Using Genetic Algorithms for Multi-depot Vehicle Routing

Chapter · September 2008

DOI: 10.1007/978-3-540-85152-3_4

2 authors:

Beatrice Ombuki-Berman
Brock University
33 PUBLICATIONS **175** CITATIONS
SEE PROFILE

Franklin T. Hanshar
University of Guelph
11 PUBLICATIONS **340** CITATIONS
SEE PROFILE

Francisco Baptista Pereira and Jorge Tavares (Eds.)

Bio-inspired Algorithms for the Vehicle Routing Problem

Studies in Computational Intelligence, Volume 161

Editor-in-Chief
Prof. Janusz Kacprzyk
Systems Research Institute
Polish Academy of Sciences
ul. Newelska 6
01-447 Warsaw
Poland
E-mail: kacprzyk@ibspan.waw.pl

Francisco Baptista Pereira
Jorge Tavares
(Eds.)

Bio-inspired Algorithms for the Vehicle Routing Problem

Springer

Francisco Baptista Pereira
ISEC - Coimbra Institute of Engineering
Rua Pedro Nunes - Quinta da Nora
3030-199 Coimbra, Portugal
Centre for Informatics and Systems
University of Coimbra
Pólo II - Pinhal de Marrocos
3030-290 Coimbra, Portugal
Email: xico@dei.uc.pt

Jorge Tavares
INRIA Lille - Nord Europe
40, avenue Halley
Bât.A, Park Plaza
59650 Villeneuve d'Ascq, France
Email: jorge.tavares@ieee.org

ISBN 978-3-540-85151-6 e-ISBN 978-3-540-85152-3

DOI 10.1007/978-3-540-85152-3

Studies in Computational Intelligence ISSN 1860949X

Library of Congress Control Number: 2008933225

Typeset & Cover Design: Scientific Publishing Services Pvt. Ltd., Chennai, India.

Printed in acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

Preface

The vehicle routing problem (VRP) is one of the most famous combinatorial optimization problems. In simple terms, the goal is to determine a set of routes with overall minimum cost that can satisfy several geographical scattered demands. A fleet of vehicles located in one or more depots is available to fulfill the requests. A large number of variants exist, adding different constraints to the original definition. Some examples are related to the number of depots, the ordering for visiting the customers or to time windows specifying a desirable period to arrive to a given location.

The original version of this problem was proposed by Dantzig and Ramser in 1959 [1]. In their seminal paper, the authors address the calculation of a set of optimal routes for a fleet of gasoline delivery trucks. Since then, the VRP has attracted the attention of a large number of researchers. A considerable part of its success is a consequence of its practical interest, as it resembles many real-world problems faced everyday by distribution and transportation companies, just to mention a few applications areas. In this context, the development of efficient optimization techniques is crucial. They are able to provide new and enhanced solutions to logistic operations, and may therefore lead to a substantial reduction in costs for companies. Additionally, and from a research oriented perspective, the VRP is a challenging NP-hard problem providing excellent benchmarks to access the efficiency of new global optimization algorithms.

Given the nature of the problem, exact methods (such as branch-and-bound, which is probably the exact algorithm most widely applied to the VRP) are only able to solve small instances and therefore cannot be applied to real-world situations, heavily constrained and with a large number of customers. Heuristic methods are then the most reliable and efficient approach to address the VRP, as they are able to provide high-quality approximate solutions in a reasonable time. In the past 40 years, a large number of classical heuristics were proposed for the VRP, ranging from the well-known savings algorithm presented by Clarke and Wright in 1965 to more recent approaches based on simulated annealing, tabu search, variable neighborhood search, among many others [2, 3, 4].

Biological inspired computation is a field devoted to the development of computational tools modeled after principles that exist in natural systems. The adoption of such design principles enables the production of problem solving techniques with enhanced robustness and flexibility, able to tackle complex optimization situations. Prominent examples of bio-inspired approaches include evolutionary algorithms, swarm intelligence, neural networks or artificial immune systems. The first publications reporting the application of bio-inspired approaches to the VRP are from the early 1990's [5, 6]. Since then, a large number of proposals appeared for different variants of the problem. Results presented in recent publications show that bio-inspired approaches are now highly competitive with other state-of-the-art heuristics [7, 8, 9, 10, 11].

Given these circumstances, it is our conviction that this is an excellent occasion for the publication of this book. The fundamental principles related to the application of bio-inspired techniques to the VRP are now well established. In addition, this field of research is extremely active, as it can be confirmed by an inspection of recent issues of the most important journals and conference proceedings from this area. The goal of the volume is to present a collection of state-of-the-art contributions describing recent developments concerning the application of bio-inspired algorithms to the VRP. Over the 9 chapters, different algorithmic approaches are considered and a diverse set of problem variants are addressed. Some contributions focus on standard benchmarks widely adopted by the research community, while others address real-world situations.

The first chapter by Jean-Yves Potvin presents a comprehensive overview of the most relevant bio-inspired algorithms that have been applied to the VRP. This contribution starts with a definition of the basic VRP and of its most popular variants. Then it briefly introduces the main bio-inspired algorithms and reviews their application to the VRP.

In chapter 2, Christian Prins proposes several simple VRP heuristics based on Iterated Local Search (ILS): a pure ILS, an extension that allows the generation of several offspring per iteration (Evolutionary Local Search or ELS) and some hybrid forms combining either ILS or ELS with GRASP. Results show that ILS is able to create fast and efficient algorithms to the VRP. In particular, the hybrid GRASP$\times$ELS is competitive with state-of-the-art heuristics for this problem

The following chapter by Panagiotis P. Repoussis, Christos D. Tarantilis and George Ioannou addresses the Open Vehicle Routing Problem with Time Windows variant. The authors propose an algorithm based on the $(\mu + \lambda)$ evolution strategy, where offspring are generated exclusively via mutation based on the ruin-and-recreate principle. Each chromosome encodes both a solution to the problem and a set of strategy parameters that determine the mutation rate of the corresponding individual. Each generated offspring is locally improved using a Tabu search algorithm. Results on benchmark data sets demonstrate the competitiveness of the approach.

In their chapter, Beatrice Ombuki-Berman and Franklin T. Hanshar propose a genetic algorithm for the fixed destination multi-depot vehicle routing problem. In this variant several depots exist, each one with its own fleet. Customers can be served by a vehicle based on any of the depots. The algorithm presented in this chapter adopts an indirect encoding and an adaptive interdepot exchange strategy, coupled with capacity and route-length restrictions. The approach is competitive with other evolutionary algorithms previously applied to this variant, although there is still some room for improvement to better compete with Tabu-based approaches.

In chapter 5, Chun Yew Cheong and Kay Chen Tan address the VRP with stochastic demand, a probabilistic variant where the actual customer demand is only revealed when the vehicle reaches the corresponding location. In their contribution, the authors propose and incorporate in a multi-objective evolutionary algorithm, a set of hybrid operators. Several metaheuristics, such as simulated annealing or tabu search are combined with problem specific components, used for local exploitation and to generate routes that satisfy all the demands. Results show that the hybrid operators help the evolutionary algorithm to discover high quality solutions.

Jano van Hemert and J. A. La Poutré present a dynamic model for a pick-up-and-delivery problem, where all loads collected from various locations must be delivered to a central depot. The situation addressed is dynamic in the sense that customer requests are handled on-line and solutions must by found in real-time. The authors introduce the concept of fruitful regions, which consist of spatially clustered customers that together have a high potential of generating a demand. An evolutionary algorithm with anticipatory routing is used to generate solutions. In anticipatory routing, vehicles are routed to regions that have not yet demanded service, but exhibit a high potential for the origin of new loads. Results show that the exploration of fruitful regions can increase the effectiveness of a routing algorithm.

In the next chapter, Anna I. Esparcia-Alcázar, Manuel Cardós, J. J. Merelo, Anaís Martínez-García, Pablo García-Sánchez, Eva Alfaro-Cid and Ken Sherman propose a two-level algorithm to address the inventory and transportation problem. The goal for this problem is to minimize both the transport and inventory costs of a retail chain supplied from a central depot. The top level adopts an evolutionary algorithm to determine the weakly delivery pattern for each shop. The bottom level has to solve a daily VRP to obtain the transportation costs associated with a given delivery policy. Different methods, ranging from the classical Clarke and Wright's savings algorithm to an ant colony optimization approach, are used to find solutions for the VRP from the bottom level. The approach is tested in a set of benchmarks partially built with real-world data obtained from a major regional Spanish drugstore chain.

The chapter by Nubia Velasco, Philippe Castagliola, Pierre Dejax, Christelle Guéret and Christian Prins describes a real-world pick-up-and-delivery problem, involving the transportation of personnel within oil platforms. The proposed approach relies on a memetic algorithm with an indirect representation and a set of local search procedures used to improve the quality of the generated solutions. Results show that this method is able to significantly improve the quality of the existing solutions for the real instances.

The last chapter by Israel Beniaminy, Dovi Yellin, Uzi Zahavi and Marko Žerdin provides an original perspective to this book. The authors present a class of complex real-world field service optimization problems and show that their properties are similar to the ones found on the VRP with time windows variant. Field service optimization problems require finding an appropriate matching of resources, demands and times. Both resources and demands have physical locations. This chapter includes a discussion concerning the most important components that are relevant from a business oriented perspective and how they translate into constraints that must be included in the problem to solve. A genetic algorithm, an ant colony optimization method and an hybrid combining GRASP with ant colony optimization are used to solve some problem instances.

The overview of the contributions reveals that they are varied in subject and scope, even though they all address some variant of the VRP. We believe this diversity improves the quality of the volume and enhances its usefulness to practitioners, researchers and graduate students in optimization, natural computing and operations research courses. We do hope you find the book appealing and take benefit from it.

Acknowledgments

This book would never have become a reality without the contribution of many. First and foremost, we express our gratitude towards all the authors, who made the volume possible. We would also like to thank Springer's editorial staff, in particular to Professor Janus Kacprzyk, who invited us to publish this book and provided encouragement and support throughout the entire process. Finally, we are thankful to Penousal Machado, who provided valuable support and enthusiasm for this project.

Coimbra
July 2008

Francisco B. Pereira
Jorge Tavares

References

1. Dantzig, G.B., Ramser, J.H.: The Truck Dispatching Problem. Management Science 6, 80–91 (1959)
2. Laporte, G., Gendreau, M., Potvin, J.Y., Semet, F.: Classical and Modern Heuristics for the Vehicle Routing Problem. International Transactions in Operational Research 7, 285–300 (2000)
3. Gendreau, M., Laporte, G., Potvin, J.Y.: Metaheuristics for the Capacitated VRP. In: Toth, P., Vigo, D. (eds.) The Vehicle Routing Problem, pp. 129–154. SIAM, Philadelphia (2002)
4. Cordeau, J.F., Gendreau, M., Hertz, A., Laporte, G., Sormany, J.: New Heuristics for the Vehicle Routing Problem. In: Langevin, A., Riopel, D. (eds.) Logistic Systems: Design and Optimization, pp. 279–298. Wiley, Chichester (2005)
5. Thangiah, S., Nygard, K., Juell, P.: GIDEON: A Genetic Algorithm System for Vehicle Routing Problem with Time Windows. In: Proceedings of the Seventh IEEE Conference on Artificial Intelligence Applications, pp. 322–328 (1991)

6. Blanton, J., Wainwright, R.: Multiple Vehicle Routing with Time and Capacity Constraints Using Genetic Algorithms. In: Forrest, S. (ed.) Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 452–459. Morgan-Kaufmann, San Francisco (1993)
7. Pereira, F.B., Tavares, J., Machado, P., Costa, E.: GVR: A New Genetic Representation for the Vehicle Routing Problem. In: O'Neill, M., et al. (eds.) Proceedings of the 13th Irish Conference on Artificial Intelligence and Cognitive Science. LNCS, pp. 95–102. Springer, Heidelberg (2002)
8. Prins, C.: A Simple and Effective Evolutionary Algorithm for the Vehicle Routing Problem. Computers & Operations Research 31, 1985–2002 (2004)
9. Reimann, M., Doerner, K., Hartl, R.F.: D-ants: Savings Based Ants Divide and Conquer the Vehicle Routing Problem. Computers & Operations Research 31, 563–591 (2004)
10. Tan, K.C., Chen, Y.H., Lee, L.H.: A Hybrid Multiobjective Evolutionary Algorithm for Solving Vehicle Routing Problem with Time Windows. Computational Optimization and Applications 34, 115–151 (2006)
11. Mester, D., Bräysy, O.: Active-Guided Evolution Strategies for Large-Scale Capacitated Vehicle Routing Problems. Computers & Operations Research 34, 2964–2975 (2007)

Contents

List of Contributors

Eva Alfaro-Cid
Complex Adaptive Systems Group
Instituto Tecnológico de Informática
Valencia, Spain

Israel Beniaminy
ClickSoftware Technologies Ltd
2 Rechavam Ze'evi Street
Givat Shmuel, 54017 Israel
`israel.beniaminy@clicksoftware.com`

Manuel Cardós
Dept. de Organización de Empresas
Universidad Politécnica de Valencia
Valencia, Spain
`mcardos@doe.upv.es`

Philippe Castagliola
IRCCyN, Université de Nantes
2 avenue du Professeur Jean Rouxel
BP 539 44475 Carquefou, France
`philippe.castagliola@univ-nantes.fr`

Chun Yew Cheong
Department of Electrical and Computer Engineering
National University of Singapore
4 Engineering Drive 3, Singapore
`cheongchunyew@nus.edu.sg`

Pierre Dejax
IRCCyN, École des Mines de Nantes
La Chantrerie
BP 20722 44307 Nantes, France
`dejax@emn.fr`

Anna I Esparcia-Alcázar
Complex Adaptive Systems Group
Instituto Tecnológico de Informática
Valencia, Spain
`anna@iti.upv.es`

Pablo García-Sánchez
Dept. de Arquitectura y Tecnología de Computadores
Universidad de Granada
Granada, Spain

Christelle Guéret
IRCCyN, École des Mines de Nantes
La Chantrerie
BP 20722 44307 Nantes, France
`gueret@emn.fr`

Franklin T. Hanshar
University of Guelph
Ontario, Canada
`fhanshar@uoguelph.ca`

J.I. van Hemert
National e-Science Centre, School of Informatics
University of Edinburgh
15 South College Street

Edinburgh EH8 9AA, United Kingdom
J.vanHemert@ed.ac.uk

George Ioannou
Department of Management Science & Technology
Athens University of Economics & Business
Evelpidon 47A & Leukados 33,
GR-11362, Athens, Grecce
ioannou@aueb.gr

Anaís Martínez-García
Complex Adaptive Systems Group
Instituto Tecnológico de Informática
Valencia, Spain

J.J. Merelo
Dept. de Arquitectura y Tecnología de Computadores
Universidad de Granada
Granada, Spain
jmerelo@geneura.ugr.es

Beatrice Ombuki-Berman
Brock University
Ontario, Canada
bombuki@brocku.ca

Jean-Yves Potvin
Département d'Informatique et de Recherche Opérationnelle
Centre Interuniversitaire de Recherche sur les Réseaux d'Entreprise, la Logistique et le Transport
Université de Montréal
Montreal, QC, Canada H3C 3J7
potvin@iro.umontreal.ca

J.A. La Poutré
Dutch National Research Institute for Mathematics and Computer Science
NL-1090 GB Amsterdam, The Netherlands
School of Technology Management
Eindhoven University of Technology
5600 MB Eindhoven, The Netherlands
Han.La.Poutre@cwi.nl

Christian Prins
Institute Charles Delaunay
University of Technology of Troyes
BP 2060, 10010 Troyes Cedex, France
christian.prins@utt.fr

Panagiotis P. Repoussis
Department of Management Science & Technology
Athens University of Economics & Business
Evelpidon 47A & Leukados 33,
GR-11362, Athens, Grecce
prepousi@aueb.gr

Ken Sharman
Complex Adaptive Systems Group
Instituto Tecnológico de Informática
Valencia, Spain

Kay Chen Tan
Department of Electrical and Computer Engineering
National University of Singapore
4 Engineering Drive 3, Singapore
eletankc@nus.edu.sg

Christos D. Tarantilis
Department of Management Science & Technology
Athens University of Economics & Business
Evelpidon 47A & Leukados 33,
GR-11362, Athens, Grecce
tarantil@aueb.gr

Nubia Velasco
Universidad de los Andes
Carrera 1 Este No. 19A -40
Bogotá, Colombia
nvelasco@uniandes.edu.co

Dovi Yellin
ClickSoftware Technologies Ltd
2 Rechavam Ze'evi Street
Givat Shmuel, 54017 Israel
`dovi.yellin@clicksoftware.com`

Uzi Zahavi
Bar-Ilan University
Ramat Gan, 52900 Israel
`zahaviu@cs.biu.ac.il`

Marko Žerdin
Zany Ants Ltd.
Exceter, United Kingdom
`marko.zerdin@zanyants.com`

A Review of Bio-inspired Algorithms for Vehicle Routing

Jean-Yves Potvin

Département d'Informatique et de Recherche Opérationnelle and
Centre Interuniversitaire de Recherche sur les Réseaux d'Entreprise, la Logistique et le Transport,
Université de Montréal,
C.P. 6128, succ. Centre-Ville, Montreal, QC, Canada H3C 3J7
potvin@iro.umontreal.ca

Summary. This chapter reviews biologically inspired algorithms for solving a class of difficult combinatorial optimization problems known as vehicle routing problems, where least-cost collection or delivery routes are designed to serve a set of customers in a transportation network. From a methodological standpoint, the review includes evolutionary algorithms, ant colony optimization, particle swarm optimization, neural networks, artificial immune systems and hybrids. From an application standpoint, the most popular vehicle routing variants are considered, starting with the classical vehicle routing problem with capacity constraints.

1 Introduction

This paper is interested in biologically inspired computing, a branch of natural computing [14] where algorithms inspired from nature are developed to solve highly complex problems, in particular problems that cannot be addressed in a satisfactory way with traditional approaches. Under this paradigm, algorithmic models of processes observed in nature are developed and implemented on computers to explore solution spaces.

In the scientific literature, there is a growing interest in the exchange of ideas between natural systems and computational systems. Bio-inspired algorithms are representative of this trend and have been applied to a large variety of problems, including optimization problems. Among them, vehicle routing problems have generated a lot of attention, because they are inherently complex and hold a central place in real-world distribution activities. They generalize the well-known traveling salesman problem (TSP), a canonical combinatorial optimization problem that has been widely studied in the literature [45]. Given a graph with a number of vertices and a cost associated with each arc, the TSP looks for a least cost tour that visits each vertex exactly once (where the cost of a tour is the sum of its arc costs). In vehicle routing problems, side constraints are introduced to limit the number of vertices that can be visited with a single tour, thus leading to solutions that cover all vertices with different routes that start from and end at a particular vertex, called the depot. These problems are used to model various real-world applications. In the manufacturing sector, for example, they model transportation activities within the supply chain, that is, the movement of goods from suppliers to factories and then to customers. In the service sector, they include activities such as mail delivery, garbage collection, public transportation, etc.

The chapter is organized as follows. Section 2 introduces the basic Vehicle Routing Problem (VRP) and its main variants. This is followed in Section 3 by a brief review of some classical problem-solving methodologies. Then, Sections 4 to 8 introduce the main bio-inspired algorithms, namely Evolutionary Algorithms (EA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Neural Networks (NN) and Artificial Immune Systems (AIS), in this order, and describe their application to vehicle routing problems. Section 9 is then devoted to hybrids that combine ideas from different bio-inspired algorithms. Concluding remarks follow in Section 10.

2 Vehicle Routing

The aim of vehicle routing is to determine optimal collection or delivery routes for a fleet of vehicles in a transportation network [43, 44, 85]. These NP-hard problems generalize the classical Traveling Salesman Problem (TSP) by requiring both an assignment of vertices to vehicles and a sequencing of these vertices within each vehicle route to obtain a solution. A large number of variants are reported in the literature, depending on the objective function to be optimized and the types of constraints to be satisfied. These variants are reviewed in the following.

2.1 Vehicle Routing Problem (VRP)

The VRP can be formally stated as follows. Let $G = (V, A)$ be a graph where $V = \{1, 2, ..., n\}$ is the vertex set and A is the arc set. Vertex 1 is the depot for a fleet of identical vehicles of capacity Q that collects a demand q_i at each vertex (customer) $i \in V - \{1\}$. A non negative cost c_{ij} is also associated with every arc $(i, j) \in A$, $i \neq j$. This cost is often interpreted as a distance or a travel time, depending on the context. Unless otherwise stated, it is assumed in the following that the problem is symmetrical, that is, $c_{ij} = c_{ji}$ for every arc (i, j). The problem is then to determine a set of least cost vehicle routes such that:

- each vertex, apart from the depot, is visited exactly once by exactly one vehicle to serve its demand;
- all vehicle routes start and end at the depot;
- the total demand on each route does not exceed the vehicle capacity;

Sometimes, a fixed cost is associated with each vehicle. The objective is then aimed at minimizing a weighted sum of fixed costs and travel costs. A maximum distance or time constraint can also be considered, in addition to the capacity constraint, for each vehicle route.

2.2 Vehicle Routing Problem with Time Windows (VRPTW)

In this extension of the VRP, a time interval or time window $[a_i, b_i]$ constrains the beginning of service at each vertex $i \in V - \{1\}$. There is also a window $[a_1, b_1]$ at the depot to constrain the start time and end time of each vehicle route. The upper bound b_i at vertex i can be either a hard of a soft constraint. In the latter case, the vehicle is allowed to arrive late, but a penalty is incurred in the objective. A waiting time is introduced in the route schedule when the vehicle arrives before the lower bound a_i. In most papers, a hierarchical objective is considered. The first objective is to minimize the number of vehicles and, for the same number of vehicles, the distance, travel time or scheduling time (travel time + waiting time) is minimized. The VRP with time Deadlines (VRPD) is a special case of the VRPTW where only a time upper bound is associated with each vertex.

2.3 Vehicle Routing Problem with Backhauls (VRPB)

Here, the set of vertices is partitioned into two subsets: linehaul vertices for which the demand is delivered from the depot and backhaul vertices for which the demand is picked up and brought back to the depot. It has been quickly recognized that substantial cost savings can be achieved by allowing empty vehicles to pick up inbound products when they return to the depot. In the grocery industry, for example, linehauls could be supermarkets where goods are delivered while the backhauls could be grocery suppliers. In the classical VRPB, there is a strict precedence relationship between the linehaul and backhaul vertices, that is, all linehauls must be visited before the backhauls. Without this constraint, goods could be picked up while other goods would not have yet been delivered, thus potentially leading to a rearrangement of the goods inside the vehicle. In other variants, this constraint is relaxed to allow a vehicle to pick up goods if the capacity utilization is sufficiently low (i.e., if only a few linehaul vertices have not been visited yet). Finally, in the mixed VRPB, linehaul and backhaul vertices can be freely mixed as long as the capacity constraint is satisfied. When time windows are associated with the vertices, the VRPBTW is obtained.

2.4 Time-Dependent Vehicle Routing Problem (TDVRP)

In the TDVRP, the travel times are not fixed but rather depend both on the distance between two vertices and the time of the day (e.g., it takes longer to get from one location to another during rush hours). The TDVRP is thus aimed at more closely modeling situations observed in the real world.

2.5 Pick-Up and Delivery Problem (PDP)

In this problem, each customer request corresponds to a pair of vertices. That is, for a given pair of vertices $i+$ and $i-$, the demand must be picked up at vertex $i+$ and delivered at vertex $i-$. Clearly, both vertices must be in the same vehicle route (pairing constraint) and vertex $i+$ must be visited before $i-$ (precedence constraint). When time windows are associated with the vertices, the PDPTW is obtained. When people are transported, as in transportation-on-demand services, trip duration constraints must also be taken into account. The latter problem is known as the dial-a-ride problem (DARP).

2.6 Other Vehicle Routing Problems

Without being exhaustive, we also want to mention the following problems:

- *multiple-depot* problems where each vehicle can start (end) its route from (to) a different depot;
- *mixed fleet and size* problems where the fleet size must be determined based on different types of vehicles with different characteristics (e.g., capacity);
- *periodic vehicle routing* problems where routes are determined over an horizon that spans a number of periods and where each vertex must be visited at a given frequency within this horizon.
- *location-routing* problems where strategic decisions about the location of different facilities (e.g., depots, warehouses) are taken concurrently with the determination of vehicle routes;
- *inventory routing* where each vertex has an inventory and delivery routes are determined to replenish the inventory so as to avoid any inventory shortage.
- *stochastic vehicle routing* problems where some data are not known with certainty. For example, the demand might not be known with certainty at a given vertex, but the probability distribution is known and can be used to minimize the expected solution value.
- *dynamic vehicle routing* problems where new information are obtained as the vehicle routes are executed. For example, new service requests occur and must be integrated into the current routes or the travel times suddenly change due to unforeseen events.

There are also a number of vehicle routing problems where arcs must be visited instead of vertices, but these problems are not addressed here (see [20, 21, 22]). Arc routing problems are useful to model applications where the customers are located along the streets of the transportation network, as in postman delivery routes.

3 Classical Problem-Solving Methodologies

In this section, we briefly review some classical methodologies for solving vehicle routing problems. These methodologies can be divided into two main classes: exact methods and heuristics. Exact methods are divided into (1) tree search methods, like branch-and-bound, (2) dynamic programming and (3) integer programming-based methods. Although these methods produce optimal solutions, they can be computationally expensive, even for relatively small instances. We refer the reader to [43] for further details on this topic.

In the case of heuristics, we can distinguish between construction heuristics, improvement heuristics and metaheuristics, as it is explained below.

3.1 Construction Heuristics

In this category, we have pure construction heuristics and two-phase construction heuristics. Pure construction heuristics are:

Insertion heuristics

These heuristics are widely used to quickly construct a solution. At each iteration, a vertex is selected among all unvisited vertices and a feasible, least cost, insertion place between two consecutive vertices is looked for. This insertion process is repeated until all vertices are visited. This is a generic approach that can easily account for different types of side constraints. Figure 1 illustrates the insertion of vertex k between two vertices i and j with the corresponding insertion cost or detour $c_{ik} + c_{kj} - c_{ij}$.

In sequential variants, routes are constructed one by one. That is, the vertices are inserted in the current route until it is not possible to add any new vertex without violating one or more constraints. At this point, a new route is constructed. In the parallel variants, all routes are constructed at once. Thus, a vertex can be inserted in any particular route, depending on the location of the best feasible insertion place.

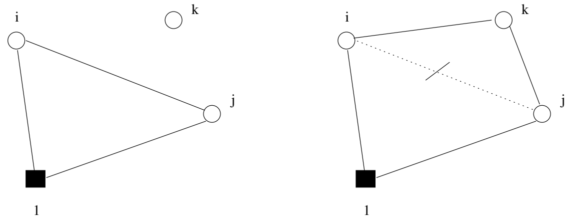


Fig. 1. Inserting a vertex

Savings heuristic

The savings heuristic is a well-known problem-solving approach for the VRP [11]. Starting from a configuration where each vertex is visited by an individual route connected to the depot, two routes are selected at each iteration and merged together, as illustrated in Figure 2. The merging process is guided by the savings measure. That is, for a given pair of vertices i and j, the initial cost of the two individual routes is $2c_{1i} + 2c_{1j}$. By merging these two routes into a single route, the cost becomes $c_{1i} + c_{1j} + c_{ij}$ for a savings $s_{ij} = 2c_{1i} + 2c_{1j} - (c_{1i} + c_{1j} + c_{ij}) = c_{1i} + c_{1j} - c_{ij}$. These savings are thus calculated for every pair of vertices and sorted in non increasing order. Pairs of routes are then merged together, by

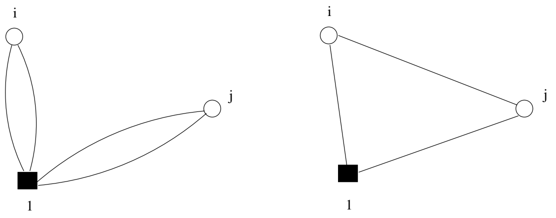


Fig. 2. Merging two routes

using the sorted list of savings to guide the merging process (starting with the largest one) until it is not possible to merge two routes without violating the side constraints.

Two-phase construction heuristics are:

Cluster-first, route-second

Here, vertices are first grouped into distinct subsets based on some proximity measure. Then, within each group, the vertices are ordered to form a route using, for example, an insertion heuristic.

Route-first, cluster-second

This an alternate approach to cluster-first, route-second, where a giant route that visits all vertices is first constructed. Then, this giant route is partitioned into smaller feasible routes.

3.2 Improvement Heuristics

Once an initial solution has been constructed, it can be further improved with a local search heuristic. In this case, a class of modifications to the current solution is defined to generate a neighborhood of solutions. The best solution in this neighborhood, if better than the current solution, then becomes the new current solution. The procedure is repeated until there is no better solution in the neighborhood of the current solution. Some basic modifications, based either on nodes or arcs in the solution, are the following:

- *vertex move*: a vertex is removed from the solution and inserted at another place;
- *vertex swap*: two vertices exchange their position in the solution;
- *arc exchange*: r arcs are removed from the solution and are replaced by r new arcs to produce a new valid solution. The most widely used exchanges of this type involve $r = 2, 3$ arcs and are called 2-opt and 3-opt [48].

Numerous extensions and adaptations of these basic movements for different types of vehicle routing problems are reported in the literature. For example, it is possible to move or swap sequences of consecutive vertices (rather than a single vertex) or to consider arc exchanges that maintain the current orientation of the routes for problems with time windows. Also, these modifications can be applied to each route individually, to modify the sequence of vertices in the route, or they can be applied to several routes at once to modify the vertex assignment.

3.3 Metaheuristics

The emergence of metaheuristics for solving difficult combinatorial optimization problems, including vehicle routing problems, is one of the most notable achievement of the last two decades in operations research. Metaheuristics offer global search strategies for exploring the solution space that are then adapted to a particular class of problems. Some metaheuristics are high-level abstraction of optimization processes observed in nature and will be described in the next sections. Among the other types of metaheuristics that have been successful on different types of vehicle routing problems, tabu search [32] is certainly the most notable one. Some applications of simulated annealing [41] and variable neighborhood search [54] are also reported in the literature. In all cases, the definition of an appropriate neighborhood structure is a crucial point. Some researchers favor streamlined implementations with a few parameters and simple neighborhood structures, like the unified tabu search [12]. Others propose complex modifications to the current solution, thus leading to large or very large neighborhood searches [1]. The reader is referred to [27] for further details on this issue.

In the following sections, the application of the main bio-inspired algorithms for solving vehicle routing problems are reviewed, starting with EAs.

4 Evolutionary Algorithms

Evolutionary algorithms are stochastic search methods that operate on a population of solutions by simulating, at a high level of abstraction, the evolution of species observed in nature. Genetic algorithms (GA) [34] were the first evolutionary algorithms to be applied to combinatorial optimization problems. In a GA, a population of solutions evolves over a number of generations through the application of operators, like selection, crossover and mutation, that mimic the corresponding genetic processes observed in nature. Typically, the solutions are encoded using strings of fixed length (in classical GAs, bitstrings are used). More formally, a GA can be sketched as follows:

```
1. Create an initial population of $P$ solutions.
2. Evaluate each solution.
3. Repeat for a fixed number of generations:
   3.1 Repeat until $P$ offspring solutions are created:
       3.1.1 Select two parent solutions in the population (with replacement)
             using a randomized selection procedure based on the solution values.
       3.1.2 Apply crossover to the two parent solutions to create two offspring
             solutions.
```

parent 1:	**1 2 3 4**	**5 6**
parent 2:	1 6 5 4	3 2
offspring 1:	**1 2 3 4**	3 2
offspring 2:	1 6 5 4	**5 6**

Fig. 3. One-point crossover

```
        3.1.3 Apply mutation (with a small probability) to each offspring.
        3.1.4 Include the two offspring in the new population.
    3.2 Evaluate each offspring in the new population.
    3.3 Replace the old population by the new one.
4. Return the best solution found.
```

First, an initial population of solutions is created, either randomly or through heuristic means. Then, the value of each solution, called fitness in GA terminology, is computed. The next population is then created from parent solutions chosen in the current population. A typical randomized selection procedure for the GA is the proportional or roulette wheel selection, where the selection probability of each solution is proportional to its value. When two parents are selected, they are modified, with a certain probability, by the crossover operator. Here, two offspring solutions are created from two parents through an exchange of solution components. For example, in the case of a bitstring encoding, the classical one-point crossover selects a cut point on the two parent strings and exchanges their end parts. Mutation is a secondary operator that processes each offspring position by position and flips the bit value at each position with a small probability. Together, selection and crossover search the problem space by combining solution components associated with good solutions to create new solutions with hopefully better values. Mutation acts as a perturbation operator to prevent premature convergence to bad suboptimal solutions.

Although some theoretical results that characterize the behavior of a GA have been reported for bitstring encodings, not all problems lend themselves easily to this representation. This is the case, in particular, for vehicle routing problems where an integer representation is more appropriate. In this case, an integer stands for a vertex, while a string of integers represents a vehicle route. For example, the string 1-5-2-3-1-4-6-1 would mean that, starting from the depot, one vehicle visits vertices 5, 2 and 3, in this order, before returning to the depot; a second vehicle visits vertex 4 first and then vertex 6 before returning to the depot. Note that the depot index 1 acts as a separator between the two routes. With this representation, called the path representation, a straightforward application of a classical GA does not work. As illustrated in Figure 3, if we assume a single route on six vertices numbered from 1 to 6, where vertex 1 is the depot, the one-point crossover produces two invalid offspring 1-2-3-4-5-6 and 1-6-5-4-3-2 when the cut point is chosen between the 4th and 5th position (note that the return to vertex 1 at the end to close the tour is not shown and is implicit). On both offspring, vertices are duplicated while others are missing.

parent 1:	**1 2**	**3 4**	**5 6**
parent 2:	1 6	5 4	3 2
offspring:	6 5	**3 4**	2 1

Fig. 4. *OX* crossover

Numerous studies have thus been conducted to address vehicle routing problems with GAs based on innovative representation schemes and specialized operators. For example, order-based crossover operators, like the Order crossover *OX* [59], have been designed to allow valid offspring routes to be generated. In the case of *OX*, two cut points are randomly chosen and the substring between the two cut points on the first parent is copied to the offspring. Then, the remaining positions are filled by following the ordering on the second parent, starting at the position just after the second cross point. When the end of the string is reached, the procedure resumes at position 1. Figure 4 shows an example on a tour with six vertices. The substring made of vertices 3 and 4 in parent 1 is first copied to the offspring. Then, the vertices are processed in the order 3, 2, 1, 6, 5, 4 on the second parent. Vertex 3 is discarded because it is already included in the offspring. Vertex 2 is then inserted at position 5, followed by vertex 1 at position 6, vertex 6 at position 1 and vertex 5 in position 2. Finally, vertex 4 is discarded. This operator thus produces a valid ordering. Many order-based crossover operators like this one are reported in the TSP literature to produce valid offspring orderings [62]. The experience gained with the TSP have also led researchers to use powerful local search-based mutation operators that fully optimize each offspring solution produced through crossover.

From the previous discussion, it is clear that a number of GAs for solving vehicle routing problems are direct extensions of GAs developed for the TSP. A good example is the edge assembly crossover EAX, an operator that proved very powerful for the TSP and was later adapted to the VRP [57]. In the case of the VRP, this operator can be described as follows:

1. Construct a graph by combining the edges of the two parent solutions.
2. Partition the set of edges in this graph with cycles created by alternately selecting an edge from the first and second parents.
3. Select a subset of cycles.
4. Generate an intermediate solution as follow. Take one parent and remove all edges that are in the selected subset of cycles. Then, the edges in the subset of cycles that come from the other parent are added. At this point, the intermediate solution is a collection of routes connected to the depot plus subtours that are not connected to the depot.
5. Create a complete solution. A greedy heuristic is applied where, at each iteration, a subtour is merged at least cost to a route or to another subtour. The procedure is repeated until a set of routes is obtained that visits all vertices.
6. Eliminate capacity violations. Using a penalty function for route overcapacity, restricted 2-opt arc exchanges [48] and vertex swaps are applied until a

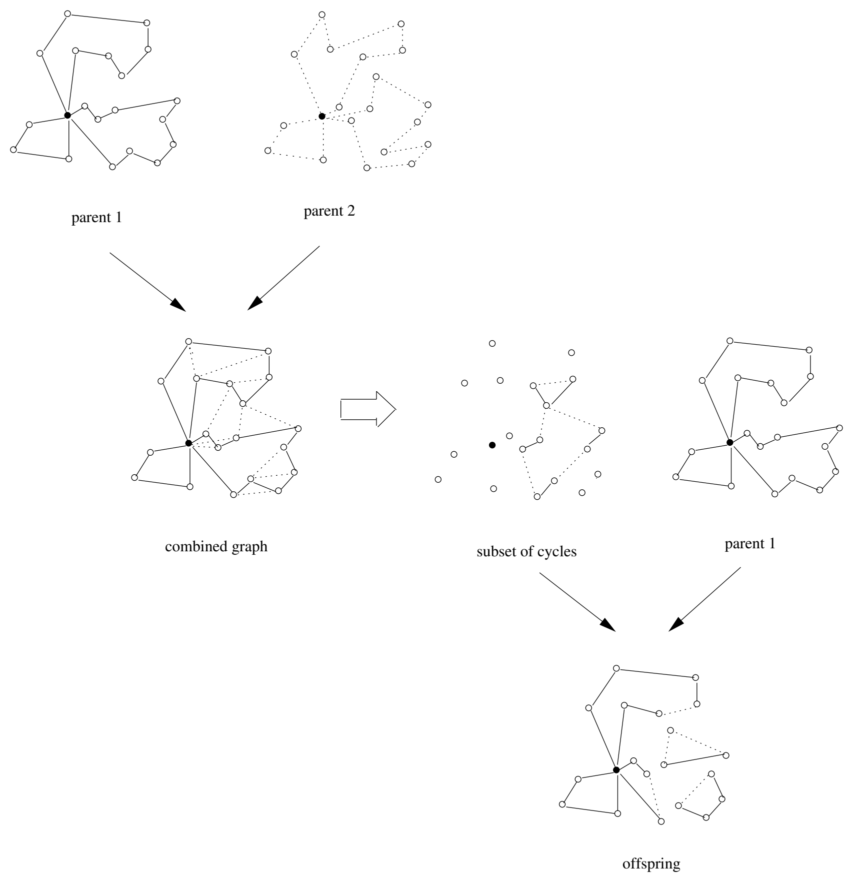


Fig. 5. *EAX* crossover

feasible solution is obtained. These modifications are said to be restricted because an infeasible route must be involved.

The *EAX* crossover is illustrated in Figure 5. From two parents, a combined graph is obtained. Then, based on the particular set of cycles shown in this example, an offspring is produced. The latter is obtained from parent 1, by removing the edges of parent 1 and by adding the edges of parent 2 that are in the set of cycles. Clearly, the offspring is not a valid solution since it contains two subtours that are not connected to the depot. These subtours thus need to be merged with the routes that are connected to the depot to obtain a valid solution.

A GA with *EAX* reached the best solution on a subset of VRP instances taken from Christofides, Mingozzi and Toth [10] and it found ten new best solutions on the data set of Golden et al. [33]. This work has also been extended to the VRPTW in [58]. In this case, an additional term for time window violations is included in the penalty function when feasibility is restored. Another good example of a crossover operator that has been extended from the TSP to the VRPTW is the natural crossover (for a description of this operator, see [38]).

In the following subsections, we present GA methodologies that have been specifically developed for vehicle routing problems. They are illustrative of different ways of handling side constraints.

4.1 Cluster-First, Route-Second

The methods presented in this section follow the cluster-first, route-second problem-solving approach, where clusters of vertices that naturally fit in the same vehicle route (due, for example, to spatial proximity) are first identified. Then, the vertices are ordered within each cluster to form routes. Here, the GA is used in the clustering phase, while the routing is done through insertion and improvement heuristics.

GIDEON

GIDEON [82] was applied to the VRPTW by first clustering the vertices in a way similar to the sweep heuristic of Gillett and Miller [31]. In the sweep heuristic, a seed point is chosen and, using the depot as a pivot, the ray from the depot to the seed is swept clockwise or counter-clockwise. Vertices are added to the current cluster as they are swept until the capacity constraint forbids the addition of the next vertex. This vertex then becomes the seed point for the next cluster. This is repeated until all vertices are assigned to a particular cluster. Routes are then constructed within each cluster by sequencing the vertices.

In the case of GIDEON, we assume a set of n vertices with planar coordinates (x_i, y_i) and polar angle p_i. The method divides the vertices into m clusters by identifying a set of seed angles, s_0, ..., s_m and by drawing a ray between the depot and each seed angle. The initial seed angle s_0 is assumed to be 0^o. The first cluster thus lies between seed angles s_0 and s_1, the second one between seed angles s_1 and s_2, and so on. GIDEON assigns vertex i to cluster k if $s_{k-1} < p_i \leq s_k$, $k =$ 1,..., m, as illustrated in Figure 6. Each angle s_k is obtained from the previous one by adding a fixed angle F and an offset e_k. That is, $s_k = s_{k-1} + F + e_k$, $k =$ 1,..., m. The fixed angle is the minimum angle value and insures that each cluster is represented. The offset is the extra angle that allows the cluster to encompass a larger or smaller area. If the addition of the fixed angle F and the offset e_k to s_{k-1} exceeds 360^o, then s_k is set to 360^o, thereby allowing the method to consider less than m vehicles in a solution. Within this clustering scheme, a set of m integer offsets is encoded in base-2 notation on a bitstring, and the GA searches for the offsets that result in the best solution.

Once the clusters are identified, the ordering of the vertices within each cluster is done with a least-cost insertion heuristic, where the cost accounts for the total distance and penalties for time window and capacity violations. Through these penalties, infeasible solutions are less attractive and less likely to be selected as parents for the next generation. At the end, a local search heuristic is applied to

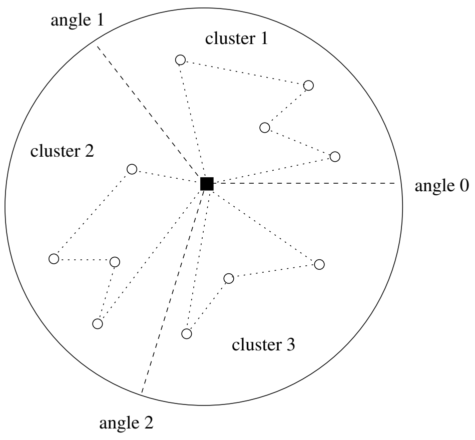


Fig. 6. Clustering phase of GIDEON

the best solution produced by GIDEON. The neighborhood structure is based on λ-interchanges [60], with $\lambda = 2$. In this case, one or two vertices are moved to another route or exchanged with one or two vertices from another route. This final phase is particularly important to reduce or eliminate capacity or time window violations. GIDEON led to the development of GenClust, which is described in the following.

GenClust

Quite often, it is desirable to have vehicle routes with particular geometric shapes, like petal or circular shapes. GenClust [81] explores this idea, while following the general principles of GIDEON. Here, a geometric shape is described via a set of attributes. For example, two attributes may be used to describe a circle, namely, its center (x, y) and its radius r. These numeric attributes are then encoded on a bitstring in base-2 notation. More precisely, each string encodes m different circles, one for each cluster, so that all vertices within a given circle are assigned to the same vehicle route (when a vertex is not inside any circle, it is simply associated with the closest one). The GA thus searches for the set of circles that lead to the best solution. GenClust was later applied to a multi-depot extension of the VRP [83]. In this case, for each set of circles produced by the GA, a solution to the multi-depot problem is obtained by first associating each circle or cluster with the closest depot.

4.2 Route-First, Cluster-Second

This approach uses a path representation to encode a (likely infeasible) unique giant tour that visits all vertices. With this representation, classical order-based crossover and mutation operators previously developed for the TSP can be used, given that a single tour is involved. A final solution is obtained by partitioning the giant tour into individual feasible routes. In [65], an exact polynomial-time algorithm is proposed to perform the partition in the case of the VRP. This algorithm works on an auxiliary directed acyclic graph with vertex set {1, 2,..., n}, where vertex 1 is the depot. An arc (i, j) is added to the graph when a route from vertex $i+1$ to vertex j, based on the ordering in the giant tour, is feasible. The length of the arc is the total length of the corresponding route. A solution is then obtained by solving a shortest-path problem from node 1 to node n in this graph. This can be done in polynomial time due to the acyclic nature of the graph.

4.3 Ruin and Recreate

Here, the GAs are inspired from the ruin and recreate paradigm, where a fraction of a solution is destroyed and then reconstructed in some alternate way [73]. For example, we can choose to remove q vertices from a solution to ruin it. Since each selection of q vertices is likely to lead to a different solution after reconstruction, the ruin and recreate principle can also be used to define a neighborhood structure for a local search heuristic. In this example, the size of the neighborhood grows quickly with q, thus leading to a large neighborhood search.

A good example of this approach is found in the work of Berger and his colleagues for the VRPTW. In [3], for example, an insertion-based crossover operator is proposed. First, a route r is probabilistically selected from the first parent solution, with a bias toward routes with large waiting times. A number of vertices are then removed from route r (ruin), based on criteria indicating that a more suitable relocation of these vertices into alternate routes is likely, due to a large distance to the immediate successor or a large waiting time. Vertices in a subset of routes from the second parent that are close to route r are then inserted, if possible, in the first parent solution. This procedure is repeated for a number of routes in the first parent solution. At the end, the offspring solution is completed with the remaining routes from the first parent. If there are unvisited vertices, they are handled by constructing additional routes with a nearest neighbor heuristic. A second child is obtained by interchanging the role of the two parents. Three mutation operators are also used to either reduce the number of routes or to reorder the vertices within the routes.

In a follow-up work [2], the authors relax the time constraints and evolve two populations with different objectives. Population 1, which contains at least one feasible solution, minimizes the total distance, while population 2 minimizes the time constraint violations, both subject to a fixed number of routes (i.e., m and $m-1$ routes, respectively). When a feasible solution with $m-1$ routes emerges in population 2, population 1 is replaced by population 2 and a special mutation operator is applied to population 2 to obtain a new population made of solutions with one fewer route. The number of routes is reduced in this way until no feasible solution emerges in population 2. A suite of five mutation operators is also proposed, including a large neighborhood search-based mutation that follows the guidelines in [74]. Here, related vertices (due to proximity or a common route assignment) are removed from the solution and reinserted with a variant of Solomon's I1 insertion heuristic [76].

A Sequence-Based crossover (SBX) is reported in [63] that merges the first part of a route in the first parent solution to the end part of another route in the second parent solution. The new route then replaces the old one in the first parent solution. In the process, vertices are likely to be duplicated or removed. Hence, a repair operator is applied to obtain a valid solution. First, each duplicate is eliminated by removing one of the two copies; then, vertices that were left apart are reinserted in the solution through a simple insertion heuristic. Other related operators where fragments of routes or whole routes are transferred to the offspring are reported in [61, 79, 80]. These operators are applied to a detailed multi-part solution representation, where each part is a string that encodes a route using the classical path representation.

4.4 Decoder

A decoder is a hybrid system where a GA is coupled with a construction heuristic. In a vehicle routing context, insertion heuristics are typically used to add vertices one by one into the current solution. The insertion order is specified by a string of vertices. For example, the string 4-2-3-6-5 means that vertex 4 is the first to be inserted in the solution, followed by vertices 2, 3, 6 and 5, in this order (note that vertex 1 is the depot and does not appear in the string). At any point, a new route can be created to visit a vertex if there is no feasible insertion place in the current set of routes, due to the side constraints. In the case of string 4-2-3-6-5, for example, an individual route is first created for vertex 4. Then, vertices 2 and 3 are inserted in this route. At this point, if we assume that there is no feasible insertion place for vertex 6, a new route is created for this vertex. Finally, vertex 5 is inserted in one of the two routes, depending on its best feasible insertion place. Clearly, different insertion orders are likely to lead to different routes. The GA thus searches for the ordering that will lead to the best solution. This problem-solving approach is attractive because the GA does not need to explicitly consider side constraints, since they are handled by the insertion heuristic. In a decoder system, specialized crossover and mutation operators are used to produce new offspring orderings from two parent orderings, like those developed for the TSP [62].

We illustrate this idea with a decoder system for the VRPTW reported in [5]. This GA uses two order-based crossover operators *MX1* and *MX2* to produce new insertion orders. Since it is generally desirable to insert vertex i before vertex j in a route if the time window at i occurs before the time window at j, the two operators are strongly biased by this time window-based precedence relationship. In Figure 7, we show how the *MX1* operator produces a valid ordering from two parent orderings.

This example is based on five vertices (apart from the depot 1), numbered from 2 to 6, and with the following time window precedence relationship $2 < 3 < 4 < 5 < 6$ (i.e., the time window at vertex 2 is the earliest and the time window at vertex 6 is the latest). Starting at position 1, the vertices on the two

(a)

parent 1: $\underline{\mathbf{2}}$ **6 3 4 5**
parent 2: $\underline{5}$ 4 2 3 6
offspring: **2** - - - -

(b)

parent 1: **2** $\underline{\mathbf{6}}$ **3 4 5**
parent 2: 2 $\underline{4}$ 5 3 6
offspring: **2** 4 - - -

(c)

parent 1: **2 4** $\underline{\mathbf{3}}$ **6 5**
parent 2: 2 4 $\underline{5}$ 3 6
offspring: **2** 4 **3** - -

(d)

parent 1: **2 4 3** $\underline{\mathbf{6}}$ **5**
parent 2: 2 4 3 $\underline{5}$ 6
offspring: **2** 4 **3** 5 -

(e)

parent 1: **2 4 3 5** $\underline{\mathbf{6}}$
parent 2: 2 4 3 5 $\underline{6}$
offspring: **2** 4 **3** 5 **6**

Fig. 7. MX1 crossover

parent orderings are compared and the vertex with the earliest time window is added to the offspring. Then, the procedure is repeated for the next position, until all positions are done. In Figure 7, vertices 2 and 5 are first compared. Since the time window at vertex 2 occurs before the time window at vertex 5, vertex 2 is selected and takes the first position in offspring (a). To produce a valid ordering, vertices 2 and 5 are then swapped in parent 2. Vertices 3 and 5 are then compared at position 2, and vertex 3 is selected to take the second position in offspring (b). Vertices 5 and 3 are then swapped in parent 1, etc. The resulting offspring is shown in (e). Note that *MX1* pushes vertices with early time windows to the front of the resulting ordering. These vertices are thus the first to be inserted in the solution by the insertion heuristic.

Specialized order-based mutation operators are also used in decoder systems to modify the orderings, for example by moving a vertex at another position or by exchanging the position of two vertices within the ordering. The generality of the decoder approach has led to its application to difficult vehicle routing problems with many side constraints. For example, a successful implementation for a vehicle routing problem with backhauls and time windows is described in [64]. In this work, solutions within 1% of the optimum (on average) have been produced on problem instances with up to 100 vertices.

Before closing this section, we need to mention some results obtained with Evolution Strategies (ES), an evolutionary approach developed in the 60's and 70's at the Technical University of Berlin by Rechenberg and Schwefel (for an introductory text on the subject, see [4]). Like genetic algorithms, ES evolve a population of solutions through mutation and recombination (crossover). One distinctive feature is the concurrent evolution of strategy parameters which are typically properties of the mutation operator, like a mutation step size. There is also a specific population replacement mechanism. For a population of size μ, either the μ best solutions are selected out of λ offspring solutions ($\mu < \lambda$), which is known as a (μ, λ)-selection, or out of the union of the parent and offspring solutions, which is a $(\mu + \lambda)$-selection. The latter scheme is elitist and guarantees a monotonically improving performance. A special case is the $(1 + 1)$-ES where there is no recombination operator, only a mutation operator.

Homberger and Gehring [35] were the first to apply ES to the VRPTW using a (μ, λ)-ES. The mutation operator is based on local search heuristics and the mutation step size is the number of modifications to the current solution. An additional strategy parameter associated with the mutation operator specifies the objective to be favored (either minimization of the total distance or minimization of the number of vehicles). Two evolution strategies, called *ES1* and *ES2* are proposed. In the simple variant *ES1*, there is no recombination, only mutation. Also, the two mutation strategy parameters are directly transferred to the offspring without any modification. This is to be opposed to the second variant *ES2*, where a crossover operator is applied to the mutation step sizes of the parents. In a follow-up paper [36], the authors describe a two-stage approach where $ES1$ is first applied with the objective of minimizing the number of vehicles, followed by a tabu search heuristic to minimize the total distance.

5 Ant Colony Optimization

In their search for food sources, ants initially look around in a random manner. When they find one, they come back to the nest, laying down an aromatic substance on the ground, known as pheromone. The amount of pheromone is related to the quality of the food source, as determined by the quantity of food and its distance from the nest. The next ants will thus search in a less random fashion, as they will be attracted by the pheromone trails. More ants will be attracted on paths with more pheromone, which in turn will lead to more and more pheromone laid down on these paths. Ultimately, all ants will be attracted by the best path.

Ant colony optimization is motivated from the way real ants find good paths to food sources, in particular the indirect communication scheme through pheromone trails that lead to this optimization, a phenomenon called stigmergy.

