CPU(s)*	RC*	RB	RB*	RC	RNI	Avg. RC	Avg. RB	P_{ls}
1.0431	33161.8	19000	448	25	26367	0.83	22080.7	183.867	0.5
0.998965	39919	19095	502	37	25304	0.87	21483.5	211.184	0.75
0.902345	49695.9	18297	577	89	23209	0.95	20367.9	268.295	1.0

Table 21: Effect of local search probability

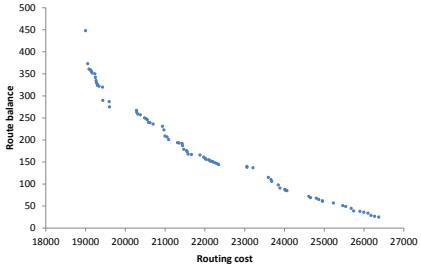

Figure 40: Pareto front of CBMix19 ($P_{ls} = 0.5$)

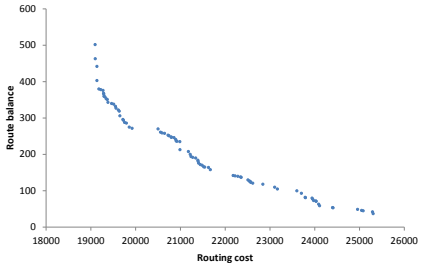

Figure 41: Pareto front of CBMix19 ($P_{ls} = 0.75$)

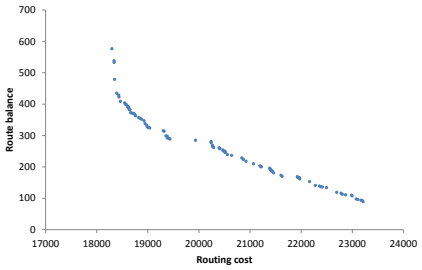

Figure 42: Pareto front of CBMix19 ($P_{ls} = 1.0$)

Based on this preliminary experiments, the local search and crossover probabilities were set equal to 1.0 and 0.95, respectively, for maximum exploration/exploitation of the search space. Thus, all off-springs obtained after the crossover operation undergo the local improvement through DBLSP. A population size of 100 solutions was maintained throughout the search. Furthermore, it is a well known fact that the performance of meta-heuristic algorithms is hugely affected by the length of the solution string in problems where some sort of sequences are needed to find out. In the present case, for example, the optimal processing order of tasks for the vehicles has to be determined. Therefore, some other parameters of the Memetic NSGA-II were designed as a function of $|J|$. As for instance, NNS ($\leq |J| - 1$) was made directly proportional to $|J|$, as shown in the Table 17). As the size of the problem increases, intensity of the local search also augments accordingly to investigate large part of the increased search space. The maximum number of iterations and stopping criteria (gen_{wc}) were set ($5000 * |J|$) and ($1000 * |J|$), respectively.

4.2 A comparative analysis

Since the reference Pareto set is not available; therefore, a comparative analysis will be performed in this section between the solutions produced by the proposed Memetic NSGA-II and a basic version of the algorithm (NSGA-II with only X-set and CMP). The results obtained by the Memetic NSGA-II and its basic version (will be called XNSGA-II henceforth) have been provided in the Table 22. The columns labeled RC* and RB* in Table 22 give the lowest values of the cost function and the route balance objective, respectively, found by the algorithms. Out of the 23 instances, Memetic NSGA-II yielded better value of RC* than XNSGA-II on the first 22 cases. Both versions reached at the same value of 780 units on CBMix23. However, Memetic NSGA-II achieved it, producing higher value of I_H. Also, the associated route balance (48 units) in the solution found by Memetic NSGA-II is lesser than the value (65 units) obtained by XNSGA-II. Moreover, the average value of RC* (7839.0 units) given by Memetic NSGA-II is significantly better than the value (9485.13 units) achieved by XNSGA-II. While dealing with the two highly conflicting objectives, Memetic NSGA-II also retrieved the best known upper bound values of the cost objective on CBMix22 and CBMix23. The Memetic NSGA-II succeeded well in finding solutions with very low route balance objective (without being biased) on CBMix instances. As shown in Table 22, the algorithm discovered solutions giving route balance in a very small range [1-20] on 9 instances. It produced an average RB* value of just 58.652 units, whereas XNSGA-II could obtain an average value of 93.304 units of RB*. These statistical analyses strongly advocate the application of Memetic NSGA-II for constructing good compromised solutions for multi-objective general routing problems.

	Memetic NSGA-II						XNSGA-II					
No. ($\|J\|$)	I_H	CPU(s)*	RC*	RB	RB*	RC	I_H	CPU(s)*	RC*	RB	RB*	RC
1(48)	1.44582	3599.03	2781	160	4	5558	0.878738	765.275	3194	396	10	3734
2(185)	0.919243	35144.6	13229	372	56	16943	0.657629	28292.4	15184	417	157	16907
3(79)	1.16019	5448.57	4250	296	17	6531	0.769882	2426.37	5181	229	46	5854
4(98)	0.924781	4795.52	7941	450	91	11090	0.83294	2610.66	9401	477	119	11944
5(65)	1.28003	8046.96	4797	503	17	8040	0.87054	1330.77	6251	258	36	7629
6(108)	1.24406	13236.7	8039	419	10	12886	0.909754	4461.82	10125	502	55	12442
7(168)	1.05667	27892.5	10939	320	24	15354	0.833492	15262	12496	466	105	15262
8(177)	0.845301	25726.3	11885	439	118	15245	0.579132	26662.7	15029	339	155	16138
9(50)	0.968628	2757.35	4187	348	63	5769	0.992061	1151.46	4559	340	42	6144
10(107)	0.887287	7374.69	7984	305	75	10666	0.671652	3298.16	8969	264	105	10510
11(82)	1.24004	6335.33	4714	452	13	8081	0.941658	2418.91	5363	396	36	6893
12(53)	1.06777	5741.11	3349	285	17	4687	0.839232	849.09	3775	263	40	4475
13(141)	1.05053	25907.8	10308	503	65	14859	0.990518	11963.5	13487	629	66	17724
14(93)	0.916485	5628.65	8993	710	157	12214	0.888958	1727.91	9786	525	143	13442
15(91)	0.728584	8950.5	8577	342	129	10378	0.646842	2197.1	9448	269	130	11374
16(169)	1.10571	36900.1	10212	323	29	15542	0.633115	22403.2	12797	324	127	14020
17(63)	1.22358	10969.3	4536	214	3	7328	0.670644	1507.89	6106	229	74	6786
18(127)	1.35262	18604.4	8306	426	23	16345	0.713326	14693.4	11554	361	99	12828
19(212)	0.902345	49695.9	18297	577	89	23209	0.747271	42023.6	23488	606	233	28741
20(73)	1.17186	4599.37	5196	503	23	8063	1.03991	2044.98	6041	407	41	8441
21(180)	0.675185	30563.2	19056	708	323	23001	0.6767	13285.7	22563	785	313	26068
22(42)	1.29898	4090.65	1941	119	2	3280	1.14618	863.486	2581	212	13	3835
23(20)	1.17465	790.388	780	48	1	1182	1.02632	151.778	780	65	1	1025
Avg.	1.07132	-	7839.0	-	58.652	-	0.824195	-	9485.13	-	93.304	-

Table 22: results on CBMix instances

Providing some more informations regarding performances of the algorithms, the columns titled CPU(s)* and I_H in Table 22 display consumed computational times to obtain the whole optimal/near-optimal Pareto sets and their qualities in terms of I_H values. As stated earlier in the section (1.5), a reference point (Z_{ref}) is needed to determine I_H. In this work, the co-ordinate of Z_{ref} is set as $(1.5, 1.5)$, and values of the objectives in the Pareto set are normalized in the range [0-1] before computing I_H. To do the normalization, each objective value is divided by its maximum value present in the Pareto set. As it can be observed, the algorithm is more effective (with respect to I_H) when the DBLSP component is switched on. The Memetic NSGA-II gave an average I_H of 1.07132. Whereas, XNSGA-II reached at an average I_H value of 0.824195 without DBLSP. From these results, it is amply clear that DBLSP very well performed the task of local exploitation and increased the convergence ability of XNSGA-II. Comparing Pareto fronts exposed in the Figures 43 & 44, it becomes very clear that the algorithm is able to find a better approximate set covering a wide range of trade-off solutions when DBLSP is in action. Nevertheless, these enhancements were achieved through DBLSP at the expense of computational time. As for instance, on CBMix 19 (the largest instance in terms of the total required tasks), the algorithm consumed 7672.3 cpu seconds more to help generate the whole optimal/near-optimal Pareto set.

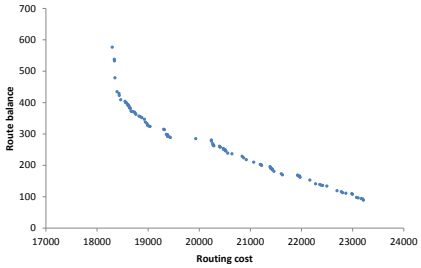

Figure 43: Pareto front of CBMix19 with DBLSP

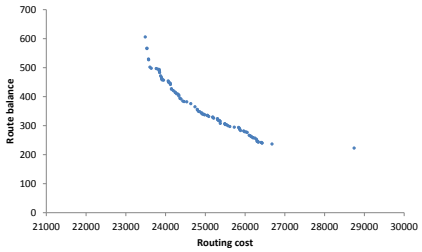

Figure 44: Pareto front of CBMix19 without DBLSP

The table 23 further reveals some additional characteristics of the discovered potentially optimal Pareto sets by the Memetic NSGA-II (termed as MNSGA-II) and its basic form XNSGA-II. It compares values of the RNI measure (the ratio between the number of obtained non-dominated solutions and the size of the population) and the average values of routing cost & route balance of Pareto sets generated by the algorithms. The Memetic NSGA-II yielded an average RNI value of 0.662, whereas XNSGA-II gave an average RNI value of only 0.563. It was found that the algorithms could obtain more number of non-dominated solutions on the complex instances comprising of large number of nodes, edges and arcs. As for instance, there are 11 nodes, 2 edges, 25 arcs and 20 required tasks in CBMix23. Despite the small size of the solution string, the algorithms could obtain RNI value of just 0.24. On the other hand, both algorithms produced RNI of 1.0 on CBMix11, which contains 69 nodes, 6 edges, 229 arcs and 82 required task. The reason being its highly dense network that provides more opportunity to create many alternative trade-off solutions. Furthermore, the average routing cost produced by Memetic NSGA-II is better than that obtained by XNSGA-II on all 23 instances. With respect to the average route balance, Memetic NSGA-II surpassed its competitor XNSGA-II 16 times. As this is the first bi-objective approach to the MCGRP, the *reference Pareto set* is hereby established for all CBMix instances. Researchers are welcomed to improve the reference set created in this research work.

4.3 Effect of CMP and X-set

In order to investigate the effects of CMP and X-set on the performance of Memetic NSGA-II, CBMix19 was selected again and the algorithm was run with and without the assistance of these ingredients. The CMP, which can also be seen as a *multi-mode mutation function*, contributed tremendously in maintaining diversity along the front as the

No.	RNI		Avg. RC		Avg. RB	
	MNSGA-II	XNSGA-II	MNSGA-II	XNSGA-II	MNSGA-II	XNSGA-II
1	0.35	0.33	3085.37	3475.21	48.4571	76.5152
2	0.75	0.91	14809.7	16030	159.853	230.879
3	0.67	0.63	5010.9	5470.25	99.7761	117.714
4	0.69	0.76	9318.52	10417.9	258.319	279.579
5	0.63	0.35	5523.7	6890.57	141.556	114.2
6	0.63	0.64	9286.81	11230.7	132.111	178.641
7	0.66	0.84	12379.0	13426.8	141.879	242.988
8	0.61	0.5	12967.0	15536.8	245.525	228.06
9	0.52	0.43	4721.5	5086.72	167.019	142.256
10	0.60	0.41	9060.77	9887.12	157.567	165.22
11	1.0	1.0	5577.82	5985.52	233.48	201.52
12	0.47	0.14	3861.74	4108.29	98.383	121.643
13	0.90	0.75	11957.6	15183.7	198.811	207
14	0.87	0.57	10182.9	10931.4	377.218	322.649
15	0.57	0.21	9200.23	10161.7	208.193	197.857
16	0.83	0.47	11979.1	13353.1	146.554	199.851
17	0.48	0.41	5421.21	6463.73	86.7292	145.927
18	0.64	0.41	10592.7	12195.1	152.016	172
19	0.95	1.0	20367.9	24989.5	268.295	362.96
20	1.0	0.73	5983.63	6767.71	209.08	194.712
21	0.72	1.0	20533.2	22563	472.667	479.78
22	0.45	0.24	2148.96	3013.96	51.7333	66.6667
23	0.24	0.24	878.083	884.208	19.5833	20.8333
Avg.	0.662	0.563	-	-	-	-

Table 23: Comparison table

algorithm was moving towards the true one. The CMP was so effective that there was not even one pair of similar solutions in the finally obtained Pareto sets for all instances. In order to instantiate this claim, the co-ordinates of all obtained non-dominated solutions of CBMix19 (the largest instance in terms of the number of required tasks) are reported in the Table 24. As it can be seen, there are no clones at all according to the adopted criteria (mentioned in the section 3.3.3).

The Pareto fronts portrayed in the Figures 49-71 further confirms the capability of the Memetic NSGA-II in generating fronts with a well-spread of trade-off solutions (non-duplicates). Only for inspecting the behaviour of the algorithm without CMP, it was permitted to navigate till Max_{gen} generations as earlier, but a different constraint has been imposed on its running time. It is stopped once RNI becomes equal to 1. The outputs have been provided in the Table 25. In the absence of CMP, the algorithm was found suffering from premature convergence due to the occurrence of genetic drift. It stopped in just 168.483 cpu seconds, giving RNI value of 1 unit. But, out of the 100 non-dominated solutions, there were only 12 solutions that occupied different positions

RC, RB	RC, RB
23209, 89	18676, 372
18297, 577	22414, 136
21399, 190	18651, 382
21393, 193	18634, 383
21420, 189	20837, 229
21383, 196	18735, 369
23198, 93	22983, 110
21431, 186	21627, 170
21427, 188	20410, 258
20395, 261	18576, 399
22356, 139	21971, 162
21066, 210	18710, 371
20404, 259	21931, 167
19932, 285	22276, 141
21604, 173	21192, 203
20508, 247	21962, 165
20505, 250	18751, 365
18987, 328	18609, 391
21461, 181	22695, 119
18924, 347	22803, 113
18978, 333	22867, 111
20264, 269	21215, 201
20270, 265	18759, 363
18945, 338	22997, 108
20470, 254	22778, 116
18336, 538	21219, 200
18560, 400	19303, 315
20472, 251	19415, 291
18430, 429	18435, 423
18848, 355	19379, 292
22163, 153	23170, 94
18821, 357	19378, 298
18341, 533	21961, 166
18665, 373	21923, 169
20285, 262	18626, 388
18604, 394	23075, 98
20520, 246	23108, 96
20276, 264	20869, 224
22384, 137	18547, 404
22496, 134	19319, 314
20638, 237	19354, 299
18351, 479	20238, 277
20556, 239	18391, 435
19038, 324	19432, 289
20920, 218	18463, 409
18606, 392	-
18876, 352	-
19013, 325	-
18621, 390	-
20236, 281	-

Table 24: Pareto set of CBMix19

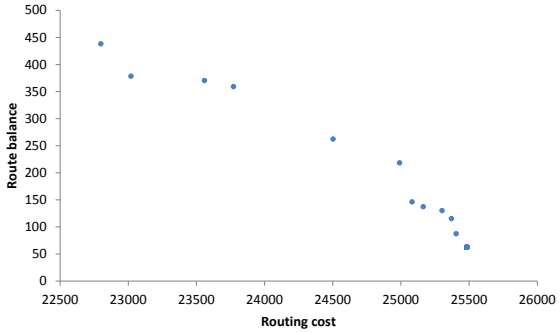

Figure 45: Pareto front of CBMix19 without CMP (S.C. $RNI = 1$)

in the objective space and rest of the 88 solutions were clones. These 12 solutions can easily be visualized in the Figure 45. On the other hand, the Memetic NSGA-II had yielded 95 different non-dominated solutions (see Table 23 and Figure 67) under the influence of the CMP component.

$S.C(RNI = 1)$	CPU(s)*	RC*	RB	RB*	RC	I_H	Avg. RC	Avg. RB	CMP
satisfied	168.483	22799	438	62	25484	0.761989	25370.5	81.58	Deactivated

Table 25: Effects of CMP on CBMix19

The table 26 summarizes the obtained results on CBMix19 by the Memetic NSGA-II while utilizing each crossover operator alone. The corresponding Pareto fronts have been shown in the Figures 46 – 48. The OX operator was found generating solutions with the low route balance objective. As presented in the Table 26, the algorithm could discover a solution giving route balance of only 34 units. Whereas, it produced the lowest route balance of 89 units (see Table 22) when the other two operators (PMX and ERX) were also used with the equal probability. In terms of the average value, Memetic NSGA-II with OX gave far better result (165.69 units of route balance) than with ERX alone and the complete algorithm. On the other hand, ERX intensively promoted the exploration in the cost-function-rich regions of the search space. Employing ERX operator, the algorithm yielded the average routing cost of 20256.7 units (better than the value outputted by Memetic NSGA-II with X-set), but consuming a large amount of computational time (124492 cpu seconds). Looking at the average route balance/routing cost values in Table 26, it can be said that the behaviour of PMX lies between OX & ERX and is mostly closer to the former operator. Together in the X-set, they drove search into different regions of the search space and helped find the diverse

sets of Pareto-optimal solutions on the CBMix dataset. Moreover, the Pareto front (Figure 67) created by Memetic NSGA-II with the X-set is better evenly spaced than using the operators alone.

I_H	CPU(s)*	RC*	RB	RB*	RC	RNI	Avg. RC	Avg. RB	Operator
1.3147	42170.3	18736	440	18	34013	0.98	22459.1	170.827	PMX
1.08052	39287.8	19957	353	34	29134	0.58	22561.4	165.69	OX
0.923709	124492	19661	556	51	24497	0.99	20526.7	345.121	ERX

Table 26: Effects of crossover operators on CBMix19

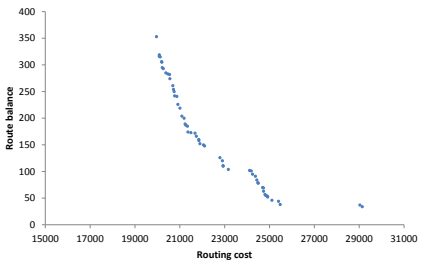

Figure 46: Pareto front of CBMix19 with OX

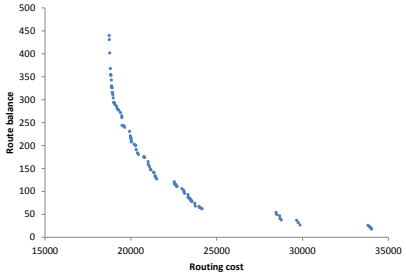

Figure 47: Pareto front of CBMix19 with PMX

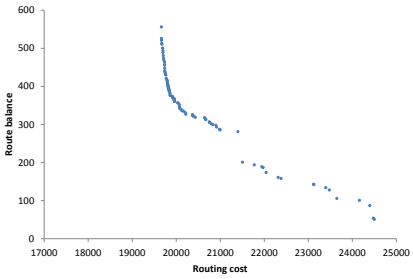

Figure 48: Pareto front of CBMix19 with ERX

4.4 Evolution of Pareto set

The table 27 shows properties of the Pareto sets generated by the Memetic NSGA-II for CBMix19 in every 5000 iterations. The changes in the RNI measure during the

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.09	0.612813	45594.1	1125.22	40892	1524	806	49781	0.281
5000	0.62	0.769446	21348	229.758	19967	385	92	23344	4463.15
10000	0.64	0.83031	21125.2	235.891	19365	483	92	23313	8188.88
15000	0.71	0.872656	21197.7	231.169	19226	615	92	23301	11498.4
20000	0.64	0.875866	21068.2	234.031	18980	589	92	23301	14542.7
25000	0.75	0.880093	20769.2	265.787	18932	595	92	23301	17549.3
30000	0.72	0.8267	21074.6	228.611	18662	420	91	23283	20338.8
35000	0.77	0.897336	20713.2	258.234	18359	577	91	23252	23371.8
40000	0.82	0.909097	20657.3	261.927	18329	615	91	23252	26795.6
45000	0.9	0.909576	20491.7	270.344	18329	615	91	23252	30189.2
50000	0.96	0.899756	20433.5	273.188	18325	577	91	23252	33876.2
55000	0.99	0.899895	20321.5	280.293	18325	577	91	23252	37507.3
60000	1	0.900328	20331.0	278.0	18300	577	89	23209	40965.7
65000	0.91	0.901493	20455.6	268.033	18297	577	89	23209	43925.1
70000	0.96	0.90202	20281.8	277.448	18297	577	89	23209	46887.5
75000	0.95	0.902345	20367.9	268.295	18297	577	89	23209	49784.8

Table 27: Pareto set evolution of CBMix19

search are neither increasing nor decreasing. The RNI value is 0.71 at the 15000^{th} iteration, but drops to 0.64 between 15001-20000 iterations. One more fall in RNI value can be observed in Table 27, that is, between 60001-65000 iterations. The algorithm finally stops at the 75000^{th} iteration, giving RNI value of 0.95 (obtained between 70001 - 75000 iterations). The final RNI value, however, is significantly better than that obtained till 40000 iterations. The hypervolume measure (I_H) improved significantly as Memetic NSGA-II kept exploring the search space. Starting with a I_H value of 0.612813, the algorithm finally achieved I_H value of 0.902345 between 70001-75000 iterations. It is worth noting that the values of RNI are equal at the 10000^{th} and 20000^{th} iterations, but I_H value is higher at the 20000^{th} iteration. Furthermore, the algorithm produced higher I_H value at the 65000^{th}, 70000^{th} and 75000^{th} iterations despite obtaining lesser number of non-dominated solutions than at the 60000^{th} iteration. These results clearly confirm the capability of the Memetic NSGA-II in finding well-spread Pareto sets.