Different ant-based metaphors are reported in the literature, starting from the original Ant System (AS) [16, 18] to more recent variants, like the Ant Colony System (ACS) [17]. Although ant-based systems have first been tested on the TSP, many other combinatorial problems have since been addressed. In the following, the basic AS algorithm for solving the TSP [17] is described and some of its variants are briefly introduced (for a full account, the reader is referred to [19]). Applications for vehicle routing problems are then reviewed.

5.1 Ant System

We assume that we have m ants, where each ant is assigned to a starting location on a vertex of the graph, according to some a priori assignment procedure (which might be random). During one iteration of the method, each ant constructs a tour by repeatedly moving to a new vertex through the application of a probabilistic nearest neighbor transition rule. When all ants have completed their respective tour, the pheromone amount on each edge is modified to favor the edges that are associated with the best tours. This procedure is repeated until a stopping criterion is met, for example a fixed number of iterations.

A simple pseudo-code description for solving the TSP is given below.

```
1. Assign each ant to a starting vertex;
2. Put some initial amount of pheromone on every edge;
3. For $T$ iterations do:
   3.1 for each ant do: construct a tour;
   3.2 update the amount of pheromone on each edge;
4. Output the best tour found.
```

In Step 3.1, each ant constructs a complete tour through the repeated addition of a new vertex to the current partial tour. Assuming that ant k is currently located at vertex i, a probabilistic transition rule determines the next vertex j to be visited, namely:

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha} \ [\eta_{ij}]^{\beta}}{\sum_{l \in N^k} [\tau_{il}]^{\alpha} \ [\eta_{il}]^{\beta}} \quad \text{if } j \in N^k. \tag{1}$$

Among all admissible edges that lead from vertex i to a vertex in set N^k, which is the set of vertices not yet visited by ant k, the probability to select edge (i, j) and go to vertex j is derived from two values: the visibility (closeness) η_{ij} which is the inverse of c_{ij} and τ_{ij} which is the amount of pheromone on edge (i, j). The amount of pheromone on each edge is the memory of the system and depends on how many times that edge was used in the past to construct good tours. Thus, a short edge with a large amount of pheromone has a higher probability of being selected, since it was part of many good tours in previous iterations and it only slightly increases the total distance traveled. In equation (1), the relative influence of the pheromone and visibility information is adjusted via parameters α and β.

Once each ant has constructed a tour, the amount of pheromone is updated in Step 3.2 with the following equation:

$$\tau_{ij} \leftarrow (1-\rho)\ \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{2}$$

where $0 < \rho \leq 1$ is the pheromone persistence (or, alternatively, $(1-\rho)$ is the pheromone evaporation). This parameter is used to avoid unlimited accumulation of pheromone on the edges and favors the exploration of new paths. In this equation, $\Delta\tau_{ij}^{k}$ is the amount of pheromone deposited by ant k on edge (i,j). It is defined as:

$$\Delta\tau_{ij}^{k} = \begin{cases} 1/C^{k} & \text{if edge } (i,j) \text{ is visited by ant } k \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where C^{k} is the total distance traveled by ant k. Thus, a larger amount of pheromone is deposited if the tour is shorter.

A number of refinements and improvements have been proposed over the years to this basic search scheme. They are presented in the following.

5.2 Elitist Ant Systems

A first variant, called the Elitist AS (EAS), reinforces the arcs belonging to the best tour found since the start of the algorithm. Accordingly, equation (2) is modified as follows:

$$\tau_{ij} \leftarrow (1-\rho)\ \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + e\Delta\tau_{ij}^{best} \tag{4}$$

where e is a parameter and $\Delta\tau_{ij}^{best}$ is defined as

$$\Delta\tau_{ij}^{best} = \begin{cases} 1/C^{best} & \text{if edge } (i,j) \text{ is in the best tour} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

with C^{best} the length of the best tour since the start of the algorithm. We thus have e elitist ants that follow the best tour and deposit additional pheromone on it. An alternative approach is the rank-based AS, called AS$_{rank}$, where the ants are sorted from best to worst according to the length of their tour [8]. The $w-1$ best-ranked ants then deposit an amount of pheromone that is weighted by their rank. More precisely, equation (2) becomes

$$\tau_{ij} \leftarrow (1-\rho)\ \tau_{ij} + \sum_{k=1}^{w-1} (w-k)\ \Delta\tau_{ij}^{k} + w\Delta\tau_{ij}^{best} \tag{6}$$

where k is a rank (from 1 for the best-ranked ant to $w-1$). Note that an additional elitist ant follows the best tour found since the start of the algorithm. This ant deposits an amount of pheromone on the edges weighted by w.

Another approach is the Max-Min AS (MMAS) reported in [77]. Its main characteristic is the exclusive reinforcement of the best tour found at the current iteration or the best tour found since the start of the algorithm. To prevent convergence to suboptimal tours due to the accumulation of pheromone on the same edges, MMAS then forces the amount of pheromone on every edge to lie within an interval $[\tau_{min}, \tau_{max}]$, where τ_{min} and τ_{max} are parameters.

5.3 Ant Colony System

The Ant Colony System (ACS) reported in [17] is more aggressive than AS during the construction of a tour by focusing on the closest vertex when an ant must decide where to go next. That is, the edge leading to the closest vertex is considered first and with high probability. The other edges will only be considered according to equation (1) if this edge is not selected. Also, the best tour found since the start of the algorithm is strongly reinforced. In the so-called global update rule, which corresponds to equation (2) in the AS algorithm, pheromone is deposited or removed only on the edges of the best tour, thus reducing the complexity of this update by an order of magnitude. There is also a local update rule, which is applied each time an ant crosses a new edge during the construction of its tour. This update significantly reduces the amount of pheromone on this edge and makes it less attractive for the other ants, thus favoring the exploration of new paths.

It is worth noting that improvements are obtained by coupling ant-based systems with local search heuristics. Namely, the solutions constructed by the ants can be further improved through the application of local search heuristics. The pheromone update rules are then based on the value of the local minima.

5.4 Ant-Based Systems for Vehicle Routing

Given that vehicle routing problems are a natural extension of the TSP, it is not a surprise if a number of applications of ant-based systems are reported in the literature.

The first work was done in the late '90s on a VRP with maximum distance constraints [6]. In this work, the transition rule of the EAS in equation (4) is modified by forcing the ant to return to the depot and start a new route whenever the inclusion of the next vertex is infeasible due to capacity or maximum distance constraints. Further improvements are then proposed through more substantial modifications to the transition rule. To account for the location of the current vertex i and the next vertex j relative to the depot, a savings term s_{ij} is added to the transition rule. Clearly, the probability to go to vertex j is higher when the savings is larger. To take advantage of the available vehicle capacity, the term $\kappa_{ij} = (Q_i + q_j)/Q$ is also included in the transition rule, where Q_i is the capacity already used, including the demand of vertex i. As this value approaches 1, the vehicle capacity gets more fully exploited. The new transition rule is then:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha \ [\eta_{ij}]^\beta \ [s_{ij}]^\gamma \ [\kappa_{ij}]^\lambda}{\sum_{l \in N^k} [\tau_{il}]^\alpha \ [\eta_{il}]^\beta \ [s_{ij}]^\gamma \ [\kappa_{ij}]^\gamma} \quad \text{if } j \in N^k, \tag{7}$$

where γ and λ are additional parameters to weight the new terms.

In another work by the same authors [7], the visibility term η_{ij} in equation (1) is replaced by a parameterized savings term $s'_{ij} = c_{1i} + c_{1j} - g\ c_{ij} + f\ |c_{1i} - c_{1j}| = s_{ij} - (g-1)\ c_{ij} + f\ |c_{1i} - c_{1j}|$, where f and g are weighting parameters. The authors also replace the EAS framework by AS$_{rank}$. In both papers, a local search heuristic based on 2-opt arc exchanges is applied to every route in every solution constructed by the ants for further improvement. The results reported on the VRP instances of Christofides, Mingozzi and Toth [10] show that the proposed approach can generate solutions of good quality, but these solutions are not really competitive with state-of-the art metaheuristics based on tabu search.

The idea of incorporating a savings measure to guide the search is explored further in [68]. Here, the authors replace the classical nearest neighbor transition rule by a probabilistic variant of the savings heuristic (see Section 3.1). Here, the attractiveness measure is defined as:

$$\eta_{ij} = [\tau_{ij}]^{\alpha}\ [s_{ij}]^{\beta} \tag{8}$$

where s_{ij} is the savings measure. The largest attractiveness values are then stored in set Ω and the transition rule selects one of them with probability

$$p^k_{ij} = \begin{cases} \frac{\eta_{ij}}{\sum_{(h,l)\in\Omega_k} \eta_{hl}} & \text{if } \eta_{ij} \in \Omega \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

After merging the two routes associated with the selected attractiveness value, the number of routes is reduced by one and the procedure is repeated until no further reduction is possible. This probabilistic savings heuristic, embedded within the AS$_{rank}$ framework, is shown to provide significant improvements over the previous results obtained in [6, 7] on the benchmark instances of Christofides, Mingozzi and Toth. Furthermore, the results are now almost at par with the best state-of-the-art methods for the VRP. A further development of this algorithm in [66] is aimed at substantially reducing the computational effort by decomposing the problem into smaller subproblems made of subsets of geographically close routes and by solving each subproblem separately. The authors also consider a local search heuristic based on the swap of two vertices between two routes to further improve the solutions, in addition to the 2-opt arc exchanges. This approach allowed to authors to solve large-scale VRP instances with a few hundred vertices in reasonable computation times.

Ant-based systems have also been applied to the VRPTW. In [24], the proposed multiple ant colony system (MCAS) defines two ant colonies: ACS-VEI that minimizes the number of vehicles (main objective) and ACS-TIME that minimizes the total travel time (secondary objective). Both colonies use the standard ACS framework, except that during the construction of a solution each ant has the choice to go either to a vertex that does not violate the capacity and time window constraints or to return to the depot. ACS-VEI searches for a solution that maximizes the number of visited vertices with a fixed number of vehicles m. When a feasible solution that includes all vertices is obtained, ACS-TIME then constructs solutions with that number of vehicles, while trying to minimize the total travel time. Also, ACS-VEI is restarted with one fewer vehicle. This is repeated until the number of vehicles cannot be reduced anymore. This approach is very similar to the one used by the EA described in Section 4.3 and reported in [2]. The proposed local search heuristic for improving the solutions is based on CROSS exchanges, where two strings of consecutive vertices are exchanged between two routes [78]. Computational experiments on the 56 Euclidean VRPTW benchmark instances of Solomon [76] have produced some best known solutions and have demonstrated that the proposed ant-based system was competitive with the best methodologies at the time.

In [67], the nearest neighbor rule is replaced by a randomized variant of Solomon's $I1$ sequential insertion heuristic [76]. This approach is used to solve the VRPBTW, where the backhauls must follow the linehauls in each route. Other variants of the VRPBTW, where backhaul and linehaul vertices can be mixed, are studied in [69]. At each iteration of the construction process, the transition rule selects an unvisited vertex i and inserts it at a feasible place between two consecutive vertices j and j' in the current route. Since the procedure involves the replacement of edge (j, j') by two new edges (j, i) and (i, j'), the probability for ant k to select vertex $i \in N^k$ and to insert it between j and j' is:

$$p_i^k = \frac{\max_{j \in R_i} [\eta_{ij} \ \frac{\tau_{ji}+\tau_{ij'}}{2\tau_{jj'}}]}{\sum_{h \in N^k} \max_{l \in R_h} [\eta_{hl} \ \frac{\tau_{lh}+\tau_{hl'}}{2\tau_{ll'}}]}$$

where R_i is the set of vertices after which vertex i can be feasibly inserted and the attractiveness η_{ij} for inserting vertex i just after j is:

$$\eta_{ij} = \max\{0, \alpha c_{1i} - \beta (c_{ji} + c_{ij'} - c_{jj'}) - (1 - \beta) \left(b_{j'}^i - b_{j'}\right)\}$$

In this formula, $b_{j'}$ is the current service time at j' and $b_{j'}^i$ is the new service time after the insertion of vertex i between j and j'. Thus, the attractiveness of vertex i is higher when it is located far from the depot, and when it introduces a small detour and a small service delay in a route. The authors also introduce an additional term in the formula, weighted by parameter γ, to further differentiate between backhaul and linehaul vertices. The transition formula thus accounts for both the attractiveness of vertex i and the amount of pheromone on the edges. In particular, when $(\tau_{ji} + \tau_{ij'})/2\tau_{jj'} > 1$, the average amount of pheromone on the two new edges is larger than the amount of pheromone on the removed edge. This approach, embedded either within the AS$_{rank}$ or ACS framework, was shown to outperform some recent heuristics for the VRPBTW [84, 88] on a data set derived from Solomon's VRPTW instances [26], but it could not match the results of a large neighborhood search [72].

6 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is also a population-based approach where each solution, called a particle, is associated with an objective value and a velocity [40]. The latter defines a direction in the search space, as well as an amplitude, for modifying the solution and is determined through interaction with other solutions. The velocity of particle i at iteration t is obtained through a randomized weighted sum of three components: the velocity of particle i at iteration $t-1$, the velocity of the best solution achieved by particle i in the past and the velocity of the best solution achieved over all particles (or over particles that are sufficiently close to particle i, depending on the variant). Although PSO has mostly been used for solving continuous optimization problems, discrete variants also exist.

In [9], a binary version of PSO is used, where a solution or particle is a binary vector. The velocity vector is made of values between 0 and 1 and the corresponding solution is obtained by interpreting each velocity value as the probability that the corresponding element in the solution is 0 (and, conversely, as 1 minus the probability that the corresponding element is 1). This approach is exploited to solve the VRP. Basically, a particle is a vector of size $n \times m$, where n is the number of vertices and m is the number of vehicles. That is, the vector of vertices is duplicated m times, one time for each vehicle. An entry of 1 means that the vertex is visited by the corresponding vehicle. The PSO thus solves the assignment subproblem. As the solution obtained is not necessarily valid (i.e., exactly one position associated with a vertex should be equal to 1), a repair operator is used. If the solution is valid but infeasible, the solution is discarded and the velocity is recalculated until a feasible solution is obtained. A simulated annealing heuristic is then used to sequence the vertices in each route.

In [89], an alternative particle representation with only $n + m - 1$ elements is used for solving the VRPTW. Each entry corresponds to the position of one of the n vertices or one of the $m - 1$ copies of the depot in the solution vector. The PSO thus solves the assignment and sequencing problems concurrently. The objective is the total distance plus penalties for violations of the capacity and time constraints. The velocity is a vector of values between -$(n + m - 2)$ and $n + m - 2$ which is added to the solution vector to update it. Normalization is then applied to obtain new integer positions.

7 Neural Networks

Neural networks (NN) are composed of units that perform in a manner similar to neurons in the human brain. These units are richly interconnected through weighted connections: a signal is sent from one unit to another along a connection and is modulated by its associated weight (which stands for the strength or efficiency of the connection). Although superficially related to their biological counterpart, NNs posses characteristics associated with human cognition. In particular, they can learn from experience and induce general concepts from specific examples. When presented with a set of inputs, they self-adjust to produce the desired response.

Neural networks were originally designed for tasks well adapted to human intelligence and where traditional computation was inadequate, such as speech understanding, artificial vision and handwritten character recognition. Starting with the pioneering work of Hopfield and Tank [37] on the TSP, they have also been applied to combinatorial optimization problems. Although neural network models can handle spatial relationships among vertices, they cannot easily handle side constraints, such as capacity constraints or time windows, that prevent a pure spatial interpretation of the problem. Therefore, the literature on NNs applications to vehicle routing problems is scant, except for a special class of models, known as deformable templates, like the elastic net and the self-organizing map [28, 29, 50, 55, 75, 86]. The work of Ghaziri [28, 29] will be used in the following to illustrate how a deformable template, here a self-organizing map, can be applied to the standard VRP with capacity constraints.

7.1 Self-organizing Maps

Self-organizing maps [42] are composed of a layer of input units fully connected to a layer of output units, where the output units are organized into a particular topology, such as a ring. These models self-organize through an iterative adjustment of their connection weights to find some regularity or structure in the input data. They are typically used to categorize or cluster data. In Figure 8, T_{11} and T_{21} denote the weights on the connections from the two input units I_1 and I_2 to output unit 1, and $T_1 = (T_{11}, T_{21})$ is the weight vector associated with output unit 1.

Assuming an ordered set of n input vectors of dimension p and a self-organizing map with p input units and q output units on a ring, the basic algorithm for adjusting the connection weights is the following.

1. *Initialization.* Set each connection weight to some initial value.
2. *Competition.* Consider the next input vector I. If all input vectors are done, restart from the first input vector. Compute the output value o_j of each output unit j as the weighted sum of its inputs, that is, $o_j = \sum_{i=1}^{p} T_{ij}\, I_i$, $j = 1, ..., q$, where T_{ij} is the connection weight between input unit i and output unit j. The winning output unit $j*$ is the unit with maximum output value.
3. *Weight adjustment.* Modify the connection weights of each output unit by setting $T_j \leftarrow T_j + f(j, j*)\,(I - T_j)$, where $T_j = (T_{1j}, T_{2j}, ..., T_{pj})$ is the weight vector of output unit j, and f is a function of j and $j*$.
4. Repeat Steps 2 and 3, until the weight vectors stabilize. At the end, each input vector is assigned to the output unit that maximizes the weighted sum in Step 2.

In Step 3, f is typically a decreasing function of the lateral distance between units j and $j*$ on the ring (i.e., if there are k units on the ring between the two units, the lateral distance is $k + 1$) and its range is the interval [0,1]. Thus, the weight vector of the winning unit $j*$ and the weight vectors of units that are close to $j*$ on the ring all move toward the input vector I, but with decreasing intensity as the lateral distance to the winning unit increases. Typically, function f is modified as the learning algorithm unfolds to gradually reduce the magnitude of the weight adjustment. At the start, all units close to the winning unit on the

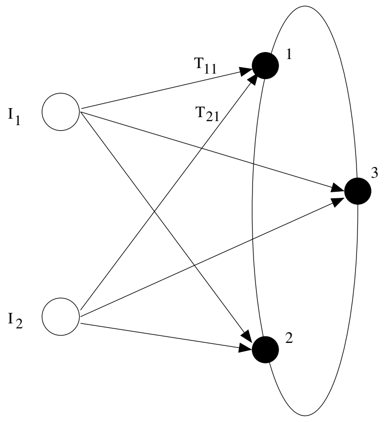


Fig. 8. Self-organizing map

ring follow that unit in order to cover a particular area. At the end, only the weight vector of the winning unit significantly moves toward the current input vector, to fix the assignment.

Self-organizing maps produce topological mappings from high-dimensional input spaces to low-dimensional output spaces. In the case of a ring, each p-dimensional input vector is associated with a 1-dimensional position on the ring. The mapping is such that two input vectors that are close in the input space are assigned to units that are close on the ring.

7.2 Self-organizing Maps for Vehicle Routing

In a VRP context, an input vector is made of the (x,y) coordinates of a vertex and a different ring is associated with each vehicle route. The learning algorithm is also slightly modified. Assuming an Euclidean distance metric, the winning output unit is the one with the closest weight vector to the coordinate vector of the current vertex (rather than the one with the largest correlation with the input vector, as obtained by maximizing the inner product, see Step 2 of the basic algorithm). Thus, the weight vector of any given output unit can be interpreted as the location of this unit in the Euclidean space. Accordingly, the procedure stops when there is a unit (weight vector) sufficiently close to each vertex. Given that multiple rings are used, the winning unit must be chosen among all units over all rings. Then, the units located on the ring of the winning unit move toward the current vertex. At the end of this procedure, each vertex is assigned to its closest unit. Through this assignment, the ordering of the units on each ring defines an ordering of the vertices on each route. Furthermore, since close units on a given ring tend to migrate toward neighboring vertices in the Euclidean plane, short routes are expected to emerge. Figure 9 depicts the evolution of the rings in a two-dimensional Euclidean plane, starting from the initial configuration illustrated in Figure 9(a).

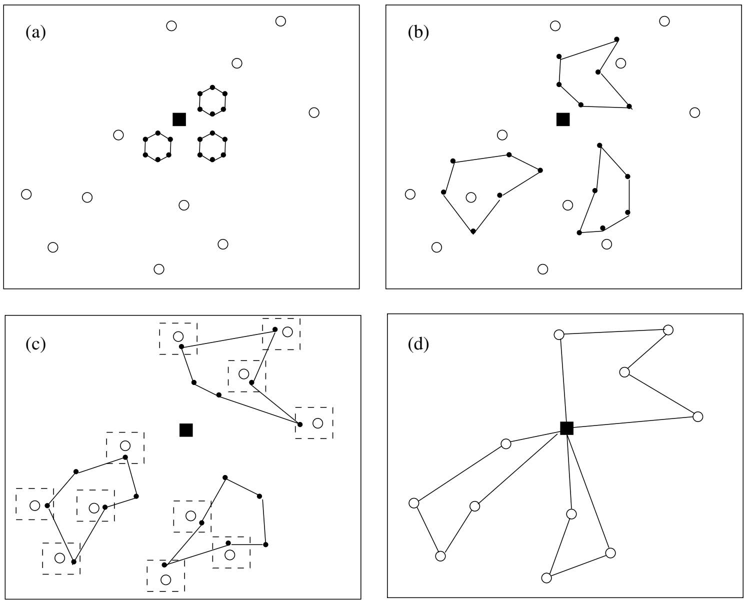


Fig. 9. Ring migration

In this figure, the vertices are the white nodes and the units are the small black nodes. Note that the location of each unit corresponds to its current weight vector. At the start, no solution of the problem can be inferred. However, as the weight vectors migrate toward the vertices, the solution progressively emerges. The final solution is determined at the end, during the final assignment of each vertex to a unit.

This type of approach is now illustrated with the so-called Hierarchical Deformable Net [29]. The model involves a competition at two different levels, namely, a deterministic competition among the units within each ring, and a stochastic competition among the rings. The number of rings is fixed, but the model is applied with an increasing number of rings (routes) until a feasible solution is obtained. In the following, n is the number of vertices, m is the number of rings, $d(x, y)$ is the Euclidean distance between the two-dimensional vectors x and y and $dL(j, l, k)$ is the lateral distance between units j and l on ring r^k, $k = 1, ..., m$.

1. Randomly order the vertices and create one ring per route. Each ring contains $2 \lceil (n/m) \rceil$ units. Set $h \leftarrow 1$ and $G \leftarrow 4 \lceil (n/m) \rceil$.
2. Repeat Step 3 to Step 8 until there is a weight vector sufficiently close to each vertex.
3. Select the next vertex i and set the current input vector I to its coordinates (if all vertices are done, restart with the first vertex).

4. Competition within each ring. Determine the winning unit o^k_{j*} on each ring r^k. This unit is the one with the closest weight vector to the current input vector I.
5. Competition among the rings. Assign the following winning probability to each ring r^k:
$$p\left(r^k\right) = \frac{f\left(r^k, h\right)}{\sum_{k'=1,\ldots,m} f\left(r^{k'}, h\right)} \tag{10}$$
$$f\left(r^k, h\right) = \frac{e^{-d\left(T^k_{j*}, I\right)/h}}{1 + e^{-\Delta Q/h}} \tag{11}$$
where ΔQ is the difference between the vehicle capacity and the total demand on ring r^k (with current vertex i).
6. Select the winning ring r^{k*}, using a selection probability for each ring r^k proportional to $p\left(r^k\right)$. Vertex i is assigned to unit o^{k*}_{j*}.
7. Move the weight vector T^{k*}_j of each unit j on the winning ring r^{k*} toward the coordinate vector I of vertex i:
$$T^{k*}_j \leftarrow T^{k*}_j + f\left(j, j*, k*\right)\left(I - T^{k*}_j\right) \tag{12}$$
$$f\left(j, j*, k*\right) = \frac{e^{-dL(j,j*,k*)^2/G^2}}{\sqrt{2}} \tag{13}$$
8. Slightly decrease the value of parameters h and G, by setting $h \leftarrow 0.99\ h$ and $G \leftarrow 0.9\ G$.

In Step 5, parameter h is used to modify the probability distribution over the rings. When the value of h is high, the selection probability of each ring is about the same. When the value of h is small, the probability distribution favors the ring that is closest to the current vertex among all rings that satisfy the capacity constraint (with the current vertex). In Step 7, parameter G controls the magnitude of the moves. When its value is reduced, the magnitude is also reduced. Hence, the values of both parameters h and G are gradually lowered to favor convergence toward good feasible solutions.

Ghaziri has applied this model to a few VRPs taken from the data set of Christofides, Mingozzi and Toth [10]. Although the solutions produced were of relatively good quality, they were not competitive with the best solutions produced with alternate problem-solving methods, like tabu search. In a later study [28], this model is extended to solve the VRP with maximum route time constraints. In this case, the selection probabilities are adjusted through an additional component that accounts for these constraints. As opposed to capacity constraints, however, the satisfaction of maximum route time constraints depends on the sequence of vertices within each route. At each iteration, a TSP procedure is thus applied to the currently assigned vertices on each ring (including the current vertex) to identify a plausible sequence. This sequence is then used to compute the selection probabilities.

Neural networks have not led to a steady stream of research for solving combinatorial optimization problems. In the case of vehicle routing, we do not know of any recent significant contribution in this field, with the exception of [30] where a self-organizing map for the VRPB is proposed. This approach is shown to be competitive with the best know methods on small sized instances, but its performance tend to deteriorate with increasing sizes.

8 Artificial Immune Systems

AIS is a recent bio-inspired algorithm motivated from the mechanisms used by our organism to defend itself against foreign microorganisms, like viruses and bacteria [13, 15]. In the natural immune system, molecular patterns at the surface of microorganism, known as antigens, lead to the proliferation of lymphocytes that are able to produce antigen-specific antibodies. This phenomenon is known as clonal selection (or clonal expansion). Since each lymphocyte has a distinct antigen specificity and the organism needs to react to different antigens, the number of lymphocytes that can differentiate into effector cells able to produce antibodies for each specific antigen is limited. Thus, it is primordial for the organism to develop a good repertoire of lymphocytes with different specificities. Furthermore, the differentiating process is not of the "all-or-nothing" type, so that antibodies typically show a variable degree of antigen affinity or specificity.

In AIS developed for optimization purposes, the antigen affinity corresponds to the solution value. Then, different mechanisms observed in the immune system, like the clonal selection mechanism, are exploited to generate good solutions. For example, artificial clonal selection involves the selection of solutions with the best objective values (high affinity), the reproduction of these solutions to obtain a number of copies or clones and mutation of these clones to generate improved solutions. As opposed to classical genetic algorithms, where mutation is a random process which is applied at low rates, the mutation operator in AIS is controlled and its rate is inversely proportional to the solution value. It is also typically applied at high rates (hypermutation) to increase population diversity. These AIS principles are often integrated within GAs to allow a better management of the population of solutions.

This is the case for example in [39], where an additional operator motivated from AIS is applied after crossover to allow the proliferation of high quality solutions in the population. This approach is applied for solving an electricity distribution network problem. The problem is modeled as a multiple depot vehicle routing problem, where the depots correspond to high/medium voltage stations and the other vertices to medium/low voltage stations. In [49], an immune-based operator is used within a GA to solve the VRPTW. Basically, the operator is able to recognize good patterns in solutions, such as subpaths that link close vertices. It then uses this information to improve solutions. For example, if vertex i is close to vertex j and the two vertices are not in the same route or are in the same route but are not consecutive, then this operator will put j just after i if an improvement is obtained.

Since the field of AIS is still in its infancy, there is only a few work related to combinatorial optimization and vehicle routing. However, this line of research should expand in the next few years with the availability of a number of paradigms to model immunological processes, like the immune network models [23].

9 Hybrids

An increasing number of bio-inspired algorithms do not rely on a single search strategy but rather combine various algorithmic ideas. These approaches are referred to as hybrids. The motivation behind hybrids is usually to exploit the strengths of a number of individual search strategies to obtain a more powerful problem-solving approach. We have already mentioned some hybrid algorithms in the preceding sections. For example, a decoder system can be viewed as the hybrid of a GA and a construction heuristic. The integration of local search-based mutation operators within GAs, known as memetic algorithms [56], is another type of hybrid. Likewise, some AIS concepts have been integrated within GAs to make them more robust.

In this regard, two recent developments for solving the VRP and VRPTW need to be mentioned. In [46, 47], different tabu searches and GAs are used to explore the search space concurrently. Their interaction is realized through a central warehouse, which is basically an adaptive memory made of elite solutions [71]. The diversity of solutions in the warehouse is exploited when new starting solutions are provided to the tabu searches and GAs. With this approach, the authors have found solutions that are competitive with state-of-the-art methods on Solomon's VRPTW instances [76], as well as on the extended data set of Gehring and Homberger [25].

In [51], an algorithm called AGES combines guided local search (GLS) [87] and ES in an iterative two-stage procedure to address both the VRP and VRPTW. GLS introduces modifications into the objective function through penalties when the search gets trapped in a local optimum. Here, the penalty counter of one arc in the solution is incremented by one, each time a local optimum is reached. The arc is selected on the basis of its length (a long arc is more likely to be penalized) and current penalty counter (an arc with a high penalty counter is less likely to be penalized again). In the first stage, GLS controls the objective function of a composite local search that exploits different types of modifications to the solution, while in the second stage, it controls the objective function of a (1 + 1)-ES. In this ES, the ruin and recreate principle is at the core of the mutation operator. That is, a number of vertices are removed from the current solution and reinserted at least cost. Refinements to this algorithm are proposed in [52, 53] with excellent results reported on classical VRP and VRPTW benchmark instances. On the VRP data sets of Christofides, Mingozzi and Toth [10] and Golden et al. [33], in particular, this approach has produced the best average solution values when compared with state-of-the-art methods.

10 Conclusion

This chapter has reviewed the main bio-inspired algorithms that have been used to solve vehicle routing problems. From this review, it is clear that evolutionary algorithms and their hybrids have shown the best performance up to now. They have been competitive and have even outperformed widely used metaheuristics, like tabu search, on some classical problems like the VRP and VRPTW. At the other end of the spectrum, neural networks have been more or less abandoned due to their deceiving performance on standard computer platforms. It is possible that the availability of specialized hardware dedicated to neural network implementations will allow researchers to exploit their full power, thus leading to their resurgence in vehicle routing studies. In-between stand ant colony optimization with some interesting results. Finally, particle swarm optimization and artificial immune systems are not yet mature fields of research. Consequently, only scarce results are reported up to now, but these approaches certainly show great promises for the future.

References

1. Ahuja, R.K., Ergun, O., Orlin, J.B., Punnen, A.P.: A survey of very large-scale neighborhood search techniques. Discrete Applied Mathematics 123, 75–102 (2002)
2. Berger, J., Barkaoui, M., Bräysy, O.: A route-directed hybrid genetic approach for the vehicle routing problem with time windows. INFOR 41, 179–194 (2003)
3. Berger, J., Salois, M., Bégin, R.: A hybrid genetic algorithm for the vehicle routing problem with time windows. In: Mercer, R.E. (ed.) Canadian AI 1998. LNCS, vol. 1418, pp. 114–127. Springer, Heidelberg (1998)
4. Beyer, H.-G., Schwefel, H.-P.: Evolution strategies: A comprehensive introduction. Natural Computing 1, 3–52 (2002)
5. Blanton, J.L., Wainwright, R.L.: Multiple vehicle routing with time and capacity constraints using genetic algorithms. In: Forrest, S. (ed.) Proceedings of the 5th International Conference on Genetic Algorithms, pp. 451–459. Morgan Kaufmann, San Mateo (1993)
6. Bullnheimer, B., Hartl, R.F., Strauss, C.: Applying the ant system to the vehicle routing problem. In: Voss, S., Martello, S., Osman, I.H., Roucairol, C. (eds.) Metaheuristics: Advances and trends in local search paradigms for optimization, pp. 285–296. Kluwer, Boston (1999)
7. Bullnheimer, B., Hartl, R.F., Strauss, C.: An improved ant system algorithm for the vehicle routing problem. Annals of Operations Research 89, 319–328 (1999)
8. Bullnheimer, B., Hartl, R.F., Strauss, C.: A new rank-based version of the ant system: A computational study. Central European Journal for Operations Research and Economics 7, 25–38 (1999)
9. Chen, A.-L., Yang, G.K., Wu, Z.M.: Hybrid discrete particle swarm optimization algorithm for capacitated vehicle routing problem. Journal of Zhejiang University SCIENCE A 7, 607–614 (2006)
10. Christofides, N., Mingozzi, A., Toth, P.: The vehicle routing problem. In: Christofides, N., Mingozzi, A., Toth, P., Sandi, C. (eds.) Combinatorial Optimization, pp. 315–338. Wiley, Chichester (1979)

11. Clarke, G., Wright, J.: Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 12, 568–581 (1964)
12. Cordeau, J.-F., Laporte, G., Mercier, A.: A unified tabu search heuristic for vehicle routing problems with time windows. Journal of the Operational Research Society 52, 928–936 (2001)
13. Dasgupta, D. (ed.): Artificial immune systems and their applications. Springer, Berlin (1999)
14. de Castro, L.N.: Fundamentals of natural computing: Basic concepts, algorithms, and applications. CRC Press, Boca Raton (2006)
15. de Castro, L.N., Timmis, J.: Artificial immune systems: A new computational intelligence approach. Springer, London (2002)
16. Dorigo, M.: Optimization, learning and natural algorithms. Ph.D. Dissertation, Politecnico di Milano, Italy (in Italian) (1992)
17. Dorigo, M., Gambardella, L.M.: Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 1, 53–66 (1997)
18. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man and Cybernetics 26, 29–41 (1996)
19. Dorigo, M., Stützle, T.: Ant colony optimization. MIT Press, Cambridge (2004)
20. Dror, M. (ed.): Arc routing: Theory, Solutions and Approximations. Springer, Berlin (2000)
21. Eiselt, H.A., Gendreau, M., Laporte, G.: Arc routing problems, Part 1: The chinese postman problem. Operations Research 43, 231–242 (1995)
22. Eiselt, H.A., Gendreau, M., Laporte, G.: Arc routing problems, Part 2: The rural postman problem. Operations Research 43, 399–414 (1995)
23. Farmer, J.D., Kauffman, S., Packard, N.H., Perelson, A.S.: Adaptive dynamic networks as models for the immune system and autocatalytic sets. Annals of the New York Academy of Sciences 504, 118–131 (1987)
24. Gambardella, L.M., Taillard, É.D., Agazzi, G.: MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In: Corne, D., Dorigo, M., Glover, F. (eds.) New ideas in optimization, pp. 63–76. McGraw-Hill, London (1999)
25. Gehring, H., Homberger, J.: A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In: Miettinen, K., Mäkelä, M.M., Toivanen, J. (eds.) Proceedings of EUROGEN 1999 - Short course on evolutionary algorithms in engineering and computer science, Reports of the Department of Mathematical Information Technology, No. A 2/1999. University of Jyväskylä, Finland, pp. 57–64 (1999)
26. Gelinas, S., Desrochers, M., Desrosiers, J., Solomon, M.M.: A new branching strategy for time constrained routing problems with application to backhauling. Annals of Operations Research 61, 91–109 (1995)
27. Gendreau, M., Laporte, G., Potvin, J.-Y.: Metaheuristics for the Capacitated VRP. In: Toth, P., Vigo, D. (eds.) The vehicle routing problem, pp. 129–154. SIAM, Philadelphia (2002)
28. Ghaziri, H.: Supervision in the self-organizing feature map: Application to the vehicle routing problem. In: Osman, I.H., Kelly, J.P. (eds.) Meta-heuristics: Theory & applications, pp. 651–660. Kluwer, Boston (1996)
29. Ghaziri, H.: Solving routing problems by a self-organizing map. In: Kohonen, T., Makisara, K., Simula, O., Kangas, J. (eds.) Artificial neural networks, pp. 829–834. North-Holland, Amsterdam (1991)

30. Ghaziri, H., Osman, I.H.: Self-organizing feature maps for the vehicle routing problem with backhauls. Journal of Scheduling 9, 97–114 (2006)
31. Gillett, B., Miller, L.: A Heuristic Algorithm for the Vehicle Dispatch Problem. Operations Research 22, 340–379 (1974)
32. Glover, F., Laguna, M.: Tabu search. Kluwer, Boston (1997)
33. Golden, B.L., Wasil, E.A., Kelly, J.P., Chao, I.-M.: The impact of metaheuristics on solving the vehicle routing problem: Algorithms, problem sets and computational results. In: Crainic, T.G., Laporte, G. (eds.) Fleet Management and Logistics, pp. 33–56. Kluwer, Norwell (1998)
34. Holland, J.H.: Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor (1975); (reprinted in 1992 by The MIT Press, Cambridge, MA)
35. Homberger, J., Gehring, H.: Two evolutionary metaheuristics for the vehicle routing problem with time windows. INFOR 37, 297–318 (1999)
36. Homberger, J., Gehring, H.: A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. European Journal of Operational Research 162, 220–238 (2005)
37. Hopfield, J.J., Tank, D.W.: Neural computation of decisions in optimization problems. Biological Cybernetics 52, 141–152 (1985)
38. Jung, S., Moon, B.-R.: A hybrid genetic algorithm for the vehicle routing problem with time windows. In: Langdon, W.B., et al. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1309–1316. Morgan Kaufmann, San Francisco (2002)
39. Keko, H., Skok, M., Skrlec, D.: Solving the distribution network routing problem with artificial immune systems. In: Proceedings of the IEEE Mediterranean Electrotechnical Conference. Dubrovnik, Croatia, pp. 959–962 (2004)
40. Kennedy, J., Eberhart, R.C.: Swarm intelligence. Morgan Kaufmann, San Francisco (2001)
41. Kirkpatrick, S., Jr Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220, 671–680 (1983)
42. Kohonen, T.: Self-organization and associative memory. Springer, Berlin (1988)
43. Laporte, G.: The vehicle routing problem: An overview of exact and approximate algorithms. European Journal of Operational Research 59, 345–358 (1992)
44. Laporte, G.: Vehicle routing. In: Dell'Amico, M., Maffioli, F., Martello, S. (eds.) Annotated bibliographies in combinatorial optimization, pp. 223–240. Wiley, Chichester (1997)
45. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Schmoys, D.B. (eds.): The traveling salesman problem. Wiley, Chichester (1985)
46. Le Bouthillier, A., Crainic, T.G.: A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. Computers & Operations Research 32, 1685–1708 (2005)
47. Le Bouthillier, A., Crainic, T.G., Kropf, P.: A guided cooperative search for the vehicle routing problem with time windows. IEEE Intelligent Systems 20, 36–42 (2005)
48. Lin, S.: Computer solutions of the traveling salesman problem. Bell System Technical Journal 44, 2245–2269 (1965)
49. Ma, J., Zou, H., Gao, L.-Q., Li, D.: Immune genetic algorithm for vehicle routing problem with time windows. In: Proceedings of the Fifth International Conference on Machine Learning and Cybernetics. Dalian, China, pp. 3465–3469 (2006)

50. Matsuyama, Y.: Self-organization via competition, cooperation and categorization applied to extended vehicle routing problems. In: Proceedings of the International Joint Conference on Neural Networks. Seattle, WA I–385–390 (1991)
51. Mester, D., Bräysy, O.: Active guided evolution strategies for large-scale vehicle routing problems with time windows. Computers & Operations Research 32, 1593–1614 (2005)
52. Mester, D., Bräysy, O.: Active guided evolution strategies for large-scale capacitated vehicle routing problems. Computers & Operations Research 34, 2964–2975 (2007)
53. Mester, D., Bräysy, O., Dullaert, W.: A multi-parametric evolution strategies algorithm for vehicle routing problems. Expert Systems with Applications 32, 508–517 (2007)
54. Mladenović, N., Hansen, P.: Variable neighborhood search. Computers & Operations Research 24, 1097–1100 (1997)
55. Modares, A., Somhom, S., Enkawa, T.: A self-organizing neural network approach for multiple traveling salesman and vehicle routing problems. International Transactions in Operations Research 6, 591–606 (1999)
56. Moscato, P., Cotta, C.: A gentle introduction to memetic algorithms. In: Glover, F., Kochenberger, G.A. (eds.) Handbook of Metaheuristics, pp. 105–144. Kluwer, Boston (2003)
57. Nagata, Y.: Edge assembly crossover for the capacitated vehicle routing problem. In: Cotta, C., van Hemert, J. (eds.) EvoCOP 2007. LNCS, vol. 4446, pp. 142–153. Springer, Heidelberg (2007)
58. Nagata, Y.: Effective memetic algorithm for the vehicle routing problem with time windows: Edge assembly crossover for the VRPTW. In: Proceedings of the Seventh Metaheuristics International Conference, Montreal, Canada (2007) (on CD-ROM)
59. Oliver, I.M., Smith, D.J., Holland, J.R.C.: A study of permutation crossover operators on the traveling salesman problem. In: Grefenstette, J.J. (ed.) Proceedings of the Second International Conference on Genetic Algorithms and Their Applications, pp. 224–230. Lawrence Erlbaum Associates, Hillsdale (1987)
60. Osman, I.H.: Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Annals of Operations Research 41, 421–451 (1993)
61. Pereira, F.B., Tavares, J., Machado, P., Costa, E.: GVR: A new genetic representation for the vehicle routing problem. In: O'Neill, M., Sutcliffe, R.F.E., Ryan, C., Eaton, M., Griffith, N.J.L. (eds.) AICS 2002. LNCS (LNAI), vol. 2464, pp. 95–102. Springer, Heidelberg (2002)
62. Potvin, J.-Y.: Genetic algorithms for the traveling salesman problem. Annals of Operations Research 63, 339–370 (1996)
63. Potvin, J.-Y., Bengio, S.: The vehicle routing problem with time windows - Part II: Genetic search. INFORMS Journal on Computing 8, 165–172 (1996)
64. Potvin, J.-Y., Duhamel, C., Guertin, F.: A genetic algorithm for vehicle routing with backhauling. Applied Intelligence 6, 345–355 (1996)
65. Prins, C.: A simple and effective evolutionary algorithm for the vehicle routing problem. Computers & Operations Research 31, 1985–2002 (2004)
66. Reimann, M., Doerner, K., Hartl, R.F.: D-Ants: Savings based ants divide and conquer the vehicle routing problem. Computers & Operations Research 31, 563–591 (2004)

67. Reimann, M., Doerner, K., Hartl, R.F.: Insertion based ants for vehicle routing problems with backhauls and time windows. In: Dorigo, M., Di Caro, G., Sampels, M. (eds.) Ant Algorithms 2002. LNCS, vol. 2463, pp. 135–147. Springer, Heidelberg (2002)
68. Reimann, M., Stummer, M., Doerner, K.: A savings-based ant system for the vehicle routing problem. In: Langdon, W.B., et al. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1317–1325. Morgan Kaufmann, San Francisco (2002)
69. Reimann, M., Ulrich, H.: Comparing backhauling strategies in vehicle routing using ant colony optimization. Central European Journal of Operations Research 14, 105–123 (2006)
70. Reinelt, G.: TSPLIB - A traveling salesman problem library. ORSA Journal on Computing 3, 376–384 (1991)
71. Rochat, Y., Taillard, É.D.: Probabilistic diversification and intensification in local search for vehicle routing. Journal of Heuristics 1, 147–167 (1995)
72. Ropke, S., Pisinger, D.: A unified heuristic for a large class of vehicle routing problems with backhauls. European Journal of Operational Research 171, 750–775 (2006)
73. Schrimpf, G., Schneider, J., Stamm-Wilmbrandt, H., Duek, G.: Record breaking optimization results using the ruin and recreate principle. Journal of Computational Physics 159, 139–171 (2000)
74. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M., Puget, J.-F. (eds.) CP 1998. LNCS, vol. 1520, pp. 417–431. Springer, Heidelberg (1998)
75. Schumann, M., Retzko, R.: Self-organizing maps for vehicle routing problems: Minimizing an explicit cost function. In: Fogelman-Soulie, F. (ed.) Proceedings of the International Conference on Artificial Neural Networks, EC2&Cie, Paris, pp. II–401–406 (1995)
76. Solomon, M.M.: Time window constrained routing and scheduling problems. Operations Research 35, 254–265 (1987)
77. Stützle, T., Hoos, H.H.: MAX-MIN ant system. Future Generation Computer Systems Journal 16, 889–914 (2000)
78. Taillard, É.D., Badeau, P., Gendreau, M., Guertin, F., Potvin, J.-Y.: A tabu search heuristic for the vehicle routing problem with soft time windows. Transportation Science 31, 170–186 (1997)
79. Tavares, J., Pereira, F.B., Machado, P., Costa, E.: GVR delivers it on time. In: Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning, Singapore, pp. 745–749 (2002)
80. Tavares, J., Pereira, F.B., Machado, P., Costa, E.: On the influence of GVR in vehicle routing. In: Proceedings of the ACM Symposium on Applied Computing, Melbourne, FL, pp. 753–758 (2003)
81. Thangiah, S.R.: An adaptive clustering method using a geometric shape for vehicle routing problems with time windows. In: Eshelman, L.J. (ed.) Proceedings of the 6th International Conference on Genetic Algorithms, pp. 536–543. Morgan Kaufmann, San Francisco (1995)
82. Thangiah, S.R., Nygard, K.E., Juell, P.L.: GIDEON: A genetic algorithm system for vehicle routing with time windows. In: Proceedings of 7th IEEE Conference on Artificial Intelligence Applications, pp. 322–328. IEEE Computer Society Press, Los Alamitos (1991)
83. Thangiah, S.R., Salhi, S.: Genetic clustering: An adaptive heuristic for the multi-depot vehicle routing problem. Applied Artificial Intelligence 15, 361–383 (2001)

84. Thangiah, S.R., Potvin, J.-Y., Sun, T.: Heuristic approaches to vehicle routing with backhauls and time windows. Computers & Operations Research 23, 1043–1057 (1996)
85. Toth, P., Vigo, D. (eds.): The vehicle routing problem. SIAM, Philadelphia (2001)
86. Vakhutinsky, A.I., Golden, B.L.: Solving vehicle routing problems using elastic net. In: Proceedings of the IEEE International Conference on Neural Networks, Piscataway, NJ, pp. 4535–4540. IEEE Press, Los Alamitos (1994)
87. Voudouris, C., Tsang, E.P.K.: Guided local search. Technical Report CSM-247, Department of Computer Science, University of Essex, Colchester, UK (1995)
88. Zhong, Y., Cole, M.H.: A vehicle routing problem with backhauls and time windows: A guided local search solution. Transportation Research E 41, 131–144 (2005)
89. Zhu, Q., Qian, L., Li, Y., Zhu, S.: An improved particle swarm optimization algorithm for vehicle routing problem with time windows. In: Proceedings of the IEEE Congress on Evolutionary Computation, Piscataway, NJ, pp. 1386–1390. IEEE Press, Los Alamitos (2006)

A GRASP × Evolutionary Local Search Hybrid for the Vehicle Routing Problem

Christian Prins

Institute Charles Delaunay, University of Technology of Troyes (UTT)
BP 2060, 10010 Troyes Cedex, France
christian.prins@utt.fr

Summary. This chapter proposes new VRP heuristics based on Iterated Local Search (ILS): a pure ILS, a version with several offspring solutions per generation, called Evolutionary Local Search or ELS, and hybrid forms GRASP×ILS and GRASP×ELS. These variants share three main features: a simple structure, an alternation between solutions encoded as giant tours and VRP solutions, and a fast local search based on a sequential decomposition of moves. The proposed methods are tested on the Christofides et al. (1979) and Golden et al. (1998) instances. Our best algorithm is the GRASP×ELS hybrid. On the first set, if only one run with the same parameters is allowed, it outperforms all recent heuristics except the AGES algorithm of Mester and Bräysy (2007). Only AGES and the SEPAS method of Tarantilis (2005) do better on the second set, but GRASP×ELS improves two best-known solutions. Our algorithm is also faster than most VRP metaheuristics.

1 Introduction

The Vehicle Routing Problem (VRP) is probably the most famous node routing problem with capacitated vehicles. It is usually defined on a complete undirected network $G = (V, E)$ with a node set $V = \{0, 1, \ldots, n\}$ and an edge set E. Node 0 is a depot with K identical vehicles of capacity Q, K can be fixed a priori or left as a decision variable. Each other node i represents a customer with a demand q_i and each edge (i, j) has a traversal cost $c_{ij} = c_{ji}$.

The VRP consists in determining a set of K vehicle trips of minimum total cost, such that each trip starts and ends at the depot, each client is visited exactly once, and the total demand handled by any vehicle does not exceed Q. Most sets of classical instances include a few *distance-constrained VRP instances* (DVRP): each customer has a service cost s_i and the total cost of each trip (travel costs plus service costs) may not exceed a fixed limit L.

The VRP and DVRP are NP-hard. Since the book by Toth and Vigo [26], significant advances have been achieved for both exact and heuristic algorithms. The most effective exact method is a recent branch-and-cut-and-price code, able to solve problems with up to 100 customers [11]. However, the development of heuristics is essential because many real instances are much larger.

Classical heuristics are useful to quickly compute solutions to large instances or initialize metaheuristics. The first one is the famous merge heuristic published in 1964 by

Clarke and Wright [4]. The main types are savings-based heuristics, two-phase methods based on the cluster-first route-second or route-first cluster-second principles, and matching procedures. All these methods are summarized in a survey by Laporte et al. [16].

In a recent survey [5], Cordeau et al. compare ten VRP metaheuristics published after 2000 on two sets of classical benchmarks: it appears that tabu search methods are now superseded by population-based methods combined with local search. Indeed, for the second set with large instances (Golden et al. [13], 200 to 483 customers), the only four methods within 1% to best-known solutions are the Active Guided Evolution Strategy (AGES) of Mester and Bräysy [19], the D-Ants algorithm of Reimann et al. [22], the memetic algorithm (MA) of Prins [21] and the Bone Route adaptive memory-based procedure of Tarantilis and Kiranoudis [25]. Since this survey, Tarantilis [24] and Pisinger and Röpke [20] have added two new heuristics to this 1% club.

The aim of this book chapter is to study even simpler metaheuristics, based on Iterated Local Search (ILS). Section 2 describes the principles of ILS in general and extensions based on the generation of several offspring solutions per iteration (Evolutionary Local Search or ELS) or on hybridization with GRASP (Greedy Randomized Adaptive Search Procedure). Specific components to tackle the VRP and DVRP are detailed in Section 3. A computational evaluation is presented in Section 4, before the conclusion.

2 Principles of GRASP, ILS, ELS and Their Hybrids

2.1 Common Notation

The purpose of this section is to recall the principles of GRASP and ILS, to introduce an evolved version of ILS, called Evolutionary Local Search or ELS, and to show how GRASP can be hybridized with ILS or ELS. In the sequel, we consider a minimization problem and use a common notation for all algorithms. S and $f(S)$ represent a solution and its cost. H is a deterministic greedy heuristic, *HR* a greedy randomized heuristic and *Mutate* a random mutation procedure like in genetic algorithms.

LS is a local search procedure. S^* and $f^* = f(S^*)$ respectively denote the best solution found by a metaheuristic and its cost. Concerning parameters, *np* denotes the number of phases (each phase using one initial local optimum), *ni* the number of iterations per phase (number of attempts to produce better local optima) and *nc* the number of children solutions generated in each iteration. The total number of calls to the local search is $ncls = np \times ni \times nc$.

2.2 Greedy Randomized Adaptive Search Procedure (GRASP)

GRASP is a simple metaheuristic used for the first time by Feo and Bard [9] and then popularized by Feo and Resende [10]. It can be thought as a multi-start local search, in which each initial solution is generated using one greedy randomized heuristic and improved by local search. The heuristic builds a solution one element at a time. To determine the next element to be added to a partial solution, the incremental cost is evaluated for each candidate element, using a greedy function. The candidates are sorted in increasing cost order and one of them is randomly selected in a restricted candidate list (RCL).

For instance, consider the *nearest neighbour heuristic* for the Travelling Salesman Problem (TSP). In the deterministic version, each iteration adds the nearest unrouted node at the end of the emerging route. To get a randomized version, one can sort the unrouted customers in increasing distance to the last node i of the partial route and draw the next customer among the k first ones.

Algorithm 1. GRASP general structure

```
1: initialize random number generator
2: $f^* := +\infty$
3: for $i := 1$ to $np$ do
4:     $HR(S)$
5:     $LS(S)$
6:     if $f(S) < f^*$ then
7:         $f^* := f(S)$
8:         $S^* := S$
9:     end if
10: end for
```

GRASP can be viewed also as a random sampling of solution space. If a starting solution falls in the attraction basin of a global optimum, the local search will pull it to this global optimum. However, in general, GRASP alone cannot compete with more agressive metaheuristics like tabu search. It must be strengthened by complementary techniques like path relinking [15]. Algorithm 1 shows the structure of a GRASP. Using our common notation, it can be characterized by $np > 1$, $ni = 1$ and $nc = 1$.

2.3 Iterated Local Search (ILS)

This other simple metaheuristic is explained for instance in a tutorial by Lourenço et al. [17]. One initial solution S^* is computed using a deterministic heuristic H and improved by local search. Then, each ILS iteration applies mutation and local search to a copy S of S^*. The latter is replaced only in case of improvement. The method is summarized in Algorithm 2. Using our notation, it can be defined by $np = 1$, $ni > 1$ and $nc = 1$.