The decreasing values of the lowest routing cost and the lowest route balance in the Table 27 (provided under the column titled RC* and RB*, respectively) prove that the algorithm moves towards the true Pareto set containing the optimum values of both objectives. It can be observed that the algorithm outputted the final hypervolume of 0.902345 between 70001-75000 iterations, consuming 46887.5-49784.8 cpu seconds. But, the best values of RC* (18297 units) and RB* (89 units) were obtained much earlier. As it can be seen, the best RC* was obtained between 60000-65000 iterations, utilizing 40965.7-43925.1 cpu seconds. Whereas, the solution giving the best RB* was encountered between 55000-60000 iterations.

No.	UB*	RC*		% gap	
		MNSGA-II	XNSGA-II	MNSGA-II	XNSGA-II
1	2589	2781	3194	6.90	18.94
2	12220	13229	15184	7.62	19.52
3	3643	4250	5181	14.28	29.68
4	7583	7941	9401	4.50	19.33
5	4531	4797	6251	5.54	27.51
6	7087	8039	10125	11.84	30.00
7	9607	10939	12496	12.17	23.11
8	10524	11885	15029	11.45	29.97
9	4038	4187	4559	3.55	11.42
10	7582	7984	8969	5.03	15.46
11	4494	4714	5363	4.66	16.20
12	3138	3349	3775	6.30	16.87
13	9110	10308	13487	11.62	32.45
14	8566	8993	9786	4.74	12.46
15	8280	8577	9448	3.46	12.36
16	8886	10212	12797	12.98	30.56
17	4037	4536	6106	11.00	33.88
18	7098	8306	11554	14.54	38.56
19	16347	18297	23488	10.65	30.40
20	4844	5196	6041	6.77	19.81
21	18069	19056	22563	5.17	19.91
22	1941	1941	2581	0.0	24.79
23	780	780	780	0.0	0.0
Avg.	-	-	-	7.59	22.31

Table 28: Percentage gap

The average values of routing cost (Avg. RC) and route balance (Avg. RB) also do not change in any particular fashion as they depend on the number of non-dominated solutions in the obtained Pareto set. However, from the 15000^{th} iteration onwards, it appears that the improvement in one has been gained by deteriorating the other. For Pareto set evolution properties of other instances, see Tables 37-58 (Appendix).

4.5 Deviation from the best known upper bound

In the Table 28, the column named UB* gives the best known upper bound values of the cost objective of CBMix instances. The column labeled RC* shows the lowest values of the routing cost found by Memetic NSGA-II and XNSGA-II. The % gap of RC* from

UB* has been calculated by (equation 4.5.1) and provided in the last column.

$$\%gap = \frac{RC* - UB*}{RC*} \times 100 \tag{4.5.1}$$

As it can be seen in the Table 28, on each instance of the CBMix dataset, the percentage gap given by the Memetic NSGA-II is significantly better than that produced by the XNSGA-II. The XNSGA-II retrieved the best known upper bound value of the cost objective on only one instance (CBMix23) and gave a percentage deviation of 22.31. On the other hand, the Memetic NSGA-II reached at UB* on two instances (CBMix22 and CBMix23), producing a comparatively very small percentage deviation of just 7.59. On 7 (out of 23) instances, Memetic NSGA-II produced the gap between RC* and UB* of less than 5%. Despite the search has been directed to find compromised solutions, Memetic NSGA-II generated Pareto sets constituting of good solutions with regard to the routing cost.

4.6 Pareto fronts

The Pareto fronts created by the Memetic NSGA-II for CBMix instances have been displayed in the Figures 49 – 71. Upon closely visualizing the curves, it is found that the proposed algorithm has created Pareto fronts with a good spread of *non-duplicate* trade-off solutions. The Pareto fronts provide a wide range of non-dominated solutions. As for instance, ranges of the Pareto curve along the cost and balance objectives on the CBMix19 are 4912 and 488 units, respectively. For CBMix3 (Figure 51), CBMix10 (Figure 58), CBMix20 (Figure 68) and CBMix21 (Figure 69), most parts of the Pareto fronts look convex. On CBMix23, the shape of the Pareto curve (Figure 71) is mostly linear. The Pareto front of CBMix15 (Figure 63) *clearly* shows that it is a mixture of concave and convex shapes. In fact, none of the curves is completely convex or concave. Hence, the use of a Pareto-dominance based multi-objective evolutionary algorithm (such as the proposed Memetic NSGA-II) seems to be the most suitable approach to solve the present bi-objective MCGRP.

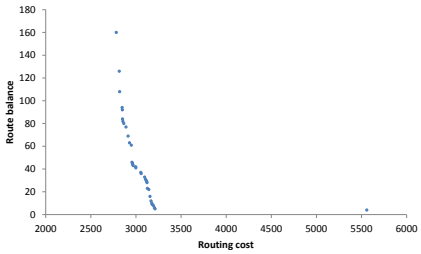

Figure 49: Pareto front of CBMix1

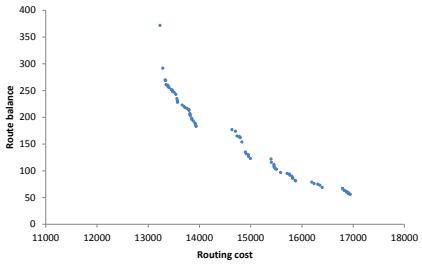

Figure 50: Pareto front of CBMix2

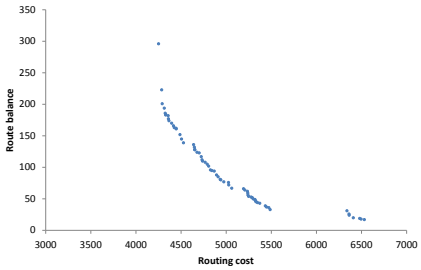

Figure 51: Pareto front of CBMix3

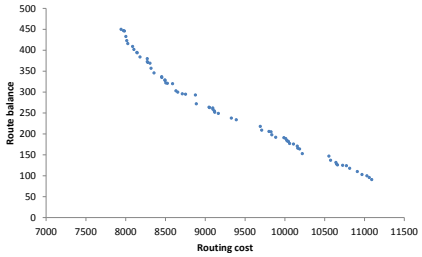

Figure 52: Pareto front of CBMix4

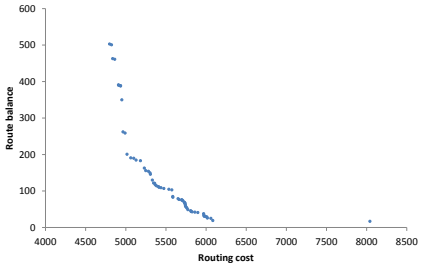

Figure 53: Pareto front of CBMix5

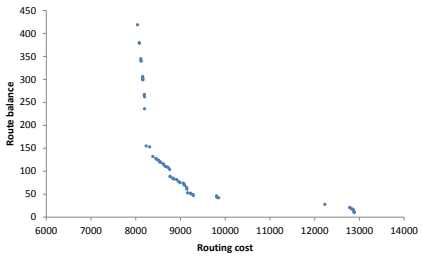

Figure 54: Pareto front of CBMix6

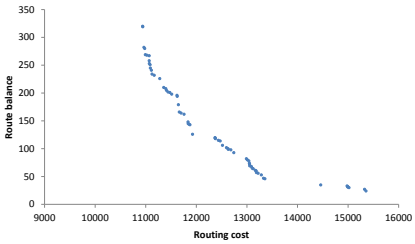

Figure 55: Pareto front of CBMix7

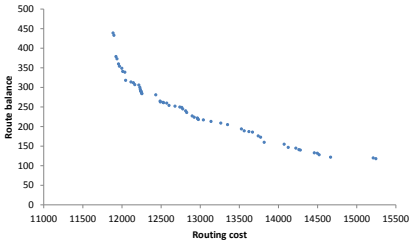

Figure 56: Pareto front of CBMix8

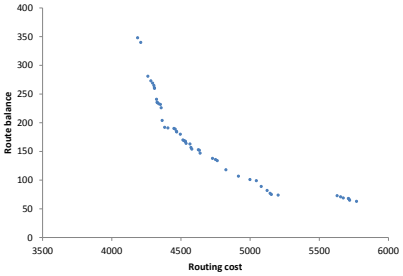

Figure 57: Pareto front of CBMix9

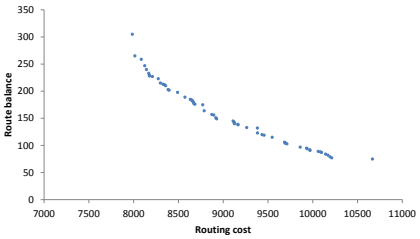

Figure 58: Pareto front of CBMix10

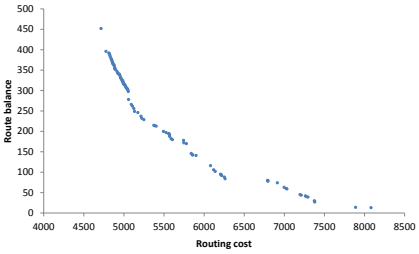

Figure 59: Pareto front of CBMix11

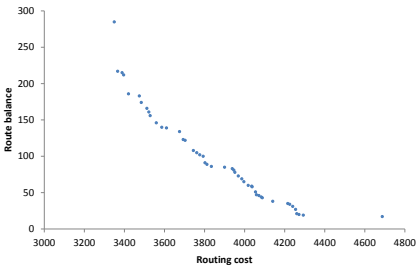

Figure 60: Pareto front of CBMix12

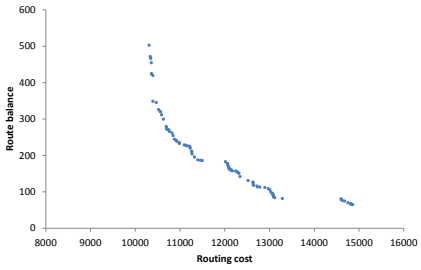

Figure 61: Pareto front of CBMix13

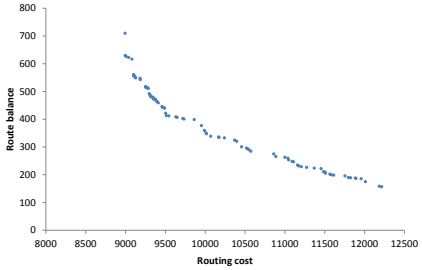

Figure 62: Pareto front of CBMix14

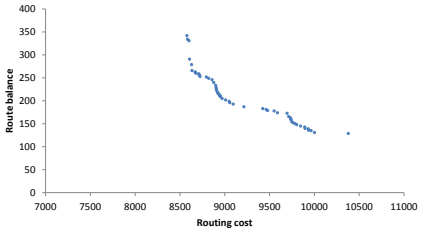

Figure 63: Pareto front of CBMix15

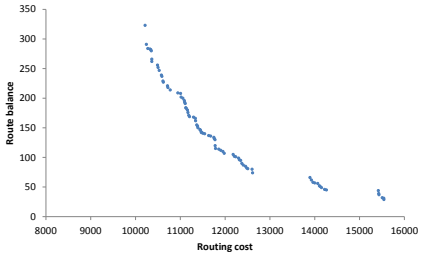

Figure 64: Pareto front of CBMix16

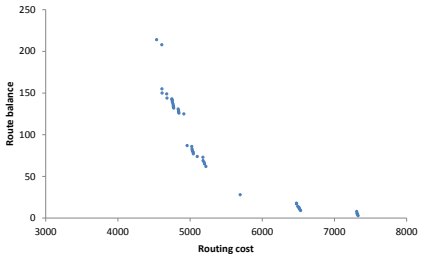

Figure 65: Pareto front of CBMix17

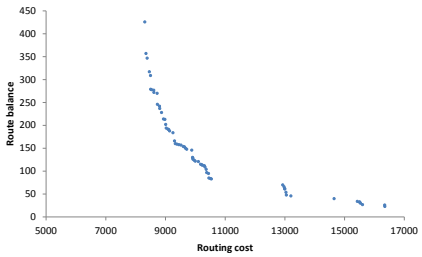

Figure 66: Pareto front of CBMix18

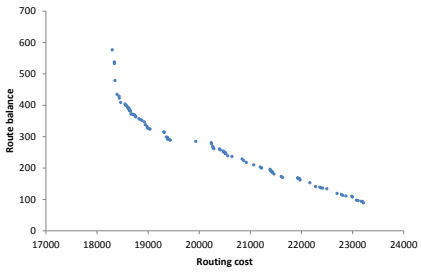

Figure 67: Pareto front of CBMix19

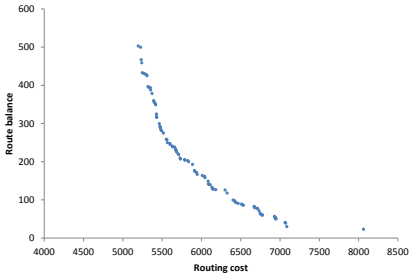

Figure 68: Pareto front of CBMix20

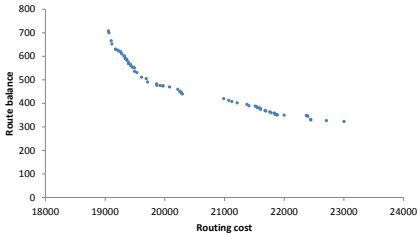

Figure 69: Pareto front of CBMix21

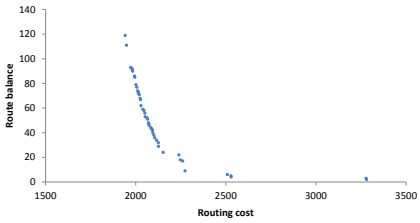

Figure 70: Pareto front of CBMix22

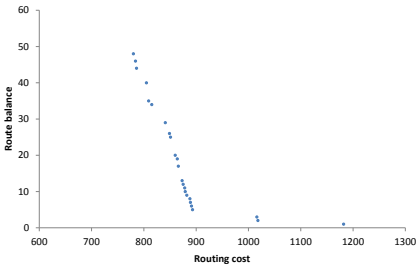

Figure 71: Pareto front of CBMix23

Conclusions and future works

A bi-objective model of the Mixed Capacitated General Routing Problem (MCGRP) has been tackled in this thesis so that the emerging practical needs of several private and public firms involved in the goods distribution (or similar) business could be fulfilled. To the best of our knowledge, this work made the first successful attempt to optimize two major objectives at the same time for the MCGRP, while satisfying the capacity constraint. One of them was the minimization of the overall routing cost and other was the minimization of the route balance (difference between the largest tour and the smallest tour in terms of the routing cost). This NP-hard problem was mathematically modelled and an architecture of a hybrid MOEA, named Memetic NSGA-II, has been designed to generate the optimal/near-optimal Pareto front with a good spread of non-dominated solutions in a single simulation run.

The proposed algorithm utilizes the framework of notable NSGA-II, which has been integrated with a dominance based local search procedure (DBLSP), a clone management principle (CMP) and a set of three crossover operators (X-set). The DBLSP employs inter-route version of three well-known tour improvement heuristics: *2-opt, Re-insertion and* λ*-interchange*, to improve the convergence ability of NSGA-II towards the true Pareto front. While, the aim of incorporating CMP was to prevent the occurrence of genetic drift that leads to premature convergence at a locally optimum Pareto front. The CMP was equipped with *swap* and *inversion* operators for the replacement of clones by their mutated structures. The rational of using three crossover operators (with equal selection probabilities) was to direct the search into different attractive regions of the solution space.

The whole concept was tested on the twenty three MCGRP benchmark instances with and without the support of these additional components. The obtained results demonstrated the effectiveness of the whole algorithmic idea and also justified the compounding of DBLSP, CMP and X-set with NSGA-II to create its Memetic version. The Memetic NSGA-II, in fact, overshadowed the performance of the basic version (NSGA-II with only X-set and CMP) with regard to the average hypervolume measure, RNI metric as well as solution quality. Furthermore, this thesis work also provides a lot of

scope for future research. It can be extended in many interesting ways on the problem and algorithm levels:

- The problem can be solved in more than two-dimension by including other important objectives, such as minimization of the number of vehicles and maximization of the route compactness.
- It will be challenging to solve the present bi-objective MCGRP incorporating some more real-world side constraints.
- Other variants of the basic MCGRP, such as MCGRP with time time windows (MCGRPTW) and multi-depot MCGRP (MDMCGRP), are needed to formulate and address.
- The Memetic NSGA-II can further be enhanced or a new multi-objective optimization algorithm can be proposed to improve the first reference set established in this work.
- A comparison between the multi-objective versions of other well-known metaheuristics in the VRP area (such as tabu search, simulated annealing, ant colony optimization and particle swarm optimization) on the present bi-objective MCGRP will be an useful work.

Finally, it is hoped that the proposed Memetic NSGA-II will also prove itself as a promising multi-objective optimization tool for various complex models of routing problems.

* * * * *

Appendix

Input: $s^{(o)}$ - the initial solution;
Output: s^* - the best found solution;
Determine the initial temperature T;
set: $s = s^{(o)}$, $s^* = s$ and $t = T$;
Repeat
  Repeat Generate a new solution s' in the neighborhood of s;
    Calculate $\triangle f = f(s') - f(s)$;
    Generate a random number $r \in [0, 1]$;
    if $\triangle f \ < \ 0$
      set $s = s'$;
    else
      if $r \ < \ \exp(- \triangle f/t)$;
        set $s = s'$;
      end if;
    end if;
    if $s \ < \ s*$
      set $s^* = s$;
    end if;
  Until (Equilibrium condition)
  Update temperature: $t = t \times \alpha \quad ; \qquad \alpha \in [0, 1]$;
Until (stopping condition is satisfied);

Table 29: Pseudocode of Simulated Annealing

Input: $s^{(o)}$ - the initial solution;
Output: s^* - the best found solution;
Initialize the Tabu List T;
set: Aspiration criteria;
set: $s = s^{(o)}$ and $s^* = s$;
Repeat
    Generate solutions in the neighborhood of s;
    Select the best possible solution $s' \notin T$ or satisfying the aspiration criteria;
    set $s = s'$;
    Insert the solution s (or its attribute) into the tabu list T;
    if $f(s) < f(s^*)$
      set $s^* = s$;
    end if;
    Update the tabu list T;
Until (stopping condition is satisfied);

Table 30: Pseudocode of Tabu Search

Input: $s^{(o)}$ - the initial solution;
$s =$ local search ($s^{(o)}$);
Repeat
    $s' =$ Mutate (s);
    $s'' =$ Local Search (s');
    if $f(s'') < f(s)$
      set $s = s''$;
    end if ;
Until (stopping condition is satisfied) ;
Return s;

Table 31: Pseudocode of Iterated Local Search

Input: Neighborhood structures N_l $\quad l = 1, 2, 3, \ldots, l_{max}$;
Generate the initial solution $s^{(o)}$;
set: $s = s^{(o)}$;
$l = 1$;
while ($l \leq l_{max}$)
    Find the best neighbor $s' \in N_l(s)$;
    if $f(s') < f(s)$
      set $s = s'$; $l = 1$;
    else
      $l = l + 1$;
    end if;
end while;
Return s;

Table 32: Pseudocode of Variable Neighborhood Decent

Input: Neighborhood structures N_l $\quad l = 1, 2, 3, \ldots, l_{max}$;
Generate the initial solution $s^{(o)}$;
set: $s = s^{(o)}$;
$l = 1$;
while ($l \leq l_{max}$)
    $s' \leftarrow$ shake (s) ; $s' \in N_l(s)$;
    $s'' \leftarrow$ Local Search(s') ;
    if $f(s'') < f(s)$
      set $s = s''$; $l = 1$;
    else
      $l = l + 1$;
    end if;
end while;
Return s;

Table 33: Pseudocode of Variable Neighborhood Search

$s \leftarrow \phi$;
Repeat
    $s \leftarrow$ Greedy Randomized Construction;
    $s' \leftarrow$ Local Search (s);
    Update s according to the acceptance criteria;
Until (maximum number of iterations);
Return best solution;
End;

Table 34: Pseudocode of GRASP metaheuristic

$s \leftarrow \phi$;
Evaluate the incremental cost of the candidate elements;
Repeat
    Build the restricted candidate list (RCL), a set of best elements;
    Select $s' \in RCL$ randomly;
    $s \cup s'$;
    Re-evaluate the incremental costs;
Until (s is not complete);
Return s;
End;

Table 35: Greedy Randomized Construction of GRASP

Input: A weight matrix (W) of size $n \times n$, where n is the number of nodes;
    E: set of paths;
set: $w_{ij} = 0 \; if \; i = j, \; w_{ij} = w(i,j) \; if \; (i,j) \in E, \; w_{ij} = \inf \; if (i,j) \notin E$;
    w_{ij}: weight/cost of path $i \rightarrow j$
set: $D^0 \leftarrow W$, where D is a matrix of size $n \times n$;
For ($k = 1 \ldots n$)
  For ($i = 1 \ldots n$)
    For ($j = 1 \ldots n$)
      $d_{ij}^k = min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$;
    end For;
  end For;
end For;
Return D^n (matrix containing the shortest paths between all pairs of nodes);

Table 36: Pseudocode of Floyd-Warshall Algorithm

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.05	0.768427	5670.8	351.2	5411	680	144	6074	0.047
5000	0.31	1.389	3169.13	37.8065	2857	84	4	5558	557.349
10000	0.32	1.42177	3145.88	39	2821	120	4	5558	1090.56
15000	0.32	1.44381	3139.72	41.3438	2781	160	4	5558	1626.82
20000	0.32	1.44446	3111.59	46.4063	2781	160	4	5558	2128.9
25000	0.35	1.44479	3088.86	48.9143	2781	160	4	5558	2638.97
30000	0.33	1.44493	3096.09	48.8485	2781	160	4	5558	3141.88
35000	0.35	1.44582	3085.37	48.4571	2781	160	4	5558	3623.67

Table 37: Pareto set evolution of CBMix1

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.02	0.358091	26846.5	721.5	26420	795	648	27273	0.218
5000	0.36	0.8052	15656.7	132.944	14384	288	61	17050	3091.53
10000	0.4	0.80113	15538.5	130.75	13908	256	61	17025	5526.2
15000	0.45	0.85432	15206	152.222	13608	308	57	16948	7899.27
20000	0.52	0.845071	15023.3	160.096	13544	285	57	16942	10309.8
25000	0.52	0.844326	15028.7	157.077	13501	280	57	16942	12652.9
30000	0.5	0.910406	15010.6	161.58	13432	395	57	16942	14987.2
35000	0.53	0.914301	15093.9	155.604	13357	395	57	16942	17292.6
40000	0.56	0.890764	15094.7	154.875	13335	340	57	16930	19629
45000	0.5	0.893951	15009.4	153.54	13335	340	57	16911	22299.2
50000	0.68	0.913035	14875.1	158.397	13245	372	57	16911	25037.6
55000	0.67	0.913977	14913.7	154.239	13245	372	57	16911	27687.6
60000	0.71	0.91846	14922.1	151.183	13229	372	56	16944	30194.2
65000	0.76	0.918867	14802.9	159.132	13229	372	56	16943	32730.7
70000	0.75	0.919243	14809.7	159.853	13229	372	56	16943	35220