The mutation level is critical in ILS. If it is too weak, S stays in the attraction basin of S^*. If it is excessive, S and S^* are unrelated and ILS behaves like a GRASP. Ideally, an adequate mutation level should move the search to an adjacent attraction basin, in which a good local optimum is likely to be found according to the *proximate optimality principle* stated by Glover and Laguna [12]. Moreover, if the distance between the solutions before and after mutation is relatively small, the local search applied afterward is in general very fast: the new solution is reoptimized in a few moves.

2.4 Evolutionary Local Search (ELS)

ELS was recently introduced by Wolf and Merz to solve a peer-to-peer network design problem [27]. It uses a population of only one individual and there is no need for

Algorithm 2. ILS general structure

```
1:  initialize random number generator
2:  $H(S^*)$
3:  $LS(S^*)$
4:  for $j$ := 1 to $ni$ do
5:      $S := S^*$
6:      $Mutate(S)$
7:      $LS(S)$
8:      if $f(S) < f(S^*)$ then
9:          $S^* := S$
10:     end if
11: end for
```

recombination. This design is motivated by the high computation cost and solution quality of the local search. Using mutation and local search, nc offspring solutions are generated, following a $(1 + nc)$ selection paradigm.

Algorithm 3 shows that ELS can be expressed as an extension of ILS, with $np = 1$, $ni > 1$ and $nc > 1$. Compared to ILS, the current best solution S^* is updated only when it is outperformed by the best child $\overline{S}$. If the mutation rate is well tuned, an ILS jumps into the first adjacent attraction basin in which it finds a better solution. The idea of ELS is to better sample the adjacent basins, before leaving the current one.

Algorithm 3. ELS general structure

```
1:  initialize random number generator
2:  $H(S^*)$
3:  $LS(S^*)$
4:  for $j$ := 1 to $ni$ do
5:      $\bar{f} := +\infty$
6:      for $k$ := 1 to $nc$ do
7:          $S := S^*$
8:          $Mutate(S)$
9:          $LS(S)$
10:         if $f(S) < \bar{f}$ then
11:             $\bar{f} := f(S)$
12:             $\overline{S} := S$
13:         end if
14:     end for
15:     if $\bar{f} < f(S^*)$ then
16:         $S^* := \overline{S}$
17:     end if
18: end for
```

2.5 Hybrid Forms

GRASP can be hybridized with ILS, as shown in Algorithm 4. The GRASP loop is used to provide the nested ILS with distinct initial solutions and can be viewed as a

Algorithm 4. GRASP×ILS hybrid

```
1: initialize random number generator
2: $f^* := +\infty$
3: for $i$ := 1 to $np$ do
4:     $HR(S)$
5:     $LS(S)$
6:     for $j$ := 1 to $ni$ do
7:         $S' := S$
8:         $Mutate(S')$
9:         $LS(S')$
10:        if $f(S') < f(S)$ then
11:            $S := S'$
12:        end if
13:    end for
14:    if $f(S) < f^*$ then
15:        $f^* := f(S)$
16:        $S^* := S$
17:    end if
18: end for
```

diversification mechanism. We do not give an algorithm but the same idea can be used to hybridize GRASP with ELS.

3 Components for the VRP

This section describes components which can be assembled to make a GRASP, an ILS, an ELS or a hybrid form for the VRP. Like most papers on VRP metaheuristics, the number K of vehicles is here a decision variable. However, a route-length limit is accepted to tackle DVRP instances.

3.1 Giant Tours, VRP Solutions and Splitting Procedure

The key-point in the efficiency of our algorithms is to alternate between solutions encoded as TSP tours, called *giant tours*, and genuine VRP solutions. This cyclic process is illustrated in Figure 1. We explain first how to convert a giant tour into a VRP solution and vice versa.

A giant tour can be converted into a VRP solution, using a procedure called *Split* which inserts copies of the depot node. This idea was proposed by Beasley [2] as a route-first cluster-second heuristic, but without numerical results. Prins reused this principle to evaluate chromosomes coded as giant tours, in an effective memetic algorithm for the VRP [21]. He showed that a) any giant tour can be split optimally, subject to its sequence, and b) there exists one giant tour which gives an optimal VRP solution after splitting. Hence, any metaheuristic based on this encoding can search the set of TSP solutions instead of the much larger VRP solution set, without loss of information.

A giant tour is encoded as a permutation $T = (T_1, T_2, \ldots, T_n)$ of the n customers. The depot at the beginning and at the end and the shortest paths between successive

$$\begin{tikzcd}
\text{Giant tour } T \arrow[r, Rightarrow] & \mathit{Mutate}(T) \arrow[r, Rightarrow] & T \text{ mutated} \arrow[d, Rightarrow] \\
\mathit{Concat}(S,T) \arrow[u, Rightarrow] & & \mathit{Split}(T,S) \arrow[d, Rightarrow] \\
S \text{ improved} \arrow[u, Rightarrow] & \mathit{LocalSearch}(S) \arrow[l, Rightarrow] & \text{VRP solution } S \arrow[l, Rightarrow]
\end{tikzcd}$$

Fig. 1. Cyclic alternation between giant tours and VRP solutions

nodes are implicit. *Split* builds an auxiliary weighted directed graph $H = (X, A, Z)$. The nodes in X are indexed from 0 to n: 0 is a dummy node while each node $i \neq 0$ represents customer T_i.

Each subsequence of customers $(T_i, T_{i+1}, \ldots, T_j)$ of the giant tour is evaluated to see if the trip $(0, T_i, T_{i+1}, \ldots, T_j, 0)$ is feasible, i.e. if its total load does not exceed vehicle capacity Q and, for a DVRP, if it satisfies the route-length limit L. If yes, the trip is modelled in the arc-set A by one arc $(i-1, j)$, with a weight $Z_{i-1,j}$ equal to the trip cost. An optimal splitting of T into feasible trips corresponds to a min-cost path from node 0 to node n in H.

Since H is acyclic by construction, Bellman's algorithm for directed acyclic graphs [6] can be used, with an $O(|A|)$ complexity. If the maximum number of customers per trip is b, there are at most b outgoing arcs per node in H and the complexity can be rewritten as $O(nb)$.

In Figure 1, we can see that true VRP solutions are used only by the local search. After the local search, a VRP solution S can be converted into a giant tour T by concatenating the sequences of customers of its different trips, which is equivalent to erasing the copies of the depot node used as trip delimiters. This task is performed by a procedure $\mathit{Concat}(S, T)$.

3.2 Heuristics for Initial Giant Tours

To initialize an ILS or ELS, we compute a VRP solution using the Clarke and Wright savings heuristic [4], called CWH in the sequel. As explained before, the resulting VRP solution must be converted into a giant tour to start the alternating cycle, using *Concat*. CWH starts from a trivial solution with one dedicated trip per customer. Each iteration evaluates all capacity-feasible mergers (concatenations) of two trips and executes the one with the largest positive saving. The process stops when no such merger can be found. The DVRP is easily handled by rejecting mergers violating the route-length limit.

Laporte et al. [16] indicate that fairly good solutions are obtained by adding a 3-opt local search as post-optimization. Our implementation applies 3-opt to the trip obtained after each merger, which gives slightly better results than a post-optimization. Moreover, when two mergers give the same saving, we break ties by selecting the merger whose the two trips have the largest discrepancy in load: on average, this simple trick brings a further improvement.

The GRASP and its hybrids use a randomized version of the nearest neighbour TSP heuristic, called RNNH. In each iteration, for an emerging route ending at customer

i, we compute the minimum and maximum distances c_{min} and c_{max} between i and unrouted customers. The RCL is then defined by the subset of unrouted customers j such that $c_{min} \leq c_{ij} \leq c_{min} + \beta \times (c_{max} - c_{min})$, where $\beta \in [0, 1]$.

3.3 Mutation Procedure

The mutation in ILS and ELS consists in swapping two distinct customers i and j, randomly selected in the giant tour. Contrary to mutation on VRP solutions, no capacity constraint needs to be respected. To adapt the mutation level, p successive swaps are executed. The level p is set to a minimum value p_{min} at the beginning and each time the best solution is improved. It is incremented each time the mutation followed by local search returns a degraded solution, but without exceeding a maximum value p_{max}.

3.4 Local Search

Our local search uses classical moves: 2-opt moves, Or-opt moves which relocate a string of nodes, and string-exchange moves which swap two strings. The length of a string is restricted to $\lambda \in \{1, 2, 3\}$ but two exchanged strings may have different lengths. These moves are applied to one or two trips and we use a fast implementation in which each move is decomposed into a sequence of partial moves, which can be pruned using bounds on partial gains.

This technique called *sequential search* was recently proposed by Irnich et al. [14]. To implement it, a VRP solution with K trips must be stored in a circular list of $m = n + K$ nodes. Nodes 1 to n are customers, while nodes $n + 1$ to $n + K$ are copies of the depot. Each trip $k < K$ starts with depot $n + k$ and ends with depot $n + k + 1$. The last trip $k = K$ starts at depot $n + K$ and ends at depot $n + 1$. This list is easily implemented using two vectors *next* and *prev* of m integers, to get the successor and predecessor of each node (customer or depot copy). A list of neighbors $NL(i)$ is also pre-computed for each node i: it contains the other nodes, sorted in increasing order of distance to i.

To test in $O(1)$ the feasibility of a move, five m-vectors are also needed. The rank of the trip containing customer i is given by $trip(i)$. By convention, the depot $n + k$ at the beginning of trip k belongs to trip k, i.e. $trip(n + k) = k$, but the depot at the end belongs to the next trip. Q_i^b and Q_i^a denote the total demand serviced *before* and *after* node i (included) in $trip(i)$. Similarly, C_i^b and C_i^a denote the cost of $trip(i)$ before and after node i, including this node and service costs. For instance, two customers i and j are in the same trip if $trip(i) = trip(j)$. In that case, the total demand of the segment from i to j (included) is $Q_j^b - Q_i^b + q_i = Q_i^a - Q_j^a + q_j$.

We now give an example of sequential search for a special 2-opt move called 2-opt* or crossover move, to show that the technique is not trivial. The other moves have even more complicated implementations, sketched in [14]. The crossover move, depicted in Figure 2, replaces two edges $[t_1, t_2]$ and $[t_3, t_4]$ by $[t_2, t_3]$ and $[t_1, t_4]$, in such a way that the circular list is split into two sub-tours. Such an operation is infeasible for the TSP, but acceptable for the VRP if each sub-tour contains at least one depot. Note that the classical 2-opt move consists in adding the two edges $[t_1, t_3]$ and $[t_2, t_4]$.

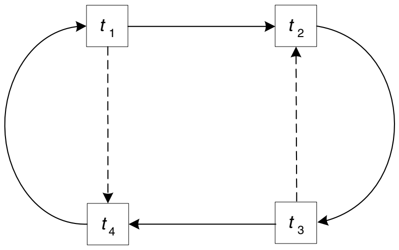


Fig. 2. Crossover move – $[t_1, t_2]$ and $[t_3, t_4]$ are removed

Algorithm 5. Efficient search of the best crossover move

```
1: $G^* := 0$
2: **for** $t_1 := 1$ **to** $m$ **do**
3:    $t_2 := next(t_1)$
4:    $B := c(t_1, t_2) - G^*/2$
5:    **for each** $t_3 \in NL(t_2)$ **while** $c_{t_2,t_3} < B$ **do**
6:       $t_4 := next(t_3)$
7:       $t2t3ok := (t_2 > n)$ or $(t_3 > n)$ or $(trip(t_2) \neq trip(t_3))$
8:       $t1t4ok := (t_1 > n)$ or $(t_4 > n)$ or $(trip(t_1) \neq trip(t_4))$
9:       $capaok := (Q^b_{t_1} + Q^a_{t_4} \leq Q)$ and $(Q^b_{t_3} + Q^a_{t_2} \leq Q)$
10:      $timeok := (C^b_{t_1} + C(t_1, t_4) + C^a_{t_4} \leq L)$ and $(C^b_{t_3} + C(t_3, t_2) + C^a_{t_2} \leq L)$
11:      **if** $t1t4ok$ and $t2t3ok$ and $capaok$ and $timeok$ **then**
12:         $G := c(t_1, t_2) - c(t_2, t_3) + c(t_3, t_4) - c(t_4, t_1)$
13:         **if** $G > G^*$ **then**
14:            $G^* := G$
15:            $t^*_1 := t_1$
16:            $b^*_3 := t_3$
17:         **end if**
18:      **end if**
19:   **end for**
20: **end for**
```

The best crossover move can be determined using Algorithm 5. A classical implementation would examine all pairs of distinct edges $([t_1, t_2], [t_3, t_4])$. The efficient implementation initializes the gain G^* of the best move to 0 and examines each edge $[t_1, t_2]$. For one given $[t_1, t_2]$, it evaluates only the edges $[t_3, t_4]$ such that c_{t_2,t_3} does not exceed the upper bound $B := c_{t_1,t_2} - G^*/2$, which is easily done by browsing the list of neighbours of t_2. This structure is based on two properties [14]:

a) To find an improving move ($G > 0$), only the pairs of edges $([t_1, t_2], [t_3, t_4])$ with a partial gain $c_{t_1,t_2} - c_{t_2,t_3} > 0$ need to be tested.
b) If we have an improving move ($G^* > 0$) and want a better one ($G > G^*$), only the pairs $([t_1, t_2], [t_3, t_4])$ with $c_{t_1,t_2} - c_{t_2,t_3} > G^*/2$ need to be tested.

For a given move defined by two edges $[t_1, t_2]$ and $[t_3, t_4]$, *legitimacy conditions* are checked first in lines 7-8, by computing two Booleans $t2t3ok$ and $t1t4ok$, true if and

Algorithm 6. Local search overview

```
1: procedure LS (inout $S$: solution; in $bi$: boolean; in $\lambda$: integer)
2: repeat
3:     Classical2Opt ($S$,$bi$,$i_2$)
4:     CrossoverMove ($S$,$bi$,$i_1$)
5:     SwapTwoNodes ($S$,$bi$,$i_3$)
6:     OrOptMove ($S$,$bi$,$\lambda$,$i_4$)
7:     StringExchange ($S$,$bi$,$\lambda$,$i_5$)
8: until not ($i_1$ or $i_2$ or $i_3$ or $i_4$ or $i_5$)
9: end procedure
```

only if the subtours containing $[t_2, t_3]$ and $[t_1, t_4]$ (respectively) include a depot. Note how this is done, on $[t_2, t_3]$ for instance: the sub-tour contains at least one depot if t_2 or t_3 is a depot, otherwise the two nodes are customers and they must be in distinct trips, i.e., $trip(t_2) \neq trip(t_3)$.

Vehicle capacity Q and maximum trip duration L are checked using two Booleans $capaok$ (line 9) and $timeok$ (line 10). If all constraints are satisfied, the gain G is computed (line 12). Each time a better move is found, G^* is updated and the nodes t_1 and t_3 defining the move are saved in t_1^* and t_3^* (lines 15-16). This best-improvement local search can be converted into a first-improvement version, by leaving the two *for* loops just after line 16.

Sequential search implementations can be found in [14] for the classical 2-opt and 3-opt moves, the Or-opt move and the string-exchange move, but the non-trivial legitimacy conditions are not detailed.

In a classical local seach, several moves can be evaluated in parallel. For instance, if we browse all pairs of nodes (i, j) in a trip, we can evaluate an exchange of i and j, a relocation of i after j, a relocation of j after i and even a classical 2-opt move, which is equivalent to an inversion of the subsequence of nodes from i to j (included). This is no longer possible in sequential search: each move uses its own loops and bounds and must be implemented as a separate procedure, like in our local search depicted in Algorithm 6.

The arguments are a solution S (modified in case of improvement), one Boolean bi and the maximum string length λ. LS calls one dedicated procedure for each neighborhood mentioned at the beginning of this subsection. However, we got slightly better results by adding a *SwapTwoNodes* procedure, special case of Or-opt move with $\lambda = 1$.

Each procedure searches for the best improvement (if bi = true) or first improvement (if bi = false). If one improving move is found, the procedure executes it, which modifies S, and sets a Boolean flag (last argument) to true. Otherwise, the flag is set to false. All neighborhoods are searched again if at least one of them brings an improvement. This local search is quite close to a variable neighborhood descent, except that the different neighborhoods are explored in a cyclic way.

3.5 Resulting Generic Algorithm

Algorithm 7 summarizes our GRASP×ELS hybrid for the VRP and DVRP. S^* is the best solution found over the np phases and f^* its cost. If $np = 1$, a VRP solution $\overline{S}$ is computed using the Clarke and Wright heuristic (CWH), otherwise a giant tour T is built with the randomized nearest neighbor heuristic (RNNH) and converted into a VRP solution $\overline{S}$ using *Split*. In both cases, $\overline{S}$ undergoes the local search LS and is converted back into a giant tour $\overline{T}$.

Algorithm 7. GRASP×ELS hybrid for the VRP and DVRP

```
1: initialize random number generator
2: $f^* := +\infty$
3: $p := p_{min}$
4: for $i := 1$ to $np$ do
5:     if $np = 1$ then
6:         $CWH(\overline{S})$
7:     else
8:         $RNNH(T)$
9:         $Split(T, \overline{S})$
10:    end if
11:    $LS(\overline{S})$
12:    $Concat(\overline{S}, \overline{T})$
13:    for $j := 1$ to $ni$ do
14:        $\hat{f} := f(\overline{S})$
15:        for $k := 1$ to $nc$ do
16:            $T := \overline{T}$
17:            $Mutate(T, p)$
18:            $Split(T, S)$
19:            $LS(S)$
20:            if $f(S) < \hat{f}$ then
21:                $\hat{f} := f(S)$
22:                $\hat{S} := S$
23:            end if
24:        end for
25:        if $\hat{f} < f(\overline{S})$ then
26:            $\overline{S} := \hat{S}$
27:            $Concat(\overline{S}, \overline{T})$
28:            $p := p_{min}$
29:        else
30:            $p := \min(p + 1, p_{max})$
31:        end if
32:    end for
33:    if $f(\overline{S}) < f^*$ then
34:        $f^* := f(\overline{S})$
35:        $S^* := \overline{S}$
36:    end if
37: end for
```

The internal ELS starts with the pair $(\overline{T}, \overline{S})$ and performs ni iterations. Each iteration generates nc children-tours by taking a copy T of $\overline{T}$, which is then mutated, splitted to get a VRP solution S and improved by local search. The best child-solution is recorded in $\hat{S}$ and its cost in $\hat{f}$. $\overline{S}$ is updated if the best child is better. The associated giant tour $\overline{T}$ is deduced using *Concat*. This gives a new pair $(\overline{T}, \overline{S})$ for the next ELS iteration. At the end of each phase, the best ELS solution is used to update the global best solution S^*.

The number p of random swaps executed in *Mutate* is variable. It is initialized to a lower bound p_{min} and incremented each time $\overline{S}$ is not improved by the best child, but without exceeding an upper bound p_{max}. After an improvement, p is reset to p_{min}. As mentioned in Section 2, Algorithm 7 boils down to a GRASP if $ni = nc = 1$, to an ELS if $np = 1$ and to an ILS if $np = nc = 1$. A GRASP×ILS hybrid is obtained by setting $nc = 1$.

4 Computational Evaluation

4.1 Implementation and Instances

Algorithm 7 was implemented in the Pascal-like language Delphi 7 on a desktop PC with a 2.8 GHz Pentium 4 and Windows XP Pro. The evaluation was conducted on two sets of *Euclidean* benchmark problems, i.e, the vertices are defined by points in the plane and the cost of each edge (i, j) is the Euclidean distance between i and j.

The first set contains 14 classical problems proposed by Christofides et al. [3]. They have 50 to 199 customers. Files 6 to 10, 13 and 14 have a maximum trip duration (DVRP) and non-zero service times. The others are pure VRPs. The second set contains 20 large-scale instances (200 to 483 customers) proposed by Golden et al. [13]. Only instances 1 to 8 are DVRPs, but with null service times. These two sets can be found in a library of VRP instances created by Alba and Dorronsoro, at `http://neo.lcc.uma.es`, in the directory `radi-aeb/WebVRP/Problem_Instances/CVRPInstances.html`.

4.2 Warning: Comparison of VRP Metaheuristics

While the community working on exact algorithms use rounded costs, all papers on VRP metaheuristics use double-precision real numbers and we have respected this convention to guarantee pertinent comparisons. Another convention is to give only the total length of the trips as solution cost. Service costs of DVRPs are not included, they are only used to check that the length of a trip plus the service costs of its customers does not exceed the limit L.

Barr et al. [1] proposed in 1995 a set of rules for a fair comparison of heuristics: the tables of results must include solution values for one single run and standard parameters, then a column for the best results obtained using several parameter settings may be added. We have observed a regrettable drift in the vehicle routing community: many comparisons mix algorithms which comply with Barr's rules and others which present the best result of several runs. In particular, ant colony researchers often give the best result of several runs but with the average duration per run.

Moreover, some recent papers contain typos in the deviations to best-known solutions and several best-known solutions have been improved since their publication. To fix these problems, we collected the most recent results starting from the 2005 survey of Cordeau et al. [5]. These results have been updated using three papers: the SEPAS algorithm published in 2005 by Tarantilis [24], the AGES method modified in 2007 by Mester and Bräysy [19] for the VRP (the method was initially designed for the VRP with time windows [18]) and the ALNS heuristic of Pisinger and Röpke [20], issued also in 2007. All results which do not comply with Barr's rules have been discarded and all average deviations and running times have been recomputed using Excel.

4.3 Preliminary Experiments

Christofides instances were used for preliminary experiments with Algorithm 7. They revealed that a first-improvement local search (bi = false in Algorithm 6) with $\lambda = 3$ gives the best results. Using these values, we decided to allocate 5000 calls to the local search to each algorithm and to distribute them among np, ni and nc. The best combinations obtained are listed in Table 1.

Table 1. Comparison for 5000 local searches – Christofides et al. instances

Version	$np \times ni \times nc$	p	β	*dbks*	*nbks*	*time*	*nls*	*mnls*	*LS/s*
GRASP	$5000 \times 1 \times 1$	–	0	1.827	1	9.76	608	4563	225
	$5000 \times 1 \times 1$	–	5	0.810	5	15.28	1338	4494	178
ILS	$1 \times 5000 \times 1$	1-2	–	0.314	7	5.80	1022	4876	312
	$1 \times 5000 \times 1$	1-4	–	0.205	6	12.82	1539	2753	232
ELS	$1 \times 100 \times 50$	1-2	–	0.199	7	6.85	1187	2995	335
	$1 \times 50 \times 100$	1-2	–	0.163	7	9.79	2126	4954	346
GRASP×ILS	$5 \times 1000 \times 1$	1-2	0	0.199	8	8.53	1583	4630	325
	$10 \times 500 \times 1$	1-2	0	0.169	8	4.82	1060	4171	325
	$20 \times 250 \times 1$	1-2	0	0.184	8	8.30	1579	3733	326
GRASP×ELS	$5 \times 100 \times 10$	1-1	0	0.152	8	3.05	894	3769	429
	$10 \times 50 \times 10$	1-1	0	0.148	7	5.37	1482	4781	432
CWH+3-OPT	–	–	–	6.620	0	0.06	–	–	–
MA	–	–	–	0.235	8	153.81	–	–	–

The first column indicates the kind of algorithm: GRASP, ILS, ELS and hybrid forms GRASP×ILS and GRASP×ELS. The corresponding values of np, ni and nc are given in column 2. Column 3 indicates the extreme values p_{min} and p_{max} for the mutation. The parameter β used for the randomized nearest neighbor solution (RNNH) is given in % in column 4. The *dbks* and *nbks* columns display the average deviation in % to the most recent best-known solutions [19] and the number of best-known solutions (out of 14) retrieved by each algorithm. The remaining columns respectively give the average time in seconds to reach the best solution (*time*), the average and maximum numbers of calls to the local search to reach this solution (*nls* and *mnls*) and the average number of local searches executed per second (*LS/s*).

Concerning parameters, Table 1 shows that the required values for p_{max} are quite small. As already mentioned, even one single swap can give a different solution after splitting. A mutation applied to VRP solutions instead of giant tours would probably require larger values. The value $\beta = 0\%$ works well in general, because many edges have equal lengths in the Christofides et al. instances and the nearest neighbor heuristic has on average 2.5 candidates with a minimum distance c_{min} to the last routed customer. However, the results are not enough diversified for the basic GRASP which generates more randomized solutions (5000): we took $\beta = 5\%$ for this algorithm.

Concerning results, the table shows that the two GRASPs are outperformed by the other algorithms, *for the same number of calls to the local search*. Moreover, looking at the last column, we see that the number of local searches executed per second is always larger for the other algorithms: after a mutation, a small number of moves is often sufficient to reoptimize the solution.

ILS needs a larger p_{max} value (4) than the other versions to reach its best average deviation (0.205%) and we can see that no improvement occurs after iteration 2753. The ELS and the hybrid GRASP×ELS show better deviations, slightly greater than 0.16%, while being faster than the best ILS. But the most efficient and fastest algorithm is the GRASP×ELS hybrid, the only one able to go below 0.15%. Figure 3 illustrates the convergence curves of the best versions of GRASP, ILS and GRASP×ELS, as from iteration 100 for a better readability: we see that GRASP×ELS still finds improvements when they are exceptional for ILS. However, ILS is better below 2500 calls to the local search.

The two last lines of the table have been added to compare our heuristics with the CWH heuristic with 3-opt and the memetic algorithm (MA) of Prins [21]. Even with the help of a 3-opt local search, the savings heuristic with 6.620% is far from the deviations achieved by our algorithms but it is very fast. In terms of deviation, the MA is effective (0.235%) but outperformed by our new heuristics except the two GRASPs and the ILS with $p \in \{1, 2\}$. Moreover, our best algorithm, GRASP-ELS with $np = 10$, is 28.6 times faster!

4.4 Results for the Christofides et al. Instances

After the preliminary experiments with 5000 local searches, we decided to keep the following parameters from GRASP×ELS: a first-improvement local search (bi = false) with $\lambda = 3$, $\beta = 0\%$ in RNNH and $p_{min} = 1$ in the mutation. Better results can be obtained by allocating more calls, at the expense of longer running times. In that case, p_{max} must be increased to 2 to bring enough diversity. For $np = 5$, $ni = 20$ and $nc = 100$ (10000 calls), GRASP×ELS finds on average a deviation of 0.094% in 9.29s. For $np = 5$, $ni = 40$ and $nc = 100$ (20000 calls), the two indicators become 0.071% and 15.93s.

To limit the size of our tables, we decided to make a comparison with metaheuristics able to achieve a 0.3% deviation to best-known solutions. Cordeau's survey [5] cites four candidates only: the memetic algorithm (MA) of Prins [21], the Bone Route adaptive memory procedure of Tarantilis and Kiranoudis [25], the Active Guided Evolution Strategy (AGES) of Mester and Bräysy [18] and the Very Large Neighborhood Search (VLNS) of Ergun et al. [8]. VLNS was discarded because of its results given for 5 runs.

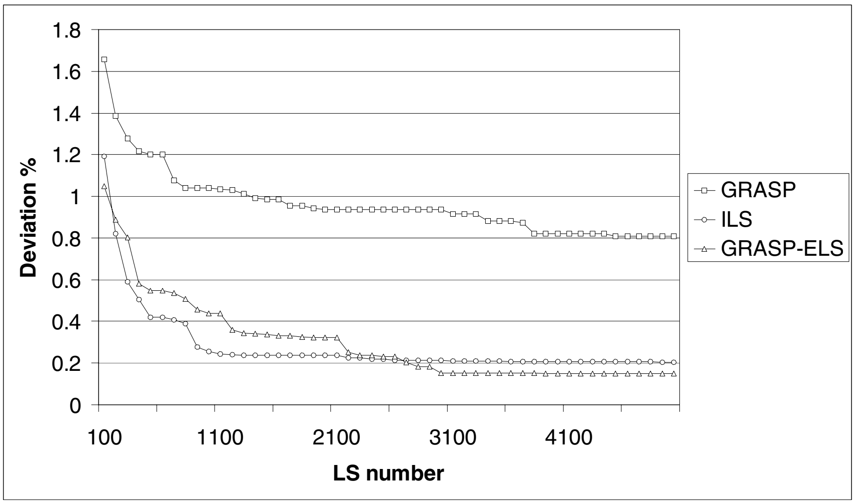


Fig. 3. Convergence curves for GRASP, ILS and GRASP-ELS

AGES has two versions, a standard one *AGES Best* and a less effective but very fast one called *AGES Fast*. They were initially designed for the VRP with time windows [18], but Cordeau's survey includes VRP results obtained via a private communication with Mester. Since the survey, Mester and Bräysy got improved results, using a better adaptation to the VRP [19]. We took these results instead of the older ones of the survey. We have also included the adaptive memory method SEPAS recently proposed by Tarantilis [24].

Cordeau's survey also includes results for the D-Ants algorithm of Reimann et al. [22], but only for Golden's instances. Results for Christofides problems can be found in [22]: the best deviation for 10 runs is excellent (0.15%) but the average deviation (0.48%) exceeds our 0.3% limit. The ALNS heuristic of Pisinger and Röpke [20] published in 2007 was ignored for the same reason: the best version has a 0.11% deviation for 10 runs, but a 0.31% average deviation.

The comparison is presented in Table 2. The BKS column indicates best-known solutions. The next columns give the solution values found by the MA, Bone Route, SEPAS and the two versions of AGES. The last columns show the solution costs found by the GRASP×ELS with $np = 5$, $ni = 40$ and $nc = 100$ (GRELS) and the running times in seconds to reach these solutions on our PC. Boldface numbers indicate best-known solutions retrieved by the algorithms. The running times per instance for the other heuristics can be found in the original papers. Recall that the costs given for each algorithm concern one single run with identical parameters for all instances.

The seven last lines respectively indicate the average deviation to the best-known solutions in % (*Dbks*), the number of best-known solutions retrieved (*Nbks*), the kind of processor used (*Proc.*), its clock speed (*Clock*), the average running time in seconds on this processor, the scaling factor for our 2.8 GHz PC according to Dongarra [7] and the scaled time. This scaled time is only an approximation: it can be affected by the

Table 2. Metaheuristics below 0.3% for the Christofides et al. benchmarks

File-n	BKS	MA	Bone Route	SEPAS	AGES Best	AGES Fast	GRELS	Time
01-050	524.61	**524.61**	**524.61**	**524.61**	**524.61**	**524.61**	**524.61**	0.06
02-075	835.26	**835.26**	**835.26**	**835.26**	**835.26**	**835.26**	**835.26**	15.01
03-100	826.14	**826.14**	**826.14**	**826.14**	**826.14**	**826.14**	**826.14**	3.19
04-150	1028.42	1031.63	**1028.42**	1029.64	**1028.42**	**1028.42**	1029.48	44.84
05-199	1291.29	1300.23	1311.48	1311.48	**1291.29**	1294.25	1294.09	11.85
06-050	555.43	**555.43**	**555.43**	**555.43**	**555.43**	**555.43**	**555.43**	0.14
07-075	909.68	912.30	**909.68**	**909.68**	**909.68**	**909.68**	**909.68**	5.18
08-100	865.94	**865.94**	**865.94**	**865.94**	**865.94**	**865.94**	**865.94**	0.32
09-150	1162.55	1164.25	**1162.55**	1163.19	**1162.55**	1164.54	**1162.55**	6.58
10-199	1395.85	1420.20	1407.21	1407.21	1401.12	1404.67	1401.46	125.57
11-120	1042.11	**1042.11**	**1042.11**	**1042.11**	**1042.11**	**1042.11**	**1042.11**	0.64
12-100	819.56	**819.56**	**819.56**	**819.56**	**819.56**	**819.56**	**819.56**	0.17
13-120	1541.14	1542.97	1544.01	1544.01	**1541.14**	1543.26	1545.43	9.25
14-100	866.37	**866.37**	**866.37**	**866.37**	**866.37**	**866.37**	**866.37**	0.27
Dbks		0.236	0.183	0.196	0.027	0.084	0.071	
Nbks		8	11	9	13	10	10	
Proc.		P3	P2	P2	P4	P4	P4	
Clock		1000	400	400	2800	2800	2800	
Time		311.17	313.03	337.50	162.66	3.01	15.93	
Factor		0.4943	0.1972	0.1972	1.0000	1.0000	1.0000	
Scaled		153.81	61.73	66.56	162.66	3.01	15.93	

programming language used, the operating system, some PC equipment like the cache memory etc. The computations with Excel allowed us to detect two typos in published papers: Tarantilis [24] gives an average deviation of 0.50% instead of 0.235 for the MA, and Mester and Bräysy 0.07% instead of 0.084 for AGES Fast [19].

The table shows that GRASP×ELS becomes the second best metaheuristic in terms of average deviation (0.071%): only AGES Best does better with 0.027%. It is also faster than the other algorithms, except AGES Fast which provides a slightly greater deviation (0.084%) in only 3 seconds. GRASP×ELS is probably the simplest method: it integrates simple components in a simple structure. It is controlled by eight parameters (n, ni, nc, bi, λ, β, p_{min} and p_{max}) but four of them are rather named constants used as a good programming rule, since they are frozen in practice: bi = false, $\lambda = 3$ and $\beta = 0$. In comparison, SEPAS requires eight parameters, the MA ten and AGES twelve.

Using several sets of parameters, GRASP×ELS found 1028.42 instead of 1029.48 for instance 4 (best-known solution) and 1542.86 instead of 1545.43 for instance 13, which corresponds to an average deviation of 0.052%. It is fair to say that all best-known solutions except one were found by Rochat and Taillard in 1995 [23], but using several parameter settings.

The only improvement was found in 2007 by Mester and Bräysy for instance 5 ($n =$ 199 customers), with 1291.29 instead of 1291.45 previously. It is likely that most best-known solutions are optimal.

4.5 Results for the Golden et al. Instances

Due to their size, the running times are much larger on Golden's instances. Moreover, the special structure of these problems, with rings of customers, force the local search to execute numerous moves with small costs variations. A faster convergence is obtained using a first-improvement local search with $\lambda = 4$, instead of 3 previously. Table 3 gives some good compromises between solution quality and running time. Solutions are very sensitive to mutation, since the combinations with $p_{max} = 1$ are faster and achieve smaller deviations to best-known solutions. The value of β can be set to zero in all cases.

The best combination is again a GRASP×ELS: using 50,000 calls to local search, it reaches in 436 seconds (7.3 minutes) an average deviation of 0.315% to best-known solutions. Moreover, all GRASP×ELS versions find a new best solution, indicated by a plus sign in colum *nbks*. In fact, only instance 8 is improved, but down to 11643.90: the previous best-known cost was 11663.55 [19].

Results for the GRASP are not included, because it is 5 times slower than the fastest ILS, with a doubled deviation. The results of the savings heuristic with 3-opt are poor, even if this method is very fast. In terms of deviation, the memetic algorithm [21] is superseded by all the new algorithms. Concerning the speed, the fastest algorithm (GRASP×ILS) is 30 times faster and the best method (GRASP×ELS) 4.5 times.

Table 3. Some good combinations of parameters for Golden's instances

Version	$np \times ni \times nc$	p	β	*dbks*	*nbks*	*time*	New BKS
ILS	$1 \times 5000 \times 1$	1-1	–	1.005	3	73.52	
	$1 \times 5000 \times 1$	1-2	–	1.159	2	113.96	
GRASP×ILS	$5 \times 1000 \times 1$	1-1	0	0.860	3	65.82	
GRASP×ELS	$5 \times 40 \times 100$	1-2	0	0.663	3+1	309.68	8: 11656.52
	$5 \times 100 \times 50$	1-1	0	0.526	4+1	209.96	8: 11652.31
	$5 \times 200 \times 50$	1-1	0	0.315	4+1	436.21	8: 11643.90
CWH+3-OPT	–	–	–	9.240	0	1.43	
MA	–	–	–	1.343	2	1984.12	

To prepare Table 4, we selected the VRP metaheuristics with a less than 1% average deviation to the most recent best-known solutions listed in [19], to compare them with our best GRASP×ELS. These best-known solutions do not include the new ones obtained in this paper. The selected algorithms are SEPAS and the two versions of AGES. Bone Route was discarded because it was evaluated only on the DVRP instances (1 to 8).

In [22], Reimann et al. gives the best result of 10 runs for each instance. For the average result, they only give a deviation in percent with two significant digits: this corresponds to an error of up to 100 units on the actual solution costs, which cannot be computed exactly. Hence, the D-Ants algorithm is perhaps in the 1% group, but we cannot include it, due to the lack of exact figures. Anyway, when the paper was published, the best results had a 0.12% deviation and the average results 0.42%. Recomputed with the most recent best-known solutions, the first deviation becomes 0.614% (and not 0.32% as written in [19]): it is unlikely that the other deviation is still below

Table 4. Metaheuristics below 1% for the Golden et al. benchmarks

File-n	BKS	SEPAS	AGES Best	AGES Fast	GRELS	Time	BGRELS
01-240	5627.54	5676.97	**5627.54**	5644.00	5644.52	238.02	5644.52
02-320	8447.92	8459.91	**8447.92**	8468.00	**8447.92**	93.12	***8444.50**
03-400	11036.22	**11036.22**	**11036.22**	11146.00	**11036.22**	384.94	**11036.22**
04-480	13624.52	13637.53	**13624.52**	13704.52	**13624.52**	349.82	**13624.52**
05-200	6460.98	**6460.98**	**6460.98**	**6460.98**	**6460.98**	62.76	**6460.98**
06-280	8412.80	8414.28	8412.88	8539.61	8412.90	84.44	8412.90
07-360	10181.75	10216.50	10195.56	10240.42	10195.59	463.21	10195.59
08-440	11663.55	11936.16	**11663.55**	11918.75	***11643.90**	298.05	***11643.90**
09-255	583.39	585.43	**583.39**	588.43	586.23	78.62	586.18
10-323	741.56	746.56	**741.56**	752.92	744.36	437.87	744.36
11-399	918.45	923.17	**918.45**	925.94	922.40	272.82	922.40
12-483	1107.19	1130.40	**1107.19**	1120.67	1116.12	1746.33	1116.12
13-252	859.11	865.01	**859.11**	865.20	862.32	128.69	860.55
14-320	1081.31	1086.07	**1081.31**	1097.68	1089.35	546.93	1084.82
15-396	1345.23	1353.91	**1345.23**	1354.76	1352.39	613.36	1352.39
16-480	1622.69	1634.74	**1622.69**	1634.99	1634.27	1262.99	1634.27
17-240	707.79	708.74	**707.79**	710.22	708.85	61.63	**707.79**
18-300	997.52	1006.90	998.73	1009.53	1002.15	175.55	1002.15
19-360	1366.86	1371.01	**1366.86**	1381.88	1371.67	1078.54	1371.67
20-420	1820.09	1837.67	**1820.09**	1840.57	1830.98	346.49	1830.98
Dbks		0.615	0.013	0.914	0.315		0.274
Nbks		2	17	1	4+1		4+2
Proc.		P2	P4	P4	P4		
Clock		400	2800	2800	2800		
Time		2728.80	1461.22	13.49	436.21		
Factor		0.1972	1.0000	1.0000	1.0000		
Scaled		538.12	1461.22	13.49	436.21		

1%. Concerning the ALNS heuristic of Pisinger and Röpke, the best version displays a deviation of 0.49% for the best result of 10 runs, but 1.02% for the average solution cost. This is why ALNS was not chosen.

Table 4 has the same format as Table 2 and running times are still given in seconds. The GRELS column corresponds to the GRASP×ELS, while the BGRELS provides the best results obtained during our experiments, using several sets of parameters. Numbers in boldface indicate solutions at least as good as best-known solutions and asterisks denote new best solutions.

Again, GRASP×ELS is the second best method: with its 0.315% deviation, it takes place between AGES Best (0.013% but more than three times slower) and SEPAS (0.615%). As usual, AGES Fast has remarkably small running times but its average deviation is close to our 1% selection limit. Using one setting of parameters, GRASP×ELS finds one new best solution for instance 8. During our tests with several parameter settings, we also discovered an improved solution for instance 2.

5 Conclusion

This chapter show that the simple principles of iterated local search can be strengthened to lead to fast and powerful algorithms for the vehicle routing problem. The resulting heuristics are still simple, except the non-trivial sequential decomposition of moves in the local search.

A key-point in the efficiency of these new methods is the alternation between giant tours and VRP solutions: this principle seems very fruitful, since we have already applied it successfully to the VRP, the capacitated arc routing problem (CARP), the periodic CARP and the VRP and CARP with time windows. Since the splitting procedure is polynomial for all these problems, it is clear that GRASP$\times$ELS can be generalized to tackle them.

We tried to add the following refinements: a frequency-based long-term memory, a distance measure in solution space to spread the trajectories from initial solutions, a tabu list, a path-relinking procedure, better initial solutions and other mutations. All these techniques had no significant impact on solution quality: the simpler, the better! The algorithms could probably be improved in two ways. The first one would be to design heuristics better adapted to the Golden et al. instances: they have a special structure which is absolutely unlikely in real distribution problems. The second one would be to add some mechanism for an automatic tuning of parameters.

Acknowledgements. This research has been funded by the Champagne-Ardenne Regional Council (France) and by the European Social Fund.

References

1. Barr, R.S., Golden, B.L., Kelly, J.P., Resende, M.G.C., Stewart Jr., W.R.: Designing and reporting on computational experiments with heuristic methods. Journal of Heuristics 1, 9–32 (1995)
2. Beasley, J.E.: Route-first cluster-second methods for vehicle routing. Omega 11, 403–408 (1983)
3. Christofides, N., Mingozzi, A., Toth, P.: The vehicle routing problem. In: Christofides, N., Mingozzi, A., Toth, P., Sandi, C. (eds.) Combinatorial Optimization, pp. 315–338. Wiley, Chichester (1979)
4. Clarke, G., Wright, J.W.: Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 12, 568–581 (1964)
5. Cordeau, J.F., Gendreau, M., Hertz, A., Laporte, G., Sormany, J.S.: New heuristics for the vehicle routing problem. In: Langevin, A., Riopel, D. (eds.) Logistic Systems: Design and Optimization, pp. 279–298. Wiley, Chichester (2005)
6. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press, Cambridge (1990)
7. Dongarra, J.J.: Performance of various computers using standard linear equations software. Technical Report CS-89-85, Computer Science Department, University of Tennessee (2006)
8. Ergun, O., Orlin, J.B., Steele-Feldman, A.: Creating very large scale neighborhoods out of smaller ones by compounding moves: a study on the vehicle routing problem. Technical report, Massachusetts Institute of Technology (2003)
9. Feo, T.A., Bard, J.: Flight scheduling and maintenance base planning. Management Science 35, 1415–1432 (1989)

10. Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedures. Journal of Global Optimization 6, 109–133 (1995)
11. Fukasawa, R., Lysgaard, J., Poggi de Aragão, M., Reis, M., Uchoa, E., Werneck, R.F.: Robust branch-and-cut-and-price for the capacitated vehicle routing problem. Mathematical Programming 106, 491–511 (2006)
12. Glover, F., Laguna, M.: Tabu search. Kluwer, Dordrecht (1997)
13. Golden, B.L., Wasil, E.A., Kelly, J.P., Chao, I.M.: The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets and computational results. In: Crainic, T.G., Laporte, G. (eds.) Fleet management and logistics, pp. 33–56. Kluwer, Dordrecht (1998)
14. Irnich, S., Funke, B., Grünert, T.: Sequential search and its application to vehicle-routing problems. Computers & Operations Research 33, 2405–2429 (2006)
15. Labadi, N., Prins, C., Reghioui, M.: GRASP with path relinking for the capacitated arc routing problem with time windows. In: Giacobini, M. (ed.) EvoWorkshops 2007. LNCS, vol. 4448, pp. 722–731. Springer, Heidelberg (2007)
16. Laporte, G., Gendreau, M., Potvin, J.Y., Semet, F.: Classical and modern heuristics for the vehicle routing problem. International Transactions in Operational Research 7, 285–300 (2000)
17. Lourenço, H.R., Martin, O., Stützle, T.: Iterated local search. In: Glover, F., Kochenberger, G. (eds.) Handbook of metaheuristics, pp. 321–353 (2002)
18. Mester, D., Bräysy, O.: Active guided evolution strategies for large scale vehicle routing problems with time windows. Computers & Operations Research 32, 1593–1614 (2005)
19. Mester, D., Bräysy, O.: Active guided evolution strategies for large scale capacitated vehicle routing problems. Computers & Operations Research 34, 2964–2975 (2007)
20. Pisinger, D., Röpke, S.: A general heuristic for vehicle routing problems. Computers & Operations Research 34, 2403–2435 (2007)
21. Prins, C.: A simple and effective evolutionary algorithm for the vehicle routing problem. Computers & Operations Research 31, 1985–2002 (2004)
22. Reimann, M., Doerner, K., Hartl, R.F.: D-ants: savings based ants divide and conquer the vehicle routing problem. Computers & Operations Research 31, 563–591 (2004)
23. Rochat, Y., Taillard, E.: Probabilistic diversification and intensification in local search for vehicle routing. Journal of Heuristics 1, 147–167 (1995)
24. Tarantilis, C.D.: Solving the vehicle routing problem with adaptive memory programming methodology. Computers & Operations Research 32, 2309–2327 (2005)
25. Tarantilis, C.D., Kiranoudis, C.T.: Bone route: an adaptive memory-based method for effective fleet management. Annals of Operations Research 115, 227–241 (2002)
26. Toth, P., Vigo, D.: The vehicle routing problem. SIAM, Philadelphia (2002)
27. Wolf, S., Merz, P.: Evolutionary local search for the super-peer selection problem and the p-hub median problem. In: Bartz-Beielstein, T., Blesa Aguilera, M.J., Blum, C., Naujoks, B., Roli, A., Rudolph, G., Sampels, M. (eds.) HCI/ICCV 2007. LNCS, vol. 4771, pp. 1–15. Springer, Heidelberg (2007)

An Evolutionary Algorithm for the Open Vehicle Routing Problem with Time Windows

Panagiotis P. Repoussis, Christos D. Tarantilis, and George Ioannou

Athens University of Economics & Business, Department of Management Science & Technology, Evelpidon 47A & Leukados 33, GR-11362, Athens, Grecce
{prepousi,tarantil,ioannou}@aueb.gr

Summary. This chapter presents an Evolutionary Algorithm (EA) for solving the Open Vehicle Routing Problem with Time Windows (OVRPTW). The OVRPTW is a well known hard combinatorial optimization problem that seeks to design a set of non-depot returning vehicle routes in order to service a set of customers with known demands. Each customer is serviced only once by exactly one vehicle, within fixed time intervals that represent the earliest and latest times during the day that service can take place. The objective is to minimize the fleet size following routes of minimum distance. The proposed EA manipulates a population of μ individual via an $(\mu + \lambda)$-evolution strategy. At each generation, a new intermediate population of λ offspring is produced via mutation based on arcs extracted from parent individuals. The selection and combination of arcs is dictated by a vector of strategy parameters. The values of these parameters depend on the appearance frequency of each arc within the population and the current diversity of the latter, while their self-adaptation is facilitated utilizing a multi-parent recombination operator. Finally, each new offspring is further improved via a short-term memory Tabu Search algorithm and a deterministic scheme is followed for the selection of survivors. Experimental results on benchmark data sets of the literature illustrate the efficiency and effectiveness of the proposed bio-inspired solution approach.

1 Introduction

The physical distribution of goods and services plays an important role, since the relevant costs constitute a large portion of the overall field expenses of the distributor. Towards this end, the Vehicle Routing Problem (VRP) is a focal problem of distribution management and has motivated intense theoretical work and development of effective models and sophisticated algorithms. Especially, in cases where the vehicle fleet is hired, effective planning is a critical success factor for the operational efficiency and for the resulting service level, since non-company assets-resources are responsible for the physical interface with the final customer [1].

This chapter presents a sophisticated bio-inspired algorithm for solving the Open Vehicle Routing Problem with Time Windows (OVRPTW) [2]. The OVRPTW can be described as the problem of designing least cost open routes from a depot to a set of customers with known demand. The latter are geographically dispersed with a distance radius that allows their daily service via a homogeneous fleet of vehicles. Furthermore, each customer is associated with a time window during which delivery must take place, and each vehicle has a fixed capacity, and perhaps, a route-length restriction that limits the maximum distance it can travel. The routes must be designed such that each customer is visited only once by exactly one vehicle, while the accumulated load up to any customer of a route must not exceed the capacity of the vehicle. After all deliveries have been performed, vehicles do not return to the depot, and the delivery process is terminated as soon as the final customer is served. The objective is to determine the minimum number of vehicles that are required to service all customers following open routes of minimum distance.

The above described OVRPTW operational framework is faced by a company which either does not own a vehicle fleet or the owned fleet is inappropriate or inadequate to satisfy customers' demand [3]. As such, the company has to contract all or part of its delivery or pick up activities to external contracted carriers. Furthermore, in cases where the customers' demand varies over time, the solution of the OVRPTW will provide the proper combination of owned and hired vehicles [4]. Companies that have a large number of deliveries encounter the same type of problem. However, one should note that there is also a trade-off. Although hiring vehicles is more expensive per unit distance traveled, a number of costs, such as capital, maintenance and depreciation costs, do not occur [5].

In the literature, several real-life applications that fit the OVRPTW framework have been addressed. Bodin et al [6] considered an express airmail distribution problem with delivery and pickup time windows, caps on the total route length, vehicle capacities and open routes. Two routing problems (deliveries and pickups) are separately solved, using the well-known savings heuristic of Clarke and Wright [7]. The latter is appropriately modified to account for the openness of the routes, among other side constraints.

Atkinson [8] considered an open vehicle routing and scheduling problem for the delivery of school meals, involving simultaneous pickup and delivery within predefined time windows. A sequential insertion-based greedy construction heuristic is proposed that incorporates a look-ahead capability by examining the effect that the selection of a customer for service may have on other customers, not routed yet. In particular, at each stage of construction, a customer is selected for insertion such that a function that combines, through appropriate weights, travel times and distances and time window information is minimized.

Fu and Wright [9] considered a train planning model for the British Rail freight services through the Channel Tunnel. Train planning involved the specification of the origin and destination of trains and details of pick ups or deliveries along the routes. The objective is to minimize the number of trains moving through the tunnel, the train mileage and the amount of marshalling, such that capacity and time window constraints relative to the stations opening times and the time slots are satisfied.

Li and Fu [10] addressed a school bus routing problem. During morning hours a bus leaves its parking place and arrives at the first pickup point where the route starts, and travels along a predetermined route, picking up pupils at several points and taking them to school, within predefined time windows. In the afternoon, the procedure is reversed. The main objective is to find the set of school bus routes that satisfies the bus capacity, to minimize the total number of buses required, the total travel time and to balance the loads and travel times between buses.

Repoussis et al [11] proposed a multi-start hybrid metaheuristic algorithm to tackle a lubricant distribution problem modeled as an OVRPTW. The Greedy Randomized Adaptive Search Procedure (GRASP) [12] solution framework is utilized for diversification combined with a Reactive Variable Neighborhood Descent [13] (ReVND) algorithm for intensification local search. The latter employs a specialized shaking mechanism to restart the search and escape from local optimum solutions [14]. The GRASP construction phase utilizes the sequential construction heuristic proposed for the OVRPTW by Repoussis et al [2]. At each stage of construction a customer is selected for insertion by minimizing a function that combines both myopic and look-ahead greedy criteria. By doing so, it achieves small inter-customer distances within the routes and retains flexibility for subsequent customers to meet their own time window constraints.

Russell et al [15] addressed a real-life newspaper delivery problem that fits the OVRPTW operational framework with zoning constraints. A Tabu Search (TS) [16] metaheuristic is presented to aid in the coordination and synchronization of the production and delivery of multi-product newspapers to bulk delivery locations. The methodology is applied to the newspaper production and distribution problem of a major metropolitan area. Computational experiments on real life data sets shows significant improvement compared to the existing newspaper company's operations.

Finally, Fu et al (2008) [17] proposed a TS algorithm for the OVRP with soft time windows. Two types of soft time window models are examined, while detailed results are also reported for the OVRPTW. The initial solution of TS is generated randomly. That is, a route is built up where the next customer is chosen at random and added to the end of the route unless this violates the capacity or route length constraints, in that case the route is completed at the last added customer and a new route starts until all customers are assigned. The TS is equipped with four neighborhood structures, namely vertex reassignment, vertex swap, 2-opt and tails swap, while an intensification mechanism is introduced to reduce the number of vehicles. For the evaluation of solutions, an objective function equipped with penalties, that show the extent of constraint violations, is utilized.

From a graph theoretical viewpoint, the solution of the OVRPTW consists of finding a set of *Hamiltonian paths*, instead of *Hamiltonian cycles* as is the case in the classical VRPTW. As stated by Syslo et al [18], the minimum Hamiltonian path problem is $\mathcal{NP}$-hard, because it can be converted into an equivalent Hamiltonian cycle problem. For this reason the focus of most researchers have been given towards the design of metaheuristic approaches capable of producing high quality solutions within reasonable computational times. Interested readers may refer to Li et al [19], Fleszar et al [20] and Tarantilis et al [21] regarding the latest algorithmic developments for the OVRP.