Table 38: Pareto set evolution of CBMix2

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.05	0.484804	9041.8	357.6	8458	439	285	9407	0.062
5000	0.51	1.07905	5076.12	96.0392	4559	230	18	6519	799.123
10000	0.6	1.09516	5120.17	96.0833	4553	262	18	6507	1712.6
15000	0.59	1.08325	5162.49	90.7458	4501	224	18	6507	2418.63
20000	0.56	1.09327	5176.41	83.7321	4469	216	17	6532	3028.44
25000	0.61	1.09927	5106.98	91.1148	4303	201	17	6532	3509.09
30000	0.61	1.1008	5148.54	86.3279	4297	201	17	6532	3983.19
35000	0.66	1.15827	5040.62	102.121	4250	296	17	6532	4482.17
40000	0.7	1.15864	5050.31	99.6	4250	296	17	6531	4962.48
45000	0.67	1.16019	5010.9	99.7761	4250	296	17	6531	5449.18

Table 39: Pareto set evolution of CBMix3

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.03	0.443433	14065.3	555	13398	666	459	14455	0.094
5000	0.3	0.820832	9686.6	231.467	8483	360	97	11087	508.164
10000	0.4	0.82536	9626	226.8	8475	360	96	11077	1000.05
15000	0.45	0.825885	9584	230.444	8460	359	96	11070	1478.2
20000	0.49	0.829977	9700.78	219.143	8460	359	95	11091	1917.1
25000	0.54	0.849865	9588.87	229.241	8413	381	95	11091	2391.3
30000	0.54	0.851437	9582.89	228.37	8399	381	95	11091	2860.46
35000	0.52	0.910697	9591.87	229.288	8145	449	91	11093	3311.65
40000	0.61	0.918973	9460.36	245.148	8020	450	91	11093	3778.69
45000	0.64	0.919605	9358.94	254.703	8020	450	91	11090	4295.35
50000	0.69	0.924781	9318.52	258.319	7941	450	91	11090	4798.32

Table 40: Pareto set evolution of CBMix4

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.06	0.718843	11393.5	442.667	10289	838	287	12301	0.062
5000	0.47	1.26149	5623.62	150.723	4898	503	17	8044	1060.36
10000	0.49	1.26258	5659.63	150.163	4898	503	17	8044	2015.05
15000	0.55	1.26453	5617.44	151.145	4881	503	17	8044	2953.93
20000	0.53	1.2642	5622.68	152.509	4881	503	17	8040	3890.66
25000	0.56	1.26442	5664.55	147.214	4881	503	17	8040	4812.86
30000	0.56	1.2793	5522.73	140.714	4797	503	17	8040	5721.47
35000	0.58	1.27996	5517.21	143	4797	503	17	8040	6656.72
40000	0.6	1.28001	5507.67	146.117	4797	503	17	8040	7597.13
45000	0.63	1.28003	5523.7	141.556	4797	503	17	8040	8530.36

Table 41: Pareto set evolution of CBMix5

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.05	0.57319	18545.6	952.2	18098	1242	574	19428	0.11
5000	0.34	1.11243	10320.1	74.3824	8634	155	24	14015	1135.4
10000	0.62	1.26473	9476.47	145.129	8308	386	22	13997	2332.87
15000	0.38	1.21059	9446	120.211	8254	386	12	12888	3812.03
20000	0.57	1.22796	9184.37	151.982	8131	425	12	12888	4930.58
25000	0.69	1.23397	9267.33	140.812	8097	425	12	12878	6314.82
30000	0.74	1.23573	9266.55	143.541	8097	425	11	12882	7641.13
35000	0.79	1.23631	9313.61	133.215	8097	425	11	12882	8849.13
40000	0.81	1.23639	9240.85	137.679	8097	425	11	12882	9911.59
45000	0.85	1.23699	9164.95	153.988	8097	425	11	12881	10996.6
50000	0.83	1.23752	9190.23	164.349	8097	425	11	12881	12152
55000	0.63	1.24406	9286.81	132.111	8039	419	10	12886	13238.6

Table 42: Pareto set evolution of CBMix6

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.04	0.52181	25994.3	848.5	24053	1113	667	27332	0.187
5000	0.31	0.990521	12539.7	154.355	11217	275	33	15477	2854.24
10000	0.44	1.0549	12514	160.295	11118	387	27	15443	5413.94
15000	0.5	1.06127	12407.3	150.76	11091	387	27	15419	7966.93
20000	0.5	1.06428	12517.6	142.58	11091	387	27	15414	9975.19
25000	0.57	1.0646	12582.1	136.456	11091	387	27	15395	11884.4
30000	0.59	1.02694	12714.9	127.492	11005	276	27	15395	13848.5
35000	0.64	1.02861	12599	133.891	10992	276	27	15395	15978.8
40000	0.7	1.03063	12567.8	133.986	10992	276	26	15393	18084.7
45000	0.68	1.03212	12429.5	137.824	10992	276	26	15393	20095
50000	0.62	1.02845	12572.9	128.435	10992	269	26	15371	22029.4
55000	0.6	1.03119	12516.8	128.717	10992	269	24	15354	23989.7
60000	0.6	1.05642	12487.2	132.3	10939	320	24	15354	26025
65000	0.66	1.05667	12379	141.879	10939	320	24	15354	27994.5

Table 43: Pareto set evolution of CBMix7

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.09	0.520475	28167.8	816.222	25940	1010	656	30272	0.203
5000	0.42	0.732085	14113.3	208.238	12596	346	119	15379	2196.41
10000	0.33	0.745519	13961.4	197.273	12306	334	119	15357	4351.99
15000	0.37	0.77374	13651.5	216.324	12173	359	119	15350	6322.38
20000	0.42	0.767401	13543.3	219.929	12171	350	119	15330	8208.34
25000	0.41	0.769361	13535.4	217.634	12171	350	119	15330	9911.84
30000	0.43	0.770573	13417.6	224.279	12171	350	119	15330	11476.8
35000	0.51	0.782805	13320.6	227.784	12098	359	119	15330	13148.5
40000	0.55	0.808536	13315.7	226.982	12062	392	119	15330	14769.9
45000	0.56	0.808691	13285.3	228.482	12062	392	119	15330	16421.8
50000	0.55	0.808311	13262.3	232.545	12024	389	119	15330	18012.3
55000	0.6	0.857797	13104.9	247.067	11920	468	119	15330	19682.9
60000	0.55	0.847807	13313.3	224.945	11920	439	118	15357	21453.4
65000	0.59	0.849098	13122	239.78	11917	439	118	15357	23462.3
70000	0.61	0.845301	12967	245.525	11885	439	118	15245	25795.2

Table 44: Pareto set evolution of CBMix8

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.05	0.521478	7027.8	305.6	6762	410	221	7217	0.047
5000	0.36	0.939588	4761.61	161.528	4243	348	68	5718	382.608
10000	0.42	0.966291	4755.33	164.714	4187	348	65	5792	789.116
15000	0.46	0.971598	4813.46	154.609	4187	348	64	5806	1355.71
20000	0.44	0.967267	4736.82	164.568	4187	348	63	5769	1633.18
25000	0.49	0.967659	4740.06	164.939	4187	348	63	5769	1913.02
30000	0.5	0.967951	4708.14	169.82	4187	348	63	5769	2192.87
35000	0.5	0.968349	4729.86	167.32	4187	348	63	5769	2504.78
40000	0.52	0.968628	4721.5	167.019	4187	348	63	5769	2780.72

Table 45: Pareto set evolution of CBMix9

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.05	0.419876	13827.8	381.6	13223	417	311	14344	0.125
5000	0.23	0.817029	9315.57	150.435	8383	259	75	10708	737.252
10000	0.3	0.789264	9198.3	148.2	8259	227	75	10666	1434.16
15000	0.32	0.791256	9205.94	146.156	8259	227	75	10666	2120.94
20000	0.37	0.80642	9200.73	147.27	8245	238	75	10666	2775.43
25000	0.5	0.876721	9076.8	160.04	8131	305	75	10666	3551.83
30000	0.5	0.88608	9078.36	158.42	7984	305	75	10666	4225.11
35000	0.53	0.886708	9002.04	163.585	7984	305	75	10666	4864.33
40000	0.56	0.886882	9014.64	162.125	7984	305	75	10666	5479.43
45000	0.6	0.886928	9034.88	160.3	7984	305	75	10666	6139.69
50000	0.58	0.887193	9044.16	159.052	7984	305	75	10666	6764.64
55000	0.6	0.887287	9060.77	157.567	7984	305	75	10666	7403.58

Table 46: Pareto set evolution of CBMix10

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.07	0.707301	11030.3	447	9695	715	315	12364	0.094
5000	0.36	1.14979	6386.97	137.889	5224	348	13	8081	373.443
10000	0.46	1.15604	6079.11	159.457	5118	322	13	8081	765.192
15000	0.63	1.21195	5858.21	196.968	4907	425	13	8081	1261.74
20000	1	1.22511	5542.13	248.42	4832	448	13	8081	1990.92
25000	1	1.22567	5499.75	253.53	4828	448	13	8081	2813.13
30000	1	1.23876	5500.03	259.5	4823	496	13	8081	3625.97
35000	1	1.24055	5514.09	257.49	4800	496	13	8081	4422.27
40000	0.72	1.2446	5732.11	221.583	4764	496	13	8081	5201.73
45000	0.99	1.24952	5568.28	238.162	4734	496	13	8081	5815.36
50000	1	1.24004	5577.82	233.48	4714	452	13	8081	6353.5

Table 47: Pareto set evolution of CBMix11

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.08	0.694694	8137.25	448.375	7317	684	254	8832	0.062
5000	0.35	1.00679	4055.31	86.9714	3501	229	18	4723	714.662
10000	0.32	1.00964	3947.69	95.5625	3485	218	17	4706	1438.3
15000	0.36	1.03537	3984.08	87.75	3415	241	17	4697	2155.34
20000	0.36	1.03312	3896	94.9722	3415	230	17	4697	2880.82
25000	0.45	1.04348	3888.89	95.6667	3405	251	17	4687	3607.26
30000	0.47	1.06429	3845.17	105	3349	285	17	4687	4337.42
35000	0.47	1.06678	3846.51	103.277	3349	285	17	4687	5063.69
40000	0.47	1.06777	3861.74	98.383	3349	285	17	4687	5775.94

Table 48: Pareto set evolution of CBMix12

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.04	0.622683	28626.3	1169.5	25772	1598	767	31092	0.203
5000	0.47	0.974713	12451.6	165.83	10871	383	65	14979	2496.75
10000	0.6	1.02492	12213	181.117	10702	495	65	14932	4461.91
15000	0.68	1.00704	12044.8	191.382	10553	420	65	14932	6549.81
20000	0.86	1.04257	11778.8	225.407	10433	504	65	14913	8856.66
25000	0.84	1.04217	11769.5	224.774	10433	504	65	14889	11288.3
30000	0.77	1.04326	11843.6	217.675	10425	504	65	14889	13786.7
35000	0.77	1.04354	11849.9	210.065	10424	504	65	14887	16087.7
40000	0.83	1.04668	11884	208.988	10385	504	65	14887	18132
45000	0.79	1.04795	11968.4	202.025	10344	504	65	14861	20114.1
50000	0.85	1.04911	11907.4	205.941	10308	503	65	14859	22152.4
55000	0.95	1.04976	11768.6	223.021	10308	503	65	14859	24212.3
60000	0.9	1.05053	11957.6	198.811	10308	503	65	14859	26259.4