This chapter is focused on the development of an efficient, effective and robust population-based bio-inspired algorithm for the OVRPTW. The proposed solution approach follows the basic principles of the neo-Darwinian model of evolution combined with a memory-based trajectory local search algorithm. The main effort is to monotonically evolve a population of individuals in the basis of the well-known $(\mu+\lambda)$-evolution strategy. The latter selects at each generation the μ survivors from the union of parents and offsprings. Given an initial population of μ adequately diversified individuals, at each generation a new intermediate population of λ individuals is produced, utilizing a discrete arc-based representation of individuals combined with a binary vector of strategy parameters. In particular, the reproduction framework works as follows: Offsprings are generated exclusively via mutation (one offspring per population individual) based on the ruin-and-recreate principle. The goal is first to ruin part of the parent individual and preserve only those components (arcs or sequences of arcs) that appear frequently in all other individuals of the population, while the removed arcs are reinserted to the partially ruined solution in a probabilistic heuristic fashion. The selection and combination of components (mutation rate) is dictated by the values of the strategy parameters. At each generation a multi-parent recombination operator is applied to the strategy parameters, based on the frequency of appearance of each arc within the population and the current diversity of the latter. Finally, each offspring is further modified-improved through a short-term memory TS algorithm tuned for intensification local search. For the selection of survivors a deterministic scheme is followed.

The remainder of the chapter is organized as follows: Section 2 introduces the necessary notation. Section 3 describes the proposed solution method and its main algorithmic components-mechanisms. Computational experiments assessing the proof of concept and the value of the suggested approach is presented in Section 4. The chapter concludes in Section 5 offering pointers for further research.

2 Problem Definition and Notation

The OVRPTW can be defined on a incomplete graph $G = (V, A)$, where $V = \{0, 1, \ldots, n\}$ is the node set, $A = \{a_{i,j} : 0 \leq i, j \leq n, i \neq j\}$ is the arc set and the depot is represented by the node 0. Each customer i has a demand d_i, a service time s_i and is associated with a time window $[e_i, l_i]$ during which service must take place. Each route has a start time E_0 and a finish time L_0, if required. There is a known non-negative cost (or distance or time) c_{ij}, associated with the traveling from i to j. Let m denote the minimum necessary number of vehicles, required to service all customers in a feasible manner, each traveling exactly one route and $M = \{1, 2, \ldots, m\}$ be the set of vehicles. The available fleet of vehicles is homogeneous and each vehicle $k \in M$ has a known maximum capacity C and a maximum allowed traveling time bound L.

A solution Ω consists of a set of feasible vehicle routes $\Omega = \{\rho_1, \rho_2, \ldots, \rho_n\}$ that correspond to Hamiltonian paths in G that start from node 0. Feasible vehicle routes must satisfy loading and time window constraints. The former state that the accumulated service up to any customer must not exceed the capacity of the vehicle. Time window constraints state that a vehicle cannot service a customer i before the lower bound e_i and after the upper bound l_i. However, a vehicle can arrive before the earlier time window and wait (waiting time w_i) until the allowable service time begins. Furthermore, there is an upper limit with respect to the total traveling time of each vehicle route. Finally, each customer must be serviced only once by exactly one vehicle. The objective of the OVRPTW is first to determine the minimum number of vehicles required and second the optimum sequence of customers visited by each vehicle, in terms of total distance traveled, such that all customers are served and all constraints are satisfied. Interested readers may refer to [2] for a comprehensive multi-commodity flow model formulation of the OVRPTW.

As described in [2] the OVRPTW covers three types of sub-problems, namely the delivery, the pickup and the simultaneous pick up and delivery. In this chapter, the focus is given on the delivery problem of products from a central depot to a set of customers. Although all these sub-problems share many similarities, the time window constraints induce a specific orientation for each route and thus different solutions occur for each sub-problem. For example, in the delivery sub-problem the start time of each vehicle route is E_0. On the other hand, in the pickup sub-problem the start time of each vehicle route depends on the time window of the first customer visited at the other end of the route. Another issue is the symmetricalness among these sub-problems. If the problem solved is symmetric, that is $c_{ij} = c_{ji}$, the three sub-problems may have the same optimal solution. However, if the problem is asymmetric, that is $c_{ij} \neq c_{ji}$, different optimal solutions occur since the orientation of the sequence of customers served changes [17].

3 Solution Method

Evolutionary Algorithms (EA) represent a large class of problem solving methodologies that mimic the neo-Darwinian model of organic evolution, including Genetic Algorithms (GA)[22], Evolution Strategies (ES)[23] and Evolutionary Programming (EP) [24]. The rationale behind EA is the evolution of a population manipulated by a set of operators (evaluation, selection, recombination and mutation). The evolution is achieved through the reproduction of offspring, on a basis of information exchange among individuals via an iterative process, until some convergence criteria are met. The better an individual performs the greater is the chance to live longer and generate offspring. The latter leads to a penetration of the population with genetic information of well-fitted individuals, while the non-deterministic nature of reproduction leads to a permanent production of novel genetic information.

Initiating from a population of adequately diversified individuals generated via a greedy randomized construction heuristic, our main intuition is to monotonically evolve a population of individuals on the basis of an $(\mu + \lambda)$-ES. Let $P(t)$ denote a population of size μ at generation t and $P'(t)$ a population of λ offspring. According to the $(\mu+\lambda)$-ES, at each generation the μ survivors from the union of parents $P(t)$ and offspring $P'(t)$ are selected to form the next generation population $P(t + 1) = [P'(t) \cup P(t)]$.

A strictly generational course of evolution is adopted, such that one offspring is produced per population individual. Recombination in terms of solution vectors is omitted and new offspring are produced only via mutation. To this end, a discrete representation of individuals is adopted, combined with a binary vector of the so-called strategy parameters. These parameters determine the corresponding mutation rates. Herein, these rates are evolve with respect to a recombination operator, in order to broadcast promising correlations between the object variables of the solution vectors.

The proposed mutation operator exploits the information provided by the strategy parameters and initiates a new offspring from a single parent individual. In particular, the mutation operator works on the basis of the ruin-and-recreate principle. The goal is first to ruin part of the parent preserving only the components dictated by the mutation rates, and then reinsert the removed nodes (customers) via a greedy randomized construction heuristic. For this purpose, the sequential construction heuristic proposed by Repoussis et al [2] is utilized combined with a probabilistic mechanism in order to inject a degree of diversification during the reproduction of offspring.

Furthermore, an effort is made to intensify the search. In particular, each offspring generated via mutation is further modified-improved via a TS algorithm. The proposed implementation utilizes simple edge-exchange neighborhood structures and a short-term memory structure. Finally, in order to ensure a balance between quality and genetic diversity, for the selection of survivors a deterministic scheme is followed considering both the quality of individuals and the corresponding distances (total number of different arcs) with respect to current best-fitted individual of the population. The proposed EA can described as follows:

```
Evolutionary Algorithm
  //Parameters: $\mu, \alpha, tenure, z$//
  $t \leftarrow 0$, $P(t) \leftarrow$INITIALIZATION$(\mu, \alpha)$, EVALUATION$(P(t))$
  While termination conditions Do{
    For all individuals $(\mathbf{x}, \boldsymbol{\sigma}) \in P(t)$ Do{
      $(\mathbf{x}, \boldsymbol{\sigma}') \leftarrow$RECOMBINATION$(P(t), \mathbf{x}, \boldsymbol{\sigma})$}
    For all individuals $(\mathbf{x}, \boldsymbol{\sigma}') \in P(t)$ Do{
      $(\mathbf{x}', \boldsymbol{\sigma}') \leftarrow$MUTATION$(\mathbf{x}, \boldsymbol{\sigma}', \alpha)$
      $(\mathbf{x}', \boldsymbol{\sigma}') \leftarrow$TS$(\mathbf{x}', \boldsymbol{\sigma}', tenure, z)$
      $P'(t) \leftarrow (\mathbf{x}', \boldsymbol{\sigma}')$}
    EVALUATION$(P'(t))$
    $P(t+1) \leftarrow$SELECTION$(P'(t) \cup P(t))$
    $t \leftarrow t+1$
  }
Return $P(t)$
```

The motivation behind the above described EA is to trace "promising" features of the solutions visited during the search and to generate offspring via mutation in order to keep track and inherit these components exhibited in parent individuals. The solution components selected to guide the evolution process are frequently observed arcs.

3.1 Greedy Randomized Construction Heuristic

During the initialization phase, it is important to ensure that the individuals forming the initial population are adequately diversified in order to spread the search over different regions of the solution space. For this purpose, a greedy randomized construction heuristic is utilized, similar to Greedy Randomized Adaptive search Procedures [12]. Such probabilistic construction mechanisms allow the better sampling of the solution space.

The sequential construction heuristic proposed for the OVRPTW by Repoussis et al [2] is adopted herein. The latter belongs to the family of insertion-based heuristics. The rationale is to build iteratively a solution by selecting and adding nodes-customers sequentially to partially built routes, until a feasible solution is generated. The selection of customers and insertion positions is made with respect to greedy criteria. Towards this end, the construction heuristic of [2] incorporates a look-ahead capability to examine the effect of a possible customer insertion to other unrouted customers. The latter property states that at each iteration of construction, a customer is selected for insertion into a route by minimizing a function that combines, through appropriate weights, travel times-distances and time window utilization criteria. By doing so, it achieves small inter-customer distances within the routes and retains flexibility for subsequent customers to meet their own time window constraints.

Two major modifications are introduced compared to the original scheme proposed by Repoussis et al [2]. First, instead of using a sequential scheme of construction, a parallel framework is adopted, considering several partially constructed routes simultaneously. Thus, given a initial set of ρ routes, at each iteration an unrouted customer is selected and it is being inserted between two consecutive customers of a partially constructed route. If at some iteration an unassigned customer cannot be inserted in any of the existing set of routes, an unassigned "seed" customer is identified and an additional route is initialized. The overall procedure is repeated until all customers are assigned to vehicles routes.

The second modification refers to the introduction of a probabilistic mechanism. As described earlier, the parallel construction scheme involves two types of greedy criteria; the first is used to select the customer for insertion and the second to select the insertion position for this particular customer. Towards this end, a greedy randomized procedure is adopted with respect to the customer selection. The proposed mechanism work as follows: At each iteration of the parallel construction scheme, all unrouted customers are ordered according to their adaptive greedy function. Next, α customers with the highest evaluation are placed into a list, called restricted candidate list (RCL). The probabilistic component is determined by randomly choosing one of the best-evaluated customers from the list (equal selection probability among all elements of the list), but not necessary the top of the list. The selected customer is then added to partially constructed solution at its best insertion position according to the second greedy criterion. Obviously, the value of α determines the extent of "randomization" and "greediness" of construction. In the proposed implementation, the RCL is cardinality-based and fixed to a predefined size.

```
**INITIALIZATION**($\mu, \alpha$)
  $P(t) \leftarrow \emptyset$
  **While** $P(t) < \mu$ **Do**$\{$
    $s \leftarrow \emptyset$
    **While** all customers are served **Do**$\{$
      $RCL \leftarrow$ Build Restricted Candidate List($s, \alpha$)
      $c \leftarrow$ Select Customer At Random($RCL$)
      $i, j \leftarrow$ Best Insertion Position($s, c$)
      $s \leftarrow s \cup \{$c$\}$
```

```
        Update Customer Selection Greedy Function($s$)
      }
      $P(t) \leftarrow s$
    }
    Return $P(t)$
```

3.2 Representation of Individuals

Contrary to other EAs, ES simulate the evolution process directly on the level of phenotypes and typically use a real-valued coding of the solution vector. Each individual $I = (\mathbf{x}, \boldsymbol{\sigma})$ consists of an n-dimensional set of object variables $\mathbf{x} \in \mathcal{R}^n$ and a vector of a so-called strategy parameters $\boldsymbol{\sigma} \in \mathcal{R}_n^+$. Mutation is performed at each vector element x_i $(1 \leq i \leq n)$ by adding a normally distributed random value $q_i \sim N(0, \sigma_i)$ with a mean value of zero and a standard deviation σ_i. Thus, an individual is mutated by varying each object of the solution vector according to the following mutation rule $x_i' = x_i + N_i(0, \sigma_i)$.

Clearly, the above described real-valued vector $\mathbf{x}$ of object variables has to be modified in order to capture the discrete combinatorial optimization nature of the OVRPTW. Thus, a set of discrete elements is required. Let a solution $\Omega = \{r_1, r_2, \ldots, r_n\}$ that consists of a set of feasible vehicle routes. Ω can be seen either as a vector of nodes, i.e., $\mathbf{x} = \{0, i, j, \ldots, 0, u, z, \ldots\}$), where $i, j, u, z \in V$, such that the sequence of routes is delimited by the depot nodes. The node vector can be transformed into a vector of arcs, i.e., $\mathbf{x} = \{a_{0,i}, a_{i,j}, \ldots, a_{0,u}, a_{u,z}, \ldots\}$ with $n + k$ ranked discrete coordinates, where k denotes the cardinality of set Ω.

On the other hand, in order to control the mutation step size σ_i, both the solution vector and the strategy parameters needs to be incorporated into the representation of individuals and

must evolve simultaneously. This will enable a learning process which tends to lead to better solutions and to more suitable values for the strategy parameters. In the proposed representation, the vector of strategy parameters $\boldsymbol{\sigma}$ is defined as follows: Each element x_i of $\mathbf{x}$ is allocated to exactly one element σ_i of $\boldsymbol{\sigma}$. Since the proposed $\mathbf{x}$ is composed of discrete elements, a binary vector $\boldsymbol{\sigma} \in \{0, 1\}^{n+k}$ is considered instead. Our main intuition is the strategy parameters to contain encoded information with respect to the individual's components-arcs that will be subject to mutation. On the basis of the above, if the value of the element σ_i is 1, this indicates that the corresponding element x_i of the individual will be inherited directly to the next generation offspring; otherwise it will be subject to mutation.

3.3 Recombination and Mutation Operators

As described earlier, the values of the strategy parameters guide the overall reproduction process and determine the corresponding mutation rates. In order to facilitate the self-adaptation of the strategic parameters a multi-parent recombination operator is applied to the strategy parameters of each solution vector. The rational behind the proposed operator is to select and extract promising solution components from each individual, based on the appearance frequency of each arc and the current diversity of the population.

Initially, the frequency of appearance $\theta_{i,j}$ of each arc $a_{i,j}$ is determined. Next, a threshold frequency of acceptance $\kappa \in \mathcal{N}^+$ is defined with respect to the size and the diversity of the population. The threshold κ indicates the least number of population individuals that must include a particular arc. To this end, the arc $a_{i,j}$ of a solution vector $\mathbf{x}$ is considered as promising, if $a_{i,j}$ is exhibited in more than κ population individuals. Given the appearance frequencies of each arc and the threshold, at each generation the elements $\sigma_i \in \boldsymbol{\sigma}$ of each individual is set to 1, if $\theta_{i,j}$ of the corresponding element $x_i \in \mathbf{x}$ is greater than or equal to κ; otherwise it is set to 0.

Let $D_s^{s'}$ denote the level of similarity between individuals s and s', expressed as the total number of common arcs between them:

$$D_s^{s'} = \sum_{a_{i,j} \in s, s'} \xi_{ij}, \tag{1}$$

where

$$\xi_{ij} = \begin{cases} 1 \text{ if } a_{i,j} \text{ is exhibited in both } s \text{ and } s' \\ 0 \qquad\qquad \text{otherwise} \end{cases} \tag{2}$$

Let also $M_P^{s'}$ denote the median position (average similarity) of all individuals $s \in P - \{s'\}$ relatively to s' as follows:

$$M_P^{s'} = \frac{\sum_{s \in P - \{s'\}} D_s^{s'}}{|P| - 1} \tag{3}$$

A reasonable threshold κ can be derived from the average similarity of the population individuals $M_P^{s_*}$ with respect to the best-fitted solution $s_* \in P$ divided by the total number of dimensions ψ of the solution vector $\mathbf{x}_{s_*}$ (total number of arcs), times the population size μ.

$$T = \frac{M_P^{s_*}}{\psi}\mu \tag{4}$$

The adaptive threshold rule 4 shows that the larger the similarity among individuals the larger is the threshold. Assuming that during early stages of the evolution large distances among individuals are observed, κ is forced to small values (low mutation rates) in order to ensure convergence reliability. Contrary, over the course of evolution and as individuals getting more and more similar and tend to converge to the best encountered solution, κ will progressively increase (high mutation rates) in order to diversify the search.

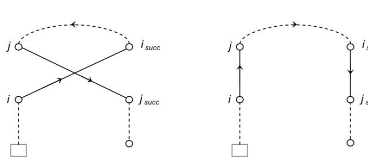
The strategy parameters dictate which elements of the solution vector will be removed and preserved during mutation. In the proposed evolution scheme a generational course of evolution is followed, such that one offspring is produced per population individual. The goal is first to remove a set of customers from the parent individual, and then reinsert back the removed customers to the partially ruined individual in a probabilistic-heuristic fashion. The mutation framework can be described as follows. Initially, all elements $a_{i,j}$ of the solution vector $\mathbf{x}$ in which the corresponding strategy parameters equals 0 are removed from $\mathbf{x}$. Next, the removed customers are declared as unrouted and the probabilistic construction heuristic described is Section 3.1 is triggered. Finally, the resulting partially ruined solution vector is re-constructed iteratively until all removed customers are routed.

3.4 Tabu Search

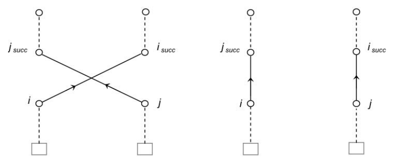
TS explores the solution space by moving at each iteration from a solution s to the best admissible solution s' (best-accept strategy) in a subset $\Phi_y(s)$ of a neighborhood structure y. Furthermore, it allows solutions to deteriorate from one iteration to the next in order to escape from local optimum solutions and uses the history of the search (short term memory) to avoid cycling. The short term memory keeps track of recently visited solutions and forbids moves towards them. The neighborhood $\Phi_y(s)$ of the current solution s is therefore restricted to non-tabu solutions, since solutions possessing attributes of recently visited ones are temporarily declared "tabu" and stored in the tabu list. The new solution s' is then added to the tabu list and the oldest is removed from the list. The duration a solution remains tabu is called tabu $tenure$, while the tabu status of a neighboring solution can only be overridden by predefined aspiration criteria. The overall procedure iterates until some termination conditions are met and the best encountered local optimum solution s_* is returned upon termination.

The neighborhood structures y used within the proposed implementation are based on traditional edge-exchange local moves (see for details [25]), namely inter- and intra-route 2-Opt, 1-0 Relocate and 1-1 Exchange (see Figure 1). The selection of neighborhood structures is purely stochastic, while equal selection probability level is assigned to all structures. At each iteration, given the allowed set of neighbors Φ_y(s), the best admissible neighbor s' replaces the current solution s, while the forward and reversal attributes of the corresponding local move are stored within the tabu list. During the exploration of the neighboring space, the typical aspiration criterion is followed, that is, higher evaluation of a neighbor compared to the current best solution found during the exploration of the solution space. Finally, the termination condition z bounds the maximum number of TS iterations without observing any further improvement.

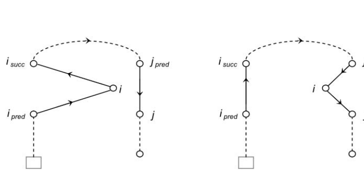
```
Tabu Search($tenure, z$)
  $counter$←0,$s_*$←$s$
  While $counter \leq z$ do{
    $y$←Select At Random()
    $\Phi_y(s)$←Build Allowed Set($s, y$)
    $s \leftarrow \min_{s' \in \Phi_y(s)}\{f(s')\}$
    Update Tabu List($s, tenure$)
    If($f(s) < f(s_*)$) do{
      $counter$←0, $s_*$←$s$}
    Else $counter$←$counter$+1
  }
  $s$←$s_*$
Return $s$
```



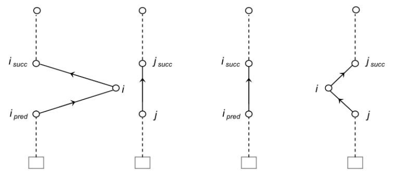
(a) Open 2-Opt



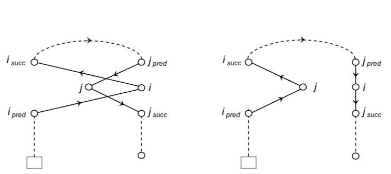
(b) Open 2-Opt*



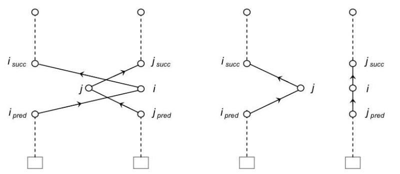
(c) Open Intra-Relocate



(d) Open Inter-Relocate



(e) Open Interchange



(f) Open Exchange

Fig. 1. The Neighboring Space

In the literature, several search techniques have been proposed to speed up the neighborhood examination at no quality loss. When handling edge-exchange neighborhood structures it is not necessary to evaluate the objective function for each neighboring solution, but only to keep track of the differences with respect to the edges (arcs) that change state (being deleted or added). Additionally, it is unnecessary to evaluate the whole neighborhood at each step, since the effect of such edge-exchanges is limited to the routes they modify, which are at most 2 in our case. Thus, once an move is executed, only those exchanges involved within the modified routes are re-evaluated. Finally, it is sufficient to memorize only the best non-tabu exchange of each structure.

Another important factor for the effective evaluation of local moves is to prune infeasible solutions and to avoid unnecessary feasibility checks (e.g., using lexicographic search). As suggested in [25] one can reduce the computational complexity of feasibility checks if sequences of nodes are handled just as single nodes, provided that time windows and travel times are suitably defined. In the proposed implementation, we keep track at each node the vehicle departure time. The latter allows to consider only insertion positions between two consecutive customers i and j that are compatible for a segment of nodes $\{q, \ldots, u\}$ with respect to the corresponding time windows and vehicle's departure times. For example, the cost and feasibility of the insertion are computed only if the vehicle departure time plus the distance $d_{i,q}$ is smaller than the latest allowed service time l_q and the deadline l_j is greater than the allowed earliest start of service e_u.

Finally, a major concern during the local search is to eliminate any surplus vehicle routes. Starting from a solution which makes use of k vehicles, surplus routes are repeatedly eliminated, if possible, until the lower bound in terms of fleet size is reached.

More specifically, the vehicle routes with the smallest cardinality are removed iteratively from the routing plan and all corresponding customers are placed into a waiting list. At each iteration of TS, the possibility of the feasible relocation of the waiting listed customers is examined to the vehicle routes modified by the local search. If one or more feasible relocation moves (insertion positions) are found, these are immediately executed and the corresponding solution is denoted as a new TS local optimum solution. The above procedure is repeated until all waiting listed customers are served. The major advantage of this simple route elimination procedure is that TS deals only with feasible solutions whether the best solution found upon termination is sub-optimal or optimal in terms of fleet size.

3.5 Evaluation and Selection

The evaluation and selection of survivors at each generation affects significantly the performance of the proposed EA. The selection scheme followed in the proposed implementation is deterministic and elitist towards high quality individuals. Furthermore, it favors adequately dissimilar (with respect to current best-fitted individual of the population) and highly evaluated individuals. Both these properties enable the search to spread towards distant regions of the solution space and ensures the evolvability of the population.

During evaluation, an effort is made to account attractiveness in terms of number of vehicles deployed, distance traveling and level of dissimilarity among individuals. Initially, all individuals s of $P(t)$ are temporarily placed into $P(t+1)$. Then, each offspring s' of $P(t)'$ is compared to the corresponding parent individual s, and if s' performs better than s, $P(t+1)$ is updated accordingly. Next, the best and the worst fitted individuals s_* and s_w of $P(t{+}1)$ are identified and the median similarity $M^{s*}_{P(t+1)}$ is determined. Subsequently, the remaining offsprings s' of $P'(t)$ (if any) are competing against individuals currently placed into $P(t+1)$ with respect to the level of similarity. More specifically, if s' performs better than s_r and its level of similarity $D^{s*}_{s'}$ is lower than the current $M^{s*}_{P(t+1)}$, then s' replaces s_r. In addition, if s' performs better than any individual $s \in P(t+1)$ and its level of similarity $D^{s*}_{s'}$ is lower than D^{s*}_{s} of s, then s replaces s'.

4 Computational Results

4.1 Benchmark Data Sets

For the evaluation of the proposed EA various computational experiments are conducted using the benchmark data sets proposed by Solomon [27] for the VRPTW. These data sets consists of 56 problem instances each categorized into six different sets, R1, C1, RC1, R2, C2 and RC2. Each set contains between eight and twelve 100-node problems over a service area defined on a 100x100 grid. The Cartesian coordinates of customers in problem sets R1 and R2 are randomly generated from a uniform distribution. In contrast, sets C1 and C2 have clustered customers. Finally, sets RC1 and

RC2 contain semi-clustered customers, i.e., a combination of clustered and randomly distributed customers. Note that sets R1, C1 and RC1 have tight time windows, short scheduling horizons and a vehicle capacity of 200 units. Contrary, the problem instances of sets R2, C2, RC2 have longer scheduling horizons and vehicle capacity ranging from 100 to 1000 units, thus allowing a larger number of customers to be served per route. These data sets can be used directly for the evaluation of the delivery sub-problem of the OVRPTW.

4.2 Parameter Settings and Influence of Components

The proposed EA introduces four parameters, namely the size α of the RCL, the tabu tenure, the maximum number of TS iterations z without observing any further improvement and the size μ of the population. Fortunately, most of these parameters are relatively insensitive to the characteristics of the problems considered. Thus, using only small adjustments one may determine very well performing settings with modest effort. Below, the sensitivity of parameters is discussed and suitable value ranges are provided.

Parameter α

It is very important to ensure that the individuals forming the initial population are adequately diversified, in order to avoid to the extend possible the premature converge of the evolution process. As discussed earlier, the selection and mutation operators of the proposed EA maintain the diversity of the population and allow the penetration of novel genetic information respectively. However, the degrees of freedom of the evolution are significantly reduced if the initial population contains similar individuals.

Parameter α (size of RCL) determines the extent of randomization and greediness of construction. Repoussis et al [26] conducted a series of experiments for the VRPTW and suggests that values of $\alpha > 10$ can generate adequately distant solutions (expressed in terms of non-common arcs). Our findings using the construction of [2] are similar. A range of values between 6 and 12 is found to be suitable for most OVRPTW problem instances.

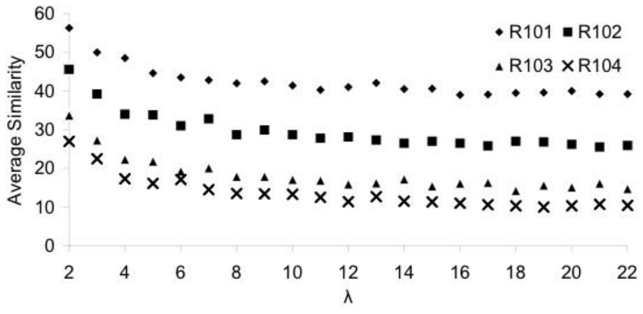


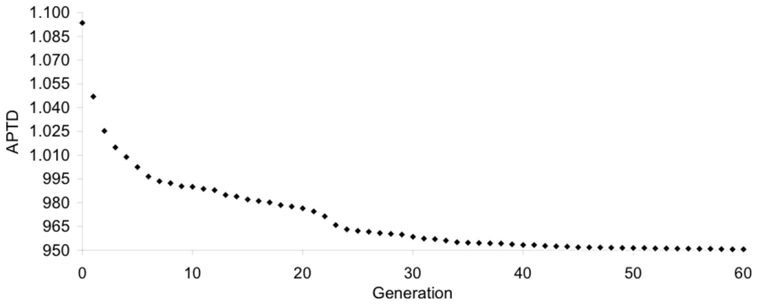
Fig. 2. RCL size α vs $M_P^{s'}$

Figure 2 illustrates the average similarity of a population of individuals produced with different values of α for several problem instances with different time window constraints. Each point of the figure corresponds to the average similarity $M_P^{s'}$ of all population individuals $s \in P$ ($\mu = 30$) produced assuming a predefined α with respect to the pure greedy solution s' found with $\alpha = 1$. In order for the analysis to be fair and objective all solutions are produced using fixed and equal weighted parameter settings. As moving towards problem instances with tighter time window constraints, it is hard to produce distant solutions even for large values of α. Furthermore, the average similarity reaches a lower limit as α increases. Obviously, it is essential to trigger the evolution process initiating from a population of non-similar high quality individuals. However, one should note as the effect of randomization is getting larger, the loss in terms of solution quality also increases and the gain regarding population diversity is small. On the basis of the above, $\alpha = 8$ found to provide a good compromise between diversity and quality of the initial population for most problem instances.

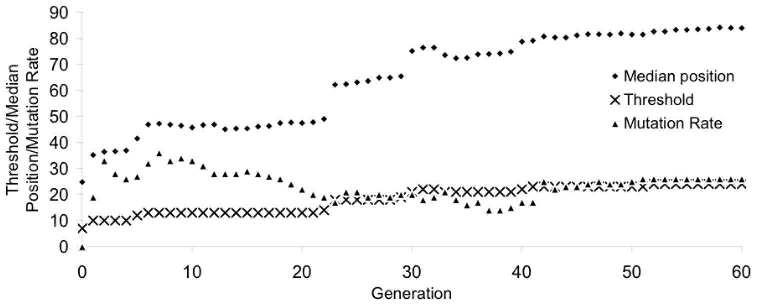
Figure 3 provides an insight of the importance of seeding the EA with solutions produced via the proposed greedy randomized construction heuristic. In particular, two simulation runs are performed for problem instance R205 with different settings regarding parameter α. Figures 3 (a) and (c) describe the average population traveling distance cost (APTD) of each generation over the course of evolution. Given that the selection strategy adopted is elitist in terms of solution quality, APTD converges towards the cost of the best-fitted individual. In the first case ($\alpha = 8$), the best solution is found at the 28^{th} generation and after 148 seconds with a distance traveling cost equal to 943,33. In the second case ($\alpha = 2$), the best solution is obtained during the last stages (952,34 at the 54^{th} generation) of the evolution process. The inferior performance of the latter can be explained as follows. Figures 3 (b) and (d) illustrate the behavior of three key elements, namely the threshold frequency κ, the median position of P with respect to the current best-fitted individual of the population M_P^{best} and the actual mutation rate expressed as the average of the total number of customers mutated-modified (changed positions) for the reproduction of offspring. Evidently, parameter α had a major impact especially during the early stages of evolution. Its small value results in less diversified individuals and rapid increase of the average similarity without the analogous solution quality output. The latter also caused non-smooth fluctuations of the actual mutation rate and premature convergence. On the other hand, the larger value of α provides a better sampling of the solution space in the short term and ensures a reasonable convergence velocity to high quality individuals in the long term.

Construction Parameters

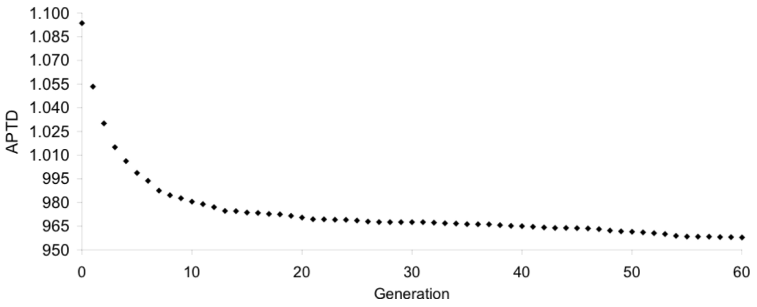
Apart from α, a factor dominating the quality of solutions produced via the greedy randomized parallel construction heuristic is the value of the parameters associated with the greedy functions (see for details [2]). Although these parameters are interrelated, their settings heavily depend on the problem characteristics. Furthermore, as described in [28] it is hard either to determine a robust value for each one of them or to find an evident correlation among them, which would provide good results for all problem instances. In our study, the interrelated construction parameters associated with the greedy functions (customer selection and route insertion criteria) are randomly selected



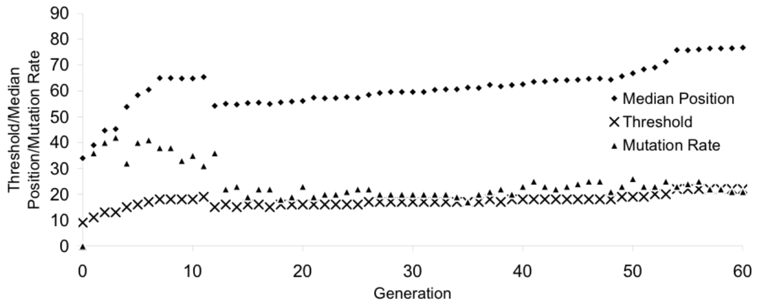
(a) Average population traveling distance cost - R205 ($\alpha = 8$)



(b) Threshold, Median Position & Mutation Rate - R205 ($\alpha = 8$)



(c) Average population traveling distance cost - R205 ($\alpha = 2$)



(d) Threshold, Median Position & Mutation Rate - R205 ($\alpha = 2$)

Fig. 3. Performance analysis over 60 generations regarding problem instance R205 ($\mu = 30, tenure = 30, z = 60$)

within the parameter's value ranges suggested in [2]. Note that such an approach allows the production of more distant solutions with the same average quality compared to the experiments described earlier with fixed construction parameter settings.

Local search

The proposed TS implementation is very simple and invokes two parameters, namely the size of the short term memory (tabu tenure) and the number of TS iterations without observing any further improvement. Regarding tabu tenure, a range between 10 and 40 is mostly used by standard TS implementations of the literature for intensification search. Since both the forward and reversal attributes of local moves are stored in the proposed implementation, tabu tenure is set equal to 20. It is also important to control the overall amount of time invested for local search. One may expect that large values of z will increase the efficiency. However, a balance between is needed, since large values of z may result in excessive computational time consumption. Towards this end, a range between 30 and 120 found to provide a good compromise.

In the literature, it is very common to combine evolutionary algorithms with local search. Such a combination is completely cooperative since it enables the search to intensify within regions of the solution space close to local optimum solutions produced during the evolutionary phase. In the case of the OVRPTW, the importance of local search is even greater due to the fact that it is the only mechanism that minimizes the number of vehicles. Indeed, minimization of the fleet size cannot be simulated explicitly via the evolution mechanism of the proposed EA. Overall, the role of local search within the proposed EA is threefold. First, it intensifies the search, second it speeds up the convergence and third it minimizes the fleet size. As such, the synergy between EA and TS increases significantly the effectiveness and the accuracy of the proposed solution method.

Population Size

In general terms, the size μ of the population affects the overall convergence reliability and velocity of the proposed EA. Small values of μ (less than 15) are not sufficient to handle the amount of information included into the individuals. On the other hand, although large values (greater than 50) fully exploit the large diversity of strategy parameters, this is made at the expense of convergence velocity. During our experiments μ is fixed to 30.

Simulation Runs and Termination Conditions

Given on the above defined parameter value ranges, several intuitively selected settings were experimentally tested, choosing the one that yielded the best average output over all instances. All computational experiments reported in subsequent sections are produced using the following parameter settings, $\mu = 30$, $z = 60$, $tenure = 20$ and $\alpha = 8$. Furthermore, in order to identify statistically significant differences in the results, 30 simulation runs are performed for each problem instance. Finally, a maximum allowed CPU time consumption of 600 seconds is assumed for each run on a Pentium IV 3.0GHz PC. The latter limit allows on average the evolution of more than 80 generations.

4.3 Comparative Analysis

Computational results for the OVRPTW are ranked according to a hierarchical objective function. The primary objective is to minimize the fleet size, and for the same number of vehicles, the secondary objective is to minimize the total traveled distance. However, one should note that these two objectives are often conflicting since the reduction of the total number of vehicles may increase the total traveling distance. Thus, comparisons among different algorithms are valid only if the above described objective function is followed.

In order to assess the proof of concept of the proposed EA, computational experiments are also conducted without invoking the proposed evolutionary mechanism. Clearly, if the values of the strategy parameters are fixed to zero at all times, the proposed solution method degenerates to a simple greedy randomized multi-start trajectory local search algorithm (GRMSTS). To this end, the comparison between the proposed EA and the GRMSTS with the same configurations and settings will indicate the efficiency and effectiveness of the evolution mechanism.

Table 1. Detailed results for data sets R1, C1 and RC1

Set	**EA**				**GRMSTS**			**Repoussis et al. [2]**			**Fu et al. [17]**		
	TD	NV	MTD	MNV	%D	TD	NV	%D	TD	NV	%D	TD	NV
R101	1192,85	19	1192,85	19	-0,38	1197,42	19	-24,06	1479,9	19	-1,21	1207,23	19
R102	1079,39	17	1081,65	17	-2,50	1106,42	17	-	1501,59	18	-2,83	1109,97	17
R103	1016,78	13	1017,28	13	-2,28	1039,94	13	-26,04	1281,52	13	-2,48	1041,96	13
R104	869,63	9	844,32	9,53	-	800,17	10	-	1021,73	10	-	764,84	10
R105	1055,04	14	1055,98	14	-0,58	1061,15	14	-21,89	1285,94	14	-1,62	1072,15	14
R106	1000,95	12	1001,04	12	-1,80	1018,92	12	-29,37	1294,88	12	-6,01	1061,14	12
R107	912,99	10	915,82	10	-	897,05	11	-	1102,7	11	-5,61	964,23	10
R108	760,30	9	760,30	9	-1,97	775,26	9	-	898,94	10	-3,76	788,85	9
R109	934,53	11	934,77	11	-	927,86	12	-	1150,42	12	-6,40	994,36	11
R110	846,49	10	851,01	10	-	899,67	11	-	1068,66	12	-10,28	933,55	10
R111	895,21	10	902,45	10	-	913,31	11	-	1120,45	11	-6,30	951,59	10
R112	811,73	9	814,33	9,47	-	814,91	10	-	966,64	10	-	755,35	10
C101	556,18	10	556,18	10	0,00	556,18	10	-27,60	709,71	10	0,00	556,18	10
C102	556,18	10	556,18	10	0,00	556,18	10	-86,45	1036,98	10	0,00	556,18	10
C103	556,18	10	556,18	10	0,00	556,18	10	-106,21	1146,89	10	0,00	556,18	10
C104	555,41	10	555,41	10	0,00	555,41	10	-63,32	907,08	10	-0,01	555,47	10
C105	556,18	10	556,18	10	0,00	556,18	10	-24,97	695,08	10	0,00	556,18	10
C106	556,18	10	556,18	10	0,00	556,18	10	-56,75	871,81	10	0,00	556,18	10
C107	556,18	10	556,18	10	0,00	556,18	10	-22,22	679,76	10	0,00	556,18	10
C108	555,80	10	555,80	10	0,00	555,80	10	-51,81	843,74	10	0,00	555,80	10
C109	555,80	10	555,80	10	0,00	555,80	10	-34,21	745,95	10	0,00	555,80	10
RC101	1227,37	14	1228,76	14	-	1183,50	15	-	1399,15	15	-	1170,36	15
RC102	1203,05	12	1205,65	12	-	1104,95	13	-	1305,52	13	-	1084,34	13
RC103	923,15	11	927,62	11	-2,98	950,68	11	-	1180,77	12	-0,96	931,98	11
RC104	787,02	10	789,21	10	-5,05	826,76	10	-	1046,16	11	-2,84	809,37	10
RC105	1195,20	13	1220,96	13	-	1152,41	14	-	1331,22	14	-	1201,82	14
RC106	1095,65	11	1113,20	11	-	1024,98	12	-	1258,37	12	-	1003,17	12
RC107	861,28	11	862,44	11	-0,82	868,31	11	-23,50	1063,66	11	-3,79	893,94	11
RC108	831,09	10	832,05	10	-	863,52	11	-	1026,57	11	-16,44	967,73	10

On the basis of the above, Tables 1 and 2 provide the best and the average results obtained, among 30 simulation runs, for each problem instance with respect to the Solomon's RC1, R1, C1 and RC2, R2, C2 data sets, respectively. In addition, the results provided by the TS algorithm of Fu et al [17] and the construction heuristic of Repoussis et al [2] are also reported. Abbreviations TD, NV, MTD and MNV stands for the traveling distance, the number of vehicles routes, the mean traveling distance and the mean number of vehicles, respectively. Columns %D report the percentage deviation of each methods with respect to the proposed EA. The latter has been calculated as follows: $100 * (TD - EA)/EA$. Finally, note that in cases where the total number of vehicles is different none comparison is made.

Clearly, the proposed EA outperforms the TS of [17] producing higher or at least equal quality solutions with cost reductions up to 16,44%. It is also worth to notice the efficiency of the proposed EA in reducing both the total number of vehicles and the associated delivery costs. More specifically, the total number of vehicles produced is equal to the corresponding best known total number of vehicles of the well-known VRPTW on Solomon data sets. Furthermore, the EA matches or improves (up to 5,05) the best solutions obtained by GRMSTS, with the same configuration settings and computational burdens. The latter strongly indicates the efficiency of the evolutionary mechanism of the proposed EA. It also illustrates the effective cooperation of a trajectory local search algorithm, tuned for intensification search, within an population-based evolutionary solution framework.

Table 2. Comparison of the proposed to the [3] methodology

Set	**EA**				**GRMSTS**			**Repoussis et al. [2]**			**Fu et al. [17]**		
	TD	NV	MTD	MNV	%D	TD	NV	%D	TD	NV	%D	TD	NV
R201	1182,43	4	1182,43	4	-0,50	1188,35	4	-37,12	1621,38	4	-1,29	1197,74	4
R202	1149,59	3	1150,07	3	-0,14	1151,16	3	-40,53	1615,53	3	-0,45	1154,81	3
R203	889,12	3	894,34	3	-1,25	900,19	3	-50,25	1335,94	3	-2,16	908,31	3
R204	801,46	2	803,54	2	-3,68	830,96	2	-	1021,57	3	-2,90	824,71	2
R205	943,33	3	950,21	3	-1,91	961,37	3	-38,39	1305,46	3	-2,53	967,15	3
R206	869,27	3	870,96	3	-1,63	883,44	3	-44,48	1255,91	3	-1,16	879,37	3
R207	857,08	2	865,93	2,77	-5,74	906,27	2	-	1109,51	3	-3,41	886,31	2
R208	700,53	2	700,67	2	-1,46	710,73	2	-	928,23	3	-1,57	711,53	2
R209	851,69	3	853,86	3	-2,74	874,99	3	-55,19	1321,73	3	-1,64	865,66	3
R210	892,45	3	894,38	3	-2,60	915,69	3	-53,40	1368,97	3	-0,99	901,32	3
R211	886,90	2	887,41	2	-2,41	908,30	2	-	994,01	3	-1,74	902,32	2
C201	548,51	3	548,51	3	0,00	548,51	3	-22,26	670,63	3	0,00	548,51	3
C202	548,51	3	548,51	3	0,00	548,51	3	-	971	4	-0,30	550,13	3
C203	548,13	3	548,13	3	0,00	548,13	3	-	1281,73	4	0,00	548,13	3
C204	547,55	3	547,55	3	0,00	547,55	3	-	1245,54	4	0,00	547,55	3
C205	545,83	3	545,83	3	0,00	545,83	3	-27,23	694,44	3	0,00	545,83	3
C206	545,45	3	545,45	3	0,00	545,45	3	-45,94	796,02	3	0,00	545,45	3
C207	545,24	3	545,24	3	0,00	545,24	3	-33,02	725,29	3	0,00	545,24	3
C208	545,28	3	545,28	3	0,00	545,28	3	-26,80	691,4	3	0,00	545,28	3
RC201	1303,73	4	1309,06	4	-1,03	1317,10	4	-36,62	1781,1	4	-3,05	1343,44	4
RC202	1321,43	3	1329,52	3	-	1251,37	4	-	1630,95	4	-4,50	1380,89	3
RC203	993,29	3	995,02	3	-1,71	1010,32	3	-42,55	1415,89	3	-1,27	1005,95	3
RC204	718,97	3	719,92	3	-3,17	741,77	3	-60,95	1157,2	3	-0,37	721,60	3
RC205	1189,84	4	1190,67	4	-0,96	1201,23	4	-44,33	1717,32	4	-1,61	1208,95	4
RC206	1091,79	3	1092,09	3	-1,53	1108,45	3	-	1457,12	4	-0,43	1096,49	3
RC207	998,70	3	1005,32	3	-3,12	1029,81	3	-63,07	1628,57	3	-2,81	1026,77	3
RC208	769,40	3	772,76	3	-1,33	779,63	3	-49,62	1151,17	3	-1,92	784,18	3

Table 3. Summary of results obtained over 30 simulation runs

Data Set		Best	Worst	Average
$R1$	MNV	11,92	12,08	12,00
	MTD	947,99	946,86	946,54
$C1$	MNV	10,00	10,00	10,00
	MTD	556,01	556,01	556,01
$RC1$	MNV	11,50	11,50	11,50
	MTD	1015,48	1035,81	1022,43
$R2$	MNV	2,73	2,82	2,80
	MTD	911,26	917,90	913,65
$C2$	MNV	3,00	3,00	3,00
	MTD	546,81	546,81	546,81
$RC2$	MNV	3,25	3,25	3,25
	MTD	1048,39	1057,37	1052,21

Similar are the figures for the results obtained with respect to the R2, RC2 and C2 data sets. Table 2 reports the results in the same fashion as those shown in Table 1. In all cases, EA outperforms the TS of [17] producing higher quality solutions with cost reductions up to 4,50%. Additionally, in several instances the EA managed to reduce the number of vehicles with respect to GRMSTS, while significant cost reductions are also achieved.

Table 3 summarizes the results obtained by the proposed EA. For each data set the best, worst and average mean number of vehicles (MNV) and mean traveling distance (MTD) is provided. Considering the clustered data sets C1 and C2, each run yield the same results. Minor also are differences regarding the data sets RC1 and RC2. In particular, the proposed EA obtained the optimum number of vehicles for each problem instance at all times, while a maximum standard deviation of 3,6 and 2,1 is occurred, in terms of distance traveled, respectively. Similar are the figures considering the data sets R1 and R2. Differences are mainly observed with respect to the total number of vehicles deployed. More specifically, during some simulation runs the proposed method found sub-optimal solutions regarding problem instances R104, R112 and R207 (see for details the fifth column of Tables 1 and 2).

Lastly, in order for the competition to be fair and objective the computational running times of each approach must be investigated. As expected, the construction heuristic of Repoussis et al. [2] is the most time effective method. However, in order to find suitable parameter settings the actual computational time consumption may range from a few seconds to a few minutes. On the other hand, the TS of Fu et al [17] was implemented on a PIV 3.00 GHz machine similar to EA. As mentioned in [17] each execution required on average 30 seconds and the best solution reported is obtained from 20 simulation runs. On the basis of the above, the proposed EA seems to be slower compared to the TS of [17]. However, it improves upon all best known solutions of the literature with fairly adequate computational burdens and fixed parameters over all problem instances.

4.4 Conclusions

This chapter presented a bio-inspired solution method for the OVRPTW. The proposed method manipulates a population of individuals on the basis of an $(\mu + \lambda)$-ES. Following a strictly generational course of evolution and a deterministic selection scheme, offspring are produced via mutation out of arcs extracted from parent individuals. A discrete arc-based representation of individuals is utilized, combined with a binary vector of strategy parameters that dictated the mutation rates. To this end, a recombination operator is applied to the strategy parameters, based on the appearance frequency of each arc and the diversity of the population. Each resulting offspring is further improved via a short-term memory TS tuned for intensification search. Results on benchmark data sets of the literature demonstrated the competitiveness and accuracy of proposed EA. Compared to other approaches, it proved to be highly efficient and effective matching or improving all best known solutions. Furthermore, it obtained the minimum published fleet size for all problem instances. Finally, the relatively few and instance independent parameters and the fairly simple algorithmic structure, indicate the applicability of the proposed method. A worth-pursuing research direction is towards the investigation of more sophisticated recombination operators and advanced pattern-identification mechanisms.

Acknowledgements

This work is supported by the General Secretary of Research and Technology under contract EP-1253-01. The author is indebted to the editors of the volume Professors Francisco Pereira and Jorge Tavares for their accurate comments and suggestions that helped improve the work presented in this chapter. Thanks also deservers Professor Zhuo Fu for providing his detailed computational results.

References

1. Tarantilis, C.D., Kiranoudis, C.T., Ioannou, G., Prastacos, G.P.: J. Oper. Res. Soc. 56, 588–596 (2005)
2. Repoussis, P.P., Tarantilis, C.D., Ioannou, G.: J. Oper. Res. Soc. 58, 355–367 (2006)
3. Fu, Z., Englese, R.W., Li, L.Y.O.: J. Oper. Res. Soc. 56, 267–274 (2005)
4. Sariklis, D., Powell, S.: J. Oper. Res. Soc. 51, 564–573 (2000)
5. Tarantilis, C.D., Diakoulaki, D., Kiranoudis, C.T.: Eur. J. Oper. Res. 152, 437–453 (2004a)
6. Bodin, L., Golden, B., Assad, A., Ball, M.: Compt. Oper. Res. 10(2), 63–211 (1983)
7. Clarke, G., Wright, J.W.: Oper. Res. 12, 568–589 (1964)
8. Atkinson, J.B.: J. Opl. Res. Soc. 41, 703–711 (1990)
9. Fu, Z., Wright, M.: J. Opl. Res. Soc. 45, 384–391 (1994)
10. Li, L.Y.O., Fu, Z.: J. Opl. Res. Soc. 53, 552–558 (2002)
11. Repoussis, P.P., Paraskevopoulos, D.C., Zobolas, G., Tarantilis, C.D., Ioannou, G.: Eur. J. Oper. Res (2008) doi:10.1016/j.ejor.2007.11.004
12. Feo, T., Resende, M.G.C.: J. Global. Optim. 6, 109–154 (1995)
13. Mladenović, N., Hansen, P.: Comput. Oper. Res. 24, 1097–1100 (1997)
14. Paraskevopoulos, D.C., Repoussis, P.P., Tarantilis, C.D., Ioannou, G., Prastacos, G.P.: J. Heuristics (2008), doi:doi10.1007/s10732-007-9045-z

15. Russell, R., Chiang, W.C., Zepeda, D.: Eur. J. Oper. Res. 35, 1576–1588 (2008)
16. Glover, F.: Comp. Oper. Res. 13, 533–549 (1986)
17. Fu, Z., Li, L.Y.O., Eglese, R.: An Improved Tabu Search Heuristic for the Open Vehicle Routing Problem with Soft Time Windows. Technical Report. Lancaster University Management School, Lancaster, UK (2008)
18. Syslo, M.M., Deo, N., Kowalik, J.S.: Discrete optimization algorithms with pascal programs. Prentice-Hall Inc., New Jersey (1983)
19. Li, F., Golden, B., Wasil, E.: Compt. Oper. Res. 34, 2918–2930 (2007)
20. Fleszar, K., Osman, I.H., Hindi, K.S.: Eur. J. Oper. Res. (2007), doi:doi.10.1016/j.ejor.2007.06.064
21. Tarantilis, C.D., Ioannou, G., Kiranoudis, C.T., Prastacos, G.P.: RAIRO 38, 345–360 (2004)
22. Holland, J.H.: Adaptations in Natural and Artificial Systems. Michigan Press (1975)
23. Rechenberg, I.: Evolutionsstrategie. Fromman-Holzboog, Stuttgart (1973)
24. Fogel, D.B.: Evolutionary Computation: The Fossil Record. IEEE Press, Piscataway (1998)
25. Kindervater, G.A.P., Savelsbergh, M.W.P.: Vehicle Routing: handling edge exchanges. In: Aarts, E., Lenstra, J.K. (eds.) Local Search in Combinatorial Optimization. Wiley Publications, UK (1998)
26. Repoussis, P.P., Paraskevopoulos, D.C., Tarantilis, C.D., Ioannou, G.: A Reactive Greedy Randomized Variable Neighborhood Tabu Search for the Vehicle Routing Problem with Time Windows. In: Almeida, F., Blesa Aguilera, M.J., Blum, C., Moreno Vega, J.M., Pérez Pérez, M., Roli, A., Sampels, M. (eds.) HM 2006. LNCS, vol. 4030, pp. 124–138. Springer, Heidelberg (2006)
27. Solomon, M.M.: Oper. Res. 35, 254–265 (1987)
28. Bräysy, O.: INFORMS J. Comput. 15, 347–368 (2003)

Using Genetic Algorithms for Multi-depot Vehicle Routing

Beatrice Ombuki-Berman[1] and Franklin T. Hanshar[2]

[1] Brock University, Ontario, Canada
bombuki@brocku.ca
[2] University of Guelph, Ontario, Canada
fhanshar@uoguelph.ca

Summary. Efficient routing and scheduling of vehicles has significant economic implications for both the public and private sectors. Although other variants of the classical vehicle routing problem (VRP) have received much attention from the genetic algorithms (GAs) community, we find it surprising to identify only one GA in the literature for the fixed destination multi-depot vehicle routing problem (MDVRP). This paper aims to bridge this gap by proposing an application of genetic algorithms approach for MDVRP. The proposed GA employs an indirect encoding and an adaptive inter-depot mutation exchange strategy for the MDVRP with capacity and route-length restrictions. The algorithm is tested on a set of 23 classic MDVRP benchmark problems from 50 to 360 customers. Computational results show that the approach is competitive with the existing GA upon which it improves the solution quality for a number of instances. A comparison of the GA's approach with other non-GA approaches show that although GAs are competitive for the MDVRP, there is still room for further research on GAs for MDVRP, compared to Tabu search.