Table 49: Pareto set evolution of CBMix13

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.06	0.654833	17773.5	745.333	15438	1027	575	20090	0.047
5000	0.49	0.880232	10211.4	388.755	9004	630	159	12240	697.21
10000	0.63	0.882563	10128	396.27	9004	630	159	12238	1365.71
150000	0.73	0.913456	10100.4	397.342	9002	710	159	12238	1967.93
20000	0.79	0.914144	10187.9	385.696	9002	710	159	12238	2506.12
250000	0.78	0.912192	10348	360.167	9002	710	159	12210	3024.27
30000	0.76	0.91114	10236.8	372.079	8993	710	159	12184	3553.41
35000	0.8	0.911709	10262	369.488	8993	710	159	12184	4090.55
40000	0.77	0.912014	10272.4	368.481	8993	710	159	12184	4642.78
45000	0.81	0.922511	10245.3	371.938	8993	710	157	12290	5153.79
50000	0.87	0.916485	10182.9	377.218	8993	710	157	12214	5643.11

Table 50: Pareto set evolution of CBMix14

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.06	0.531115	15897.7	500	13737	643	426	16908	0.047
5000	0.34	0.619199	9523.65	180	8824	254	130	10448	728.316
10000	0.38	0.637061	9414.89	187.184	8728	259	129	10460	1549.68
15000	0.65	0.728789	9188.86	213.323	8613	342	129	10422	2484.3
20000	0.57	0.729912	9270.84	202.175	8577	342	129	10403	3449.04
25000	0.57	0.72854	9260.54	204.789	8577	342	129	10386	4434.04
30000	0.56	0.728795	9253.95	204.411	8577	342	129	10386	5376.52
35000	0.56	0.728165	9249.88	204.607	8577	342	129	10378	6275.11
40000	0.58	0.728356	9250.79	204.362	8577	342	129	10378	7149.17
450000	0.54	0.72849	9247.2	203.759	8577	342	129	10378	8084.09
50000	0.57	0.728584	9200.23	208.193	8577	342	129	10378	9086.61

Table 51: Pareto set evolution of CBMix15

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.04	0.499946	29778	957.25	27054	1130	734	31518	0.187
5000	0.39	0.975561	12752.8	128.359	11256	255	40	15637	3012.08
10000	0.4	0.966633	12661.3	121.4	11005	215	39	15603	6872.75
15000	0.47	0.984877	12560.8	123.021	10948	215	33	15592	9770.41
20000	0.41	0.997752	12755.9	116.732	10920	215	29	15591	12359.4
25000	0.43	1.00172	12572.1	122.233	10910	215	29	15566	14868.5
30000	0.51	1.08839	11916.2	155.529	10401	317	29	15550	18412.5
350000	0.65	1.09328	11867.6	157.431	10336	316	29	15542	21454.7
40000	0.73	1.08679	11872.9	156.493	10312	299	29	15542	24358.6
45000	0.76	1.08907	11793.8	157.408	10275	299	29	15542	27171.6
50000	0.77	1.08952	11899.1	151.922	10275	299	29	15542	29813.8
55000	0.82	1.0993	12000.3	144.768	10271	316	29	15542	32323.3
60000	0.87	1.10489	12017.9	143.437	10219	323	29	15542	34737.9
65000	0.83	1.10571	11979.1	146.554	10212	323	29	15542	37099.4

Table 52: Pareto set evolution of CBMix16

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.05	0.494823	11302.8	427.6	10132	537	371	11948	0.062
5000	0.2	1.2062	6069.35	77.4	4553	273	5	7340	2039.16
10000	0.34	1.21237	5607.79	93.7059	4547	273	5	7334	3050.59
15000	0.4	1.21568	5480.7	98.825	4544	273	5	7333	4208.58
20000	0.46	1.21752	5651.07	85.913	4544	273	5	7333	4979.17
25000	0.49	1.2217	5432.39	85.3061	4536	214	3	7332	7550.99
30000	0.47	1.2217	5424.4	87.3617	4536	214	3	7328	8759.31
35000	0.5	1.22257	5410.04	89.02	4536	214	3	7328	9760.74
40000	0.48	1.22358	5421.21	86.7292	4536	214	3	7328	11026.3

Table 53: Pareto set evolution of CBMix17

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.05	0.620606	25402	923.2	24099	1321	594	27429	0.094
5000	0.34	1.2512	12220.3	115.324	9332	284	27	17282	1631.8
10000	0.4	1.23099	11747.9	129.5	9330	250	27	17282	3300.17
15000	0.34	1.33913	10696.5	160.412	8682	501	24	16375	4990.92
20000	0.41	1.34042	11416.9	135.537	8536	466	23	16371	6425.76
25000	0.42	1.3447	11077.3	142.667	8536	466	23	16371	7784.74
30000	0.53	1.34227	10710.5	155.736	8422	426	23	16353	9341.68
35000	0.46	1.34442	10420.4	155.935	8422	426	23	16353	11011.9
40000	0.47	1.34737	10890.5	145.34	8380	426	23	16345	12584.5
450000	0.55	1.34816	10451.1	158.509	8380	426	23	16345	14078.1
50000	0.6	1.34855	10549.5	158.95	8380	426	23	16345	15654.5
55000	0.63	1.34878	10476.2	161.746	8380	426	23	16345	17196.3
60000	0.64	1.35262	10592.7	152.016	8306	426	23	16345	18658.6

Table 54: Pareto set evolution of CBMix18

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.05	0.611709	10666.8	584.4	10484	797	262	10896	0.078
5000	0.49	1.07796	6290.69	162.898	5593	332	27	8092	431.692
10000	0.53	1.14149	6176.32	191.774	5403	490	27	8089	927.858
15000	0.63	1.14659	6162.21	174.27	5260	446	27	8083	1401.64
20000	0.76	1.16734	6080.28	188.671	5196	503	27	8083	1968.05
25000	0.92	1.16804	6026.83	200.185	5196	503	27	8083	2540.44
30000	0.92	1.16853	6038.3	196.859	5196	503	27	8083	3086.4
35000	0.99	1.17044	6006.72	208.364	5196	503	23	8063	3605.8
40000	0.96	1.17138	6014.92	204.531	5196	503	23	8063	4125.43
1	1.17186	5983.63	209.08	5196	503	23	8063	4632.35	

Table 55: Pareto set evolution of CBMix20

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.05	0.549553	39183.6	889.4	36379	1231	712	41307	0.234
5000	0.26	0.60709	21176.7	455.154	19825	633	323	23033	2032.83
10000	0.27	0.586196	21321.1	422.704	19688	586	323	23028	4274.03
15000	0.33	0.614504	21122.4	434.545	19586	626	323	23028	6425.82
20000	0.4	0.665649	20795.5	470.025	19390	714	323	23021	8764.43
25000	0.55	0.686383	20574.7	480.945	19158	746	323	23002	11288.9
30000	0.71	0.687674	20387.1	497.789	19141	746	323	23002	13790.8
35000	0.75	0.674514	20451.1	487.373	19095	712	323	23002	16023.7
40000	0.77	0.672739	20476.1	484.013	19095	708	323	23002	18242
45000	0.71	0.67298	20539.6	468.887	19095	708	323	23002	20427.7
50000	0.72	0.674018	20581.6	467.319	19070	708	323	23001	22637.8
55000	0.66	0.670561	20632	464.773	19068	700	323	23001	24655.2
60000	0.72	0.674934	20617.4	469.347	19056	708	323	23001	26646.2
65000	0.71	0.675073	20509.3	476.268	19056	708	323	23001	28595.5
70000	0.72	0.675185	20533.2	472.667	19056	708	323	23001	30574.7

Table 56: Pareto set evolution of CBMix21

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.02	0.398718	4105.5	109.5	3726	117	102	4485	0.047
5000	0.31	1.2368	2365.29	44.871	2071	150	3	3280	628.008
10000	0.29	1.27286	2249.45	39.4138	1975	98	2	3282	1077.46
15000	0.42	1.30258	2158.24	49.6667	1941	131	2	3282	1700.14
20000	0.4	1.29837	2169.63	46.4	1941	120	2	3282	2331.87
25000	0.42	1.2979	2167.29	46.7619	1941	120	2	3280	2924.45
30000	0.45	1.29816	2156.8	48.6222	1941	119	2	3280	3510.5
35000	0.45	1.29898	2148.96	51.7333	1941	119	2	3280	4106.73

Table 57: Pareto set evolution of CBMix22

Iteration no.	RNI	I_H	Avg. RC	Avg. RB	RC*	RB	RB*	RC	CPU(s)*
1	0.03	0.732796	1147.67	52	1066	106	17	1196	0.031
5000	0.22	1.17352	880.136	20.1818	780	48	1	1182	234.385
10000	0.24	1.17361	880.208	19.4167	780	48	1	1182	467.392
15000	0.26	1.17423	876.962	19.8462	780	48	1	1182	696.939
20000	0.24	1.17465	878.083	19.5833	780	48	1	1182	924.071
22082	0.24	1.17465	878.083	19.5833	780	48	1	1182	1020.05

Table 58: Pareto set evolution of CBMix23

Bibliography

Aggarwal, C. C., Orlin, J. B., and Tai, R. P. (1997). Optimized crossover for the independent set problem. *Operations Research*, 45(2):226–234.

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network flows: theory, algorithms and applications*. Prentice Hall.

Alba, E. and Dorronsoro, B. (2004). *Evolutionary Computation in Combinatorial*, chapter Solving the vehicle routing problem by using cellular genetic algorithms, pages 11–20. Springer.

Bach, L., Hasle, G., and Wøhlk, S. (2013). A lower bound for the node, edge, and arc routing problem. *Computers and Operations Research*, 40(4):943–952.

Basseur, M., Seynhaeve, F., and Talbi, E. (2005). Path relinking in pareto multi-objective genetic algorithms. In Coello, C., Aguirre, A., and Zitzler, E., editors, *Evolutionary multi-criterion Optimization*, volume 3410, pages 120–134. Springer.

Beasley, J. E. (1983). Route first-cluster seconds methods for vehicle routing. *Journal of Management Science*, 11(4):403–408.

Belenguer, J. M. and Benavent, E. (2003). A cutting plane algorithm for the capacitated arc routing problem. *Computers and Operations Research*, 30(5):705–728.

Bell, J. E. and McMullen, P. R. (2004). Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics*, 18:41–48.

Benavent, E., Campos, V., Corberán, A., and Mota, E. (1992). The capacitated chinese postman problem: Lower bounds. *Networks*, 22(7):669–690.

Berger, J. and Barkaoui, M. (2003). A new hybrid genetic algorithm for the capacitated vehicle routing problem. *Journal of the Operational Research Society*, 54(12):1254–1262.

Beullens, P., Muyldermans, L., Cattrysse, D., and Oudheusden, D. V. (2003). A guided local search heuristic for the capacitated arc routing problem. *European Journal of Operational Research*, 147(3):629–643.

Bock, F. (1998). An algorithm for solving traveling-salesman and related network optimization problems. unpublished manuscript associated with talk presented at the 14th ORSA National Meeting.

Boonkleaw, A., Suthikarnnarunai, N., and Srinon, R. (2009). Strategic planning and vehicle routing algorithm for newspaper delivery problem: Case study of morning newspaper, bangkok, thailand. In *Proceedings of the World Congress on Engineering and Computer Science*, volume II, San Francisco, USA.

Bosco, A., Laganà, D., Musmanno, R., and Vocaturo, F. (2013). Modeling and solving the mixed capacitated general routing problem. *Optimization Letters*, 7(7):1451–1469.

Bramel, J., Coffman, E. G., Shor, P. W., and Simchi-Levi, D. (1991). Probabilistic analysis of the capacitated vehicle routing problem with unsplit demands. *Operations Research*, 40(6):1095–1106.

Brandáo, J. and Eglese, R. (2008). A deterministic tabu search algorithm for the capacitated arc routing problem. *Computers and Operations Research*, 35(4):1112–1126.

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

Chen, P., Huang, H.-K., and Dong, X.-Y. (2010). Iterated variable neighborhood decent algorithm for the capacitated vehicle routing problem. *Expert Systems With Applications*, 37:1620–1627.

Cheung, B. K. S., Langevin, A., and Villeneuve, B. (2001). High performing evolutionary techniques for solving complex location problems in industrial system design. *Journal of Intelligent Manufacturing*, 12(5-6):455–466.

Christofides, N., Mingozzi, A., and Toth, P. (1981). Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical prograaming*, 20:255 – 282.

Christofides, N., Mingozzi, A., Toth, P., and Sandi, C., editors (1979). *Combinatorial Optimization*, chapter 11, pages 315–328. John Willey, Chichester.

Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581.

Corberán, A., Farnandez, E., Laguna, M., and Marti, R. (2002). Heuristic solutions to the problem of routing school buses with multiple objectives. *Journal of Operational Research Society*, 53:427–435.

Cordeau, J.-F., Laporte, G., and Mercier, A. (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of Operational Research Society*, 52(8):928–936.

Croes, G. A. (1958). A method for solving traveling-salesman problem. *Operations Research*, 6(6):791–812.

Cunha and Mutarelli (2007). A spreadshet-based optimization model for the integrated problem of producing and distributing a major weekly newsmagzine. *European Journal of Operational Research*, 176:925–940.

Dantzig, G. B. and Ramser, J. M. (1959). The truck dispatching problem. *Management Science*, 6(1):81–91.

Darwin, C. (1859). *The Origin of Species*. John Murray, London.

Davis, L. (1985). Applying adaptive algorithms to epistatic domains. In *Proceedings of the 9th international joint conference on Artificial intelligence*, volume 1, pages 162–164.

DeArmon, J. S. (1981). A comparison of heuristics for the capacitated chinese postman problem. Master's thesis, University of Maryland, College Park, MD.

Deb, K., Agrawal, S., Pratap, S., and Meyarivan, A. (2000). A fast ellitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Parallel Problem Solving from Nature-PPSN VI*, pages 849–858, Paris, France.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transaction on Evolutionary Computation*, 6:182–197.

Deng, X., Zhu, Z., Yang, Y., Li, X., Tian, Y., and Xia, M. (2007). A genetic algorithm for the capacitated arc routing problem. In *Proceedings of the IEEE International Conference on Automation and Logistics*, pages 1551–1556.

Edgeworth, F. Y. (1881). *Mathematical Psychics*. C. Kegan Paul and Co., London, England.

Eglese, R. W. (1994). Routing winter gritting vehicles. *Discrete Applied Mathematics*, 48(3):231–244.

Eraslan, E. and Derya, T. (2010). Daily newspaper distribution planning with integer programming: an application in turkey. *Transportation Planning and Technology*, 33(5):423–433.

Ericsson, M., Resende, M., and Pardalos, P. (2002). A genetic algorithm for the weight seting problem in ospf routing. *Journal of Combinatorial Optimization*, 6:299–333.

Eswaran, K. P. and Tarjan, R. (1987). Augmentation problems. *SIAM Journal on Computing*, 5(4):653–665.

Fisher, M. L., Greenfield, A. J., and Jaikumar, R. (1982). A computrized vehicle routing application. *Interfaces*, 12(4):42–52.

Fleischer, M. (2003). The measure of pareto optima applications to multi-objective metaheuristics. In *Proceedings of the 2nd International Conference on Evolutionary Multi-Criterion Optimization*, pages 519–533.

Flood, M. M. (1956). The traveling-salesman problem. *Operations Research*, 4(1):61–75.

Floyd, R. W. (1962). Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345. doi: 10.1145/368996.369016.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artificial Intelligenece through Simulated Evolution*. John Wiley and Sons.

Fonseca, C. M. and Fleming, P. L. (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In Forrest, S., editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 416–423, Urbana-Champaign, IL, USA. Morgan Kaufmann.

Frederickson, G. N. (1979). Approximations algorithms for some postman problems. *Journal of the ACM*, 26(3):538–554.

Gaskell, T. J. (1967). Bases for vehicle fleet scheduling. *Operational Research Society*, 18(3):281–295.

Gendreau, M., Hertz, A., and Laporte, G. (1992). New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40(6):1086–1094.

Gendreau, M., Hertz, A., and Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290.

Ghiani, G., Guerriero, F., Laporte, G., and Musmanno, R. (2004). Tabu search heuristics for the arc routing problem with intermediate facilities under capacity and length restrictions. *Journal of Mathematical Modelling and Algorithms*, 3(3):209–223.

Gillett, B. and Miller, L. (1974). A heuristic for the vehicle dispatching problem. *Operations Research*, 22:340–349.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549.

Goldberg, D. E. and Deb, K. (1991). *Foundations of Genetic Algorithms*, chapter A comparative analysis of selection schemes used in genetic algorithms. Morgan Kaufmann, California.

Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodel function optimization. In *In Second International Conference on Genetic Algorithms*, pages 41–49.

Goldberg, D. E. and Robert Lingle, J. (1985). Alleles loci and the travelling salesman problem. In *Proceedings of the 1st International Conference on Genetic Algorithms and their applications*, pages 154–159.

Golden, B. L., Assad, A. A., and Wasil, E. A. (2002). *Routing vehicles in the real world*, chapter Applications in the Solid Waste, Beverage, Food, Diary, and Newspaper Industries. SIAM. Section: 10.4.

Golden, B. L., Dearmon, J. S., and Baker, E. K. (1983). Computational experiments with algorithms for a class of routing problems. *Computers and Operations Research*, 10(1):47–59.

Golden, B. L., Wasil, E. A., Kelly, J. P., and Chao, I. M. (1998). *Metaheuristics in vehicle routing*. Springer, Kluwer, Boston.

Golden, B. L. and Wong, R. T. (1981). Capacitated arc routing problems. *Networks*, 11(3):305–315.

Gómez-Villouta, G., Hamiez, J.-P., and Hau, J.-K. (2010). *IEA/AIE, Part-I, LNAI*, chapter Tabu search with consistent neighbourhood for strip packing. Springer.

Goncalaves, J. F. and Almeida, J. (2002). A hybrid genetic algorithm for assembly line balancing. *Journal of Heuristics*, 8:629–642.

Grandinetti, L., Guerriero, F., Laganá, D., and Pisacane, O. (2012). An optimization-based heuristic for the multi-objective undirected capacitated arc routing problem. *Computers Operations Research*, 39(10):2300–2309.

Gutiérrez, J. C. A., Soler, D., and Hervás, A. (2002). The capacitated general routing problem on mixed graphs. *Revita Investigacion Operacional*, 22(5):15–26.

Haimes, Y. Y., Lasdon, L. S., and Wismer, D. A. (1971). On a bicriterion formulation of the problems of integrated system identiïňĄcation and system optimization. *IEEE Transaction on Systems, Man, and Cybernetics*, 1(3):296–297.

Haimovich, M. and Kan, A. (1985). Bound and heuristics for capacitated routing problems. *Mathematics of Operations Research*, 10:527–542.

Hansen, P. and Mladenovi, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467.

Hasle, G. (2012). Routing applications in newspaper delivery. Report A23753, SINTEF, Oslo, Norway. ISBN: 978-82-14-05310-4.

Hasle, G., Kloster, O., and Smedsrud, M. (2011). A capacitated clustering-based method for newspaper delivery routing. In *19th Triennial Conference of the International Federation of Operational Research Societies (IFORS)*, Melbourne, Australia. Presented paper.

Hertz, A., Laporte, G., and Hugo, P. N. (1999). Improvement procedures for the undirected rural postman problem. *INFORMS Journal on Computing*, 11(1):53–62.

Hertz, A., Laporte, G., and Mittaz, M. (1997). A tabu search heuristic for the capacitated arc routing problem. In *Centre for research on transportation*. Montreal, Canada, crt-97-03 edition.

Hertz, A., Laporte, G., and Mittaz, M. (2000). A tabu search heuristic for the capacitated arc routing problem. *Operations Research*, 48(1):129–135.

Hertz, A. and Mittaz, M. (2001). A variable neighborhood decent algorithm for the undirected capacitated arc routing problem. *Transportation Science*, 35(4):425–434.

Hirabayashi, R., Saruwatari, Y., and Nishida, N. (1992). Tour construction algorithm for the capacitated arc routing problem. *Asia-Pacific Journal of Operational Research*, 9:155–175.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: The University of Michigan Press.

Horn, J. and Nafpliotis, N. (1993). Multiobjective optimization using the niched pareto genetic algorithm. Technical report, University of Illinois at Urban-Champaign. IlliGAL Report 93005.

Jozefowiez, N., Semet, F., and Talbi, E. G. (2006). Enhancements of NSGA II and its application to the vehicle routing problem with route balancing. In *Proceedings of the 7th international conference on Artificial Evolution*, pages 131–142.

Jozefowiez, N., Semet, F., and Talbi, E. G. (2007). Target aiming pareto search and its application to the vehicle routing problem with route balancing. *Journal of heuristics*, 13(5):455–469.

Jozefowiez, N., Semet, F., and Talbi, E. G. (2008). Multi-objective vehicle routing problems. *European Journal of Operational Research*, 189(2):293–309.

Jozefowiez, N., Semet, F., and Talbi, E. G. (2009). An evolutionary algorithm for the vehicle routing problem with route balancing. *European Journal of Operational Research*, 195(3):761–769.

Kirkpatrick, S., Gelatt, J., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Knowles, J. D. and Corne, D. W. (1999). The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 98–105.

Kokubugata, H., Moriyama, A., and Kawashima, H. (2007). A practical solution using simulated annealing for general routing problems with nodes, edges, and arcs. In Stuetzle, T., Birattari, M., and Hoos, H. H., editors, *Proceedings of the International conference on Engineering stochastic local search algorithms: designing, implementing and analyzing effective heuristics*, volume 4638 of *Lecture Notes in Computer Science*, pages 136–149, Berlin, Heidelberg. Springer.

Lacomme, P., Prins, C., and Ramdane-Cherif, W. (2004a). Competitive memetic algorithms for arc routing problems. *Annals of Operations Research*, 131(1-4):159–185.

Lacomme, P., Prins, C., and Sevaux, M. (2006). A genetic algorithm for a bi-objective capacitated arc routing problem. *Computers and Operations Research*, 33(12):3473–3493.