1 Introduction

Vehicle routing problems (VRPs) are classical combinatorial optimization problems which have received much attention in recent years due to their wide applicability and economic importance. Due to their complexity and importance in determining efficient distribution strategies to reduce operational costs, VRPs form an interesting line of research and variants of VRPs have been studied extensively in the literature. Many of these problems are NP-hard, and they usually define challenging search problems that are good for analyzing new heuristic search techniques. A discussion on the complexity of vehicle routing and scheduling problems can be found in Lenstra and Rinnooy [28]. A survey on the classification and application of VRPs is given in Laporte [25] and Desrosier *et al.* [13].

A typical VRP can be stated as follows: a set of geographically dispersed customers with known demands are to be serviced by a homogenous fleet of vehicles with limited capacity. Each customer is to be fully serviced exactly once and each vehicle is assumed to start and end at exactly one depot, and the primary objective is to minimize the total distance travelled by all vehicles. However, in a large number of practical situations and to satisfy real-life scenarios, additional constraints are usually defined for variants of the VRP. For example, large companies such as those found in the gas

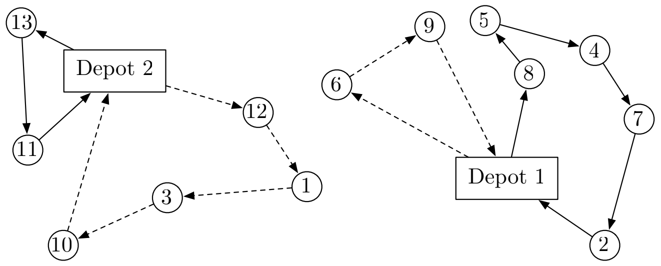


Fig. 1. A MDVRP example with 2 depots, 4 routes and 13 customers

industry [44] have more than one depot from which their stations are serviced. We study the multi-depot VRP (MDVRP), which is an extension of the classical VRP with the exception that every customer should be serviced by a vehicle based at one of several depots. Further practical significance of the MDVRP has been documented in various contexts within distribution management and logistics, including case studies such as the delivery of newspapers [20], chemical products [3], packaged foods [35], and soft drinks [21], among others.

The MDVRP can further be classified as a fixed destination or a non-fixed destination problem [14], [15], [40] and [39]. In the fixed destination MDVRP model, each vehicle starts and terminates its routes in the same depot whereas in the non-fixed destination MDVRP, vehicles originate and terminate at different depots. Although a few articles exist for the use of genetic algorithms for the non-fixed destination problem [14], [15], [40] and [39], only one GA is found in the literature as given by Thangiah and Salhi [41] which is a detailed extension to their preliminary work in [38]. In this paper, we focus on the use of GAs for the fixed destination MDVRP. An example scenario of fixed destination MDVRP with 2 depots with 2 routes each is shown in Figure 1.

The MDVRP is a classic example of a NP-hard [17] combinatorial optimization problem. Even for relatively small problem sizes, the MDVRP is NP-hard and difficult to solve to optimality. The combinatorial explosion is obvious, and obtaining optimal solutions for this type of problems is computationally intractable. As reported by Crevier *et al.* [11], only a few exact algorithms exist for the MDVRP, and these are only practical for relatively small problem sizes. Laporte, Nobert and Arpin [26] were the first to report optimal solutions for problem sizes up to 50 customers by use of a branch and bound technique. Another exact approach for asymmetric MDVRPs by Laporte, Nobert and Taillefer [27] first transformed the problem into an equivalent constraint assignment problem, and then applied a branch and bound technique to problem instances containing up to 80 customers and 3 depots.

Various meta-heuristics have been proposed for MDVRP problems. These approaches seek approximate solutions in polynomial time instead of exact solutions which would be at intolerably high cost. Early heuristics were based on simple construction techniques incorporating improvement procedures. The work by Tillman [43] was one of the early heuristics for the MDVRP; it used the Clarke and Wright savings method [9]. Chao *et al.* [7] introduced a multi-phase heuristic which also incorporated

Table 1. Summary of past work on MDVRP

Authors	**Methodology and Application**
Ball, Golden, Assad and Bodin [3]	Savings method and route first cluster second approach; distribution of chemical product
Benton [4]	A savings method approach combined with a B-B algorithm; delivery of bakery products to retail outlets
Benton and Srikar [5]	(T-C) and concept of frequency of customer allocation to depots
Cassidy and Bennett [6]	Methods similar to (W-H); school meals delivery
Chao, Golden, and Wasil [7]	Composite heuristics that employ infeasibility and refinements
Crevier, Cordeau and Laporte [11]	Tabu search and integer programming to MDVRP with inter-depot routes
Gillett and Johnson [18]	Clustering procedure and Sweep heuristic in each depot
Golden, Magnanti and Nguyen [20]	(T-C) method for borderline customers and savings for routing; newspaper delivery
Klots, Gal and Harpaz [24]	Exact methods and heuristics for MDVRP with backhauls; a daily industry application
Laporte, Nobert and Taillefer [27]	Mixed integer linear formulation with LP relaxation
Min, Current and Schilling [30]	Combination of exact methods and heuristics for MDVRP with backhauls
Perl (P) [33]	(T-C) with modified distance in a variant of savings method
Perl and Daskin [34]	Similar to (P) in a location routing formulation distribution of manufacturing products in USA
Renaud, Laporte and Boctor [36]	Tabu Search with intensification and diversification
Salhi and Sari [37]	An extension of the fleet-mix to the MDVRP
Thangiah and Salhi [41]	Genetic clustering and post-optimization
Tillman and Cain (T-C) [42]	Method of saving with a modified distance formula
Tillman (TL) [43]	Used a Clarke-Wright savings approach [9]
Wren and Holliday (M-H) [45]	Savings method in each depots with refinements

the savings criterion. Other recent heuristics for the MDVRP have been reported by Renaud *et al.* [36] who introduced a tabu search with diversification and intensification, Salhi and Sari [37] developed a multilevel composite heuristic for an extension of the fleet-mix to the MDVRP, and Thangiah and Salhi [41] and Salhi *et al.* [38] used a genetic clustering approach. Table 1 gives a summary of past work on MDVRP and related variants; further details are available in Chao *et al.* [7] and Cordeau *et al.* [10].

However, the available literature on meta-heuristics, in particular genetic algorithms, for the fixed and non-fixed destination MDVRP is limited compared to its counterpart with single depot VRP and variants. Filipec *et al.* ([14] and [15]) introduced a three-phased approach for the non-fixed destination MDVRP. The first phase focused on clustering whereby a heuristic algorithm is used to divide the set of customers into regionally bounded clusters based on depots' capacities and travelling costs between depots and customers. In the next phase, known as the radial routing phase, a GA is applied to the supply area of each depot separately to optimize radial routes leaving the depot. The GA is applied again in the final phase to connect the radial routes into the link network structure. In related work, Skok *et al.* ([40] and [39]) further investigated the use of genetic algorithms for the non-fixed destination MDVRP. They noted (as evidenced by the quality of non-fixed destination MDVRP solutions) that there maybe a downside in solution quality due to the use of decomposition of the problem to relatively independent subsets that are separately investigated, and thus focused on using GA based algorithms ([40] and [39]) that enables routing of all vehicles in non-fixed destination MDVRP.

On the other hand, although a few articles exist for the use of GAs for the non-fixed destination MDVRP [14], [15], [40] and [39], we find it surprising to identify only GenClust GA [41] in literature (which was originally introduced in [38]) for the fixed destination MDVRP studied here, given GA's successes in other variants of VRP such as vehicle routing problem with time windows (VRPTW). Thus, a major contribution of this work is aimed at narrowing this gap, by proposing a relatively simple, but effective GA that employs an indirect encoding and an adaptive inter-depot exchange strategy for the MDVRP with capacity and route-length restrictions.

The proposed GA uses an effective crossover operator and makes use of indirect coding based on permutations of customers: the individuals in the population do not represent direct encodings of solutions. Instead, the GA incorporates an intelligent route scheduler that builds routes from the permutations using the problem constraints as a guide. This encoding scheme was motivated by our previous work in Ombuki, Ross and Hanshar [31]. However, since the MDVRP has the added variable of multiple depots, additional considerations are needed in applying genetic operators to the permutations. Related applications using indirect encoding strategies for other problems are available; for example, Aickelin and Dowsland [1], and Palmer and Kershenbaum [32]. Unlike some cases where a scheduler or decoder [1] is not guaranteed to always produce feasible solutions, our routing scheduler always builds feasible solutions.

Thangiah and Salhi's GenClust [41] is the first and the only major article found in the literature that applied GA to the fixed destination MDVRP we study here. As mentioned earlier, their algorithm was originally introduced in [38] and thereafter, further experimental analysis was presented in [41]. They obtained interesting results where unlike previous approaches for the fixed destination MDVRP, they focused on minimizing both the total number of vehicles and distance. Thangiah and Salhi describe an adaptive method based on geometric shapes which uses a GA that first clusters customers to depots, and then in each cluster a solution is obtained using an insertion technique. In contrast, we use a simple and fast procedure that visits one customer at a time, assigning each customer to its nearest depot, and then we apply a GA strategy to each cluster. We design a dynamic inter-depot mutation operator which may re-assign borderline customers during the evolution process to the final depot placement. Our GA's ability to find good solutions demonstrates that the solution for the MDVRP problem does not strongly depend on the clustering algorithm used to initially assign customers to depots. Furthermore, a number of works that employed the nearest depot clustering technique usually have refinements added to improve the final obtained solution. For example, Thangiah and Salhi [41] used a post-optimizer to improve the final best solution obtained through genetic clustering. We do not perform any local searches on the final best solution by the GA, although our crossover operator has an aspect of local search.

Our GA is competitive with Thangiah and Salhi's GA and improves upon the solution quality especially in reducing the total distance travelled in a number of instances.The proposed GA finds 12 out of 23 new GA solutions using weighted sum fitness measure as compared to the GA described by Thangiah and Salhi. Whenever a weighted sum fitness measure is undertaken, the vehicle and distance dimensions are essentially evaluated as a unified score. The advantage of this is that single solutions are obtained as a result. If one scans the literature for MDVRP most researchers clearly place priority on minimizing the total distance travelled over the number of vehicles. Although this might be reasonable in some instances, it is not inherently preferable over minimizing the number of vehicles, and may not always be what the user wants. Minimizing the number of vehicles affects vehicle and labour costs, while minimizing distance affects time and fuel resources. Thangiah and Salhi's GenClust [41] focused on optimizing both the total distance and vehicles. To give a fair comparison of the proposed GA with their work, besides using the weighted sum fitness measure, our GA employed the Pareto ranking technique [12]. An advantage of this approach is that it is unnecessary to derive weights for a weighted sum scoring formula and aims to optimize both the total distance and the number of vehicles. The solution quality of the GA using Pareto also found 10 out of 23 new GA solutions compared to Thangiah and Salhi's GenClust [41]. When compared to other well-known non GA-based meta-heuristics for the MDVRP, the solution quality of the proposed GA performs well giving good results, although they also indicate the need for further research by the GA community on MDVRP to better compete with Tabu search.

The remainder of this paper is structured as follows: Section 2 gives a formal description of the MDVRP. We then present the details of the proposed GA in Section 3. Next, we provide the results of the comparison of our GA with other meta-heuristics in Section 4. Finally, Section 5 provides concluding remarks.

2 Multi-depot Vehicle Routing Problem

Given a logistics system, assume that the customer size, location, and individual customer demands, and the number and location of all potential depots are known. The vehicle type and size are also given. We now use the MDVRP model adopted by Renaud *et al.* [36]. Let $G = (V, A)$ be a directed graph, where V is the vertex set, and A is the edge set. The vertex set V is further partitioned into two subsets $V_c = \{v_1, ..., v_n\}$ representing the set of *customers*, and $V_d = \{v_{n+1}, ..., v_{n+p}\}$, the set of *depots*. We associate a non-negative *demand* d_i and a *service time* ρ_i with each customer, $v_i \in V_c$. The arc set A denotes all possible connections between the nodes (including nodes denoting depots). We define a *cost* matrix $C = (c_{ij})$ on A corresponding to *travel times*. We adopt travel times to mean Euclidean distance as employed in other related work on MDVRP. We focus our attention on problems for which C is symmetric and satisfies the triangle inequality, i.e., $c_{ij} = c_{ji}$ for all i, j and $c_{ik} \leq c_{ij} + c_{jk}$, for all i, j, k. Based at each depot $v_{n+k} \in V_d$ are t_k identical vehicles of capacity Q, where t_k belongs to some interval $[t_k, t_k^{'}]$. We assume that $t_k = 0$ and $t_k^{'} = n(k = 1, ..., p)$, that is, not all depots are necessarily used. The objective of the MDVRP is to construct a set of routes in such a way that (1) the total number of vehicles used to service the customers is minimized, (2) the total routing cost (a.k.a distance traveled) is minimized, (3) each customer is serviced exactly once by a vehicle, (4) each vehicle route begins and ends at the same depot, (5) the total demand on each route is less than or equal to the total capacity of the vehicles assigned to that route, and (6) the total trip duration (including travel and service time) does not exceed a preset limit.

3 GA Methodology for MDVRP

Genetic Algorithms belong to a class of meta-heuristic methods that mimic Darwin's theory of evolution and survival of the fittest. While most stochastic search methods operate on a single solution, GA is an adaptive heuristic search technique that operates on a *population* of solutions. Holland [22] and his students developed the basic concepts of GAs in the 1970s, although evolutionary computation can be traced further back (an extensive review of early approaches can be found in Fogel [16]). The practicality of using GA to solve complex problems was demonstrated by De Jong [23] and Goldberg [19]. Further details about GAs can be found in such works as Alander [2] and Michalewicz [29].

The details of the chromosome representation, fitness evaluation, and other GA components used for the MDVRP are provided here. When the GA is initialized, a simple clustering strategy is employed to assign each customer to an initial depot. The clustering strategy assigns the customers one by one to a given depot until all the customers have been assigned.

1. *Generate an* **initial population**, *POP*;
2. **Evaluate** *the fitness $F(x)$ of each chromosome x of the population, and calculate the average fitness*;
3. *Create a new population by repeating the following steps until the new population is complete*;
   - **Selection** *Select two parent chromosomes from the population by using tournament selection*;
   - **Recombination** *Apply crossover with a probability to the parents to form new offspring. If no crossover is performed, offspring is an exact copy of parents*;
   - **Mutation** *With mutation probabilities, apply intra-depot or inter-depot mutation to mutate new offspring*;
   - **Acceptance** *Place a new offspring in the population, replacing the parents*;
   - **Elitism** *Randomly replace 1% of the population with the best 1% parents' population*;
4. *Update the old population with the newly generated population*;
5. *If the preset number of generation is reached, stop, return the average fitness, and the fitness of the best (chromosome) solution in the current population*;
6. *Else go to step 2*;

Fig. 2. An outline of the genetic routing system

Next, an initial pool of potential solution candidates (chromosomes) is randomly generated. Each of the chromosomes is transformed by a route scheduler into a set of routes, and then the chromosomes are subjected to an evolutionary process until a minimum possible number of routes is attained, or the termination condition is satisfied. The evolutionary process is carried out as in ordinary GAs using crossover and selection operations on chromosomes. The GA incorporates an adaptive inter-depot mutation which allows the initial customer to depot assignments to be improved where possible. A tournament selection with elite retaining model [19] is used to perform fitness-based selection of individuals for further evolutionary reproduction. A problem-specific crossover operator that ensures that solutions generated through genetic evolution are all feasible is applied to the MDVRP. Figure 2 outlines the GA methodology.

3.1 Initial Depot-Clustering

Initially each customer is assigned to the nearest depot in terms of Euclidean distance. This strategy is simple, fast, and viable since no capacity limit is imposed on each depot. During this initial assignment, some customers are further identified as borderline cases (that is, they are within a certain distance from more than one depot). During the evolution process, an inter-depot mutation may re-assign borderline customers to their final placement to a given depot as discussed in Section 3.9.

3.2 Population Structure and Initialization

A chromosome for a MDVRP problem must specify the number of routes (that is, vehicles), and the delivery order within each route. In this paper, the individuals in the population do not represent direct encodings of solutions. That is, the GA uses an indirect coding based on permutations of customers: it incorporates an intelligent route scheduler that builds routes from these permutations using the problem constraints as a guide. This indirect encoding allows the GA to remain flexible, even when new constraints or objectives are introduced. The MDVRP genetic representation consists of n integer vectors where n corresponds to the number of depots. Each vector consists of a cluster of routes, each of them composed of an ordered subset of customers (genes), as shown in Figure 3.

We can now discuss in detail the chromosome representation in reference to one depot. A gene in a given chromosome indicates the customer number, and the sequence of genes in the chromosome string dictates the order of customer visitations. Below is an example of a chromosome representing a sub-solution in depot d_1:

$$6\ 9\ 8\ 5\ 4\ 7\ 2$$

This chromosome string contains a sequence of routes for a given depot, but no delimiter is used to indicate the beginning or end of a respective route in a given chromosome. To generate the initial population, we use random permutations of N customer nodes per depot.

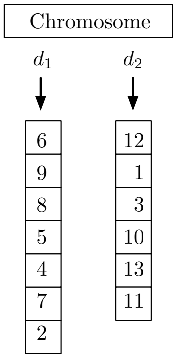


Fig. 3. Example of chromosome representation for a problem with only two depots, d_1 and d_2 with no delimiters to differentiate between routes from a given depot

3.3 Route Scheduler

Because we employed an indirect encoding scheme, the actual correspondence between a chromosome and the routes is achieved by a two-phased route scheduler. The scheduler is designed to consider the feasibility of the routes, with a bias towards solution quality.

In Phase 1, a vehicle must depart from the depot. The first gene of a chromosome indicates the first customer the vehicle is to service. A customer is appended to the current route in the order that customer appears on the chromosome. The routing procedure takes into consideration that the vehicle capacity and route length constraints are not violated before adding a customer to the current route. A new route is initiated every time a customer is encountered that cannot be appended to the current route due to constraints violation (s). This process is continued until each customer has been assigned to exactly one route.

In Phase 2, the last customer of each route r_i, is relocated to become the first customer to route r_{i+1} . If this removal and insertion maintains feasibility for route r_{i+1}, and the sum of costs of r_i and r_{i+1} at Phase 2 is less than the sum of costs of r_i + r_{i+1} at Phase 1, the routing configuration at Phase 2 is accepted, otherwise the route network before Phase 2 (that is, at Phase 1) is maintained.

3.4 Fitness Evaluation

Once each chromosome has been transformed into a feasible network topology using the route scheduler given above, the fitness value of each chromosome is determined by using a weighted-sum fitness function or the Pareto ranking technique [12].

Weighted-sum fitness scoring

This method requires adding the values of fitness functions together using weighted coefficients for each individual objective. The fitness $F(x)$ of an individual x is returned as:

$$F(x) = \alpha \cdot \mid V \mid + \beta \cdot \sum_{k \in V} V_d$$

where

$$V_d = \sum_{i \in N} \sum_{j \in N} C_{ij}$$

α and β are weight parameters associated with the number of vehicles and the total distance traveled by the vehicles, respectively. The weight values of the parameters used in this function were set at $\alpha = 100$ and $\beta = 0.001$ which effectively focuses the evolution on minimization of the number of routes first, then the minimization of route length.

Pareto Ranking Procedure

The Pareto ranking scheme [12] has often been used for multi-objective optimization problem (MOP) applications of genetic algorithms. A MOP is one where the solution is composed of two or more objectives which contribute to the overall result. These objectives often affect one another in complex, nonlinear ways. The challenge is to find a set of values for them which yields an optimization of the overall problem at hand. The Pareto ranking scheme is easily incorporated into the fitness evaluation process within a GA by replacing the raw fitness scores with Pareto ranks. These ranks stratify the current population into preference categories whereby lower ranks store more desirable solutions. Figure 4 shows how the Pareto ranking scheme of Figure 5 is incorporated with the genetic algorithm.

```
Create initial population.
**Repeat until** max. generation reached {
    For each chromosome $i$ in population:
        Evaluate route, determining number of vehicles and cost.
            $\Rightarrow (n_i, c_i)$
        Determine Pareto rank.
            $\Rightarrow rank_i$
    Apply GA to population, using $rank_i$ as fitness value.
            $\Rightarrow$ new population
}
```

Fig. 4. GA with Pareto Ranking

The idea of Pareto ranking is to preserve the independence of individual objectives by treating the current solutions as stratified sets or ranks of possible solutions. In order for a solution to occupy a lower rank it must be clearly superior (i.e., non-dominated) to the others in all objectives of the problem. Hence, solutions that occupy the same rank are considered indistinguishable from each other. This contrasts with a pure GA's attempt to assign a single fitness score to a multi-objective problem, perhaps as a weighted sum. The following definition of the notion of dominance is based on a discussion in [12].

```
Curr_Rank := 1
N := (population size)
m := N
**While** N $\neq$ 0 {     /* process entire population */
    **For** i := 1 to m {     /* find members in current rank */
        **If** $\mathbf{v}_i$ is non-dominated {
            rank($\mathbf{v}_i$) := Curr_Rank
        }
    }
    **For** i := 1 to m {    /* remove ranked members from population */
        **if** rank($\mathbf{v}_i$) $=$ Curr_Rank {
            Remove $\mathbf{v}_i$ from population
            N := N-1
        }
    }
    Curr_Rank := Curr_Rank + 1
    m := N
}
```

Fig. 5. Pareto Ranking algorithm

We assume that the multi-objective problem is a minimization problem, in which lower scores are preferred.

Definition: A solution $\mathbf{v}$ is **Pareto optimal** if there is no other vector $\mathbf{u}$ in the search space that dominates $\mathbf{v}$.

Definition: For a given MOP, the **Pareto optimal set** $\mathcal{P}^*$ is the set of vectors $\mathbf{v}_i$ such that $\forall v_i : \neg\exists \mathbf{u} : \mathbf{u} \preceq \mathbf{v}_i$.

Definition: For a given MOP, the **Pareto front** is a subset of the Pareto optimal set.

Many MOP's will have a multitude of solutions in its Pareto optimal set. Therefore, in a successful run of a genetic algorithm, the Pareto front will be the set of solutions obtained.

As mentioned earlier, a Pareto ranking scheme is incorporated into a genetic algorithm by replacing the chromosome fitnesses with *Pareto ranks*. These ranks are sequential integer values that represent the layers of stratification in the population obtained via dominance testing. Vectors assigned rank 1 are non-dominated, and inductively, those of rank *i+1* are dominated by all vectors of ranks 1 through *i*. Figure 5 shows how a Pareto ranking can be computed for a set of vectors.

3.5 Selection

At every generational stage, we select parents for mating and reproduction. The tournament selection strategy with elite retaining model [36] is used to generate a new population. The tournament selection strategy is a fitness-based selection scheme that works as follows. A set of k individuals are randomly selected from the population. This is known as the tournament set. In this paper, we employed a binary tournament slection and hence the set size is simply two. In addition, we also select a random number *r*, between 0 and 1. If *r* is less than 0.8 (0.8 set empirically from 0.6 to 1.0), the fittest individual in the tournament set is then chosen for reproduction. Otherwise, any of the two chromosomes is chosen for reproduction from the tournament set.

An elite model is incorporated to ensure that the best individual is carried on into the next generation. The advantage of the elitist method over traditional probabilistic reproduction is that it ensures that the current best solution from the previous generation is copied unaltered to the next generation. This means that the best solution produced by the overall best chromosome can never deteriorate from one generation to the next. In our GA, although the preceding fittest individual is passed unaltered to the next generation, it is forced to compete with the new fittest individual.

3.6 Recombination

In [31] we developed *Best Cost Route Crossover* (BCRC), a problem specific crossover operator for the vehicle routing problem with time windows (VRPTW) which ensured that the solutions generated through genetic evolution are all feasible. We further show the applicability of BCRC by extending it to the MDVRP, but with some slight improvements. The example in Figure 6 shows how BCRC constructs two offspring, c_1 and c_2 from two parents, $p1$ and $p2$, using an arbitrary problem instance of customer size 9 and 2 depots. Since each crossover operation is applied to only one depot per generation, depot d_1 is randomly selected. According to step a), $d_1 \in p1$ has two routes, while $d_2 \in p1$ has three routes. Also in step a), for each parent, a route is chosen randomly. In this case, for $d_1 \in p1$, route 2 with the customers 1 and 8 is chosen. Likewise for $d_1 \in p2$, route 3 with customers 6 and 7 is randomly selected.

Next, in step b), for a given parent, the customers (selected in step a) from the opposite parent are removed. For example, in step b), for parent $p1$, customers 6 and 7 which belong to the randomly selected route in $p2$ are removed from $p1$. Likewise, customers 1 and 8 which were selected from a route in $p1$ are removed from the routes in $p2$, as shown in step b). In steps c) and d), the removed customers of each parent are re-assigned either to a best feasible location, or any other feasible location according to a given probability. The following notations and algorithm summarizes the crossover operator as follows.

Given a population $P = \{p_1, ..., p_n\}$ of viable MDVRP chromosomes, where each $p_i = \{d_{i_j}, ..., d_{i_m}\}$, where m is the number of depots and d_{i_j} is the chromosome for the jth depot of the ith parent. Each $d_{i_j} = \{r_k, ..., r_v\}$ is a non-empty set of routes. V_d is the set of depots and $r_k = \{c_1, c_2, ..., c_l\}$ is an ordered list of customers.

1. Randomly select $p_1, p_2, \in P$.
2. Randomly select a depot $\Phi \in V_d$ to undergo reproduction
3. Randomly select a route from each parent $r_1 \in p_{1_\Phi}, r_2 \in p_{2_\Phi}$
4. a) Remove all customers $c \in r_1$ from p_2
   b) Remove all customers $c \in r_2$ from p_1

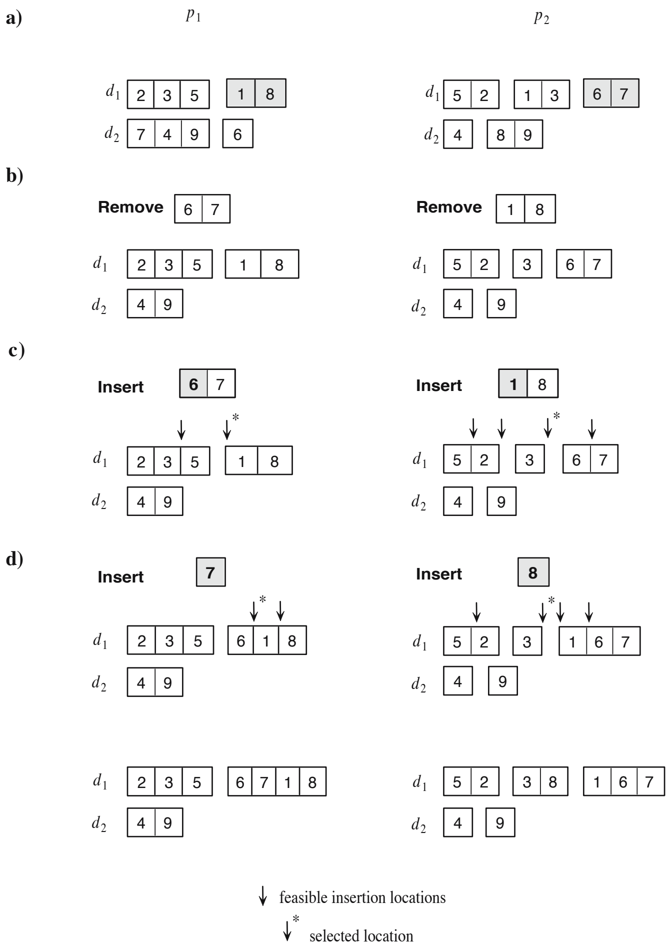


Fig. 6. Example of improved Best Cost Route Crossover (BCRC) operator

5. For each $c \in r_1$
   - Compute the insertion cost of $c_l \in r_1$ into each location in p_{2_Φ} and store these in an ordered list. Also for each insertion location, store in this list whether the insertion maintained feasibility or infeasibility. (If insertion broke an existing route into two routes, this would be infeasible.) location in p_{2_Φ}.
   - Generate a random number $k \in (0, 1)$.
   - If $k \leq 0.8$, choose the first feasible insertion location. (If there exists no feasible insertion locations, create new route with c_l as the only customer.)
     Else if $k > 0.8$, choose the first entry in the list, regardless of feasibility.
6. Repeat for r_2 and p_{1_Φ} taking place of r_1 and p_{2_Φ} respectively in the above loop (from step 5)

3.7 Replacement Scheme

The GA employed a generational scheme in which eligible offspring are introduced into the population once they are produced, replacing the parents so that the population size remains constant.

3.8 Intra-depot Mutation

The mutation step was necessary to increase diversity by introducing new characteristics to current individuals. We employed various mutations to allow for multiple ways to break free from local minima, and to improve route costs when possible.

Mutations that involve single depots are called *intra-depot* mutations, and are effective in incorporating diversity into the routes of each depot. Three types of intra-depot mutations were investigated, each with an equal chance of being applied. In any given generation, if mutation is applicable, it occurs only within one randomly chosen depot.

(a) Reversal mutation. We propose an adaptation of a simple widely used mutation, usually referred to as inversion [29]. Using the example given, in Figure 2, we assume that depot d_1 is selected for mutation. Two cutpoints are selected in the chromosome associated with d_1, and the genetic material between these two cutpoints is reversed. Given a chromosome $\langle 6\ 9\ 8\ 5\ 4\ 7\ 2 \rangle$, two cutpoints are generated before 9 and 7, the genetic material is reversed between the cutpoints giving $\langle 6\ 4\ 5\ 8\ 9\ 7\ 2 \rangle$.
(b) Single customer re-routing. Re-routing involves randomly selecting one customer, and removing that customer from the existing route. The customer is then inserted in the *best feasible insertion location* within the entire chromosome. This involves computing the total cost of insertion at every insertion locale, which finally re-inserts the customer in the most feasible location.
(c) Swapping. This simple mutation operator selects two random routes and swaps one randomly chosen customer from one route to another.

3.9 Inter-depot Mutation

Inter-depot mutation allows for the swapping of customers from one depot to another, which helps improve solution quality, since the initial static clustering of customers to depots makes an imposing assumption about where each customer is to be placed. The application rate of inter-depot mutation is set by a parameter *APPRATE* which was empirically set at every 10 generations, starting at generation 0. This means that at every 10 generations, if mutation is applicable, instead of applying any of the intra-depot mutations described above, an inter-depot mutation operator is used. We also tried running the GA for a number of generations, 1000 generations for example, before applying the inter-depot mutation, but the prior strategy proved sufficient. In the inter-depot mutation, customers can lower the total route costs or the number of vehicles used by re-assigning them to different depots.

The clustering algorithm before evolution identifies a set of customers all deemed swappable. Although allowing any customer to be swappable depot-wise is possible, there exist instances where swapping certain customers can worsen the overall fitness, and may not be feasibly re-inserted into any other depot(s) routes because of route distance constraints. Besides, customers within a certain distance to their previously assigned depots need not be swapped.

As mentioned in Section 3.1, during the initial customer clustering, a list of customers who may be reassigned to other depots during the evolutionary process is also created. This list is termed the *swappable-customer list*, and it typically consists of borderline customers. Although each customer is initially assigned only to its nearest depot during the GA initialization, if any other depot is within a certain distance D from a given customer, that customer will be added to the swappable-customer list. Each customer in this swappable-customer list contains a *candidate list* which is the customer's set of possible depot assignments. This list for a given customer c is made up of the nearest depot to the customer and any other depot d_i in which the following inequality holds:

$$\frac{distance(c, d_i) - min}{min} \leq BOUND$$

where $distance(c, d_i)$ is the Euclidean distance from the customer c to depot d_i, *min* is the distance from c to the nearest depot, and $BOUND$ is a constant value, experimentally set at 2.

4 Computational Results and Discussions

A summary of the experimental results and discussions is given next.

4.1 Implementation and Benchmarks

The GA was implemented in Java 1.3, on a Pentium IV 2.6 GHz PC with 512MB memory under the operating system Windows 2000. We first compare our GA results using both weighted-sum fitness evaluation and Pareto ranking with those obtained by GenClust GA by Thangiah and Salhi [41]. Next, we compare our GA with non-GA based approaches by considering the other works using the same data as [41], that is, Tabu Search by Chao *et al.* (CGW) [7] and the Tabu Search by Renaud *et al.* (RBL) [36].

The evaluation is performed on two sets of 23 MDVRP *Euclidean* benchmark problems, that is, the depot and customer vertices are defined by points in the Cartesian plane, where the cost for each edge (i, j) is the Euclidean distance between customers i and j. The first set with 11 benchmark problems is described by Christofides and Eilon [8], and the second set with 12 problems was introduced by Chao *et al.* [7]. The problems consist of customer sizes varying from 50 to 360, and depots ranging from 2 to 9. About half the problems, namely, 8-11, 13-14, 16-17, 19-20, and 20-23, have route-length restrictions. The basic characteristics of these test problems are summarized in [36] and [41].[1]

[1] Data files can be downloaded from http://neo.lcc.uma.es/radi-aeb/WebVRP.

Table 2. Experimental parameters

Parameter	Setting
population size	400
population type	generational
chromosome initialization	random
generation span	3000
number of elite	4 (1%)
probability of crossover	0.60
probability of intra-depot mutation	0.20
prob. inter-depot mutation	0.25
attempt rate of inter-depot mutation:	every 10 generations

4.2 Experimental Parameters

Obtaining good GA parameter settings that work for a given problem is a non-trivial task. In order to determine robust parameter settings, the critical factors to be considered include the population size, encoding scheme, selection criteria, genetic operator probabilities, and evaluation (fitness) techniques. De Jong's PhD work [23] was among the first systematic attempts to seek out optimum settings for GA parameters. For example, if the population size is too small, the GA may converge too quickly on a local optimal solution. On the other hand, too large a size may take too long for the GA to converge, or result in long computing times for significant improvements. Another concern in parameter settings is mutation rate. If this rate is too high, the mutation tends to introduce too much diversity, hence taking a longer time to get the optimal (or satisfactory solution), and can lead to search instability. Generally, it is now held that optimum parameter settings may be problem-specific, implying that one must first parameterize the GA in the context of a particular problem. Each GA optimization run was carried out with similar parameter settings, as shown in Table 2.

4.3 Comparison of GA Using Weighted Sum-Fitness Scoring with Existing GA on Benchmark Data

We first discuss the performance of the proposed GA using weighted-sum fitness evaluation as compared to GenClust GA by Thangiah and Salhi [41]. Figures 7 and 8 illustrate some of the network topologies obtained by the GA. Figure 7 represents networks for data sets of 100 customers, a capacity limit of 100, and 2, 3, and 4 depots, respectively, with no preset route durations.

On the other hand, Figure 8 represents networks for data sets with larger problem instances (that is, 249 customers), 2-4 depots, and a vehicle capacity of 500. In addition, the networks in Figure 8 represent data sets where the route durations (including travel and service times) exist, and a preset limit of 310 was not to be exceeded.

A comparison of the proposed GA, with GenClust by Thangiah *et al.* [41] for the given problems is given in Table 3 using a format similar to that found in [41]. Route costs are measured by average Euclidian distance. The column labeled *GenClust* gives the best published solutions in [41] while the column *GA* gives the best solution in

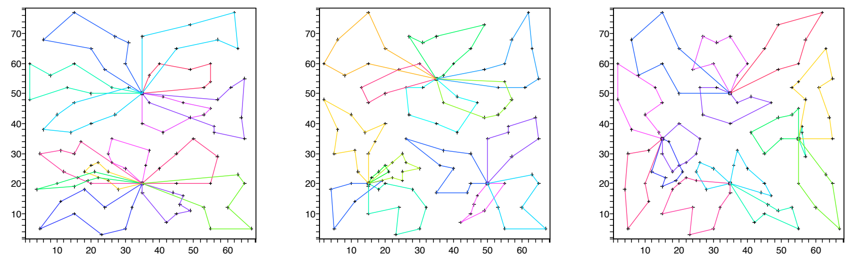

(a) p04 (b) p06 (c) p07

Fig. 7. Network topology with 100 customers, 2,3,4 depots with the capacity limit of 100. Test problems 4,6,7.

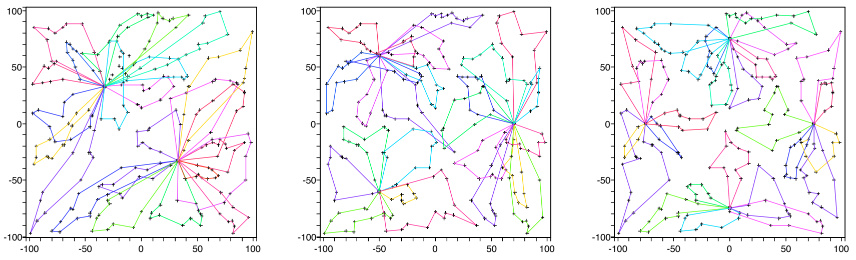

(a) p08 (b) p09 (c) p10

Fig. 8. Network topology with 249 customers, 2-4 depots with capacity limit of 500 and route limit of 310. Test problems 8-10.

10 runs by our proposed GA. The column labeled *Avg.* gives the average of the best solutions found by the GA over all the 10 runs.

The column labeled V_T shows the total vehicle differences between the GA as compared to GenClust. A negative number indicates a decrease, positive number indicates an increase, while a zero depicts that both methods employed the same number of vehicles. The GA obtained the same number of vehicles as GenClust [41] in 14 out of 23 problems, a reduction of 1-3 vehicles in 3 problems, and an increase of 1-2 vehicles in 6 problems. We conclude that the performance of the two algorithms is comparable when considering only the total number of vehicles used as an objective measure.

The column C_T depicts the distance deviations in terms of percentages between the GA and GenClust. A negative value means that the GA had a lower cost, whereas a positive value means that the proposed GA obtained a higher cost than GenClust. In order to make a fair comparison, entries in this column are only listed if the number of vehicles employed for the particular problem is the same for both the GA and GenClust, which resulted in 14 out of 23 problems. It should be noted that in one of these 14 problems (i.e., instance 13) the GA has the same cost as the GenClust for the same number of vehicles. As depicted in the column given by C_T, the GA had reduced costs in 9

Table 3. Comparison of GA using weighted-sum scoring with best-known GA, GenClust [41]

Inst.	Cust.	D	R_{Limits}	Q	GenClust	GA	Avg.	V_T	C_T
1	50	4	-	80	**591.73**[10]	622.18[10]	604.39[10.9]	0	5.2
2	50	4	-	160	463.15[5]	480.04[6]	484.61[6]	+1	
3	75	2	-	140	**694.49**[10]	706.88[10]	715.03[10.6]	0	+1.8
4	100	2	-	100	1062.38[15]	**1024.78**[15]	1044.97[15]	0	-3.5
5	100	2	-	200	**754.84**[8]	785.15[8]	804.57[8]	0	+4.0
6	100	3	-	100	976.02[15]	**908.88**[15]	922.41[15.1]	0	-6.9
7	100	4	-	100	976.48[15]	918.05[16]	935.21[16]	+1	
8	249	2	310	500	4812.52[25]	**4690.18**[25]	4794.98[25]	0	-2.5
9	249	3	310	500	4284.62[25]	**4240.08**[25]	4316.69[25.5]	0	-1.0
10	249	4	310	500	4291.45[25]	3984.78[26]	4058.17[26]	+1	
11	249	5	310	500	4092.68[25]	**3880.65**[25]	3936.53[25.6]	0	-5.0
12	80	2	-	60	1421.94[8]	**1318.95**[8]	1324.72[8]	0	-7.2
13	80	2	200	60	**1318.95**[8]	**1318.95**[8]	1326.58[8]	0	0
14	80	2	180	60	**1360.12**[8]	1365.69[8]	1372.20[8.1]	0	+0.4
15	160	4	-	60	3059.15[15]	2579.25[16]	2653.15[16]	+1	
16	160	4	200	60	2719.98[16]	**2587.87**[16]	2613.69[16]	0	-5.0
17	160	4	180	60	2894.69[16]	**2731.37**[16]	2747.59[16.2]	0	-5.6
18	240	6	-	60	5462.90[22]	3903.85[24]	4015.37[24]	+2	
19	240	6	200	60	3956.61[24]	**3900.61**[24]	3939.45[24]	0	-1.4
20	240	6	180	60	4344.81[27]	**4097.06[24]**	4130.23[24.4]	-3	
21	360	9	-	60	6872.11[34]	5926.49[36]	6070.03[36]	+2	
22	360	9	200	60	5985.32[37]	**5913.59[36]**	5939.00[36]	-1	
23	360	9	180	60	6288.04[39]	**6145.58[37]**	6305.86[37.9]	-2	

out of 14 problems, with a maximum cost reduction of -7.2%. Furthermore, besides the 13 problem instances, there are 3 problem instances (see 20, 22 and 23) where the proposed GA has a reduction in both distances and number of vehicles. Thus we conclude that the proposed GA finds better solutions than the Genclust in 12 out of 23 or a solution improvement of 52.17% problem instances overall. In summary the bolded values in columns labeled *GenClust* and *GA* indicates the problem instances where one algorithm outperforms another by reducing total distance (with same number of vehicles for both GA and GenClust) or it has reduced both the total distance and number of vehicles.

4.4 Comparison of GA Using Pareto Ranking Procedure with Existing GA on Benchmark Data

By considering a GA using weighted-sum scoring, the multi-objective MDVRP is transformed into a single-objective problem. This is not unique to our work. As evidenced in the various MDVRP related work, most researchers focus on one objective, especially on the one minimizing the total distance traveled, ignoring the number of vehicles employed. However, there maybe practical situations where the best solution maybe the one with one or more less vehicles than the best solution found, but the total distance traveled is greater as argued in [40], [41] and [31]. Solutions should be preferred if they require fewer vehicles and less total distances. However, we claim that there is no theoretical nor practical advantage to giving priority the total distance travelled over number of vehicles, or vice versa, perhaps other than having a common framework from which to compare different researchers results. Unless given prior knowledge by the user on their preferences, it should be left to the user to decide which kind of solution is best for their needs. Is the user's main objective to reduce the total distance travelled, or to reduce the number of vehicles dispatched? As implied earlier, this motivated our use Pareto ranking approach besides considering that it would provide us with a better basis of comparing solution quality of the proposed GA with that of [41] who interpreted the MDVRP as a multi-objective problem by considering the minimization of both the total distance and vehicles used. As opposed to weighted-sum scoring, when using the Pareto ranking, one has a choice of more than one solution depending on whether the user wants the best number of vehicles or best travel costs solutions.

Table 4 compares the performance of the proposed GA using Pareto ranking with the GA found in [41]. In presenting the Pareto ranking based results, we provided the best solution (considering both the minimization of distance and vehicles) per a given problem. In some instances (like problem 9 and 11), we have two solutions in each case because neither one dominates the other. For completeness, we have provided the

Table 4. Comparison of our Pareto GA with best-known GA, GenClust [41]

Inst.	Cust.	D	Q	GenClust	GA (Pareto)	V_T	C_T
1	50	4	80	**591.73**[10]	600.63[11]	+1	
2	50	4	160	**463.15**[5]	480.04[6]	+1	
3	75	2	140	**694.49**[10]	683.15[11]	+1	
4	100	2	100	1062.38[15]	**1034.59**[15]	0	-2.7
5	100	2	200	**754.84**[8]	778.01[8]	0	+3.1
6	100	3	100	976.02[15]	**916.71**[15], 900.44 [16]	[0,+1]	[-6.5,–]
7	100	4	100	976.48[15]	922.83[16]		
8	249	2	500	4812.52[25]	**4672.56**[25]	0	-3.0
9	249	3	500	**4284.62** [25]	4332.32 [25], 4243.74[26]	[0,+1]	[+1.1,–]
10	249	4	500	4291.45[25]	3953.24[26]		
11	249	5	500	4092.68[25]	**3962.17** [25], 3876.26[26]	[0,+1]	[-3.3,–]
12	80	2	60	1421.94[8]	**1318.95**[8]	0	-7.2
13	80	2	60	**1318.95**[8]	**1318.95**[8]	0	0
14	80	2	60	**1360.12**[8]	1365.69[8]	0	+0.4
15	160	4	60	3059.15[15]	2579.25[16]	+1	
16	160	4	60	2719.98[16]	**2596.83**[16]	0	-4.7
17	160	4	60	2894.69[16]	**2731.37**[16]	0	-5.6
18	240	6	60	5462.90[22]	3897.22[24]	+2	
19	240	6	60	**3956.61[24]**	3972.80[24]	0	+0.4
20	240	6	60	4344.81[27]	**4097.06[24]**	-3	
21	360	9	60	6872.11[34]	6101.68[36]	+2	
22	360	9	60	5985.32[37]	**5984.87[36]**	-1	
23	360	9	60	6288.04[39]	**6145.58[37]**	-2	

same experimental analysis as given above, by including the columns V_T and C_T which reflect the deviations (by the proposed GA compared to the GA in [41]) of the total vehicles and distances respectively, per a given problem. The bolded values in columns labeled *GenClust* and *GA* indicates the problem instances where one algorithm outperforms another by reducing total distance (with same number of vehicles for both GA and GenClust) or it has reduced both the total distance and number of vehicles. In this case we conclude that the proposed GA introduced 10 out 23 new solutions or an improvement of 43.46% whereas GenClust [41] introduced 7 out of 23 new solutions and one solution (for instance 13 is a tie).

4.5 Comparison of GA Using Weighted Sum-Fitness Scoring with Non-GA Techniques on Benchmark Data

Table 5 shows a comparison of our GA with non-GA techniques for the MDVRP, again using a format similar to that given in [41]. The column *CGW* indicates composite heuristics that employ infeasibility and refinements [7], while *RBL* is a Tabu Search algorithm [36]. The columns labeled V_a and V_b show the vehicle differences between the GA, CGW and RBL, respectively. A positive number indicates an increase by the GA, a negative number indicates a decrease, while a zero represents equal number of

Table 5. Comparison of our GAs with well-known non-GA results

Inst.	Cust.	D	CGW [7]	RBL [36]	GA	V_a	D_a	V_b	D_b
1	50	4	582.3[11]	576.87[11]	622.18[10]	-1		-1	
2	50	4	476.7[5]	473.53[5]	480.04[6]	+1		+1	
3	75	2	641.2[11]	641.19[11]	706.88[10]	-1		-1	
4	100	2	1026.9[16]	1003.87[15]	1024.78[15]	-1		0	2.8
5	100	2	756.6[8]	750.26[8]	785.15[8]	0	3.8	0	4.7
6	100	3	883.6[16]	876.50[16]	908.88[15]	-1		-1	
7	100	4	898.5[17]	892.58[15]	918.05[16]	-1		+1	
8	249	2	4511.6[27]	4485.09[25]	4690.18[25]	-2		0	4.6
9	249	3	3950.9[26]	3937.82[26]	4240.08[25]	-1		-1	
10	249	4	3815.6[28]	3669.38[26]	3984.78[26]	-2		0	8.6
11	249	5	3733.0[27]	3648.95[26]	3880.65[25]	-2		-1	
12	80	2	1327.3[8]	1318.95[8]	1318.95[8]	0	-0.6	0	0
13	80	2	1345.9[8]	1318.95[8]	1318.95[8]	0	-2.0	0	0
14	80	2	1372.5[8]	1365.69[8]	1365.69[8]	0	-0.5	0	0
15	160	4	2610.3[16]	2551.46[16]	2579.25[16]	0	-1.2	0	1.1
16	160	4	2605.7[16]	2572.23[16]	2587.87[16]	0	-0.7	0	0.6
17	160	4	2816.6[18]	2731.37[16]	2731.37[16]	-2		0	0
18	240	6	3877.4[25]	3781.04[23]	3903.85[24]	-1		0	
19	240	6	3864.0[24]	3827.06[24]	3900.61[24]	0	1.0	0	1.9
20	240	6	4272.0[28]	4097.06[24]	4097.06[24]	-4		0	0
21	360	9	5791.5[37]	5656.47[36]	5926.49[36]	-1		0	4.8
22	360	9	5857.4[37]	5718.00[36]	5913.59[36]	-1		0	3.4
23	360	9	6494.6[41]	6145.58[36]	6145.58[37]	-4		+1	

vehicles. In 15 out of 23 problems, the GA obtained a smaller number of vehicles than CGW, the same number of vehicles in 7 problems, and only required one more vehicle in one problem instance. In 5 out of 23 problems, the GA obtained better number of vehicles, the same number of vehicles in 14 problems, and required an extra vehicle in each of the remaining 4 problems compared with RBL. We conclude that when considering the total numbers of vehicles used the proposed GA does better that CGW [7] , whereas the performance of the GA is comparable to that of RBL [36]. However, given that the MDVRP was not interpreted as multi-objective optimization by CGW and RBL, to make this comparison meaningful more weight should be placed in the comparison below whereby the algorithms have used the same number of vehicles, and consider only the total distance traveled.

The percentage differences in distances between the GA, CGW and RBL in Table 5 were also computed with the outcomes listed in columns D_a and D_b, respectively. As mentioned above, to make this comparison short and meaningful we only compare problem instances where the algorithms have used the same number of vehicles, or where one algorithm has a better solution than the other in terms of both distance and number of vehicles. Entries are only entered for cases where the GA used the same number of vehicles as the CGW [7] or RBL [36] . As indicated by D_a, in 5 out of the 7 problems where CGW used the same number of vehicles as our GA, the GA obtained reduced costs, up to a maximum of -2.0% . For the remaining two problems, the maximum increase was 3.8%. However, by observing column D_b (and the overall solution quality), it is seen that the Tabu search approach presented by RBL [27] provides better solutions than the GA and CGW [26]. However, it should be noted again that the main objective of both CGW [7] and RBL [36] was to minimize the total distance for the fixed destination MDVRP. Hence we note that although Table 5 gives us some results to compare the performance of our GA with non-GA approaches, our main focus was on the previous GA works for this problem.

In summary, based on this results presented above, we conclude that the proposed GA is competitive with the two approaches, although RBL [27] gives an overall better solution quality. However, the main focus of our work was to contribute to the GA application for MDVRP. A literature review showed that although there exists published work focusing on the development of meta-heuristics for the MDVRP, little work is reported on the use of GA for the fixed destination MDVRP. This calls for further investigation, since a number of efficient GAs exist for related extensions of the classic VRP, such as VRPTW. This paper presented a new genetic algorithm for the MDVRP which improved upon the solution quality of published GA-based approach. The GA studied in this paper has an advantage of its simplicity, and yet gives effective results. The comparisons withe the non-GA based approach indicate that there is still a need for GA research for MDVRP to better compete with Tabu-based approaches.

5 Conclusions and Future Work

Vehicle routing forms an important line of research, because of its high complexity and intractable nature, as well as its importance in distribution management. A literature review showed that although there exists published work focusing on the development of heuristics for the fixed destination MDVRP, little work is reported on the use of GA for this problem. This calls for further investigation, since a number of efficient GAs exist for related extensions of the classic VRP, the VRPTW.

This paper presented a new genetic algorithm for the MDVRP. The approach is tested on a set of 23 standard problem instances taken from the literature, and compared with other approaches. The GA studied in this paper has an advantage of its simplicity and yet gives efficient solution quality with respect to existing GA and non-GA techniques. A possible future work is to extend this work to the multi-depot vehicle routing problem with time windows and to better tune the GA parameters.

Acknowledgement

This research is supported by the Natural Sciences and Engineering Research Council of Canada, NSERC. This support is greatfully acknowledged. The authors wish to thank Mario Ventresca and Dr. Michael Berman for their valuable comments.

References

1. Aickelin, U., Dowsland, K.A.: An indirect genetic algorithm for a nurse-scheduling problem. Computers and Ops. Res. 31, 761–778 (2004)
2. Alander, J.T.: An indexed bibliography of genetic algorithms in operations research. Technical report series no. 94-1-or, University of Vaasa, Vaasa, Finland (2000)
3. Ball, M.O., Golden, B.L., Assad, A.A., Bodin, L.D.: Planning for truck fleet size in the presence of a common-carrier option. Decis. Sci. 14, 103–120 (1983)
4. Benton, W.C.: Evaluating a modified heuristic for the multiple vehicle scheduling problem. Working paper rs86-14, College of Administration Science, Ohio State University, Columbus, OH (1986)
5. Benton, W.C., Srikar, B.: Experimental study of environmental factors that affect the vehicle routing problem. Journal of Business Logistics 6(1), 66–78 (1987)
6. Cassidy, P.J., Bennett, H.S.: Tramp - a multi-depot vehicle scheduling system. Operational Research Quarterly 23, 151–162 (1972)
7. Chao, I.M., Golden, B.L., Wasil, E.: A new heuristic for the multi-depot vehicle routing problem that improves upon best-known solutions. Am. J. Math. Mgmt Sci. 13, 371–406 (1993)
8. Christofides, N., Eilon, S.: An algorithm for the vehicle-dispatching problem. Operational Research Quarterly 20, 309–318 (1969)
9. Clarke, G., Wright, J.W.: Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 46, 93–100 (1964)
10. Cordeau, J.F., Gendreau, M., Laporte, G.: A tabu search heuristic for the period and multi-depot vehicle routing problems. Networks 30, 105–119 (1997)
11. Crevier, B., Cordeau, J.F., Laporte, G.: The multi-depot vehicle routing problem with inter-depot routes. European Journal of Operational Research 176(2), 756–773 (2007)
12. Deb, K.: Multi-Objective Optimization Using Evolutionary Algorithms. John Wiley, Chichester (2001)
13. Desrosier, J., Dumas, Y., Solomon, M.M., Soumis, F.: Time constraint routing and scheduling. In: Ball, M.O., Magnanti, T.L., Monma, C.L., Nemhauser, G.L. (eds.) Handbooks in Operations Research and Management Science, vol. 8, pp. 35–139. Elsevier Science Publishers, Amsterdam (1995)

14. Filipec, M., Skrlec, D., Krajcar, S.: Darwin meets computers: New approach to multiple depot capacitated vehicle routing problem 1, 421–426 (1997)
15. Filipec, M., Skrlec, D., Krajcar, S.: Genetic algorithm approach for multiple depot capacitated vehicle routing problem solving with heuristic improvements. International Journal of Model ling & Simulation 20, 320–328 (2000)
16. Fogel, D.: Evolutionary Computation: the fossil record. IEEE Press, New York (1998)
17. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to The Theory of NP-Completeness. W. H. Freeman and Company, New York (1979)
18. Gillett, B.E., Johnson, J.G.: Multi-terminal vehicle-dispatch algorithm. Omega 4, 711–718 (1976)
19. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading (1989)
20. Golden, B.L., Magnanti, T.L., Nguyen, H.Q.: Implementing vehicle routing algorithms. Networks 7, 113–148 (1977)
21. Golden, B.L., Wasil, E.: Computerized vehicle routing in the soft drink industry. Operations Research 35, 6–17 (1987)
22. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
23. De Jong, K.A.: An Analysis of the behavior of a class of genetic adaptive systems. PhD thesis, University of Michigan (1975)
24. Klots, B., Gal, S., Harpaz, A.: Multi-depot and multi-product delivery optimization problem time and service constraints. Ibm israel report 88-315, Science and Technology, Haifa, Israel (1992)
25. Laporte, G.: The vehicle routing problem: An overview of exact and approximate algorithms. European Journal of Operations Research 59, 345–358 (1992)
26. Laporte, G., Nobert, T., Arpin, D.: Optimal solutions to capacitated multidepot vehicle routing problems. Congressus Numerantium 44, 283–292 (1984)
27. Laporte, G., Nobert, T., Taillefer, S.: Solving a family of multi-depot vehicle routing and location problems. Transportation Science 22, 161–172 (1988)
28. Lenstra, J.K., Rinnooy Kan, A.H.G.: Complexity of vehicle routing problem with time windows. Networks 11, 221–227 (1981)
29. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs, 3rd edn. Springer, Heidelberg (1998)
30. Min, H., Current, J., Schilling, D.: The multiple depot vehicle routing problem with backhauling. Journal of Business Logistics 13, 259–288 (1992)
31. Ombuki, B., Ross, B., Hanshar, F.: Multi-objective genetic algorithms for vehicle routing problems with time windows. Journal of Applied Intelligence 24(1), 17–30 (2006)
32. Palmer, C., Kershenbaum, A.: Representing trees in genetic algorithms. In: Proceedings of the First IEEE International Conference on Evolutionary Computation, NY, pp. 379–384 (1994)
33. Perl, J.: The multi-depot routing allocation problem. American journal of Mathematical and Management Science 7, 8–34 (1987)
34. Perl, J., Daskin, M.S.: A warehouse location-routing problem. Transportation Research 19B, 381–396 (1985)
35. Pooley, J.: Integrated production and distribution facility planning at Ault foods. Interfaces 24, 113–121 (1994)
36. Renaud, J., Laporte, G., Boctor, F.F.: A tabu search heuristic for the multi-depot vehicle routing problem. Computers Ops. Res. 23, 229–235 (1996)
37. Salhi, S., Sari, M.: A multi-level composite heuristic for the multi-depot vehicle fleet mix problem. European J. Op. Res. 103, 95–112 (1997)

38. Salhi, S., Thangiah, S.R., Rahman, F.: A genetic clustering method for the multi-depot vehicle routing problem, 234–237 (1998)
39. Skok, M., Skrlec, D., Krajcar, S.: The genetic algorithm method for multiple depot capacitated vehicle routing problem solving, 520–526 (2000)
40. Skok, M., Skrlec, D., Krajcar, S.: The non-fixed destination multiple depot capacitated vehicle routing problem and genetic algorithms, 403–408 (2000)
41. Thangiah, S.R., Salhi, S.: Genetic clustering:an adaptive heuristic for the multidepot vehicle routing problem. Applied Artificial Intelligence 15, 361–383 (2001)
42. Tillman, F.A., Cain, T.M.: An upper bound algorithm for the single and multiple terminal delivery problem. Management Science 18, 664–682 (1972)
43. Tillman, F.A.: The multiple terminal delivery problem with probabilistic demands. Transportation Science 3, 192–204 (1969)
44. Bell, W., Dalberto, L., Fisher, M.L., Greenfield, A., Jaikumar, R., Mack, R., Kedia, P., Prutzman, P.: Improving the distribution of industrial gases with an on-line computerized routing and scheduling system. Interfaces 13, 4–22 (1983)
45. Wren, A., Holliday, A.: Computer scheduling of vehicles from one or more depots to a number of delivery points. Operational Research Quarterly 23, 333–344 (1972)

Hybridizing Problem-Specific Operators with Meta-heuristics for Solving the Multi-objective Vehicle Routing Problem with Stochastic Demand

Chun Yew Cheong and Kay Chen Tan

Department of Electrical and Computer Engineering, National University of Singapore,
4 Engineering Drive 3, Singapore
{cheongchunyew, eletankc}@nus.edu.sg

Summary. This book chapter extends a recently published work on solving the multi-objective vehicle routing problem with stochastic demand (VRPSD). In that work, a few problem-specific operators, including two search operators for local exploitation and the route simulation method (RSM) for evaluating solution quality, were proposed and incorporated with a multi-objective evolutionary algorithm (MOEA). In this chapter, the operators are hybridized with several meta-heuristics, including tabu search and simulated annealing, and tested on a few VRPSD test problems adapted from the popular Solomon's vehicle routing problem with time window (VRPTW) benchmark problems. The experimental results reveal several interesting problem and algorithmic characteristics which may have some bearing on future VRPSD research.

1 Introduction

The vehicle routing problem (VRP) is a generic name referring to a class of combinatorial optimization problems in which customers are to be served by a number of vehicles. The vehicles leave the depot, serve customers in the network and on completion of their routes, return to the depot. Each customer is described by a certain demand. Other information includes the co-ordinates of the depot and customers, the distance between them, and the capacity of the vehicles providing the service. All these information are known in advance for the purpose of planning a set of routes which minimizes transportation cost while satisfying capacity constraints [1], time constraints [2, 3], and time window constraints [4, 5, 6, 7]. However, in many real-world applications, one or more parameters of the VRP tend to be random or stochastic in nature, giving rise to the stochastic vehicle routing problem (SVRP).

There are three basic classes of SVRP [8, 9, 10]: stochastic customers [11, 12, 13], stochastic demands, and stochastic travel and service times. This chapter considers the capacity and time constrained vehicle routing problem with stochastic demand (VRPSD), where only the customer demand is stochastic and all other parameters are known a priori. This problem appears in the delivery of home heating oil [14], trash collection, sludge disposal [15], beer and soft drinks distribution, the provision of bank automates with cash, and the collection of cash from bank branches [16].

The VRPSD differs from its deterministic counterparts in that when some data are random, it is no longer possible to require that all constraints be satisfied for all realizations of the random variables [9]. The basic characteristic of the VRPSD is that the actual demand of each customer is revealed only when the vehicle reaches the customer. As such, on one hand, the vehicle routes are designed in advance by applying a particular algorithm but on the other hand, due to the uncertainty of demands at the customers, at some point along a route the capacity of the vehicle may be depleted before all demands on the route have been satisfied. Teodorović and Lucić [17], and Dror and Trudeau [18], referred to such a situation as "route failure". In the capacity constrained VRPSD, recourse or corrective actions, e.g. making a return trip to the depot to restock, have to be designed to ensure feasibility of solutions in case of route failure.

In the time constrained VRPSD, one possible corrective action is to apply a penalty when the duration of a route exceeds a given bound. This penalty would correspond to the overtime pay that a driver receives. As such, the situation of route failure, together with all its associated recourse policies, would definitely generate additional transportation cost, in terms of the travel distance for the to and fro trips to the depot and the overtime pay for drivers, which are stochastic in nature. This means that the actual cost of a particular solution to the VRPSD cannot be known with certainty before the actual implementation of the solution. One of the main obstacles to solving the VRPSD is in finding an objective function which takes into consideration all these costs. It is for this reason that Laporte and Louveaux [9], Gendreau et al. [10], and Dror [19], agree that the VRPSD and the SVRP in general are inherently much more difficult to solve than their deterministic counterparts.

Many researchers have studied the VRPSD in two frameworks, namely as a chance constrained program (CCP) [20, 21] or as a stochastic program with recourse (SPR). In CCPs, the problem consists of designing a set of vehicle routes for which the probability of route failure is constrained to be below a certain threshold. It was shown by Steward and Golden [22] that, under some restrictive assumptions, the problem can be reduced to a deterministic VRP and then solved using existing deterministic algorithms. Although the CCP tries to control the probability of route failure, the cost of such failures is ignored. In contrast, the SPR considers the demand distributions of customers and tries to minimize the expected transportation cost, which includes the travel cost as well as the additional cost generated by recourse policies. Gendreau et al. [10] commented that SPRs are typically more difficult to solve than CCPs but their objective functions are more meaningful.

According to Yang et al. [8], a single, long route has the lowest expected travel distance but it may not be feasible to implement in the context of the real-world as the vehicle may take a long time to complete the route. From this finding, it is clear that the VRPSD is inherently a multi-objective optimization problem. In minimizing the travel distance of a particular solution, an algorithm for the VRPSD must also account for the feasibility of implementation of the solution in terms of the duration of the routes, i.e. both capacity and time constraints must be considered. Therefore, it is required to minimize multiple conflicting cost functions, such as the travel distance, the remuneration for drivers including overtime pay, and the number of vehicles required, concurrently, which is best solved by means of multi-objective optimization. As such, in [23], a multi-objective evolutionary algorithm (MOEA) was proposed to tackle the VRPSD. The MOEA is equipped with several VRPSD-specific operators, including two local search heuristics, which are designed to exploit two route structures of a VRPSD solution [18], and a route simulation method (RSM), which was proposed to address the issue of evaluating the expected costs of VRPSD solutions. In this chapter, these operators are hybridized with several meta-heuristics, including tabu search and simulated annealing, and tested on a few multi-objective VRPSD test problems adapted from the popular Solomon's vehicle routing problem with time window (VRPTW) benchmark problems.

This chapter is organized as follows. Sect. 2 provides the problem formulation of the VRPSD. Sect. 3 presents the implementation details of the MOEA and the participating algorithms. Sect. 4 presents the extensive simulation and performance comparison results. Conclusions are drawn in Sect. 5.

2 Problem Formulation

This section presents the mathematical model of the VRPSD. The time and capacity constrained problem, as well as the recourse policy used, will also be explained. Fig. 1 shows a complete graph representing a model of a simple VRPSD and its solution. The solution consists of two routes, R_1 and R_2, connecting the depot to a set of customers which are each identified by a number. For a particular route, the arrows show the sequence in which the customers will be visited by the vehicle and the route must start and end at the depot.

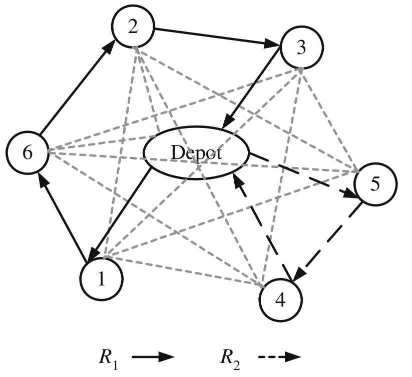


Fig. 1. Graphical representation of a simple vehicle routing problem

Definitions of some of the frequently used notations for the VRPSD, leading to the formulation of the mathematical model of the problem, are given as follows.

1) Customers and depot: The customer set $V = \{0, 1, 2, \ldots, N\}$ represents the N customers to be visited. For simplicity, the depot is denoted as customer 0, or v_0. The depot is treated as the source of service demanded by the customers. Every vehicle must start and end its route at the depot. With the exception of the depot, each customer v_i has a demand distribution D_i. D_i is a normal random variable and is described by two parameters, the mean μ_i and the variance σ_i^2. The actual demand of each customer d_i is revealed when the vehicle first arrives at the customer. There is also a service time

s_i associated with each customer and the depot, which will be incurred each time the vehicle arrives at the customer or returns to the depot for restocking.

2) Node: A node is denoted by $n_i(r)$, which represents the *i*th customer that is served in a particular route *r*. It must be an element in the customer set, i.e. $n_i(r) \in V$.

3) Vehicles and capacity constraint: All vehicles are identical and each one has a capacity limit *C*. This capacity limit acts as a constraint and a route failure occurs when this constraint is compromised.

4) Routes: A vehicle starts its route at the depot, visits a number of customers, and returns to the depot. A route *r* is represented as $\Omega(r) = \langle v_0, n_1(r), n_2(r), \ldots, n_k(r), v_0 \rangle$ where *k* is the size of the route. Since all vehicles must depart and return to the depot v_0, to simplify the representation, the depot will be omitted, i.e. $\Omega(r) = \langle n_1(r), n_2(r), \ldots, n_k(r) \rangle$.

5) Euclidean costs: The travel distance between any two points *i* and *j*, where each point can be a customer or the depot, is equal to the travel time and is denoted by c_{ij}, which is calculated using the following equation:

$$c_{ij} = \sqrt{(i_x - j_x)^2 + (i_y - j_y)^2} \tag{1}$$

where i_x and i_y are the *x* and *y* coordinates of the point *i*, respectively. c_{ij} is symmetrical, i.e. $c_{ij} = c_{ji}$, and satisfies the triangular inequality, where $c_{ij} + c_{jk} \geq c_{ik}$.

6) Route failure and recourse policy: For a route $\Omega(r) = \langle n_1(r), \ldots, n_f(r), \ldots, n_k(r) \rangle$, route failure is said to occur at the *f*th customer of the route if $\sum_{i=1}^{f} d_{n_i(r)} \geq C$ and the simple recourse policy [9, 17, 24, 25] is employed to maintain the feasibility of solutions. For the case where $\sum_{i=1}^{f} d_{n_i(r)} > C$, the recourse policy is such that the vehicle will unload all remaining goods (equivalent to $C - \sum_{i=1}^{f-1} d_{n_i(r)}$ units) at the *f*th customer, return to the depot to restock, then turn back to the *f*th customer to complete the service, and finally continue with the originally planned route. For the case where $\sum_{i=1}^{f} d_{n_i(r)} = C$ and $f < k$, the recourse policy is such that the vehicle will return to the depot to restock and continue with the planned route at the (*f* + 1)th customer. These recourse actions will of course incur additional transportation cost, in terms of the travel distance $Q_d(r)$ and time $Q_t(r)$ for the to and fro trips to the depot. $Q_t(r)$ also includes the additional service times incurred when a vehicle visits a customer more than once or returns to the depot for restocking due to route failures.

7) Time constraint and driver remuneration: The total duration $c_t(r)$ of a route $\Omega(r) = \langle n_1(r), n_2(r), \ldots, n_k(r) \rangle$ is calculated as in the following equation:

$$c_t(r) = \sum_{i=1}^{k-1} (c_{n_i(r), n_{i+1}(r)}) + c_{v_0, n_1(r)} + c_{n_k(r), v_0} + \sum_{j=1}^{k} s_j + Q_t(r) \tag{2}$$

The time constraint is such that $c_t(r)$ should not exceed a given bound *B*. This is a soft constraint and *B* is calculated as the time for a vehicle to travel diagonally across the map from one corner to the other and back. This time is assumed to be 8 hours, equivalent to a driver's workday. Remuneration is such that drivers are paid \$10 for each of the first 8 hours of work and \$20 for every additional hour of work subsequently. This is done to penalize exceedingly long routes which may not be feasible to implement in the context of the real-world.

8) Transportation costs: The transportation costs include travel distance and driver remuneration. The travel distance $c_d(r)$ for a route $\Omega(r) = \langle n_1(r), n_2(r), \ldots, n_k(r)\rangle$ is given in the following equation:

$$c_d(r) = \sum_{i=1}^{k-1}(c_{n_i(r),n_{i+1}(r)}) + c_{v_0,n_1(r)} + c_{n_k(r),v_0} + Q_d(r) \tag{3}$$

The driver remuneration $c_r(r)$ is calculated as in the following equation:

$$c_r(r) = \begin{cases} \frac{c_t(r)}{\frac{B}{8}} \times 10, & c_t(r) \leq B \\ 80 + \frac{c_t(r)-B}{\frac{B}{8}} \times 20, & otherwise \end{cases} \tag{4}$$

9) Routing plan: The routing plan *G* consists of a set of routes $\{\Omega(r_1), \ldots, \Omega(r_m)\}$. The number of routes *m* is equal to the number of vehicles used in the plan. The condition $\bigcup_{i=1}^{m} \Omega(r_i) = V$, i.e. all customers must be routed, must be satisfied.

10) Other assumptions: It is also assumed that each customer can only be serviced by one vehicle but the vehicle is allowed to service the same customer more than once. Multiple service times will be incurred if a vehicle visits a customer multiple times.

The VRPSD, therefore, involves finding a solution $G = \{\Omega(r_1), \ldots, \Omega(r_m)\}$ that minimizes the three objectives of travel distance $\sum_{i=1}^{m} c_d(r_i)$, driver remuneration $\sum_{i=1}^{m} c_r(r_i)$, and number of vehicles required *m*.

3 Description of Algorithms

Other than the MOEA, the algorithms used in this study include tabu search, simulated annealing, hill-climbing search, and random search. Like evolutionary algorithm, on which the MOEA is based, tabu search and simulated annealing belong to the class of meta-heuristics, which are widely recognized as one of the most practical approaches for solving combinatorial optimization problems. Hill-climbing search represents the class of greedy search algorithms, whereas random search acts as a sort of minimal requirement for the algorithms such that it is reasonable to request that a good algorithm for the VRPSD performs better than random search.

3.1 Common Elements of Algorithms

With reference to the design of the MOEA [23], this section details the common elements within the participating algorithms. Fig. 2 shows the algorithmic flow of the MOEA. For brevity, only the components that are common in the participating algorithms will be described in this chapter. Interested readers are referred to [23] for the implementation details of the MOEA, though these details are not required for appreciating the contents in this chapter.

Variable-Length Chromosome

The MOEA and the four participating algorithms use the variable-length chromosome representation (Fig. 3), which encodes a complete solution, including the number of

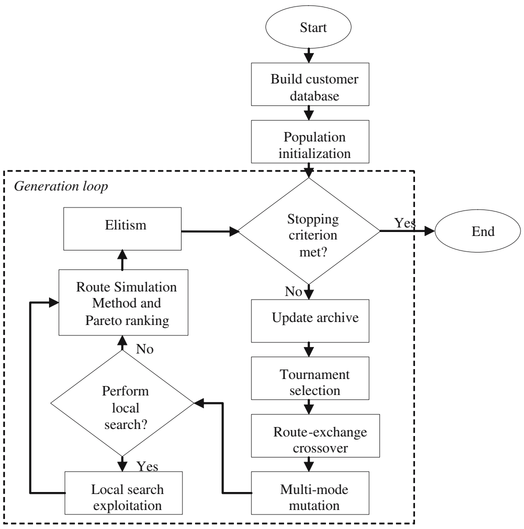


Fig. 2. Algorithmic flow of MOEA

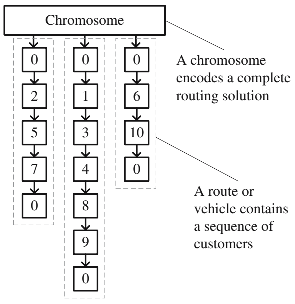

Chromosome
0 2 5 7 0
0 1 3 4 8 9 0
0 6 10 0
A chromosome encodes a complete routing solution
A route or vehicle contains a sequence of customers

Fig. 3. Variable-length chromosome representation

vehicles and the order of customers served by these vehicles. Such a representation is efficient and allows the number of vehicles to be manipulated and minimized directly for multi-objective optimization in the VRPSD.

Population-Based Search

All the algorithms are population-based in that they begin the search process by building an initial search population of 500 chromosomes. In initializing the population, the first chromosome is built such that the sum of the mean values of the customer demands on each route does not exceed the vehicle capacity. Furthermore, the sum of the travel and service times on each route must not exceed the vehicle time window. The number of vehicles required in this first chromosome is then taken as the maximum number of vehicles that each of the remaining chromosomes can use. For each chromosome, the number of vehicles is randomly picked from the feasible range. The routes are then built such that each route has approximately the same number of customers. This procedure is done so that the initial population has a wide range of chromosomes with different number of vehicles to start with. These chromosomes then form the starting points of the search. Starting the search from multiple points will allow greater parallelism and increase the chances of finding good solutions.

Unlike the MOEA, no search information will be exchanged between chromosomes during the search process for the four participating algorithms. As such, the tournament selection scheme, the route-exchange crossover operator, and the elitism mechanism of the MOEA (Fig. 2) are irrelevant to the four algorithms.

Multi-Mode Mutation

The four participating algorithms utilize the multi-mode mutation operator of the MOEA to generate new chromosomes to replace the current ones since the operator allows every point in the search space to be reached. Fig. 4 shows the operation of the multi-mode mutation.

1) Partial Swap: This operation involves a number of swap moves and for each move, two routes will be randomly chosen. A segment is then randomly selected from each route and swapped to the other route. This new segment takes the place of the previous segment that has been swapped out. In the situation where either one of the routes has only a single customer in it, a random segment is still selected from the route with more than one customer. This segment is then swapped with the solitary customer in the other route. In addition, a mechanism is in place such that the same two routes will not be selected twice in a particular partial swap operation.

2) Merge Shortest Route: This operation searches for the two routes of the chromosome with the smallest sum of travel distance and driver remuneration, and appends one route to the other. The merge shortest route will not operate on any chromosome with only one route.

3) Split Longest Route: This operation searches for the route with the largest sum of travel distance and driver remuneration, and breaks the route into two at a random point.

At the end of the multi-mode mutation, a random shuffling operation is applied on every route of each chromosome with a probability equal to the shuffle rate. Like in the MOEA, elastic rate and squeeze rate (Fig. 4) are set to 0.5 for the participating algorithms. The ways in which newly generated chromosomes replace current ones are representative of the four participating algorithms and will be elaborated in Sect. 3.3.

Local Search Exploitation

Two VRPSD-specific local search heuristics were introduced in [23]. The heuristics are inspired by the underlying structures of a VRPSD solution identified by Dror and Trudeau [18]. The operations of the two local search operators are described as follows.

1) Shortest Path Search: Dror and Trudeau [18] showed that given that the customers demands are independent random variables with non-negative means, route failures are

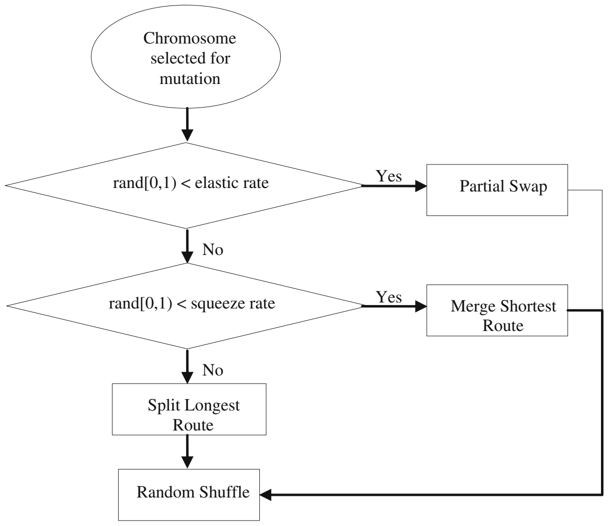


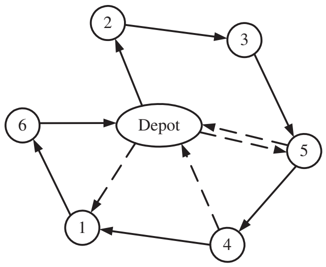
Fig. 4. Operation of multi-mode mutation

more likely to occur at the end of a route. Shortest Path Search (SPS) is designed to exploit this route structure. The SPS attempts to rearrange the order of customers in a particular route. For example, given a route that contains five customers, a new route is built by choosing the customer that is furthest from the depot as the first customer in the route, while the customer that is nearest to the depot is chosen as the last customer of the route. Next, the customer that is nearest to the first customer is chosen as the second customer, while the customer that is nearest to the last customer is chosen as the second last customer of the new route. This step continues until all the customers in the original route are re-routed. The new route will be compared against the original one and the better route will be retained. By re-routing customers in such a manner, customers that are further from the depot will be at the beginning of the route whereas those that are nearer to the depot will be at the end of the route. The rationale is to reduce the additional transportation cost that will be incurred by the recourse policy.

2) Which Directional Search: Which Directional Search (WDS) is designed to exploit the fact that the expected transportation cost of a route is dependent on the traversed direction [18]. In contrast, for the deterministic VRP, the transportation cost of a route is the same regardless of the direction in which the route is traversed. To be specific, given a route, the WDS builds a new route that runs in the opposite direction. Similarly, the new route will replace the original one if it is better.

Route Simulation Method

Another VRPSD-specific operator introduced in [23] is the route simulation method (RSM). As mentioned earlier, one of the main difficulties of solving the VRPSD is in



Customer	Real demand
1	5
2	6
3	2
4	13
5	9
6	5

Fig. 5. Example to show the operation of the RSM

finding an objective function that is able to define properly the expected transportation cost of a solution. This difficulty is overcome by the RSM, which was proposed to evaluate the expected costs of solutions. Fig. 5 will be used to illustrate the operation of the RSM.

Fig. 5 shows a route sequence, Depot → 2 → 3 → 5 → 4 → 1 → 6 → Depot. The solid arrows indicate the route that the vehicle will take if this were a deterministic VRP. In the VRPSD, due to the recourse policies in the event of a route failure, the actual route taken by the vehicle cannot be known with certainty before the route is actually implemented. However, the implementation of the route can be simulated by generating a set of demands of all the customers based on their demand distributions and treating these demands as if they were the real demands revealed when a vehicle first arrives at the customer. The set of demands generated is tabulated in Fig. 5. For this particular example, it is assumed that the vehicle capacity is 15 and each arrow indicates a unit of distance.

The vehicle first leaves the depot and arrives at customer 2. It is able to satisfy its demand with a remaining capacity of 9. The vehicle then travels to customer 3 and satisfies its demand. The capacity of the vehicle is 7 when it reaches customer 5. The vehicle then finds that it is unable to satisfy the demand of customer 5, so it unloads all remaining goods and makes a return trip to the depot to restock. This recourse is indicated by the dashed arrows between the depot and customer 5. The vehicle then unloads two units of goods and leaves customer 5 for customer 4 with a capacity of 13. After serving customer 4, the vehicle is empty and returns to the depot to restock. Since the demand of customer 4 has been satisfied, the vehicle travels to customer 1 from the depot. The vehicle then satisfies the demands of customers 1 and 6 and returns to the depot. From this simulation, the total distance traveled by the vehicle (10 units for this example) and the remuneration for the driver for a particular realization of the set of customer demands can be obtained. Due to the stochastic nature of the cost considered, there is a need to repeat the above operation N times for every route of a particular solution, using a different set of demands randomly generated based on the demand distributions of the customers each time and then taking the average to obtain the expected transportation cost of the solution. In this chapter, the N demand sets are regenerated before each fitness evaluation of the entire population.

Pareto Fitness Ranking

As mentioned in the introduction, the VRPSD is a multi-objective optimization problem where a number of objectives such as the travel distance, the remuneration for drivers, and the number of vehicles required, need to be minimized concurrently. In contrast to single-objective optimization, the solution to a multi-objective optimization problem exists in the form of alternate tradeoffs known as the Pareto optimal set. Each objective component of any non-dominated solution in the Pareto optimal set can only be improved by degrading at least one of its other objective components. In the total absence of information regarding the preference of objectives, a ranking scheme based upon the Pareto optimality is regarded as an appropriate approach to represent the fitness of each solution for multi-objective optimization. Thus, the role of multi-objective optimization in the VRPSD is to discover such a set of Pareto optimal solutions for which the decision maker can select an optimal solution based on the situation at hand. The Pareto fitness ranking scheme [26] for evolutionary multi-objective optimization is adopted in all the algorithms to assign the relative strength of solutions. The ranking approach assigns the same smallest rank for all non-dominated solutions, while the dominated ones are inversely ranked according to the number of solutions dominating them. In Fig. 6, a hypothetical solution (the black dot) is plotted in the objective domain at coordinates (1, 1, 1). Each solution will define a rectangular box encompassing the origin as shown in the figure. Another solution will dominate this solution if and only if it is within or on the box defined by the first solution but not equal to the first solution.

Archive Population

All the algorithms use an archive population to store all the best solutions found during the search. The archive population updating process consists of a few steps. The main population is first appended to the archive population. All repeated chromosomes, in

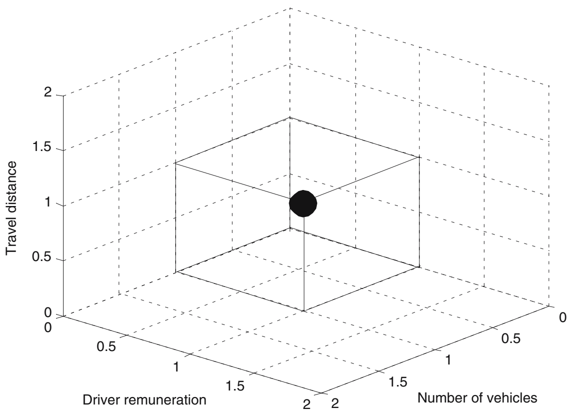


Fig. 6. Example to show how a solution is dominated by another solution

terms of the objective domain, are deleted. Pareto fitness ranking is then performed on the remaining chromosomes in the population. The higher ranked (weaker) chromosomes are then deleted such that the size of the archive population remains the same as before the updating process. The main population remains intact during the updating process. The final output of each of the algorithms is the non-dominated solutions in the archive population at the termination of the algorithm.

Computing Budget

The computing budget [27] represents a fixed amount of computational work. It is observed that the RSM requires intensive computations and can be regarded as the bottleneck of all the algorithms. As such, it makes sense to define one unit of the computing budget as one run of the RSM on a particular solution using a particular demand set. Thus, the computing budget puts a cap on the total number of times the RSM is run on solutions in the MOEA and can be regarded as the stopping criterion for all the algorithms. A computing budget of 2,000,000 is set for all the algorithms.

3.2 Different Versions of Each Algorithm

To observe the effects that the VRPSD-specific operators have on the performance of the algorithms, four versions of each of the algorithms are considered. The first version (NLSDET) does not involve any local search and instead of using the RSM, it evaluates the fitness of chromosomes based on the mean demands of customers, which represents a simplistic and deterministic approach to the stochastic problem. The second version (RANDET) is similar to the first version but it utilizes local search such that all the chromosomes have equal chance of being operated by SPS and WDS. The third version (NLSRSM) employs the RSM but does not involve any local search. The final version (RANRSM) incorporates both the RSM and the local search operators (local search setting is the same as that in RANDET) in the search process. *N* is set to 10 whenever the RSM is applied and local search occurs every 50 generations where applicable. A comparative study of the solutions obtained by these 20 settings will show how the performance of the algorithms is affected by local exploitation using SPS and WDS, and the RSM.

3.3 Implementation Details of Participating Algorithms

Random Search

As mentioned, random search (RS) is used as a basis for comparison and a good algorithm for the VRPSD should perform better than RS. This algorithm simply builds routes at random without any heuristic. It is implemented by setting the shuffle rate of the multi-mode mutation operator to 1. A new chromosome obtained by the multi-mode mutation operator will always replace the current one.

Hill-Climbing Search

In general, hill-climbing search (HC) will always try to move the search from the current state to a new state such that the new state is nearer to the optimal solution than the current state regardless of whether the search path would lead to the optimal solution.

In applying HC to the VRPSD, two implementations were used. As mentioned, for NLSDET and RANDET, a new chromosome that is generated by the multi-mode mutation operator will be evaluated based on the mean demands of customers. This new chromosome will replace the current chromosome if it dominates the current chromosome, otherwise it will be discarded and another new chromosome will be generated from the current one and evaluated. This process stops when the replacement of the current chromosome is successful and the search moves on to the next chromosome in the search population. In order to prevent the algorithm from wasting its computing budget on a particular chromosome for which it is unable to find a new chromosome that dominates the current one, the search will move on to the next chromosome in the population after 20 attempts. The versions that utilize the RSM for fitness evaluation, NLSRSM and RANRSM, differ from the implementation of the versions described above in that a new chromosome will replace the current one only if it satisfies two criteria. The first criterion is that the new chromosome's fitness when evaluated using the mean demand set must dominate that of the current one. If this criterion is satisfied, the new chromosome will then be evaluated based on the *N* randomly generated demand sets. It will replace the current chromosome if this new fitness also dominates that of the current chromosome. This is done to filter low quality chromosomes whose fitness when evaluated using the mean demand set fail to dominate that of the current chromosome. This cuts down on the usage of the computationally expensive RSM which consumes *N* units of the computing budget each time it is operated on a chromosome, thus allowing a larger search space to be explored The shuffle rate of the multi-mode mutation operator is set at 0.3 as in the MOEA.

Simulated Annealing

Simulated annealing (SA) is a stochastic relaxation technique proposed separately by Kirkpatrick et al. [28] and Cerny [29] to deal with combinatorial optimization problems. It exploits an analogy between the way in which molten metal cools and freezes into a minimum energy crystalline structure (the annealing process) and the search for a minimum in a more general system. SA involves a process in which the temperature of the system is gradually reduced during the simulation. Often, the system is repeatedly heated and cooled to reach a thermal equilibrium to allow it to surmount energetic barriers in a search for conformations with energies lower than the local-minimum energy found by energy minimization.

In terms of implementation, SA is similar to HC except that a new chromosome which does not dominate the current one is not discarded immediately but stands a chance to replace the current one. The acceptance of an inferior chromosome is analogous to the system "leaping" into a higher energy state where the current chromosome represents a lower energy state and the new chromosome represents a higher energy state. The probability of such "leaps" is higher at higher temperatures and for smaller energy differences between initial and final states and is given by the Boltzmann's energy distribution in Eqn. 5:

$$\text{Probability of accepting an inferior solution} = \exp(-\frac{\delta E}{T}) \tag{5}$$

where δE is the energy difference between the initial and final states and T is the current temperature of the system. δE is implemented as the Euclidean distance between the current and new chromosomes in the objective domain. Each chromosome in the search population has its own value of T which is initialized to 100 at the beginning of the search. The search is divided into epochs and the end of a particular epoch occurs when thermal equilibrium is reached. Thermal equilibrium is defined in this case to be achieved when the particular chromosome has not been replaced for the past 10 generations. The end of an epoch marks the start of the next epoch where T will be raised to 0.9 times of the value of T at the beginning of the previous epoch. Within each epoch, the value of T will be reduced by 0.9 times each time the search cycles through the entire population of chromosomes. This is done to simulate the repeated heating (at the end of each epoch) and cooling (within each epoch) to reach a thermal equilibrium to avoid the meta-stable states produced by quenching the molten metal.

Tabu Search

Tabu search (TS) is a memory-based meta-heuristic attributed to Glover [30, 31]. The algorithm explores part of the search space by moving at each generation to the best neighbor of the current state, even if this leads to a deterioration of the objective function. TS uses memory structures to support and encourage a non-monotonic search. It stores the most recent moves in a tabu list. Attempts that reverse the moves in the tabu list will be marked as "tabu" and be denied. However, an aspiration criterion can release this restriction if a move leads to a new global best solution. The lifetime of a tabu status in the tabu list is controlled by the tabu list size, where the first-in-first-out rule is often used for refreshing the list.

The implementation of TS is different from HC and SA in that at each attempt to replace a current chromosome with a new one, five candidate chromosomes are generated using the multi-mode mutation operator, from which one will be chosen to replace the current chromosome. For NLSDET and RANDET, the fitness of the five candidate chromosomes will be evaluated using the mean demand set, whereas for NLSRSM and RANRSM, their fitness will be evaluated using the RSM. The chromosomes will then be ranked according to the Pareto fitness ranking scheme. Going down the ranks, starting from the non-dominated (among the five) chromosomes, a candidate chromosome will replace the current one if it satisfies the aspiration criterion of dominating any non-dominated chromosome in the archive population. Otherwise, it will still be able to replace the current chromosome if none of its routes is in the tabu list. Each chromosome in the main population will have its own tabu list which will be updated each time a new chromosome successfully replaces the current one. The routes in the current chromosome that have been altered to obtain the new chromosome will be kept in the tabu list for 10 generations. The search moves on to the next chromosome in the search population after the first successful replacement. If all the five chromosomes fail to satisfy either of the two criteria for replacing the current chromosome, the current chromosome will remain intact and the search moves on to the next chromosome in the search population. Similar to the MOEA, HC, and SA, the shuffle rate associated with the multi-mode mutation operator is set at 0.3.

4 Simulation Results and Analysis

4.1 Background Results

To appreciate the results presented in this section, it is required to review some of the results and findings of Tan et al. [23], which are listed as follows.

1) The robustness of a VRPSD solution is measured by the increase in travel distance and driver remuneration from the respective expected costs after actual implementation. To quantify the robustness of a VRPSD solution, a test demand set is randomly generated based on the customers demand distributions. This test demand set represents the real demands that the vehicles of the solution would experience when the solution is actually implemented. The implementation costs of the solution are then computed based on the test demand set. The difference between the implementation costs and the respective expected costs, which are computed during the optimization process, indicates the robustness of the solution. A robust solution should have its implementation costs not too different from its expected costs since the expected costs are what the logistic manager sees when he is deciding on which solution in the Pareto set to implement.

2) The overall quality of a Pareto set of VRPSD solutions is measured by a single metric known as the average actual multiplicative aggregate. The computation of this metric is illustrated with reference to Table 1. In Table 1, expected travel distance and expected driver remuneration are the respective costs averaged over the non-dominated solutions in the archive population (Pareto solutions) at the termination of the algorithm. Increase in travel distance and increase in driver remuneration are the median values of the increase in the respective costs after implementation using a particular test demand set. Actual travel distance and actual driver remuneration are the sum of the respective expected costs and increase in costs. The actual multiplicative aggregate [32] for the test demand set is the product of actual travel distance and actual driver remuneration. The above process is repeated for a number of test demand sets, all of which are randomly generated based on the customers demand distributions. By averaging the actual multiplicative aggregates obtained over the different test demand sets, the average actual multiplicative aggregate is computed.

3) For each test problem, there is a trade-off value of *N*, the number of demand sets used by the RSM for each fitness evaluation, i.e. there is a value of *N* that will result in the lowest average actual multiplicative aggregate. It is found that a larger value of *N* will produce more robust solutions whose expected costs are better approximations of the actual costs of implementation. However, due to the fixed computing budget, the corresponding smaller number of generations used in the MOEA will result in poorer routing solutions as there is insufficient time to explore the search space.

4) It is found that given two same-sized problems, i.e. they involve the same number of customers, the decision would be to use a larger value of *N* for the problem where the

customers demands are highly unpredictable, i.e. the variances of the demand distributions are high, to obtain robust solutions. On the other hand, given two problems with equal stochastic level of customer demands, the decision would be to use a smaller value of *N* for the problem with a larger search space, i.e. the problem which involves more customers, to allow a more extensive exploration of the search space since a smaller *N* value corresponds to a larger number of generations used by the MOEA.

5) It is found that the *N* trade-off value for the 75-customer DT86 test problem of Dror and Trudeau [18] is 10. A performance comparison of the MOEA using the RSM and using the deterministic approach, where solutions are evaluated based on the mean demand set, on DT86 showed that although the deterministic approach churns out solutions with lower expected costs, the increase in costs after implementation is higher, leading to an inferior average actual multiplicative aggregate.

Table 1. Example to illustrate the computation of the actual multiplicative aggregate

Expected travel distance	Increase in travel distance	Actual travel distance	Expected driver remuneration	Increase in driver remuneration	Actual driver remuneration	Actual multiplicative aggregate ($\times 10^6$)
1086.59	33.85	1120.44	985.95	32.87	1018.82	1.142

4.2 Description of Test Problems

The test problems used in this chapter are adapted from the vehicle routing problem with time window (VRPTW) benchmark problems designed by Solomon [33]. The original design of Solomon's VRPTW benchmark problems [33] highlights several factors, which can affect the behavior of routing and scheduling algorithms, of which the topology of customers is the concern of this chapter. The benchmark problems consist of 56 data sets which can be categorized into three main classes based on the spatial distribution of customers. For the Type-R problems, all the customers are remotely located, whereas for the Type-C problems, the customers are grouped into a few clusters. The Type-RC problems consist of a mixture of remote and clustered customers. Customer information, such as the number of customers (each instance consists of 100 customers), their locations, demands, and service times, within each category is identical. Using these three sets of customer information, three test problems are created. Like the DT86 test problem of Dror and Trudeau [18], to adapt the customer information to the VRPSD, the original demand quantity is used as the mean demand of each customer. The standard deviation of the demand distribution of each customer is generated using a uniform random number generator so that it falls between zero and one-third of the mean demand of the customer. The stochastic versions of the Type-R, Type-C, and Type-RC problems of Solomon [33] will be referred to as Type-RS, Type-CS, and Type-RCS, respectively.

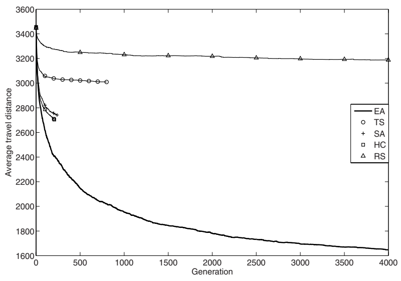
Simulations were conducted on the Type-RS, Type-CS, and Type-RCS test problems using the NLSDET, RANDET, NLSRSM, and RANRSM versions of the MOEA (EA), TS, SA, HC, and RS. Each of the 20 settings underwent 10 simulation runs to obtain statistical results. The results for each of the test problems are shown in the following sections.

4.3 Type-RS Test Problem

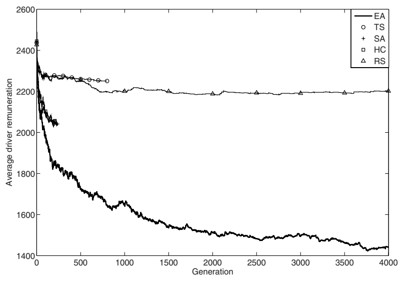
Fig. 7(a)-(h) shows the convergence traces of the 20 settings on the Type-RS test problem. Each convergence trace shows the convergence of the respective costs, averaged over all the non-dominated solutions in the archive population, over the generations.

An immediate observation from the convergence traces in Fig. 7(a)-(h) is the generation disparity between the five algorithms. Regardless of the version, for the same computing budget of 2,000,000, EA and RS use many more generations than TS, SA, and HC. This implies that TS, SA, and HC require heavy usage of the fitness evaluation function and may not be suitable for solving the VRPSD, where fitness evaluations are more computationally demanding than the deterministic VRP. Putting the generation disparity aside, it can be seen that at any generation, the performance of EA is usually the best among the algorithms. For NLSDET and NLSRSM, where the convergence traces are not affected by the sudden dips caused by the effective local exploitation every 50 generations, the gradients of the convergence traces of EA are steeper than those of the other four algorithms. This implies that the search operations of EA are more effective in exploring the search space and finding better solutions. For RANDET and RANRSM, after the initial local search operation, TS, SA, HC, and RS are usually trapped in some local optima where the convergence traces stay flat over a large number of generations. Regardless of the version, such a situation does not apply to EA since its convergence traces are able to decline steadily throughout the search, albeit in varying degrees. It is also observed that for the versions that do not use local search, NLSDET and NLSRSM, the convergence traces of EA have not converged even at the 4,000th generation. In comparison, the convergence traces of EA for RANDET and RANRSM have almost converged by the 150th generation. If the logistic manager is satisfied with a good enough solution, the search process for the versions with local search can actually be stopped at the 150th generation since the solutions found at the end of the search process for these settings are not much better than those found at the 150th generation. Local exploitation using SPS and WDS can speed up the whole search process thus saving much computational effort. It is also observed that the application of local search seems to benefit TS more than SA, HC, and RS. For NLSDET and NLSRSM, the performance of TS betters only RS. However, with local search, the performance of TS only falls behind EA. The route structures of the solutions found by TS seem to be more exploitable by the local search operators of SPS and WDS.

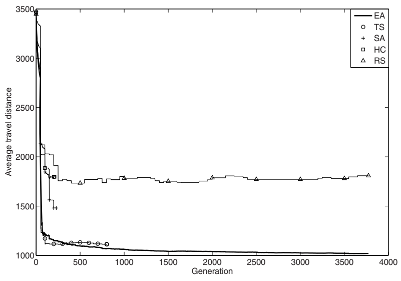
The non-dominated solutions obtained by the 20 settings are then implemented on four test demand sets and the solution robustness comparison results are represented in box plots and shown in Fig. 8(a)-(b). Each box plot represents the distribution of the deviations between the implementation and expected costs where the horizontal line within the box encodes the median, and the upper and lower ends of the box are the upper and lower quartiles respectively. The two horizontal lines beyond the box give an indication of the spread of the data. A plus sign outside the box represents an outlier.



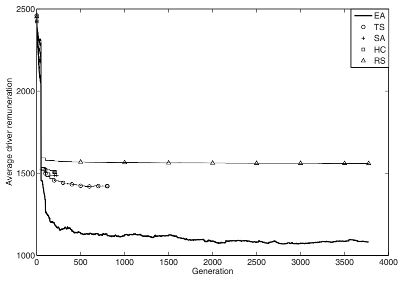
(a)



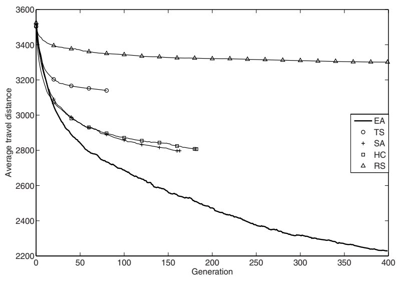
(b)



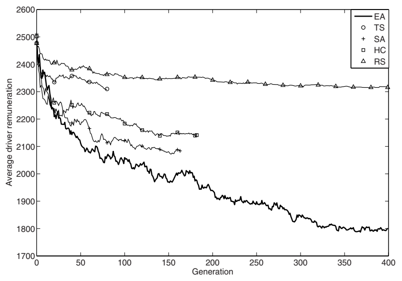
(c)



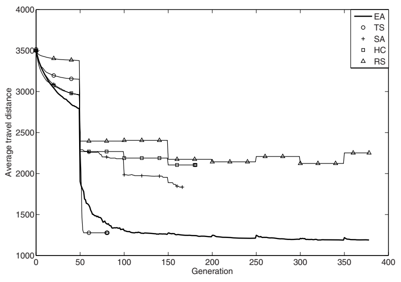
(d)



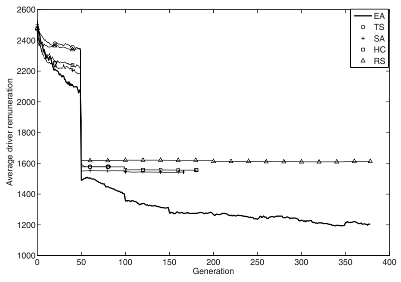
(e)



(f)

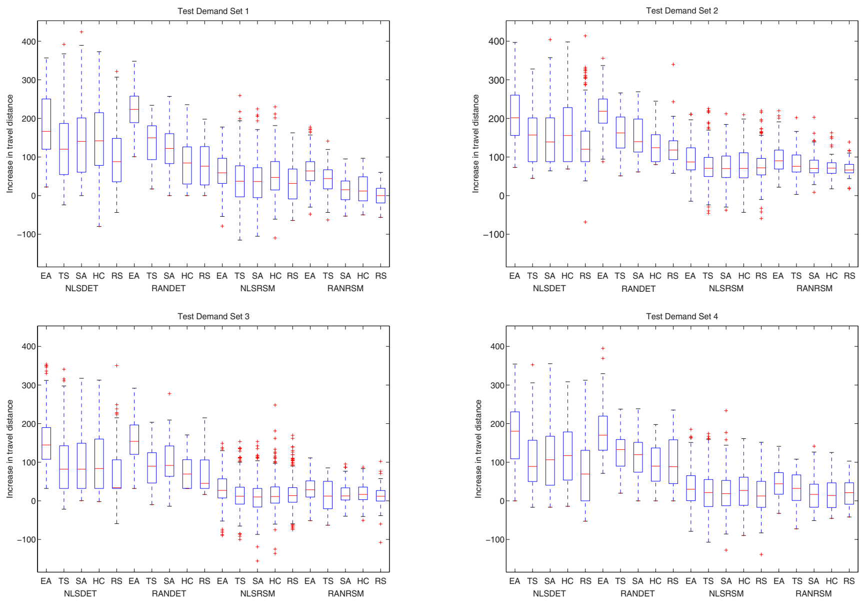


(g)

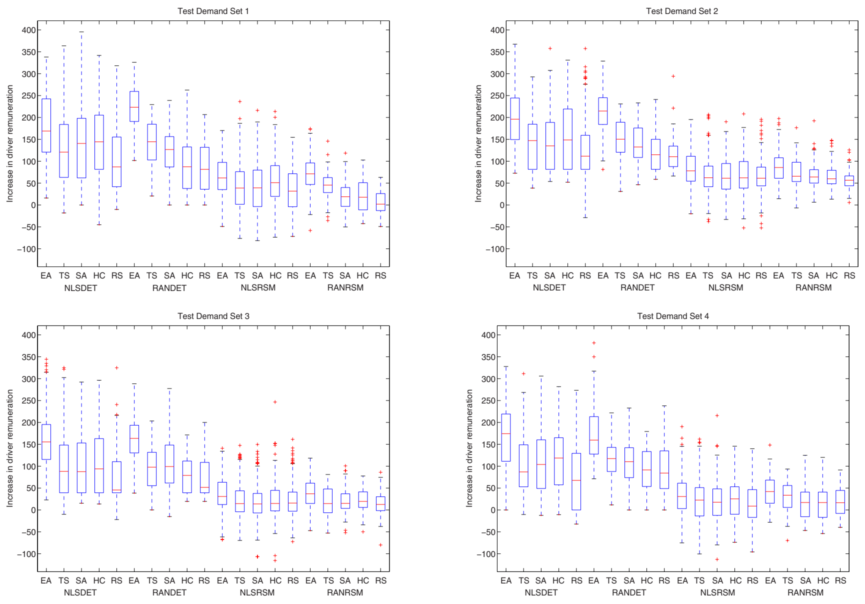


(h)

Fig. 7. Average travel distance and average driver remuneration of non-dominated solutions of the NLSDET (a)-(b), RANDET (c)-(d), NLSRSM (e)-(f), and RANRSM (g)-(h) versions of the five algorithms for the Type-RS test problem



(a)



(b)

Fig. 8. Increase in (a) travel distance and (b) driver remuneration after implementing non-dominated solutions of the four versions of the five algorithms for the Type-RS test problem

Although the convergence traces showed that the performance of EA is the best among the five algorithms, the box plots show otherwise. The solutions found by EA are not as robust to the stochastic nature of the problem as those of the other algorithms since the increases in travel distance and driver remuneration after implementing the solutions on the four test demand sets are the largest regardless of version. One probable explanation for this could be that more fitting solutions, i.e. those with lower travel distance and driver remuneration, are less robust and the actual costs of these solutions are likely to suffer greater deviations from the expected costs. This explanation is justified as the ranking of the performance of the algorithms given by the box plots is generally inversely related to the ranking of the performance given by the convergence traces. It can also be seen that the versions of the algorithms that make use of the RSM produce more robust solutions. This is to be expected from the studies in [23]. However, it is also observed that the application of the RSM seems to reduce the disparity in the robustness of solutions of the algorithms i.e. the box plots for the algorithms of NLSRSM and RANRSM are more level with one another compared to those for the algorithms of NLSDET and RANDET. This also meant that EA has benefited the most from the application of the RSM. Another important observation is that on top of the fact that the RSM is able to provide robust solutions, the local search operators of SPS and WDS seem to be able to improve on the robustness of solutions as well. It can be seen in Fig. 8(a)-(b) that the spreads of the box plots for RANDET and RANRSM are smaller than those for NLSDET and NLSRSM, respectively. This meant that the synergy of the local search operators and the RSM can lead to solutions which are highly robust to the stochastic nature of the problem.