Lacomme, P., Prins, C., and Tanguy, A. (2004b). First competitive ant colony scheme for the CARP. In *Ant Colony Optimization and Swarm Intelligence*, volume 3172 of *Lecture Notes in Computer Science*, pages 426–427.

Li, L. Y. O. and Eglese, R. W. (1996a). An interactive algorithm for vehicle routeing for winter-gritting. *Journal of the Operational Research Society*, 47(2):217–228.

Li, L. Y. O. and Eglese, R. W. (1996b). A tabu search based heuristic for arc routing with a capacity constraint and time deadline. In *Meta-heuristics Theory and Applications*, pages 633–649, Dordrecht. Kluwer Academic Publishers.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44:2245–2269.

Liu, T., Jiang, Z., and Geng, N. (2013). A memetic algorithm with iterated local search for the capacitated arc routing problem. *International Journal of Production Research*. DOI: http://dx.doi.org/10.1080/00207543.2012.753165.

Løkketangen, A., J. Oppen, J. O., and D. L, W. (2012). An attribute based similarity function for vrp decision support. *Decision making in manufacturing and services*, 6(2):65–83.

López, M. (1998). Optimización mediante técnicas de simulación monte carlo del recorrido del servico de recogida de residuos en el municipo de aldaya (valencia): caso de trazado urbano con alto número de calles con sentido de circulación prohibido. *Proyecto Final de Carrera*. Escuela Técnica Superior de Ingenieros Industriales de la Universidad Politécnica de Valencia.

Martinez, C., Loiseau, I., Resende, M. G. C., and Rodriguez, S. (2011). BRKGA algorithm for the capacitated arc routing problem. *Electronic Notes in Theoretical Computer Science*, 281:69–83.

Mei, Y., Tang, K., and Yao, X. (2011). Decomposition-based memetic algorithm for multiobjective capacitated arc routing problem. *IEEE Transactions on Evolutionary Computation*, 15(2):151–165.

Miettinen, K. (1999). *Nonlinear Multiobjective Optimization*. Kluwer, Boston.

Miettinen, K. (2003). *Multi-Objective Programming and Goal Programming: Theory and Applications*, chapter Graphical illustration of Pareto optimal solutions, pages 197–202. Springer-Verlag, Berlin, Heidelberg.

Miller, C., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of the travelling salesman problem. *Journal of the Association for Computing Machinery*, 7:326–329.

Mole, R. H. and Jameson, S. R. (1976). A sequential route-building algorithm employing a generalized savings criterion. *Operational Research Quaterly*, 27:503–511.

Morse, J. N. (1980). Reducing the size of non-dominated set: Pruning by clustering. *Computers and Operations Research*, 7(1-2):55–66.

Murata, T. and Itai, R. (2005). Multi-objective vehicle routing problems using two-fold EMO algorithms to enhance solution similarity on non-dominated solutions. In *Proceedings of the Third international conference on Evolutionary Multi-Criterion Optimization*, pages 885–896, Berlin, Heidelberg. Springer.

Nagata, Y. and Bräysy, O. (2009). Edge assembly-based memetic algorithm for the capacitated vehicle routing problem. *Networks*, 54(4):205–215.

Nazif, H. and Lee, L. S. (2012). Optimised crossover genetic algorithm for capacitated vehicle routing problem. *Applied Mathematical Modelling*, 36:2110–2117.

Oliver, I. M., Smith, D. J., and Holland, J. R. C. (1987). A study of permutation crossover operators on the travellng salesman problem. In *Proceedings of 2nd International Conference on Genetic Algorithms and Their Application*, pages 224–230.

Or, I. (1976). *Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking*. PhD thesis, Northwestern University, Evanston, Illinois.

Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41(4):421–451.

Osman, I. H. and Christofides, N. (1994). Capacitated clustering problems by hybrid simulated annealing and tabu search. *International Transactions in Operational Research*, 1(3):317–336.

Pandit, R. and Muralidharan, B. (1995). A capacitated general routing problem on mixed networks. *Computers and Operations Research*, 22(5):465–478.

Pareto, V. (1906). *Manuale di Economia Politica*. Societa Editrice Libraria, Milano, Italy.

Pasia, J. M., Derner, K. F., Hartl, R. F., and Reimann, M. (2007). A population-based local search for solving a bi-objective vehicle routing. In *European conference on Evolutionary computation in combinatorial optimization*, pages 166–175.

Pearn, W. L. and Wu, T. C. (1995). Algorithms for the rural postman problem. *Computers and Operations Research*, 22(8):819–828.

Pisinger, D. and Ropke, S. (2006a). An adaptive large neighborhood search heuristic for the pick up and delivery problem with time windows. *Transportation Science*, 40(4):455–472.

Pisinger, D. and Ropke, S. (2006b). An adaptive large neighbourhood search heuristic for the pick up and delivery problem with time windows. *Transportation Science*, 40(4):455–472.

Polacek, M., Doerner, K. F., Hartl, R. F., and Maniezzo, V. (2008). A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. *Journal of Heuristics*, 14(5):405–423.

Potvin, J.-Y. and Bengio, S. (1996). The vehicle routing problem with time windows part II: Genetic search. *INFORMS Journal on Computing*, 8:165–172.

Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers and Operations Research*, 31(12):1985–2002.

Prins, C. and Bouchenoua, S. (2004). A memetic algorithm solving the VRP, the CARP and GENERAL routing problems with nodes, edges and arcs. In Hart, W. E., Krasnogor, N., and Smith, J. E., editors, *Recent Advances in Memetic Algorithms*, volume 166 of *Studies in Fuzziness and Soft Computing*, pages 65–85. Springer, Berlin, Heidelberg.

Prins, C., Labadi, N., and Reghioui, M. (2009). Tour splitting algorithms for vehicle routing problems. *International Journal of Production Research*, 47(2):507–535.

Rechenberg, I. (1973). *Evolutionsstrategie-Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, Frommann-Holzboog Verlag.

Rego, C. (1998). A subpath ejection method for the vehicle routing problem. *Management Science*, 44(10):1447–1459.

Rego, C. and Roucairol, C. (1996). A parallel tabu search algorithm using ejection chains for the vehicle routing problem. In Osman, I. H. and Kelly, J. P., editors, *Meta-heuristics: Theory and Applications*, pages 661–675. Springer, Kluwer, Boston, Kluwer, Boston.

Reimann, M., Doerner, K., and Hartl, R. F. (2004). D-ants: Savings based ants divide and conquer the vehicle routing problem. *Computers and Operations Research*, 31(4):563–591.

Riise, A. (2002). Comparing genetic algorithms and tabu search for multi-objective optimization. In *Proceedings of the IFORS Conference*, page 29, Edinburgh, UK.

Rochat, Y. and Taillard, E. D. (1995). Probabilistic diversification and intensification in local search for vehicle routing. *Journal of heuristics*, 1:147–167.

Russel, R., Chiang, W.-C., and Zepeda, D. (2008). Integrating multi-product production and distribution in newspaper logistics. *Computers and Operations Research*, 35:1576–1588.

Santos, L., Coutinho-Rodrigues, J., and Current, J. R. (2010). An improved ant colony optimization based algorithm for the capacitated arc routing problem. *Transportation Research Part B: Methodological*, 44(2):246–266.

Savelsbergh, M. W. P. (1992). The vehicle routing problem with time windows: minimizing route duration. *INFORMS Journal of Computing*, 4:146–154.

Schwefel, H.-P. (1977). *Numerische Optimierung von Computer-Modellen*. PhD thesis, Birkhäuser.

Segerstedt, A. (2013). A simple heuristic for vehicle routing-a variant of clarke and wright's saving method. *International Journal of Production Economics*. doi:http://dx.doi.org/10.1016/j.ijpe.2013.09.017.

Shaw, P. (1998). Using constraint programming and local methods to solve vehicle routing problems. In *Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming*, pages 417–431.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265.

Song, S. H., Lee, K. S., and Kim, G. S. (2002). A practical approach to solving a newspaper logistics problem using a digital map. *Computers and Industrial Engineering*, 43:315–330.

Spears, W. and DeJong, K. (1991). On the virtues of parametrized uniform crossover. In *Proceedings of the Fourth International Conference of Genetic Algorithms*, pages 230–236, San Diego, USA.

Srinivas, N. and Ded, K. (1995). Multiobjective optimization usinf non-dominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248.

Stadler, W. and Dauer, J. P. (1992). *Structural Optimization: Status and Promise*, chapter Multicriteria optimization in engineering: a tutorial and survey, pages 211–249. Published by American Institute of Aeronautics and Astronautics, Washington DC.

Taillard, É. (1993). Parallel iterative search methods for vehicle routing problems. *Networks*, 23(8):661–673.

Taillard, E., Badeau, P., Gendreau, M., Guertin, F., and Potvin, J.-Y. (1997). A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186.

Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation*. WILEY. ISBN: 0470278587 9780470278581.

Tan, K. C., Chew, Y. H., and Lee, L. H. (2006). A hybrid multi-objective evolutionary algorithm for solving truck and trailer vehicle routing problems. *European Journal of Operational Research*, 172(3):855–885.

Tang, K., Mei, Y., and Yao, X. (2009). Memetic algorithm with extended neighborhood search for capacitated arc routing problems. *IEEE Transactions on Evolutionary Computing*, 13(5):1151–1166.

Toth, P. and Vigo, D. (2002). *The vehicle routing problem*, chapter Routing vehicles in the real world: Applications in the solid waste, beverage, food, dairy and newspaper industries. SIAM, Philadelphia.

Toth, P. and Vigo, D. (2003). The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, 15(4):333–346.

Ulusoy, G. (1985). The fleet size and mix problem for capacitated arc routing. *European Journal of Operational Research*, 22(3):329–337.

Usberti, F. L., França, P. M., and França, A. L. M. (2011). Grasp with evolutionary path-relinking for the capacitated arc routing problem. *Computers and Operations Research*. doi: http://dx.doi.org/10.1016/j.cor.2011.10.014.

VanBuer, M. G., Woodruff, D. L., and Olson, R. T. (1999). Solving the medium newspaper production/distribution problem. *European Journal of Operational Research*, 11:237–253.

Veldhuizen, D. A. V. and Lamont, G. B. (1998). Evolutionary computation and convergence to a pareto front. In *Late Breaking Papers on the Genetic Programming*, pages 221–228.

Whitley, L. D., Starkweather, T., and Fuquay, D. (1989). Scheduling problems and traveling salesman: The genetic edge recombination operator. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 133–140, George Mason University, Fairfax, Virginia, USA.

Xing, L., Rohlfshagen, P., Chen, Y., and Yao, X. (2010). An evolutionary approach to the multidepot capacitated arc routing problem. *IEEE Transactions on Evolutionary Computation*, 14(4):356–374.

Xu, J. and Kelly, J. P. (1996). A network flow-based tabu search heuristic for the vehicle routing problem. *Transportation Science*, 30(4):379–393.

Yellow, P. C. (1970). A computational modification to the saving method of vehicle scheduling. *Operations Research Quarterly*, 21(2):281–283.

Zitzler, E., Laumanns, M., and Thiele, L. (2002). *Evolutionary Methods for Design, Optimisation, and Control*, chapter SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization, pages 19–26. CIMNE, Barcelona, Spain.

Zitzler, E. and Thiele, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271.

* * * * *