Having seen that EA performed the best in the comparison of the convergence traces and performed the worst in the comparison of the box plots, it would be useful to compare the performance of the algorithms taking these two factors into consideration. Table 2 compares the overall performance of the 20 settings on the Type-RS test problem. In Table 2, inter-algorithm ranking is the performance ranking among algorithms having the same version setting, while inter-version ranking is the ranking among the different versions of a particular algorithm. Rank 1 represents the best performer, the one with the smallest average actual multiplicative aggregate.

From Table 2, it is obvious that the performance of EA is the best among the five algorithms since it has an inter-algorithm rank of 1 in all the four versions. Consistent with the earlier observation that TS has benefited the most from the application of the local search operators, the inter-algorithm rank of TS is 2 in RANDET and RANRSM but becomes 4 in NLSDET and NLSRSM. The performance of SA is generally better than HC since it allows the acceptance of inferior chromosomes and thus is less likely to be trapped in local optima. Comparison of the inter-version ranking shows that the performance of the algorithms of RANDET are generally better than their counterparts of the other three versions, including RANRSM. This contradicts the findings in [23], where it was shown that the MOEA performs better with the RSM compared to the deterministic approach. One probable explanation could be that the *N* value of 10 used in the simulations of the 20 settings on the Type-RS test problem is not the trade-off value. In [23], the experiments were conducted on the 75-customer DT86 test problem using an *N* value of 10 which was shown to be the trade-off value for that particular

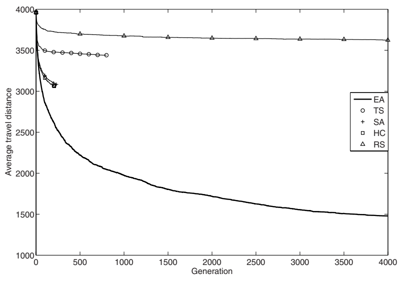
Table 2. Overall performance comparison of the 20 settings on the Type-RS test problem

Version	Algorithm	Actual multiplicative aggregate ($\times 10^6$)	Inter-algorithm ranking	Inter-version ranking
NLSDET	EA	2.938	1	3
	TS	7.372	4	3
	SA	6.169	2	4
	HC	6.175	3	3
	RS	7.446	5	3
RANDET	EA	1.541	1	1
	TS	1.930	2	1
	SA	2.566	3	1
	HC	3.023	4	1
	RS	3.103	5	1
NLSRSM	EA	4.222	1	4
	TS	7.444	4	4
	SA	5.994	2	3
	HC	6.204	3	4
	RS	7.818	5	4
RANRSM	EA	1.574	1	2
	TS	2.134	2	2
	SA	2.932	3	2
	HC	3.380	4	2
	RS	3.723	5	2

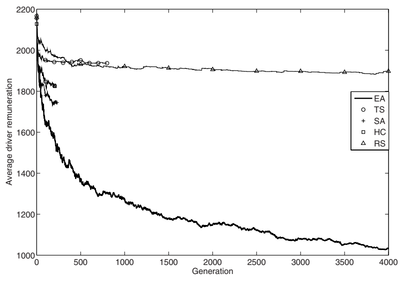
test problem. Since the method of obtaining the variances of the customer demand distributions for the 75-customer DT86 problem of Dror and Trudeau [18] was adopted in the creation of the Type-RS test problem, the stochastic level of customer demands for the two test problems are comparable. It was found in [23] that given two problems with equal stochastic level of customer demands, the decision would be to use a smaller value of *N* for the problem which involves more customers. This meant that the trade-off value of *N* for the 100-customer Type-RS test problem should be smaller than that of the 75-customer DT86 test problem.

4.4 Type-CS Test Problem

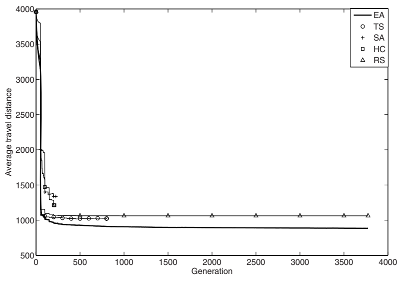
Like in the previous section, the convergence traces and box plots, considering only the non-dominated solutions, for the 20 settings on the Type-CS test problem are plotted in Fig. 9(a)-(h) and Fig. 10(a)-(b), respectively. Similar to Table 2, Table 3 compares the overall performance of the 20 settings on the Type-CS test problem. As the behaviors of the 20 settings are very similar in both the Type-CS and Type-RS test problems, only the differences will be highlighted.



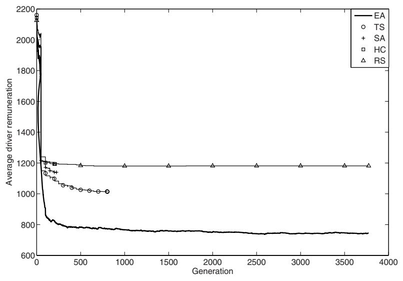
(a)



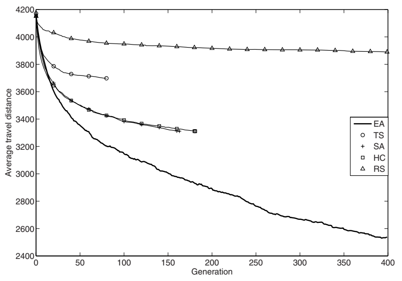
(b)



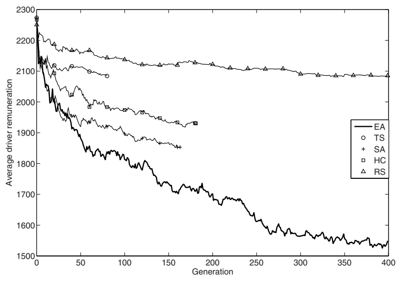
(c)



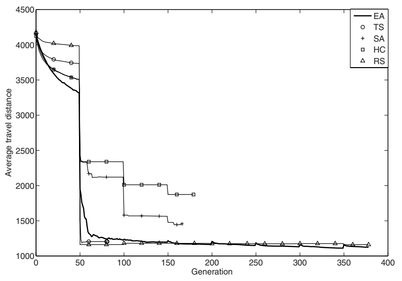
(d)



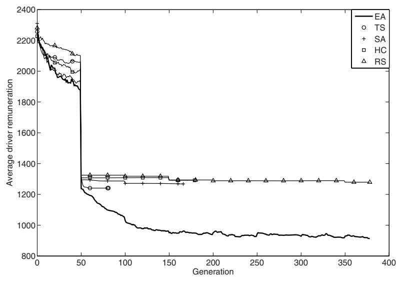
(e)



(f)

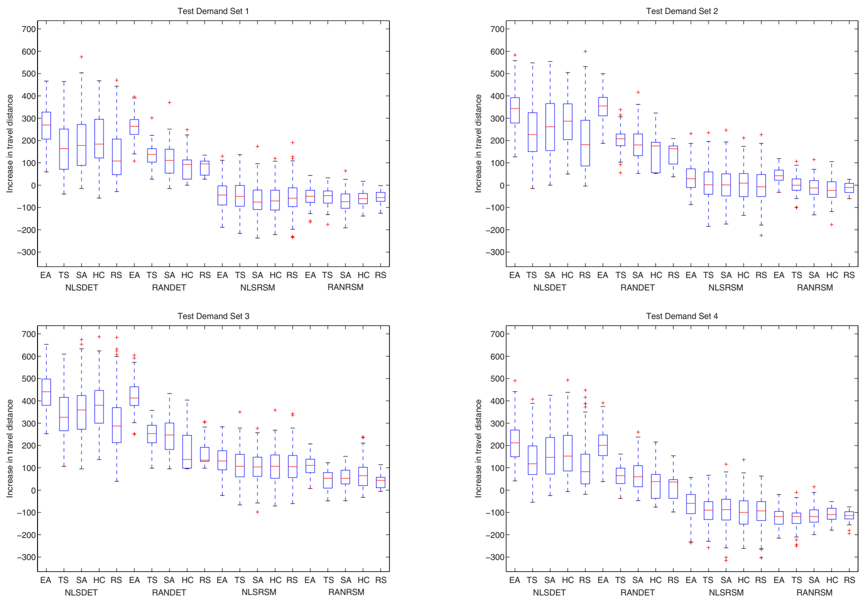


(g)

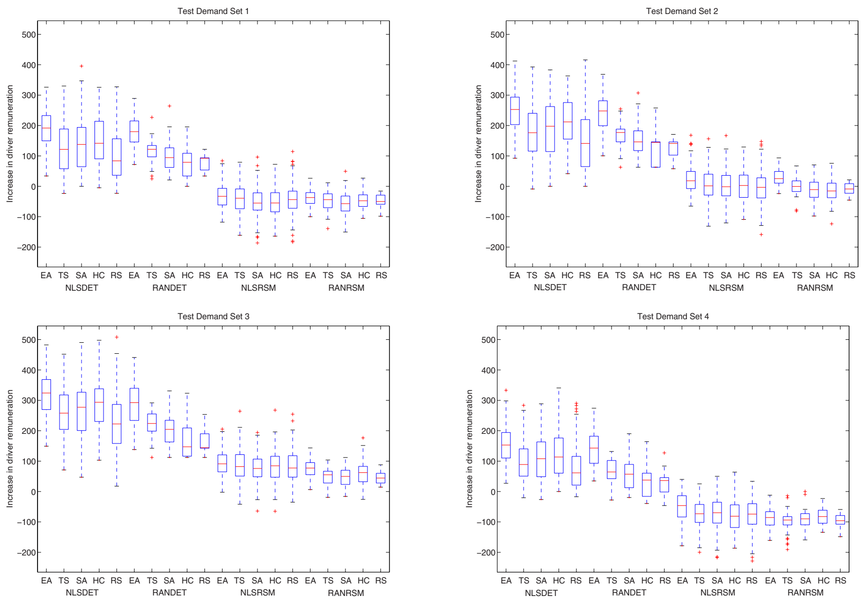


(h)

Fig. 9. Average travel distance and average driver remuneration of non-dominated solutions of the NLSDET (a)-(b), RANDET (c)-(d), NLSRSM (e)-(f), and RANRSM (g)-(h) versions of the five algorithms for the Type-CS test problem



(a)



(b)

Fig. 10. Increase in (a) travel distance and (b) driver remuneration after implementing non-dominated solutions of the four versions of the five algorithms for the Type-CS test problem

It was mentioned in the previous section that for the Type-RS test problem, the application of local search had benefited TS more than SA, HC, and RS. For the Type-CS test problem, it can be observed from Fig. 9(c) and Fig. 9(g) that the application of local search can bring the performance of TS and RS, in terms of travel distance, to the level of EA.

An interesting observation from the box plots for the Type-CS test problem is that the improvements in the robustness of solutions when the RSM is applied are generally greater than the improvements for the Type-RS test problem. This can be seen by comparing the differences between the medians of the box plots for a particular algorithm of NLSDET and NLSRSM (These two versions differ only in the usage of the RSM), or RANDET and RANRSM for the Type-CS and Type-RS test problems. The differences between the medians are generally larger for the Type-CS test problem. This observation seems to imply that the effectiveness of the RSM is more pronounced in the Type-CS test problem.

Unlike the Type-RS test problem, where the algorithms of RANDET performed better than their counterparts of the other three versions, comparing the inter-version ranking for the Type-CS test problem, RANRSM performs slightly better than RANDET with three algorithms ranked 1 and two algorithms ranked 2. It appears that the trade-off value of *N* for the Type-CS test problem is slightly closer to 10 than that for the Type-RS test problem so that conducting the simulations using an *N* value of 10 provides better results. It was shown in [23] that the trade-off value of *N* depends on the number of customers and the stochastic level of their demands. In the case of the Type-RS and Type-CS test problems, the number of customers and the stochastic level of their demands are comparable, so one probable reason for the difference in the trade-off values of *N* could be that the trade-off value depends on the topology of the customers as well.

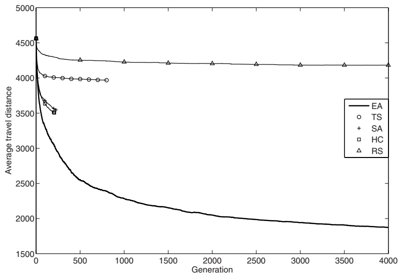
4.5 Type-RCS Test Problem

The convergence traces and box plots, considering only the non-dominated solutions, for the 20 settings on the Type-RCS test problem are plotted in Fig. 11(a)-(h) and Fig. 12(a)-(b), respectively. Table 4 compares the overall performance of the 20 settings.

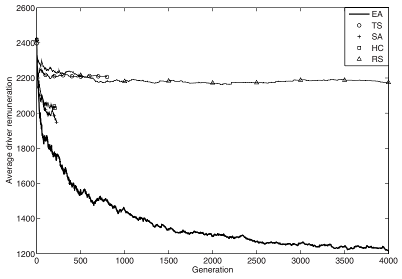
Like in the Type-CS test problem, it can be observed from Fig. 11(c) and Fig. 11(g) that the application of local search in the Type-RCS test problem can bring the performance of TS and RS, in terms of travel distance, to the level of EA.

From the box plots for the Type-RCS test problem, it can again be seen that the effectiveness of the RSM is more pronounced in the Type-RCS test problem than the Type-RS test problem. It seems that the presence of clustered customers is the reason for this observation.

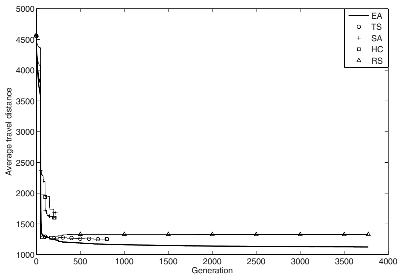
From Table 4, it can be seen that the behaviors of the 20 settings are very similar in both the Type-CS and Type-RCS test problems, where RANRSM performs slightly better than RANDET with three algorithms ranked 1 and two algorithms ranked 2. From the results of the Type-CS and Type-RCS test problems, it seems that the trade-off values of *N* for both problems are slightly closer to 10 than that for the Type-RS test problem since RANRSM is able to perform better than RANDET in both problems. It



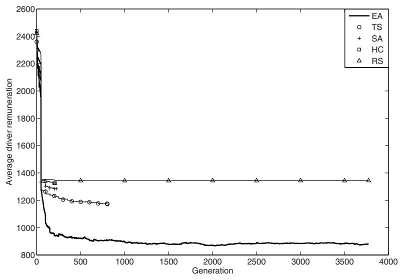
(a)



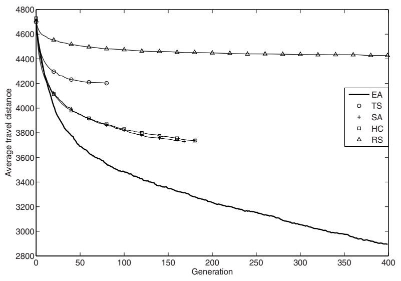
(b)



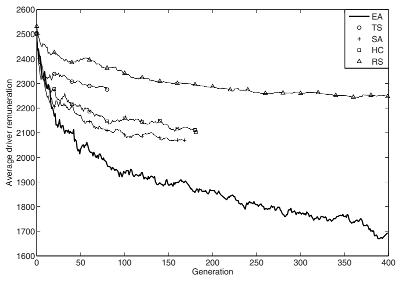
(c)



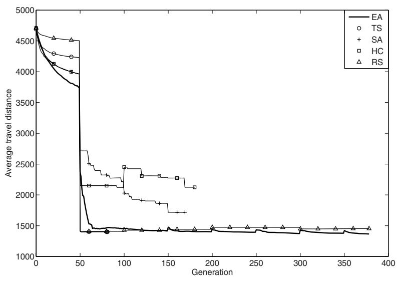
(d)



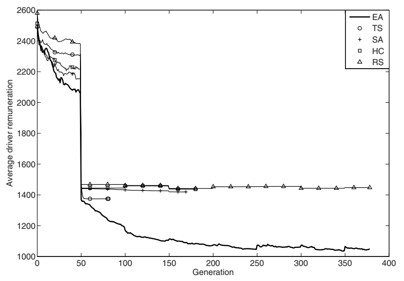
(e)



(f)

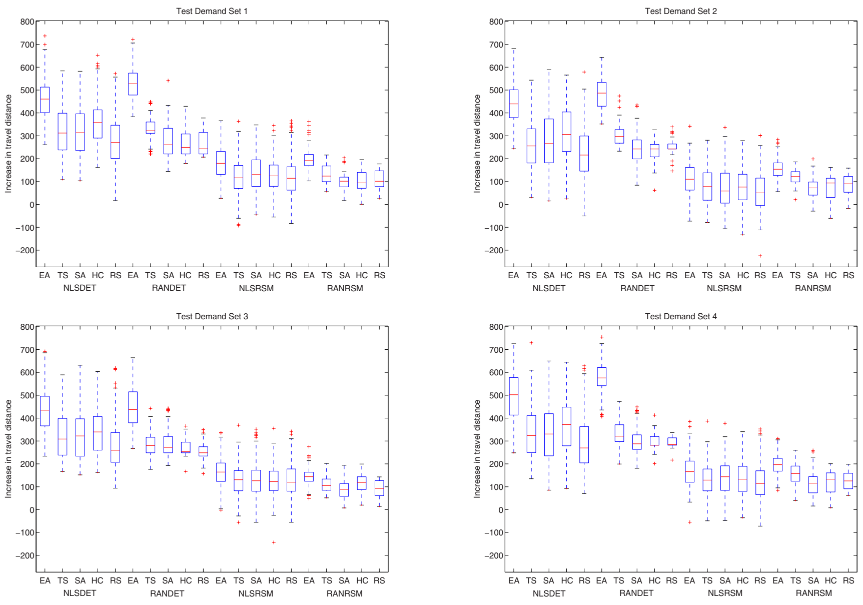


(g)

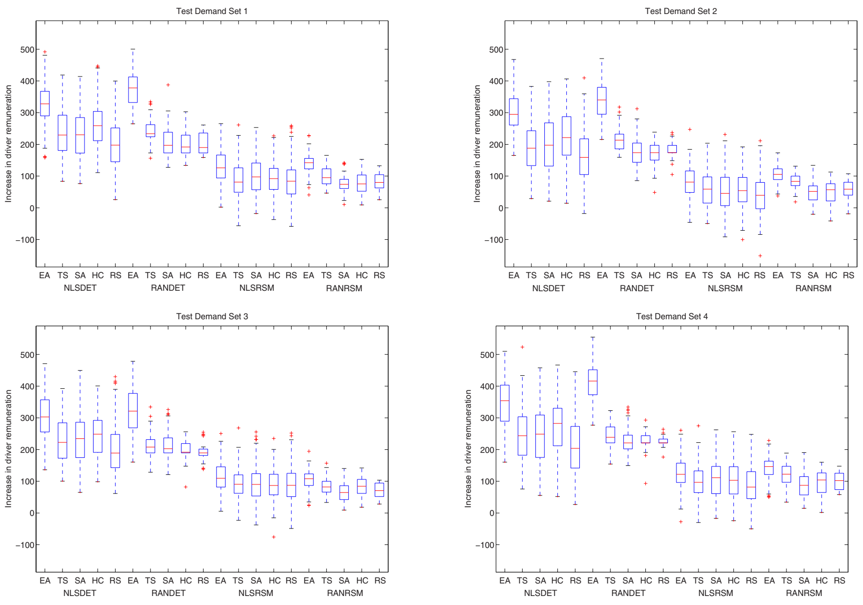


(h)

Fig. 11. Average travel distance and average driver remuneration of non-dominated solutions of the NLSDET (a)-(b), RANDET (c)-(d), NLSRSM (e)-(f), and RANRSM (g)-(h) versions of the five algorithms for the Type-RCS test problem



(a)



(b)

Fig. 12. Increase in (a) travel distance and (b) driver remuneration after implementing non-dominated solutions of the four versions of the five algorithms for the Type-RCS test problem

Table 3. Overall performance comparison of the 20 settings on the Type-CS test problem

Version	Algorithm	Actual multiplicative aggregate ($\times 10^6$)	Inter-algorithm ranking	Inter-version ranking
NLSDET	EA	2.273	1	3
	TS	7.658	4	3
	SA	6.396	2	4
	HC	6.678	3	4
	RS	7.674	5	3
RANDET	EA	1.149	1	2
	TS	1.385	2	1
	SA	1.884	5	2
	HC	1.717	4	1
	RS	1.505	3	2
NLSRSM	EA	3.978	1	4
	TS	7.668	4	4
	SA	6.072	2	3
	HC	6.331	3	3
	RS	8.045	5	4
RANRSM	EA	1.023	1	1
	TS	1.440	3	2
	SA	1.765	4	1
	HC	2.344	5	2
	RS	1.414	2	1

was also observed that the effectiveness of the RSM is more pronounced in the Type-CS and Type-RCS test problems compared to the Type-RS test problem. Table 5 shows the average improvements achieved by applying the RSM on the three test problems. Average improvement is calculated by averaging the differences between the median values of the increase in travel distance as well as the median values of the increase in driver remuneration for each of the five algorithms of NLSDET and its counterpart of NLSRSM as well as for each of the five algorithms of RANDET and its counterpart of RANRSM over the four test demand sets. A higher average improvement value signifies that the effectiveness of the RSM is more pronounced in the particular problem. It can be gathered from Table 5 that the effectiveness of the RSM gets more pronounced with the increase in the number of clustered customers. As the effectiveness of the RSM gets more pronounced, it makes sense to use a larger value of N to exploit the effectiveness of the RSM, while sacrificing some generations of the algorithm. It is for this reason that the trade-off values for the Type-CS and Type-RCS test problems are closer to 10 than that for the Type-RS test problem. From this analysis, it can be seen that given two problems with equal number of customers and equal stochastic level of the customer demands, it would be desirable to choose a larger value of N for the problem with more clustered customers.

Table 4. Overall performance comparison of the 20 settings on the Type-RCS test problem

Version	Algorithm	Actual multiplicative aggregate ($\times 10^6$)	Inter-algorithm ranking	Inter-version ranking
NLSDET	EA	3.586	1	3
	TS	10.362	4	4
	SA	8.389	2	4
	HC	8.788	3	4
	RS	10.483	5	3
RANDET	EA	2.032	1	2
	TS	2.178	2	1
	SA	2.887	5	2
	HC	2.825	4	1
	RS	2.436	3	2
NLSRSM	EA	5.499	1	4
	TS	10.174	4	3
	SA	8.288	2	3
	HC	8.420	3	3
	RS	10.503	5	4
RANRSM	EA	1.806	1	1
	TS	2.246	2	2
	SA	2.697	4	1
	HC	3.388	5	2
	RS	2.374	3	1

Table 5. Average improvements achieved by applying the RSM

Problem type	Average improvement
RS	85.146
RCS	177.581
CS	195.481

5 Conclusions

In this chapter, a few problem-specific operators, including two search operators for local exploitation and the route simulation method (RSM) for evaluating solution quality, are hybridized with several meta-heuristics, including tabu search and simulated annealing, and tested on three VRPSD test problems adapted from the popular Solomon's vehicle routing problem with time window (VRPTW) benchmark problems. For all the test problems, the MOEA, equipped with the VRPSD-specific operators, is able to produce high quality, robust solutions. The performance of the MOEA is the best among the five algorithms used in the test. In the process of comparing the algorithms, the contributions of the local search operators and the RSM to the performance of the MOEA are also displayed. It is also found that the trade-off value of N, the number of demand sets used by the RSM for each fitness evaluation, for a particular instance of the VRPSD is dependent not only on the number of customers and the stochastic level of their demands but also on the topology of the customers.

References

1. Prins, C.: A simple and effective evolutionary algorithm for the vehicle routing problem. Computers and Operations Research 31(12), 985–2002 (2004)
2. Hwang, H.S.: An improved model for vehicle routing problem with time constraint based on genetic algorithm. Computers & Industrial Engineering 42(2–4), 361–369 (2002)
3. Thangiah, S.R., Potvin, J.Y., Sun, T.: Heuristic approaches to vehicle routing with backhauls and time windows. Computers and Operations Research 23(11), 1043–1057 (1996)
4. Tan, K.C., Lee, T.H., Ou, K., Lee, L.H.: A messy genetic algorithm for the vehicle routing problem with time window constraints. In: Proceedings of the 2001 Congress on Evolutionary Computation, Seoul, Korea, vol. 1, pp. 679–686 (2001)
5. Tan, K.C., Lee, L.H., Zhu, Q.L., Ou, K.: Heuristic methods for vehicle routing problem with time windows. Artificial Intelligence in Engineering 15(3), 281–295 (2001)
6. Tan, K.C., Lee, T.H., Chew, Y.H., Lee, L.H.: A multiobjective evolutionary algorithm for solving vehicle routing problem with time windows. In: Proceedings of the IEEE International Conference on Systems Man and Cybernetics, Washington DC, USA, vol. 1, pp. 361–366 (2003)
7. Potvin, J.Y., Bengio, S.: The vehicle routing problem with time windows - part II: genetic search. INFORMS Journal on Computing 8(2), 165–172 (1996)
8. Yang, W.H., Mathur, K., Ballou, R.H.: Stochastic vehicle routing problem with restocking. Transportation Science 34, 99–112 (2000)
9. Laporte, G., Louveaux, F.V.: Solving stochastic routing problems with the integer L-shaped method. In: Crainic, T.G., Laporte, G. (eds.) Fleet Management and Logistics, pp. 159–167. Kluwer Academic Publishers, Boston (1998)
10. Gendreau, M., Laporte, G., Séguin, R.: Stochastic vehicle routing. European Journal of Operational Research 88(1), 3–12 (1996)
11. Bertsimas, D.J., Jaillet, P., Odoni, A.R.: A priori optimization. Operations Research 38, 1019–1033 (1990)
12. Jaillet, P., Odoni, A.R.: The probabilistic vehicle routing problem. In: Golden, B.L., Assad, A.A. (eds.) Vehicle Routing: Methods and Studies. North-Holland, Amsterdam (1988)
13. Jaillet, P.: Stochastic routing problems. In: Andreatta, G., Mason, F., Seraf-ini, P. (eds.) Stochastics in Combinatorial Optimization. World Scientific, New Jersey (1987)
14. Dror, M., Ball, M.O., Golden, B.L.: Computational comparison of algorithms for inventory routing. Annals of Operations Research 4, 3–23 (1985)
15. Larson, R.C.: Transportation of sludge to the 106-mile site: An inventory routing algorithm for fleet sizing and logistic system design. Transportation Science 22, 186–198 (1988)
16. Lambert, V., Laporte, G., Louveaux, F.V.: Designing collection routes through bank branches. Computers and Operations Research 20, 783–791 (1993)
17. Teodorović, D., Lucić, P.: Intelligent vehicle routing system. In: Proceedings of the IEEE International Conference on Intelligent Transportation Systems, Dearborn, MI, USA, pp. 482–487 (2000)
18. Dror, M., Trudeau, P.: Stochastic vehicle routing with modified savings algorithm. European Journal of Operational Research 23(2), 228–235 (1986)

19. Dror, M.: Modeling vehicle routing with uncertain demands as a stochastic program: Properties of the corresponding solution. European Journal of Operational Research 64(3), 432–441 (1993)
20. Charnes, A., Cooper, W.: Deterministic equivalents for optimizing and satisfying under chance constraints. Operations Research 11, 18–39 (1963)
21. Charnes, A., Cooper, W.: Chance-constrained programming. Management Science 6, 73–79 (1959)
22. Stewart, W.R., Golden, B.L.: Stochastic vehicle routing: A comprehensive approach. European Journal of Operational Research 14(4), 371–385 (1983)
23. Tan, K.C., Cheong, C.Y., Goh, C.K.: Solving multiobjective vehicle routing problem with stochastic demand via evolutionary computation. European Journal of Operational Research 177, 813–839 (2007)
24. Teodorović, D., Pavković, G.: The fuzzy set theory approach to the vehicle routing problem when demand at nodes is uncertain. Fuzzy Sets and Systems 82(3), 307–317 (1996)
25. Gendreau, M., Laporte, G., Séguin, R.: A tabu search heuristic for the vehicle routing problem with stochastic demands and customers. Operations Research 44(3), 469–477 (1996)
26. Fonseca, C.M.: Multiobjective genetic algorithms with application to control engineering problems, Ph.D. Thesis, University of Sheffield, Sheffield, UK (1995)
27. Lee, L.H., Chew, E.P.: A simulation study on sampling and selecting under fixed computing budget. In: Proceedings of the 2003 Winter Simulation Conference, New Orleans, LA, USA (2003)
28. Kirkpatrick, S., Gellat, L., Vecchi, M.: Optimization by simulated annealing. Science 220, 671–680 (1983)
29. Cerny, V.: Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. Journal of Optimization Theory and Applications 45, 41–51 (1985)
30. Glover, F.: Tabu search part I. ORSA Journal on Computing 1, 190–206 (1989)
31. Glover, F.: Tabu search part II. INFORMS Journal on Computing 2, 4–32 (1990)
32. Van Veldhuizen, D., Lamont, G.B.: Multiobjective evolutionary algorithm research: a history and analysis. Department of Electrical and Computer Engineering, Air Force Institute of Technology, Ohio, Technical Report TR-98-03 (1998)
33. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research 35(2), 254–265 (1987)

Exploiting Fruitful Regions in Dynamic Routing Using Evolutionary Computation

J.I. van Hemert[1] and J.A. La Poutré[2,3]

[1] National e-Science Centre, School of Informatics, University of Edinburgh, 15 South College Street, Edinburgh EH8 9AA, United Kingdom
J.vanHemert@ed.ac.uk

[2] Dutch National Research Institute for Mathematics and Computer Science, P.O. Box 94079, NL-1090 GB Amsterdam, The Netherlands
Han.La.Poutre@cwi.nl

[3] School of Technology Management, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Summary. We introduce the concept of fruitful regions in a dynamic routing context: regions that have a high potential of generating loads to be transported. The objective is to maximise the number of loads transported, while keeping to capacity and time constraints. Loads arrive while the problem is being solved, which makes it a real-time routing problem. The solver is a self-adaptive evolutionary algorithm that ensures feasible solutions at all times. We investigate under what conditions the exploration of fruitful regions improves upon the effectiveness of the evolutionary algorithm. A technique called anticipatory routing is used with the aim to increase the number of successfully transported loads with respect to time and capacity constraints.

1 Introduction

Vehicle routing forms an interesting line of research, its high complexity and intractable nature [1], as well as because of its importance to the transport industry. In the past, vehicle routing was studied as a static problem, but with the problems of increasing traffic, the increasing demand of flexibility of customers, and the increasing amount of computing, facilities, and communication, current studies are focusing on vehicle routing under dynamic conditions.

As a static problem, vehicle routing is studied using various models. A comprehensive overview is found in [2, 3]. Here we will focues on a pick-up-and-delivery problem, i.e., where goods are collected from various sites and then delivered to a central location. A preliminary version of this study is presented in [4].

Two different concepts of dynamic routing exist. At first, dynamic routing was studied by constructing a priori routes of which the quality was assessed afterwards in a model with stochastic uncertainty on certain aspects of the problem [5], such as whether a customer would actually need to be serviced. Later, dynamic routing was studied using real-time simulations, where the routing algorithm runs in parallel with the developing problem, which enables it to react to arising conditions in the problem [6]. Furthermore, it was addressed theoretically through the worst-case analysis of fundamental cases in on-line vehicle routing algorithms [7, 8, 9].

Within the context of real-time dynamic routing, it becomes important for a routing algorithm to react to changes in the problem. These changes can originate from normal operation when for instance, a customer calls in to put in a request for service, or from exceptions such as bad traffic conditions or a breakdown of vehicles. Most of the time, a dynamic problem changes due to normal operations, and when these constitute a large part of the problem, we are able to predict some of them, for example, by analysing the customer requests over a period in history [10]. If we incorporate the knowledge from such an analysis into a routing algorithm we might be able to improve the quality of routing by anticipating future changes. Here we shall study the influence of the constraints in a dynamic vehicle routing problem, where the central question is whether the use of this knowledge can improve the effectiveness of a routing algorithm. In our study, the effectiveness is measured as the percentage of successfully transported loads while adhereing to set time and capacity constraints. To do so, we assume that an analysis of customer requests was already performed. Moreover, we will introduce the concept of fruitful regions. These fruitful regions consist of spatially clustered customers that together have a high potential of demand. By incorporating the potential of customers into the routing algorithm we directly aim to improve the overall effectiveness, that is, we aim to increasing the ratio of successfully transported loads while respecting the time and capacity constraints.

In this chapter, dynamic routing problem instances are generated artificially and are solved simultaneously using an evolutionary algorithm, also referred to in the literature as a genetic algorithm. The process by which the instances are generated enforces the settings of the constraints that we are considering. We show that the exploration of fruitful regions can improve the solution quality of a routing algorithm by increasing the effectiveness, i.e., increasing the number of successfully transported loads without violating set time and capacity constraints. The main part involves the search for the problem conditions where this benefit is achieved. To this end we design two variants of the evolutionary algorithm: one without and one with exploring fruitful regions. For the latter, a self-adaptive mechanism [11] is added that gives the algorithm control over how much exploration is performed. We choose this technique over deterministic methods because of its flexibility with respect to dynamic problems. A change in the problem can easily be taken into account, without the need for restarting the algorithm. Furthermore, by employing a repair mechanism we can ensure a feasible solution at all times.

In the next section we discuss the concepts of dynamic and anticipatory routing. The contributions of this work are: in Section 3, a mathematical model of a dynamic pickup and deliver-to-a-depot problem. In Section 4, a formal notion of the concept of fruitful regions is given. In Section 5, an evolutionary algorithm is provided that is able to explore fruitful regions. To determine the favourable conditions for the exploration of fruitful regions, a problem generator is introduced in Section 6.1. The problem instances created with this generator are used in the empirical study described in Section 6.2 and in Section 6.3. Finally in Section 7 we elaborate on results from this study.

2 Dynamic and Anticipatory Routing

Often in practice, a problem may develop over time, that is, some variables in the problem may change or be set at a later stage such that a new evaluation of the problem is required. Basically, this inserts uncertainties and lack of information into the problem. The literature distinguishes between two ways of analysing such problems, stochastic routing and dynamic routing [12]. The latter is also referred to as real-time routing or on-line routing [8, 13]. In stochastic routing we are presented with a routing problem that has a number of its variables stochastically determined. This still requires a solution to the static problem, i.e., an a priori solution, but we can use only an expected value of the optimisation function. Examples of stochastic variables are uncertain supplies [5] and stochastic travel times [14]. The dynamic routing problem requires that the solver keeps waiting for events to occur to which it should act accordingly. Such events may be more customer requests, but they can also be changed travel times or other incidents that endanger the success of the current plan. A solver should thus make sure that it creates its solution to the current status in such a way that it may be altered successfully to changes in the future. We are interested in the second approach as these become more and more common in practice, due to an increasing amount of computing and communication resources and due to an increasing amount of flexibility demanded by customers [6, 15].

In the same book [16] where Psaraftis [17] laid out the differences between dynamic and static vehicle routing problems, Powell [18] first reviews the idea of forecasting the uncertainties within dynamic routing. Powell's work is targeted towards solving assignment problems, where the objective is to assign available jobs to candidate solution vehicles such that the whole fleet of vehicles is able to maintain a steady flow of work. In [19], a model is presented that deals with long-haul transport, which has led to a hybrid model that assigns jobs to trucks and forecasts jobs in future time periods. In this model, distances and travel times are ignored, thus, Powell's work is essentially about work flows, and not about dynamic routing.

Our objective is to use the concept of demand forecasting and use this in dynamic routing problems that occur in real-time, i.e., the vehicles are on route when the problem changes. This is different to the work of Laporte and F. Louveaux [5, 14], which is concerned with creating routes beforehand. It is also different to Powell's work on forecasting, where demand forecasting is used in combination with the assignment problem in the application area of dispatching. In the assignment problem, the routing aspect, i.e., the optimisation problem where one minimises route length or the number of vehicles, is fully ignored in favour of having the fleet spread over the network such that the expected loads are covered. We combine the concept of demand forecasting with vehicle routing in the regular combinatorial sense.

Under the assumption that knowledge on uncertain parameters in a dynamic routing problem is known beforehand, we can incorporate this knowledge in a routing algorithm to make it anticipate future changes, and thereby attempt to increase the quality of solutions. Here we shall use *anticipatory routing* as a means of using knowledge on the potential of customer demands. In anticipatory routing, vehicles are routed to locations that have not yet demanded service. By using the knowledge on demand potentials, an algorithm can then try to provide more efficient routes by strategically locating the fleet of vehicles such that it is ahead of future changes. The success of this approach depends on how well the routing algorithm can balance the routing for current events and the anticipatory routing.

Work closer to ours is described in [20, 21, 22], where the concept of anticipatory routing is used in dynamic path planning problems under stochastic customer demand. Essentially, the objective is to find the shortest path between two given nodes through a set of customers, where the set of customers is not fully specified at the start. The model is limited in a computational complexity sense, making it impossible to use multiple vehicles or enforce capacity constraints. Contrary to that model, we will study a vehicle routing problem where multiple vehicles are allowed to perform multiple rounds from and to the depot. Another notable difference is that in our model capacity and time constraints are enforced. To cope with the computational complexity that arises with this model we employ evolutionary computation to create routes in real-time.

Another approach related to our work is that of waiting strategies, where vehicles are given explicit orders to wait at a certain location in anticipation of further work [23]. In [24], an application of this approach in combination with evolutionary computation is shown to increase the number of customers serviced with less detours. In [25] a parallel Tabu search is applied to a pickup problem with a similar mechanism that lets vehicles wait, hoping for any future service requests in the neighbourhood to appear. In contrast, our approach is based on the location of anticipated future customers and not on waiting strategies, vehicles are dispatched to customers that have not yet requested service.

Summarising, in this chapter the application lies in transport that occurs in and between areas with a dense distribution of customers, where routing is required to achieve cost effective solutions. The problem is dynamic in a sense that customer requests are handled on-line, and the vehicle routing is solved in real-time. Furthermore, by using anticipatory routing, the solver can possibly make use of the knowledge about regions that have a high-potential for generating service requests. We call these fruitful regions, which we shall define in Section 4.

3 Definition of the Problem

A routing problem consists of a set of nodes N, connected by a graph $G = (N, E)$ with edges E that are labelled with the cost to traverse them, a set of unique loads L and a set of vehicles V. An optimisation function determines the goal of the routing problem, which is further specified by a set of constraints that places restrictions on the aforementioned resources. Here we consider loads that need to be picked up from nodes and then delivered to a central depot.

A load $l \in L$ is a quadruple $\langle s, d, t, \Delta \rangle$, where $s \in N$ is the source node, and $d \in \mathbb{N}$ is the size of the load, which is fixed to one here. $t \in T$ is the entry time, i.e., the time when the load is announced to the solver, with T the time period of the problem. Every $l \in L$ load should be delivered to a central depot $n_0 \in N$ within Δ time steps, i.e., before $t + \Delta$. Here we assume that Δ is equal for each load. Note that this definition allows customers to issue several requests.

A vehicle $v \in V$ is characterised by a capacity q and a start position $sp \in N$. We assume that all vehicles have the same capacity and start at the same position. Each vehicle has a speed of 1 and starts at the depot $sp = n_0$. The dynamic status of a vehicle v, i.e., its route, assigned loads and current position, is defined with two functions. Both functions use the assumption that a vehicle always travels directly from node to node, i.e., it is not possible to stop half way and choose another route. Ichoua et al. [26] show that under certain conditions the number of successful customer request may increase when a vehicle's route can be diverted on route to the next customer. As we do not allow this, the model enables expressing the position of a vehicle just in terms of nodes. In the simulation described in Section 5.2 we explain the benefits of this approach. We define the current position of a vehicle v as $p_v \in \mathbb{N}$ where p_v is the index of the node last visited by v in vector of nodes in $\mathbf{r}$. Then, we define the current situation using v and p_v as,

$$\text{current}(v, p_v) = \langle \mathbf{r}_{\leq p_v}, A_{\leq p_v}, A_{p_v} \rangle, \tag{1}$$

where $\mathbf{r}_{\leq p_v} = (r_0, \ldots, r_j)$ is an ordered list of nodes where for $0 \leq i \leq j : r_i \in N$ and $r_0 = sp$, denoting the route of the vehicle through the network up until, and including, node r_j where $r_j = p_v$. $A_{\leq p_v}$ is the set of loads that were assigned, and were collected and delivered, to the central depot by v so far. A_{p_v} is the set of loads that were assigned and were collected by v so far and not yet delivered.

$$\text{future}(v, p_v) = \langle \mathbf{r}_{> p_v}, A_{> p_v} \rangle, \tag{2}$$

where $\mathbf{r}_{> p_v} = (r_{p_v+1}, \ldots, r_k)$ is an ordered list of nodes with $r_i \in N$, which denotes the route planned for vehicle v from the next location r_{p_v+1} onwards. $A_{> p_v}$ is the set of loads that are assigned to be collected by v in the future. We note that r_{p_v+1} may not be altered by the routing algorithm, if the corresponding vehicle has already "left" r_{p_v}. This way we enforce vehicles to drive only directly from node to node. The remainder of the route, i.e., $r_{p_v+2}, \ldots, r_k$ can be changed.

We define some additional functions that are helpful in our further formulation of the problem. The function arrival-time(v, p_v) returns the time vehicle v arrives at node $r_{p_v} \in N$ as the p_v-th node of its route. The function loads(v, p_v) returns the set of all loads assigned to vehicle v; loads$(v, p_v) = A_{\leq p_v} \cup A_{p_v} \cup A_{> p_v}$. The constraints of the routing problem are given next.

$$\forall v \in V, \forall p_v : \text{future}(v, p_v) =$$
$$\langle (r_{p_v+1}, \ldots, r_k), A_{> p_v} \rangle \wedge p_v + 1 = k \Rightarrow A_{> p_v} = \emptyset \tag{3}$$
$$\forall v_1, v_2 \in V, \forall p_{v1}, p_{v2} : v_1 \neq v_2 \Rightarrow$$
$$\text{loads}(v_1, p_{v1}) \cap \text{loads}(v_2, p_{v2}) = \emptyset \tag{4}$$
$$\forall v \in V, \forall p_v : \forall \langle s, d, t, \Delta \rangle \in A_{p_v} \cup A_{> p_v} :$$
$$\text{arrival-time}(v, p_v) \geq t \tag{5}$$
$$\forall l = \langle s, d, t, \Delta \rangle \in L : \forall v \in V : \text{current}(v, p_v) =$$
$$\langle r_{\leq p_v}, A_{\leq p_v}, A_{p_v} \rangle \wedge \langle s, d, t, \Delta \rangle \in A_{p_v} \wedge \exists p_v{}' :$$
$$r_{p_v{}'} = d \wedge p_v \leq p_v{}' \Rightarrow \text{arrival-time}(v, p_v{}') \leq t + \Delta \tag{6}$$
$$\forall \langle v \rangle \in V : \forall p_v : \text{ current}(v, p_v) = \langle r_{p_v}, A_{\leq p_v}, A_{p_v} \rangle \Rightarrow$$
$$\sum_{\langle s,d,t,\Delta \rangle \in A_{p_v}} d \leq q \tag{7}$$

The constraint (3) restricts vehicles from carrying loads at the end of their route. To enforce that a load is carried by at most one vehicle the constraint (4) is used. The notion of time appears in three places; in the definition of a load, where the *entry time* is set, in constraint (5), where we make sure loads are only carried after their entry time, and in constraint (6), where we make sure loads are delivered at their destination within the fixed time Δ after entering the system. Finally, constraint (7) is introduced to have vehicles carry only as much as their capacity lets them.

Three differences with the classic static routing problem can be identified in this dynamic model:

1. vehicles may drive routes where they pass through nodes without performing a loading or unloading action,
2. loads may become available while vehicles are already on route,
3. it is not a hard constraint to have each load assigned to a vehicle.

The latter enables us to choose as the objective of the routing problem to carry as many loads as possible, i.e., to maximise the number of loads successfully delivered at the central depot: let p_v be the last position of vehicle v at the end of time period T, then the objective function is:

$$\max \sum_{v \in V} |\text{loads}(v, p_v)| \, .$$

In the experiments we shall report as a performance measure the success ratio, which is defined as the number of loads successfully delivered at the depot divided by the total number of loads generated during a run of the whole system:

$$\text{success ratio} = \frac{\sum_{v \in V} |\text{loads}(v, p_v)|}{|L|}$$

The algorithm described in Section 5 uses knowledge about the potential of customers, which tells it how likely customers are to put in a request. The potential of a customer $n \in N$ is defined as the sum of the sizes of the loads requested by n divided by the sum of the sizes of the loads requested by all the customers,

$$\text{potential}(n) = \frac{\sum_{\langle n,d,t,\Delta \rangle \in L} d}{\sum_{\langle n',d',t',\Delta' \rangle \in L} d'}$$

4 Fruitful Regions

In practice, customers are often clustered into regions as opposed to scattered around equally over the total domain [27]. Often, the amount of service requested in total from one region may differ much from another. This is due to all types of reasons, such as the capacity of production facilities and the amount of customers in one region. Depending on the distribution of the customers and the distribution of service requests, it might be beneficial for a vehicle to visit nodes in the customer graph that have a high probability for a service request in the near future. The decision to drive the additional distance to a potential customer depends also on hard constraints, such as the capacity of the vehicle.

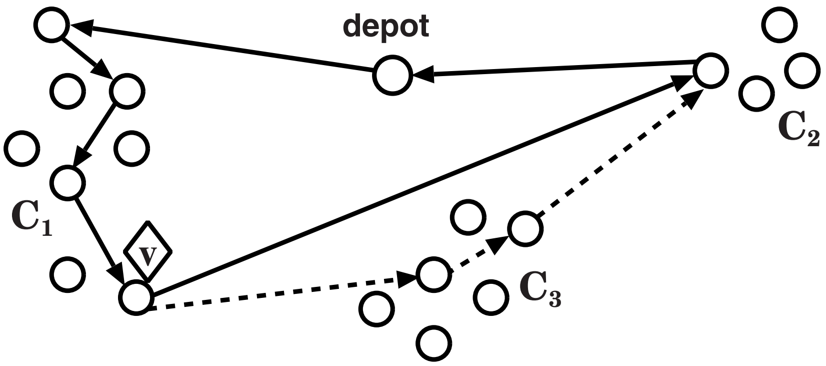


Fig. 1. Three clusters of nodes with one node to represent the depot in the centre, and a vehicle v on route from the depot now stationed in one of the nodes at C_1

The grouping of customers into clusters is achieved either a posteriori, by observing the distribution of service requests in historical data, or a priori, by modelling them explicitly. In both cases we assume that a partition $C = \{C_1, C_2, \ldots, C_c\}$ of clusters exists in the layout of the problem, where each $C_i \subseteq N$. Next, we define the vector $\mathbf{f} = (f_1, f_2, \ldots, f_{|N|})$, where f_j is the probability that a load in the simulation originates from customer $i \in N$. The following holds: $\sum_{j \in N} f_j = 1$. We define the potential of a cluster C_i as the sum over the probabilities of its nodes: $\sum_{j \in C_i} f_j$.

An example of three clusters and one central depot is presented in Figure 1. The vehicle v that left the depot and then visited several nodes in C_1, is now on its route to a node in C_2. If the potential for new loads in C_3 is high enough, it might want to change its route to visit nodes in C_3 along the dotted lines as this could provide another load without much additional driving.

5 Evolutionary Algorithm

Evolutionary computation has been applied to many different combinatorial optimisation problems. Over the past forty years [28], it has proven especially successful in practical settings because of the flexibility with which domain knowledge can be added to the general outline of an evolutionary algorithm. Moreover, many general ways for improving the efficiency and effectiveness of an evolutionary algorithm exist, often in the form of heuristics or hybrid solutions [29].

In the context of dynamic optimisation, evolutionary computation has the advantage of being able to cope with changes in the environment during its optimisation [30]. By maintaining a diverse set of candidate solutions, the evolutionary algorithm can adapt to the changes. Evolutionary algorithms can quickly converge to a suboptimal or optimal solution. Together with the current speed of processors, this makes it possible to keep providing solutions to a dynamic routing problem.

Bräysy and Gendreau [31] have reviewed many successful evolutionary algorithms and other meta-heuristic approaches for the static vehicle routing problem. These were often tested using the well known Solomon benchmark [27]. All these implementations make extensive use of heuristics and other hybrid solutions to increase the efficiency and effectiveness of the final routing algorithm. Opposite to the abundance of literature on solving static vehicle routing with evolutionary computation is the small amount of publications on applying this technique to dynamic variants. One such publication [32] shows a preliminary study on how to change the fitness function such that delays and overloading of vehicles can be taken into account in the optimisation process.

Our objective is to verify whether or not the concept of fruitful regions can help increase performance by increasing the effectiveness. We shall start with a straightforward representation of the problem together with basic operators, then add the possibility to move vehicles to fruitful regions. Precise details of the evolutionary algorithm are given next.

5.1 An Evolutionary Algorithm for Dynamic Routing

Below we describe the evolutionary algorithms used for creating the routes in the experiments in Section 6. Two variants of this evolutionary algorithm will be used, one that explores fruitful regions and one that does not explore fruitful regions.

Self-adaptive mechanism. When fruitful regions are considered, a self-adaptation mechanism [11] is used that lets a candidate solution decide for itself how much it values to explore such regions. This employs an alpha parameter (α), which is set by each candidate solution itself. This value becomes part of the hereditary process which is the main idea behind self-adaptation in evolutionary computation. Its setting lies in the range of $[0, 1]$, where small values correspond to a low importance in exploring fruitful regions. The influence of the alpha parameter on the evolutionary algorithm is described in the fitness function and in the variation operator.

The standard self-adaptation [29, Section 7.1] of the alpha parameter α is used,

$$\alpha' = \alpha \exp\left((2|V|)^{-1/2} N(0,1) + \left(2|V|^{1/2}\right)^{-1/2} N(0,1) \right)$$

where α' replaces α and will be used for the next mutation operator, $N(0,1)$ is a random number from a Gaussian distribution with a mean of 0 and a standard deviation of 1.

Representation. Each candidate solution represents a basic plan for all the vehicles in the system, where every load available for pickup is considered. It is a list of routes through customers that have a load available for pickup, where each route corresponds to exactly one vehicle. A route consists of potential assignments for that vehicle, which it will perform in that order. Assignments are either pickup assignments or moves to a location. This representation is then transformed into a feasible solution, i.e., a solution where none of the constraints is violated. Assignments that violate one or more time constraints are cancelled. The process of repairing the representation into a feasible plan is described in more detailed below. The moves to a location, i.e., anticipated moves, are used only with the variant of the evolutionary algorithm that tries to explore fruitful regions.

Initialisation and new requests. At the start, i.e., at $t = 0$, every available load $l \in \{\langle s, d, t, \Delta\rangle \in L | t = 0\}$ is randomly assigned to a vehicle. The self-adaptive parameter

α, if used, is drawn uniform randomly from the range $[0, 1]$. The population size, i.e., the number of candidate solutions, is set to thirty, which forms a trade-off between having a large population for keeping sufficient diversity and having a small population that restricts the usage of computational resources.

New requests issued by customers are inserted into the basic plan by first uniform randomly select a vehicle and then adding the request to the end of its route. Note, it is left up to the variation operator to optimise the best order of pickup and deliveries.

Repair mechanism. A repair mechanism is used to transform the representation into a feasible solution. A feasible solution consists of a set of routes, one for each vehicle, where each route contains assignments such that all the constraints in (3)–(7) are satisfied. The representation only contains assignments involving pickups of loads or anticipated moves. Such assignments are called *feasible loads* and *feasible anticipated moves*. When, after scheduling a number of pickups or anticipated moves, a vehicle reaches its capacity or when adding another assignment would violate a time constraint of any of the previously assigned loads, the repair mechanism first forces a visit to the depot. Afterwards, this vehicle may be deployed to service customers again, thus the repair mechanism continues too. The procedure described here is the same as used in [14]. The fitness of the candidate solution will be based on the repaired solution. Although the repairing process may have a large impact on the fitness landscape [33], it is necessary as in a dynamic environment we must be able to produce feasible solutions on demand.

During a simulation, the vehicles need to be fed with routes. These routes will be taken from the best candidate solution as determined by the fitness function described below. Note that, over time the routes may be changed as other candidate solutions become available. Only the current destination of vehicles must remain unaltered.

Fitness Function. The basis for the fitness function is the number of loads that can be transported successfully, i.e., the number of feasible loads found by the repair mechanism. The fitness of a candidate solution x is then defined as,

$$\text{fitness}_1(x) = \frac{\text{\# feasible loads}}{\text{\# total loads available}}.$$

Only available loads are considered, that is, loads that are not yet picked up. The rationale here is that loads are removed from candidate solutions of the evolutionary algorithm once they are picked up. As candidate solutions are only concerned with future routing these loads play no role and should therefore not be included in the evaluation of a candidate solution. Note that for assessing the overall quality of the evolutionary algorithm at the end of a run, we will use every load that existed in the system over the whole period T.

When exploring fruitful regions is switched on, the fitness function, which has to be maximised, is extended with anticipated moves, i.e., moves that have no direct loading action attached. These moves are, similar to the pickup of loads, performed only when the constraints allow this. Thus, some anticipated moves may be cancelled during repairing to make sure that the final plan satisfies the constraints placed upon a vehicle because of previous assigned actions, such as, picking up a load. If we would perform such a move, the planning would become infeasible and hence, such a move is called an *infeasible anticipated move*. A candidate solution is penalised for every infeasible anticipated move, this in order to restrict the number of anticipated moves.

The fitness of a candidate solution is decreased by the fraction of infeasible anticipated moves out of all planned anticipated moves. The amount with which this decreases the fitness function depends on the alpha parameter,

$$\text{fitness}_2(x) = \text{fitness}_1(x) - \alpha \frac{\text{\# infeasible anticipated moves}}{\text{\# total anticipated moves of } x}.$$

The effect of the self-adaptive mechanism is intricate, as it requires a period of time to find a good value for α, i.e., a value that provides a good balance between routing on the one hand and exploring fruitful regions on the other. Moreover, the exploration of fruitful regions must be sufficiently beneficial to be switched on, because for $\alpha > 0$, a penalty is incurred for infeasible anticipated moves to prevent the algorithm from flooding vehicles with anticipated moves. Individuals that do not have a good value will have a high chance of being removed from the population at a later time of the simulation, where the lack or abundance of anticipated moves is shown to be an ineffective strategy. Thus, α is determined by the ability to survive in a dynamic setting. The conditions under which a positive α is beneficial will be explored in the experiments section.

Selection. A 2-tournament selection is used to select a candidate solution for mutation. A generational scheme is used, known in evolution strategies as (μ, μ), with elitism of size one [11]. The elitism prevents the best solution from being lost. We apply a generational scheme instead of a steady state scheme as it circumvents the problem of re-evaluating individuals after a change in the environment [34].

Variation operator. A mutation operator is used only, no crossover operator is used. During a mutation operator, two vehicles, possibly the same one, are chosen uniform randomly. Then in both vehicles, one node is selected uniform randomly. If only one vehicle is chosen these nodes are chosen to be distinct. Then, the nodes are swapped. This way, visits, both for loading or as an anticipated move, can be exchanged between vehicles as well as within the route of one vehicle.

If exploration of fruitful regions is enabled, the candidate solution's alpha parameter is changed according to the common update rule from evolution strategies. A firm introduction to evolutionary strategies and its self-adaptive mechanism is found in [11, Chapter 2]. Here we make use only of one parameter, which is changed according to a Gaussian distribution.

Furthermore, each vehicle is considered for the insertion of an anticipated move. A customer is randomly chosen as the destination of this anticipated move, where the probability of selecting a customer equals its potential. The customer is added to the end of the current stack of work of the vehicle. It is up to the variation operator of the evolutionary algorithm to optimize its location within the route.

Stop condition. The algorithm is terminated once the time period T is expired, i.e., at the end of the simulation.

5.2 Dynamism

In practice, dealing with a dynamic problem means that we have limited time to contemplate the solution as vehicles need to be kept going and changes may be on their way. In reality we have the world and the solver running in parallel, and then we need to decide how often the solver's view of the world is updated. Here, we choose a time period based on the number of times the evolutionary algorithm evaluates one of its candidate solutions. These fitness function evaluations take up most of the time in an evolutionary algorithm. For each time unit of the simulation the evolutionary algorithm may perform one generation. Thus the evolutionary algorithm will perform *population size* $\times$ *time steps* $(= 30|T|)$ fitness function evaluations in a simulation run.

The whole simulation operates by alternately running the evolutionary algorithm and the simulated routing problem. An overview of the process is shown in Figure 2. The routing simulator calculates when the next event will occur, e.g., a vehicle will pickup or deliver a load, or, a load is announced for pickup. Then, the evolutionary algorithm may run up until this event occurs, i.e., it is allowed to perform X iterations of its loop. This way we simulate an interrupt of the evolutionary algorithm when it needs to

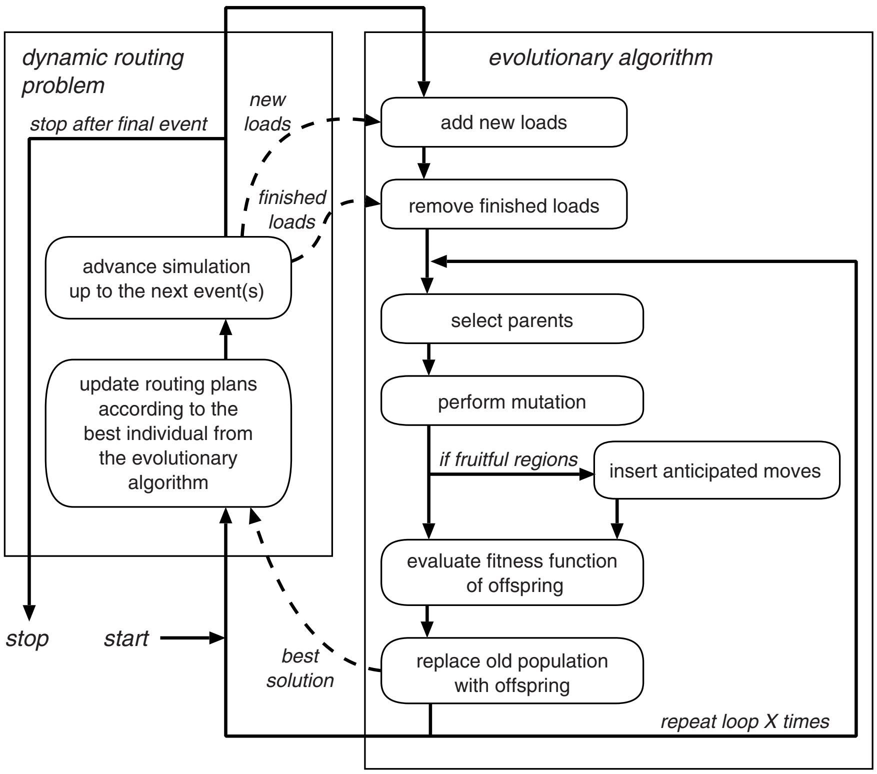


Fig. 2. Overview of the simulation, which starts with the first events in the dynamic problem and then alternates between the dynamic problem and the evolutionary algorithm

adapt to changes in the real world. The best candidate solution from the last generation before the “interrupt” is used to update the assignments of the vehicles in the routing simulation. Then, the routing problem is advanced up until the next event, which is made possible by the representation introduced in Section 3 that expresses the position of vehicles in terms of the previous and next node. Afterwards, the candidate solutions of the evolutionary algorithm are updated by removing finished assignments and adding loads recently made available. This process is repeated for time period T. Note that it is possible that multiple events occur at the same time, which will be handled in one go.

6 Experiments

6.1 Creating Test Sets

In the test problems used in Section 6, the nodes are on a 2-dimensional rectangle. The distance between any two nodes is the Euclidean distance. As all the vehicles will have the same speed, we can calculate the position of any vehicle at any particular time. This includes when the vehicle is travelling between two nodes.

To simulate a dynamic environment with fruitful regions we introduce a particular arrangements of the customers by clusters. First a set of points called the set of cluster centres C is created by randomly selecting points (x, y) in the 2-dimensional space such that these points are uniformly distributed in that space. Then for each cluster centre $(x, y) \in C$ a set of locations $R_{(x,y)}$ is created such that these locations are scattered around the cluster centre by using a Gaussian random distribution[1] with an average distance of τ to choose the diversion from the centre. This way we get clusters with a circular shape. The set of nodes N is defined as $N = \{n | n \in R_{(x,y)} \wedge (x, y) \in C\}$. The set of locations form the nodes of the graph $G = (N, E)$. This graph is a full graph and its edges E are labelled with the costs to traverse them. For each $(n_1, n_2) \in E$, this cost is equal to the Euclidean distance between n_1 and n_2.

Every cluster is assigned a potential, which is a number randomly drawn from a Gaussian distribution using the average number of loads per cluster and the deviation as provided in Table 1. The potential of a node is determined by randomly assigning it with a number of loads. This number is taken from a Gaussian distribution with as average the potential of the cluster that node belongs to, and as the deviation it uses the deviation of loads within a cluster parameter as set in Table 1.

A set of loads is randomly generated, which will represent the work that needs to be routed in the time period T. The set is constructed such that the number of loads originating from a node corresponds to the potential of that node. The target of all loads is the same; it is a central depot, which is located in the centre of the map (height$/2$, width$/2$). Every load is of size $d = 1$ and has the same delivery time constraint Δ, which will be varied throughout the experiments. On average, 333 loads were generated for a problem instance, with a standard deviation of 107. The problem of pickup at several customers and delivery to a central depot exists in the real world where several customers are

[1] Gaussian distributions were created using the common Box-Muller Transformation [35] from a uniform distribution of fractional numbers in $[0, 1)$ with a precision of 16 to prevent sampling numbers that are too large.

Table 1. Parameters of the problem instances, with experiment parameters in bold

parameter	*value*
maximum dimension of the map	200×200
number of locations	$\lvert N\rvert = 50$
number of clusters	$\lvert C\rvert = 5$
spread of locations in a cluster	$\tau = 10$
number of vehicles	$\lvert V\rvert = 10$
capacity constraint	$q \in \{1, 2, 3, 4, 5, 6\}$
delivery time constraint	$\varDelta_l \in \{20, 40, \ldots, 400\}$
average number of loads/cluster	5
deviation of loads per cluster	5
deviation of loads/node in a cluster	4
average time spread	$\lambda = 10$ $(\sigma = 50)$

producing goods that require processing at one location. For example, the industrial partner in our consortium collects and vegetables at farms to be delivered to the port of Rotterdam for shipment to Great Britain.

The entry times of loads is chosen by considering all loads one at a time. The i-th load is generated at time $i \times \lambda + N(0, \sigma)$, where λ is the average time spread and σ is the deviation of the average time between entry times (see Table 1). This way we spread the loads over time using a normal distribution.

The number of vehicles is shown in Table 1. Each vehicle has the same capacity. This capacity will be varied in the experiments from $q = 1$ to $q = 10$ and a setting of $q = 20$ will be used.

The total length of the simulation is set to the delivery time of the load with the latest delivery time: $T = \max_{\langle s,d,t,\varDelta\rangle \in L}(t + \varDelta)$.

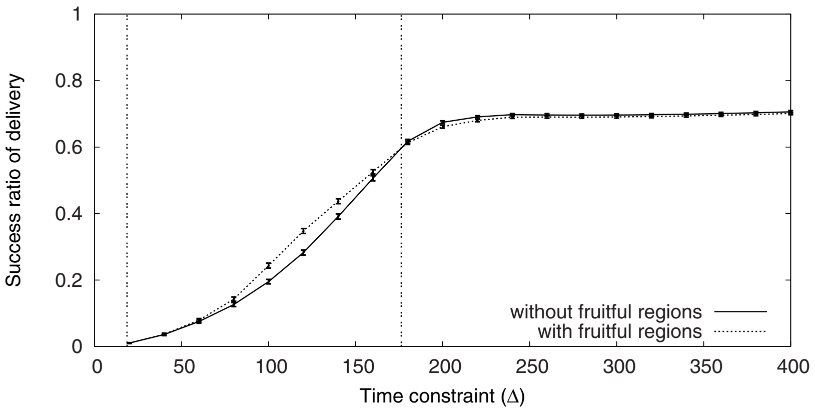
The problem generator used for generating the problem instances for the experiments reported next can be obtained from `http://www.vanhemert.co.uk/`. It is written in Perl.

6.2 Experimental Setup

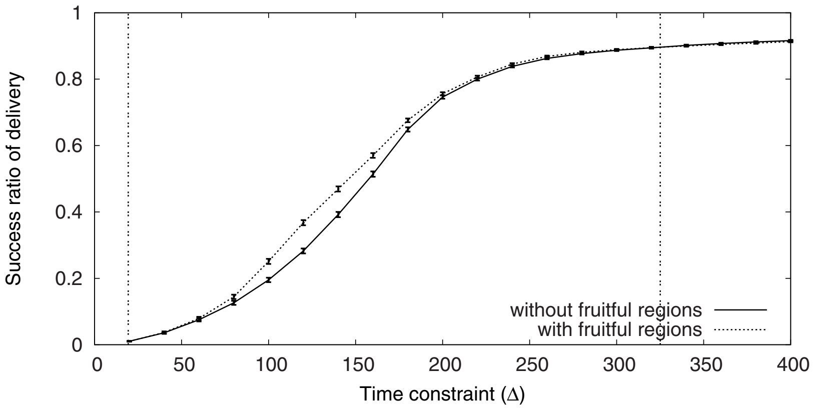
We test different settings for the capacity and time delivery constraints. The capacity constraint is varied between 1 and 6 and the time delivery constraint is varied from 20 to 400 with steps of 20. For each setting of the capacity and time constraint we randomly generate 80 independent problem instances. Then we perform 30 independent runs of both evolutionary algorithm variants, i.e., with and without anticipated routing, on each of these problem instances.

6.3 Computational Results

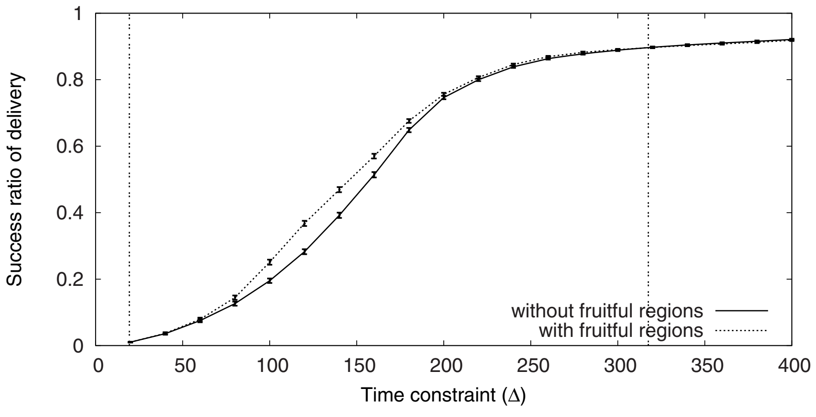
To show the effect of performing anticipated moves to fruitful regions we fix the capacity q of each vehicle to one and vary the time delivery constraint $\varDelta$. By increasing the time available for performing a delivery, we are loosening this constraint, which makes



(a) $q = 1$



(b) $q = 5$



(c) $q = 10$

Fig. 3. Average ratio of successfully delivered loads with increasing delivery time for both with and without exploring fruitful regions. The capacity of vehicles is set to q

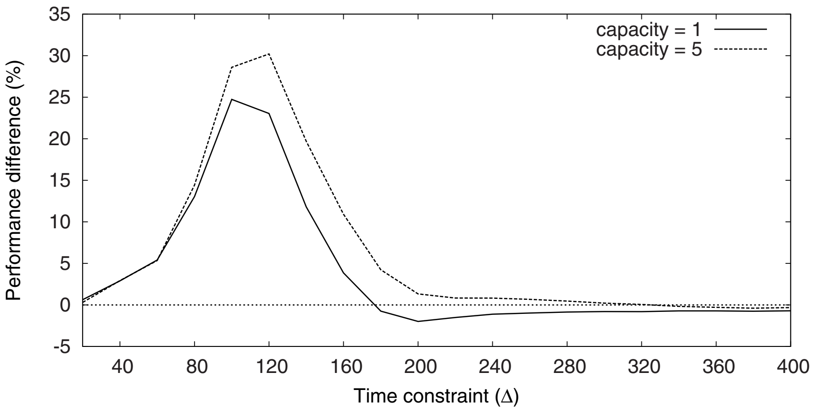


Fig. 4. Performance ratios, where the success ratio from the experiment with anticipated moves is divided by the success ratio from the experiment without anticipated moves

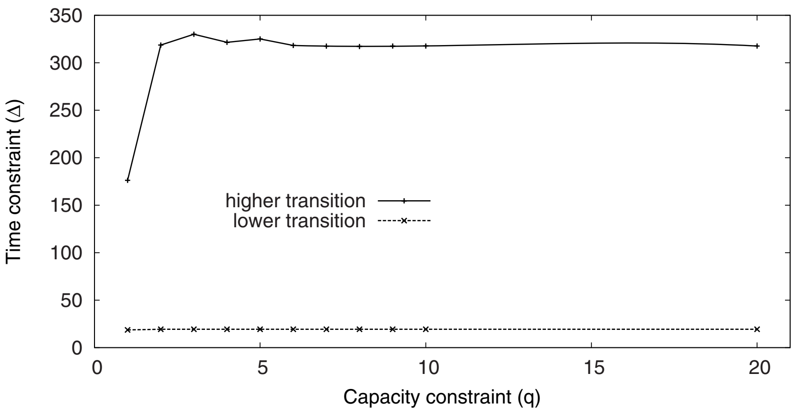


Fig. 5. Relating the capacity and time constraints using transition points to determine the boundaries wherein anticipated moves to fruitful regions may improve effectiveness

it possible that more loads are transported. Figure 3 shows the results for both with and without performing anticipated moves. Results for each setting of the time delivery constraint are averaged over 80 problem instances with 30 independent runs of the evolutionary algorithm and include 95% confidence intervals. For each setting of q, we clearly notice three stages, shown separated by two vertical lines. The first stage, where delivery time is highly restricted, and consequently only few loads can be transported. There, anticipated moves will not influence the success ratio. Then, in the second stage, the exploration of fruitful regions shows its potential for increasing the success ratio. Last, in the third stage, when the time restriction is removed further, there is plenty of time to deliver loads, thus the best result is achieved by sticking to routing vehicles for pickup and deliveries only. For this last stage, a vehicle can take plenty of time when a new load appears and then fetch it, as it will have more than enough time to return to the depot.

The transition from the first stage to the second stage we call the *lower transition point* and the transition from the second to the third stage we call the *higher transition point*. The terms lower and higher correspond to the amount of time available for delivery, where lower refers to small delivery times. The settings of the capacity influences where the higher transition points occur. For $q = 1$ shown in Figure 3(a), this transition happens at 176.21, whereas for $q = 5$, shown in Figure 3(b) it happens at 325.05. An even higher setting of $q = 10$, shown in Figure 3(c) results in bringer the transition forward again at 200.00, thus shortening the window where exploring fruitful regions is useful.

In Figure 4, we show by how much anticipated moves improves the effectiveness. The largest benefit is found at a time constraint setting of 100 and 120 for $q = 1$ and $q = 5$ respectively, where the success ratio is improved by 25% and 30%. For tight time constraints, no significant difference can be noted and for loose time constraints, the success ratio is at most 1.8% and 0.8% lower, for the settings of 1 and 5 respectively. For $q = 1$, we notice a dip around the higher transition point at $\Delta = 196$. A capacity of one means that vehicles are driving only back and forth between a customer and the depot. It seems that, at this critical setting of the time constraint, the self-adaptive mechanism is unsuccessful in finding a good balance between routing loads and exploration of fruitful regions. From the results, we infer that knowledge of the constraint settings is important, but as long as the tightness of the time constraint is not too restrictive, an exploration of fruitful regions is recommendable.

The variant of the evolutionary algorithm that uses self-adaptation of α to determine which amount of exploration of fruitful regions is beneficial, performs equally well or significantly better than the variant that does not make use of anticipated moves. This shows that this mechanism is a viable way of controlling the amount of

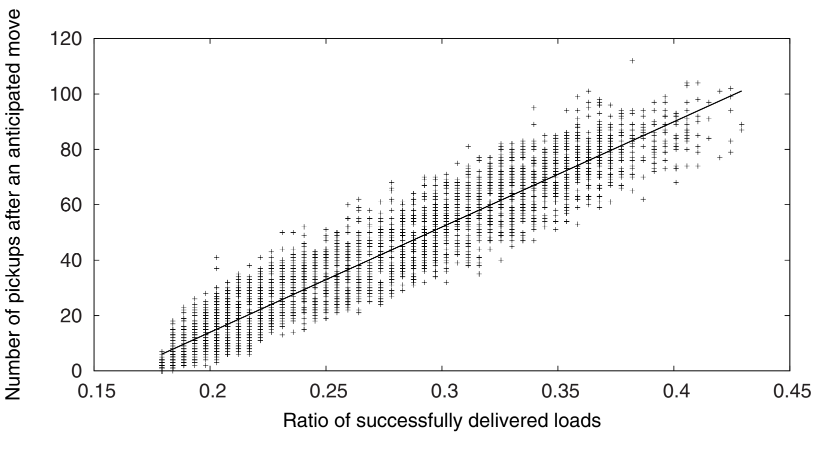


Fig. 6. Plotting the success ratio against the number of pickups immediately after an anticipated move in 3000 independent runs, and a linear regression of the same where the slope is 0.949 with a standard error of 0.0204. The rms of the residuals is 8.184

anticipated moves. Note that the evolutionary algorithm has no a priori information about the settings of the constraints and has no knowledge about how to react to certain settings. That is, the self-adaptive mechanism functions completely by learning from the dynamic environment.

To take into consideration the effect of both constraints, capacity and time, we show how they relate for the lower and higher transition points. For $q = 1$, $q = 5$, and $q = 10$ these transition points are shown as vertical lines in Figure 3. For different settings of the capacity constraint (1–10,20), we determine these transition points, which are shown in Figure 5. When the settings of the constraints lie in between the lower transition points and the higher transition points the usage of exploration of fruitful regions by performing anticipated moves is beneficial for the effectiveness of load deliveries.

We provide evidence to support the claim that adding anticipated moves to exploit fruitful regions provides a means for increasing the success ratio of delivery. We measure the number of pickups that happen immediately after an anticipated move in each run. Figure 6 shows this measure plotted against the success ratio in the delivery discussed before.

7 Conclusions

Dynamic constrained optimisation problems play an increasingly import role in many areas, not only because of faster computers and telecommunication, but also because of the increasing demand of customers for flexible and fast service. Vehicle routing is a scientifically interesting problem as its static version is known to be intractable. By moving into the realm of dynamic routing problems, we make this problem even more challenging.

In this chapter, we introduced a dynamic routing model that defines a load collection problem, i.e., the pickup and return of loads to one depot. Specifically, we introduced a model for taking into account regions with high potentials for the origin of new loads. Furthermore, we have provided an evolutionary algorithm that is able to provide solutions in real-time. The evolutionary algorithm is extended to let it perform anticipatory routing by sending vehicles to customers that have not yet requested service.

We have performed an empirical study into the benefits of exploring fruitful regions, i.e., groups of customers geographically near to each other that have a high potential for service requests. Through the use of the added self-adaptive mechanism, the evolutionary algorithm, is able to successfully change the amount of exploration of the fruitful regions, thereby making it employable in situations where the effect of constraint settings is not known in advance.

By observing the quality of the routing in terms of the ratio of successfully transported loads we have determined the transition points in terms of constraint settings, where between these this exploration of fruitful regions is of benefit. The potential of using the concept of fruitful regions appeared from the significant increase in the effectiveness for settings within these transition points.

The vehicle routing problem commonly is studied by either maximising the number of successfully handled requests or by minimising the total length of the routes. Here we have used the first as an object, and for future research we plan to address the latter. Moreover, we propose to tackle both objectives simultaneously as evolutionary algorithms are known to perform well on multi-objective optimisation tasks [36, 37].

We propose to study whether anticipated moves can be beneficial in other dynamic optimisation domains. For instance, in the area of elevator control, where most approaches consist of a deterministic algorithm, with only a few approaches tested that allow stochastic behaviour, most notably an evolutionary algorithm [38] and using fuzzy logic [39].

References

1. Lenstra, J.K., Rinnooy Kan, A.H.G.: Complexity of vehicle routing and scheduling problems. Networks 11, 221–227 (1981)
2. Fisher, M.L.: Vehicle routing. In: Ball, M., Magnanti, T., Monma, C., Nemhauser, G. (eds.) Handbook in Operations Research and Management Science: Network Routing, pp. 1–33. North-Holland, Amsterdam (1995)
3. Toth, P., Vigo, D.: The Vehicle Routing Problem. SIAM, Philadelphia (2002)
4. van Hemert, J.I., La Poutré, J.A.: Dynamic routing problems with fruitful regions: Models and evolutionary computation. In: Yao, X., Burke, E.K., Lozano, J.A., Smith, J., Merelo-Guervós, J.J., Bullinaria, J.A., Rowe, J.E., Tiňo, P., Kabán, A., Schwefel, H.-P. (eds.) PPSN 2004. LNCS, vol. 3242, pp. 690–699. Springer, Heidelberg (2004)
5. Laporte, G., Louveaux, F.: Formulations and bounds for the stochastic capacitated vehicle routing problem with uncertain supplies, pp. 443–455. Elsevier Science Publishers B.V, Amsterdam (1990)
6. Psaraftis, H.N.: Dynamic vehicle routing: status and prospects. Annals of Operations Research 61, 143–164 (1995)
7. Lipmann, M., Lu, X., de Paepe, W., Sitters, R., Stougie, L.: On-line dial-a-ride problems under a restricted information model. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 674–685. Springer, Heidelberg (2002)
8. Feuerstein, E., Stougie, L.: On-line single-server dial-a-ride problems. Theoretical Computer Science 268(1), 91–105 (2001)
9. Ascheuer, N., Krumke, S.O., Rambau, J.: Online dial-a-ride problems: Minimizing the completion time. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 639–650. Springer, Heidelberg (2000)
10. Godfrey, G., Powell, W.B.: Adaptive estimation of daily demands with complex calendar effects. Transportation Science 34(6), 451–469 (2000)
11. Bäck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, Oxford (1996)
12. Ghiani, G., Guerriero, F., Laporte, G., Musmanno, R.: Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies. Technical report, Center of Excellence for High Performance Computing, University of Calabria, Italy (2003)
13. de Paepe, W.E.: Complexity Results and Competitive Analysis for Vehicle Routing Problems. PhD thesis, Research School for Operations Management and Logistics, Technical University of Eindhoven (2002)
14. Laporte, G., Louveaux, F.V., Mercure, H.: The vehicle routing problem with stochastic travel times. Transportation Science 26(3), 161–170 (1992)
15. Bianchi, L.: Notes on dynamic vehicle routing — the state of the art. Technical report, IDSIA, Galleria 2, 6928 Manno-Lugano, Switzerland (2000)
16. Golden, B.L., Assad, A.A.: Vehicle Routing: methods and studies. Elsevier Science Publishers B.V, Amsterdam (1988)

17. Psaraftis, H.N.: Dynamic vehicle routing problems. In: Vehicle Routing: methods and studies[16], ch. 11, pp. 223–248
18. Powell, W.B.: A comparative review of alternative algorithms for the dynamic vehicle routing problem. In: Vehicle Routing: methods and studies [16], ch. 12, pp. 249–291
19. Powell, W.B.: A stochastic formulation of the dynamic assignment problem, with an application to truckload motor carriers. Transportation Science 30(3), 195–219 (1996)
20. Thomas, B.W.: Anticipatory Route Selection Problems. PhD thesis, University of Michigan (2002)
21. Thomas, B.W., White III, C.C.: Anticipatory route selection problems. Transportation Science 38(4), 473–487 (2005)
22. Thomas, B.W., White III., C.C.: The dynamic next term shortest path problem with anticipation. European Journal of Operational Research (in press, 2005)
23. Mitrović-Minić, S., Laporte, G.: Waiting strategies for the dynamic pickup and delivery problem with time windows. Transportation Research Part B 38, 635–655 (2004)
24. Branke, J., Middendorf, M., Noeth, G., Dessouky, M.: Waiting strategies for dynamic vehicle routing. Transportation Science 39(3), 298–312 (2005)
25. Ichoua, S., Gendreau, M., Potvin, J.-Y.: Exploiting knowledge about future demans for real-time vehicle dispatching. Transportation Science (in press)
26. Ichoua, S., Gendreau, M., Potvin, J.-Y.: Diversion issues in real-time vehicle dispatching. Transportation Science 34(4), 426–438 (2000)
27. Solomon, M.M.: The vehicle routing and scheduling problems with time window constraints. Operations Research 35(2), 254–265 (1987)
28. Fogel, D.B. (ed.): Evolutionary Computation: the Fossile Record. IEEE Press, Los Alamitos (1998)
29. Bäck, T., Fogel, D., Michalewicz, Z. (eds.): Handbook of Evolutionary Computation. Institute of Physics Publishing Ltd, Bristol and Oxford University Press (1997)
30. Branke, J.: Evolutionary Optimization in Dynamic Environments. In: Genetic Algorithms and Evolutionary Computation, vol. 3. Kluwer Academic Publishers, Dordrecht (2001c)
31. Bräysy, O., Gendreau, M.: Vehicle routing problem with time windows, part II: Metaheuristics. Transportation Science 39(1), 119–139 (2005)
32. Jih, W.-r., Hsu, J.Y.-j.: Dynamic vehicle routing using hybrid genetic algorithms. In: Proceedings of the 1999 IEEE International Conference on Robotics & Automation, pp. 453–458. IEEE Press, Los Alamitos (1999)
33. Reeves, C.R.: Landscapes, operators and heuristic search. Annals of Operations Research 86, 473–490 (1999)
34. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments—a survey. IEEE Transactions on Evolutionary Computation 9(3), 303–317 (2005)
35. Box, G.E.P., Muller, M.E.: A note on the generation of random normal deviates. Ann. Math. Stat. 29, 610–611 (1958)
36. Coello Coello, C.A.: An updated survey of evolutionary multiobjective optimization techniques: State of the art and future trends. In: 1999 Congress on Evolutionary Computation, pp. 3–13. Morgan Kaufmann, San Francisco (1999)
37. Coello Coello, C.A.: Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. Computer Methods in Applied Mechanics and Engineering 191(11–12), 1245–1285 (2002)
38. Alander, J.T., Tyni, T., Ylinen, J.: Optimizing elevator group control parameters using distributed genetic algorithms. In: Proceedings of the Second Finnish Workshop on Genetic Algorithms and their Applications, Finland, University of Vaasa, pp. 105–113 (1994)
39. Tan, G.V., Hu, X.: On designing fuzzy controllers using genetic algorithms. In: Proceedings of the Fifth IEEE International Conference on Fuzzy Systems, vol. 2, pp. 905–911 (1996)

EVITA: An Integral Evolutionary Methodology for the Inventory and Transportation Problem

Anna I. Esparcia-Alcázar[1], Manuel Cardós[2], J.J. Merelo[3], Anaís Martínez-García[1], Pablo García-Sánchez[3], Eva Alfaro-Cid[1], and Ken Sharman[1]

[1] Complex Adaptive Systems Group, Instituto Tecnológico de Informática, Valencia, Spain
anna@iti.upv.es

[2] Dept. de Organización de Empresas, Universidad Politécnica de Valencia, Spain
mcardos@doe.upv.es

[3] Dept. de Arquitectura y Tecnología de Computadores, Universidad de Granada, Spain
jmerelo@geneura.ugr.es

Summary. The Inventory and Transportation Problem (ITP) can be seen as a generalisation of the Periodic Vehicle Routing Problem that takes into consideration the inventory costs, plus a set of delivery frequencies instead of a single delivery frequency for each shop. Additionally, the ITP can also be viewed as a generalisation of the Inventory Routing Problem to the multiproduct case. EVITA, standing for **Ev**olutionary **I**nventory and **T**ransportation **A**lgorithm, is a two-level methodology designed to address this problem. The top level uses an evolutionary algorithm to obtain delivery patterns for each shop on a weekly basis so as to minimise the inventory costs, while the bottom level solves the Vehicle Routing Problem (VRP) for every day in order to obtain the transport costs associated to a particular set of patterns.

Here we compare four different algorithms that have been employed in the literature for solving the VRP and show how the choice of the lower level algorithm (the *VRP solver*) can play a significant part in the performance of the whole algorithm.

1 Introduction

The Inventory and Transportation Problem (ITP) arises when the objective is to minimise both the transport and inventory costs of a retail chain supplied from a central depot owned by the same company and subject to the operational constraints taking place at the shop level; for instance, just a few *delivery frequencies* are allowed for each shop[1].

The main feature that differentiates this problem from other similar supply chain management ones addressed in the literature (e.g. [10, 17, 21, 28, 33]) is that we have to decide on the frequency of delivery to each shop, which determines the size of the deliveries. The inventory costs can then be calculated accordingly, assuming a commonly-employed periodic review stock policy for the retail chain shops. Besides, for a given delivery frequency, expressed in terms of number of days a week, there can also be

[1] See [9] for a detailed explanation of this problem.

a number of *delivery patterns*, i.e. the specific days of the week in which the shop is served. Once these are established, the transportation costs can be calculated by solving the vehicle routing problem (VRP) for each day of the week.

Because a pattern assumes a given frequency, the problem is limited to obtaining the optimal patterns (one per shop) and set of routes (one per day). The optimum is defined as that combination of patterns and routes that minimises the total cost, which is calculated as the sum of the individual inventory costs per shop (inventory cost) plus the sum of the transportation costs for all days of the week (transport cost). These two objectives are in general contradictory: the higher the frequency of delivery the lower the inventory cost, but conversely, a higher frequency involves higher transportation costs.

The operational constraints at the shop level are imposed by the business logic and can be listed as follows:

1. A periodic review stock policy is applied for the shop items. So, a target cycle service level has been established for every item category, which is usually lower for the slow-moving ones. As a consequence, stockout is allowed.
2. Every shop has a limited stock capacity.
3. The retail chain tries to fulfil backorders in a few days, so there is a lower bound for delivery frequency depending on the target client service level.
4. Most of the expected stock reduction between replenishments is due to fast moving items. This reduction cannot be too high in order to avoid two problems: (a) the unappealing empty-shelves aspect of the shop just before the replenishment; and (b) replenishment orders too large to be placed on the shelves by the shop personnel in a short time compatible with their primary selling activity.
5. Conversely, the expected stock reduction of fast moving items must be high enough to perform an efficient allocation of the replenishment order.
6. Not all frequencies are admissible for all shops. For instance, frequency 1 (one delivery per week) is not applicable to most shops, for various reasons. On the one hand, many shops do not have the storage capacity required for the big deliveries imposed by such a low delivery frequency. On the other hand, big deliveries imply that the shop staff must devote a long time to the management of the stock, a time that would be more profitably employed in selling.
7. Sales are not uniformly distributed over the time horizon (week), tending to increase over the weekend. Hence, in order to match deliveries to sales, only a given number of delivery patterns are allowed for every feasible frequency. For instance, a frequency-2 pattern such as (Mon, Fri) is admissible, while another of the same frequency such as (Mon, Tues) is not.

For simplicity reasons we will consider that the shops have no time windows, i.e. the deliveries can take place at any time. However, the hours a driver can work are limited by regulations and this has to be taken into account in the time needed per delivery.

Additionally, the transport is subcontracted, which has a number of implications; the fleet is homogeneous, i.e. only one type of vehicle is used; the number of vehicles is unlimited and finally, the transport cost only includes the cost per kilometer; there is no cost attached to either the time of use of the vehicle or to the number of units delivered, nor there is a fixed cost per vehicle.

Finally, no single shop can consume more than the capacity of any one vehicle, i.e. a shop is served by one and only one vehicle; also, the load is containerised, so the number of units is an integer.

Problem data is freely available from our group website:
http://cas.iti.upv.es/bio_downloads.htm/.[2]

1.1 State of the Art

Inventory and transportation management have both received lots of attention in the logistics literature; however, this is not the case for the joint problem. For example, Constable and Whybark [13] consider a single product controlled by an order-point system and a per-unit transportation cost. Burns et al. [7] develop an approximate analytical method for minimising inventory and transportation costs under known demand, based on an estimation of the travel distance. Benjamin [5] analyses production, inventory and transportation costs in a capacitated network, yet once again transportation costs are assumed to be proportional to the units moved. Speranza and Ukovich [29] focus on a multiproduct production system on a single link. Ganeshan [22] presents a (s, Q)-type inventory policy for a network with multiple suppliers by replenishing a central depot, which in turn distributes to a large number of retailers; the author considers transportation costs, but only as a function of the shipment size. Qu et al. [27] deal with an inbound material-collection problem so that decisions for both inventory and transportation are made simultaneously; however, vehicle capacity is assumed to be unlimited so that it is solved as a traveling salesman problem (TSP). Finally, Zhao et al. [34] present a modified economic ordering quantity for a single supplier-retailer system in which production, inventory and transportation costs are all considered.

The inventory routing problem (IRP) is the superclass of models that includes the VRP. This problem arises when the vendor delivers a single product and implements a Vendor Inventory Management (VIM) [10] policy with its clients, so that the vendor decides the delivery (time and quantity) in order to prevent the clients from running out of stock while minimising transportation and inventory holding costs. This is the case of Campbell and Savelsbergh [8]. Unfortunately, retail chains do not fall into this category, as thousands of items are involved.

Also related to the VRP is the periodic vehicle routing problem (PVRP), which appears when customers have established a predetermined delivery frequency and a combination of admissible delivery days within the planning horizon. The objective is to minimise the total duration of the routes, while the restrictions usually involve a limited capacity of the delivery vehicles and a maximum duration of each itinerary. See for instance, [14] and [32] for a general description of the PVR problem.

The Inventory and Transportation Problem (ITP) [9] can be viewed as a generalisation of the IRP to the multiproduct case. Additionally, it can also be viewed as a generalisation

[2] Its use is subject to the condition that this or other papers on the same subject by the authors are mentioned.

of the PVRP, as it includes inventory costs[3] and a set of delivery frequencies instead of a unique delivery frequency for each shop.

1.2 EVITA

In view of the descriptions above, EVITA's task consists of finding:

- The *optimal vector of patterns*, $\mathcal{P}_{opt}$, with which all shops can be served. A pattern p represents a set of days in which a shop is served which implies a delivery frequency (expressed as number of days a week) for the shop
- The *optimal routes* for each day of the working week, by solving the VRP for the shops allocated to that day by the corresponding pattern

Our previous work [19, 18] showed the benefits obtained from addressing inventory and transportation management simultaneously, rather than optimising them separately, and how the choice of VRP algorithm could affect not only the transportation costs but also the inventory costs obtained. In [20] statistical analysis was used to choose from a set of simple VRP algorithms the one that provided best results (both in terms of cost and time); the chosen algorithm was then employed within the EVITA tool. However, the selection was done using a set of classic VRP problems extracted from the literature [1, 6], independently from the top level, and hence it did not take into account the interaction between the two levels.

Here we will adopt an *integral* approach, by means of a more thorough analysis of the interaction between the two levels. The issue is that since the choice of VRP algorithm affects the results of the top level inventory-minimisation algorithm, one cannot be studied in the absence of the other. Hence, the VRP algorithm will be chosen according to its performance within the methodology, and not on a set of independent experiments. The VRP algorithms studied will include tabu search [14], evolutionary computation [31], ant colony optimisation [15] and a classical VRP technique, Clarke and Wright's algorithm [11]. We will carry out an extensive set of experiments to test the methodology on a number of different problem instances. Each instance will consist of:

- A geographic layout. This will be taken from Augerat *et al.* [1, 6].
- An inventory cost table (see Table 6). The table, containing data for up to 80 shops, will be the same across all different geographical layouts used.

To these we will add a set of constraints common for each problem: the characteristics of the vehicles used and the working hours of the drivers. These are given in Table 7.

The top level evolutionary algorithm will employ a configuration of parameters that proved to be successful in previous work [20].

The best solutions obtained by each run will be analysed on two fronts: in terms of the fitness (global cost) achieved and in terms of the computational effort spent in achieving it. The results will be analysed statistically in order to determine if one of the chosen VRP algorithm performs significantly better than the others and in a commercially feasible timescale.

[3] The PVRP can be seen as an ITP in which the inventory costs are zero.

2 The Top Level: Evolutionary Algorithm

Let $\mathcal{P}$ be the set of *all possible delivery patterns*. We will represent a pattern p as an integer whose binary value represents the days the shop is served (1) or not (0). Hence, $\mathcal{P}$ will be equal to[4]:

$$\mathcal{P} = [1,\ 2^d - 1]$$

where d is the number of working days in the week. In our case, $d = 5$, so

$$\mathcal{P} = [1,\ 31]$$

So, for instance, the pattern $p = 31$ or, in binary $p = 11111$, represents that a shop is served on all five working days of the week.

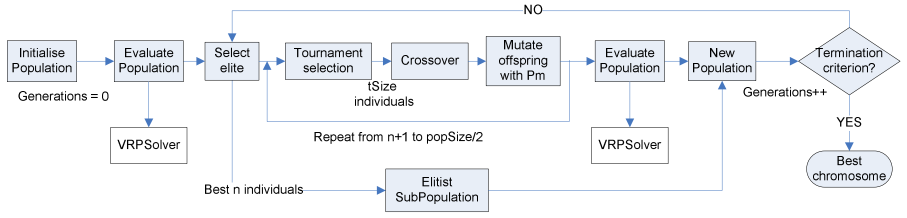


Fig. 1. Flow chart of the evolutionary algorithm employed at the top level

Table 1. Configuration of the top-level evolutionary algorithm employed

Encoding	The gene i represents the pattern for shop i. The chromosome length is equal to the number of shops ($nShops$).
Selection	Tournament in 2 steps. To select each parent, we take $tSize$ individuals chosen randomly and select the best. The best 10 individuals of each generation are preserved as the elite.
Evolutionary operators	2 point crossover and 1-point mutation. The mutation operator changes the pattern for 1 shop in the chromosome.
Termination criterion	Terminate when the total number of generations (including the initial one) equals 100.
Fixed parameters	Population size, $popSize = 100$ Tournament size, $tSize = 2$ Mutation probability, $pM = 0.2$

We will not work with the full $\mathcal{P}$. Instead, let $\mathcal{P}_A$ be the set of *admissible patterns*,

$$\mathcal{P}_A \subset \mathcal{P}$$

The components of $\mathcal{P}_A$ are chosen subject to business criteria; the ones employed here are given in Table 5.

[4] The value $p = 0$ is excluded from the interval, since it would imply no deliveries throughout the whole week.

Algorithm 1. Evaluation function

```
Procedure Evaluate
input:
    Chromosome $\{p_1, \ldots, p_{nShops}\}$,
    problem data tables
    $costPerKm$
output: Fitness
    [Calculate inventory cost]
      InventoryCost = 0
      for $i = 1$ to $nShops$
        Look up frequency $f_i$ for pattern $p_i$
        Look up cost $c_i$ for shop $i$ and frequency $f_i$
        $InventoryCost += c_i$
    [Calculate transportation cost]
      $day = Monday$;
      $totalDistance = 0$
      repeat
        Identify shops to be served on $day$
        Run VRP algorithm to get $dayDistance$
        $totalDistance += dayDistance$
        $day++$
      until $day = lastWorkingDay$;
      $TransportCost = totalDistance * costPerKm$
    [Calculate fitness]
      $TotalCost = InventoryCost + TransportCost$
    return $TotalCost$
end procedure;
```

In EVITA an *individual* that undergoes evolution is a vector of length equal to the number of shops to serve ($nShops$) and whose components $p_i, i \in [1 \ldots nShops]$ are integers representing a particular delivery pattern,

$$p_i \in \mathcal{P}_A$$

The fitness of such an individual is calculated as the sum of the associated inventory and transportation costs,

$$f = InventoryCost + TransportCost \tag{1}$$

Inventory costs are computed from the patterns for each shop by taking into account the associated delivery frequency and looking up the inventory cost per shop in Table 6. Transport cost are obtained by solving the VRP with one of the algorithms under study.

The flow diagram for the top level evolutionary algorithm is depicted in Figure 1 and the details are given in Table 1. The pseudo-code for the evaluation function is given in Algorithm 1.

3 Lower Level: Solving the VRP

Four algorithms have been tested for solving the VRP, namely:

CWLS, the classical Clarke and Wright's algorithm [11], which is presented in subsection 3.1
ACO, a bioinspired ant colony optimisation algorithm [16], described in subsection 3.2
TS, tabu search [25], which is described in subsection 3.3, and
EC, another evolutionary algorithm [2], described in subsection 3.4

3.1 Clarke and Wright's Algorithm

Clarke and Wright's algorithm [11] is based on the concept of *saving*, which is the reduction in the traveled length achieved when combining two routes. We have employed the parallel version of the algorithm, which works with all routes simultaneously.

Due to the fact that the solutions generated by the C&W algorithm are not guaranteed to be locally optimal with respect to simple neighbourhood definitions, it is almost always profitable to apply a local search to attempt and improve each constructed solution. For this we designed a simple (and fast) local search method; this combination of C&W's algorithm with local search is what we have termed CWLS and the pseudo-code for it can be found in Algorithm 2.

3.2 ACO

Ant algorithms are derived from the observation of the self-organized behavior of real ants [16]. The main idea is that artificial agents can imitate this behavior and collaborate to solve computational problems by using different aspects of ants' behavior. One of the most successful examples of ant algorithms is known as "ant colony optimisation", or ACO, which is based on the use of pheromones, chemical products dropped by ants when they are moving.

Each artificial ant builds a solution by probabilistically choosing the next node to move to among those in the neighborhood of the graph node on which it is located. The choice is biased by pheromone trails previously deposited on the graph by other ants and some heuristic function.

Besides, each ant is given a limited form of memory to store the partial path it has followed so far, as well as the cost of the links it has traversed. This, together with deterministic backward moves, helps avoiding the formation of loops [16].

The pseudo-code for the ACO employed here is given in Algorithm 3. If an ant is at shop i the *transition function* (described in Algorithm 4) determines if shop j should be

Algorithm 2. Clarke & Wright's algorithm with local search (CWLS)

```
Algorithm CWLS
    [Initialisation]
        Build $nShops$ routes as follows
        $r_i = (0, i, 0)$
    [Calculate savings]
        Calculate $s_{i,j}$ for each pair of shops $i$, $j$
        $s_{i,j} = cost_{i0} + cost_{0j} - cost_{ij}$
    [Best union ]
        repeat
          $s_{i*j*} = \max s_{i,j}$
          Let $r_{i*}$ be the route containing $i$
          Let $r_{j*}$ be the route containing $j$
          if
            $i*$ is the last shop in $r_{i*}$
            and $j*$ is the first shop in $r_{j*}$
            and the combination is feasible
          then combine $r_{i*}$ y $r_{j*}$
          delete $s_{i*j*}$;
        until there are no more savings to consider;
    [Local search]
        Improve each route $r_i$ separately
        Improve considering exchanges between routes
    return routes
end algorithm;
```

selected as the next shop in the route. This is based on the probability of selecting shop j, given by

$$p_j = \frac{\varphi_{i,j}^{\alpha} + H(i,j)^{\beta}}{\sum\limits_{k \neq visited} (\varphi_{i,k}^{\alpha} + H(i,k)^{\beta})}$$

where $\varphi_{i,j}$ is the amount of pheromone in arc (i, j), $H(i, j)$ is the heuristic function for the same arc and α and β are their respective weights, which are determined empirically. The heuristic function is given by

$$H(i,j) = \omega \, \frac{time_{max}}{time(i,j)} + \theta \, \frac{demand(j)}{demand_{max}}$$

Algorithm 3. Ant Colony Optimisation (ACO)

```
Algorithm ACO(nShops:int, L:List(shops)):routes
    [Initialise pheromones]
    pheromones = initialValue
    for each iteration i:
        for each ant h:
            [Ant solutions]
            currentShop = Depot
            solution of ant[h] $\leftarrow$ add Depot
            repeat
                if not enough time or demand then nextShop = Depot
                else nextShop = Transition(currentShop)
                solution of ant[h] $\leftarrow$ add nextShop
                Update time and demand
                Update pheromone in arc of currentShop to nextShop
                currentShop = nextShop
            until there are no more shops to consider
            solution of ant[h] $\leftarrow$ add Depot
        Update pheromones in all arcs
        [Keep best solution of iteration]
        if best solution of iteration is better than global solution then
            global solution = best solution of iteration
    routes = global solution
return routes
end algorithm;
```

which takes into account the time for traversing arc (i, j), the demand of shop j, their respective weights ω and θ (also determined empirically) and the maximum values of time and demand. The latter are external constraints corresponding to the maximum working time and the capacity of the vehicle used (see Table 7).

Mimicking the *evaporation* process in nature, the pheromone is updated globally (for all arcs) as follows:

$$\varphi_{i,j} = (1-\varepsilon)\cdot\varphi_{i,j} + \varepsilon\cdot\varphi_0$$

with φ_0 being the initial value of the pheromone, which is the same for all arcs.

Further, $\varphi_{i,j}$ is also updated locally as follows

$$\varphi_{i,j} = (1-\rho)\cdot\varphi_{i,j} + \frac{\rho}{time(i,j)\cdot costPerKm}$$

Algorithm 4. Transition function for ACO

```
Algorithm Transition(shopsNotVisitedYet:List(shops),currentShop:shops):nextShop
    [Build a table of probabilities]
    for i=1:shopsNotVisitedYet.size
        Calculate probability of shopsNotVisitedYet[i]:
    Order probabilities in ascending order
    p = random number(0,1)
    nextShop = first shop of shopsNotVisitedYet whose probability is less than p
return nextShop
end algorithm;
```

Table 2. Parameters for the ACO algorithm

Number of ants	25
Iterations	200
Parameters for transition function	Weight of pheromone, $\alpha = 0.2$ Weight of heuristic, $\beta = 0.2$
Pheromone update	Initial value: $\varphi_0 = 0.5$ Local update: Weight of evaporation $\epsilon = 0.8$ Global update: Weight of memory, $\rho = 0.2$
Parameters for heuristic function	Weight of arc time: $\omega = 0.5$ Weight of shop demand: $\theta = 0.5$
Parameters for local search	Size of neighborhood: 6 times the number of routes of best solution in iteration

Some ant systems have been applied to the VRP (see for instance [12, 24]) with various degrees of success. We have applied it in this paper with the parameter values shown in Table 2.

3.3 Tabu Search

Tabu Search (TS) is a metaheuristic introduced by Glover in order to allow Local Search (LS) methods to overcome local optima [25]. The basic principle of TS is to pursue LS whenever it encounters a local optimum by allowing non-improving moves; cycling

Algorithm 5. Main Tabu Search (TS) algorithm

```
Algorithm TS
    [Initialisation]
        currentSolution ← initialSolution
        currentSolutionCost ← calculateCost(currentSolution)
    [Main loop]
        while (not finished)
            bestNeighbour ← getBestNeighbour (currentSolution, tabuList)
            bestNeighbourCost ← calculateCost(bestNeighbour)
            if (bestNeighbourCost < currentSolutionCost)
                currentSolution ← bestNeighbour
                currentSolutionCost ← bestNeighbourCost
            if (currentSolutionCost.isBetterThan(bestsolutionCost))
                bestSolution ← currentSolution
                bestSolutionCost ← currentSolutionCost
            tabuList ← updateTenure
    return bestSolution
end algorithm;
```

back to previously visited solutions is prevented by the use of memories, called *tabu lists* [23], which last for a period given by their *tabu tenure*. The main TS loop is given in Algorithm 5.

To obtain the best neighbour of the current solution we must move in its neighbourhood, avoiding movements into older solutions and returning the best of all new solutions. This solution may be worse than the current solution. For each solution we must generate all possible and valid neighbours whose generating moves are not tabu. If a new best neighbour is created, the movement is inserted into the Tabu List with the maximum tenure. This movement is kept in the list until its tenure is over. The tenure can be a fixed or variable number of iterations.

The way to obtain the best neighbour is described in algorithm 6.

In our case a move can be

- swapping a shop with another one belonging to the same route
- swapping a shop with another one belonging to a different route
- creating a new route with that shop only

The identifier of the move is the shop number.

The termination criterion is a number of 60 iterations and the tenure of the Tabu List is 12 iterations. It means that every shop that has been selected to move must wait 12 iterations until it can be selected again. The initial solution is a random valid solution.

Algorithm 6. Best neighbour algorithm

```
Algorithm bestNeighbour
    [Initialisation]
        moved ← false
        moves ← getAllMoves
        theBestNeighbour ← currentSolution
        theBestNeighbourCost ← $\infty$
        neighbourCost ← $\infty$
    [Main loop]
        for i=1:moves.length
            move ← $moves[i]$
            neighbour ← currentSolution
            neighbour ← move.operateOn(neighbour)
            neighbourCost ← calculateCost(neighbour)
            if (neighbourCost < theBestNeighbourCost AND isNotTabu(move))
                theBestNeighbour ← neighbour
                theBestNeighbourCost ← neighbourCost
                bestNeighbourMove ← move
                moved ← false
        end for
    [Update tabu list]
         if moved == true
            tabuList.addMove(bestNeighbourMove)
    return bestSolution
end algorithm;
```

3.4 Evolutionary Computation

The final VRP solver we tested was another evolutionary algorithm. EC and, in particular, genetic algorithms, has been widely used to solve the VRP. For instance, [3] reports obtaining results which are competitive with tabu search and simulated annealing. In [31] the authors introduce a new genetic vehicle representation for addressing the VRP, while [30] presents a multiobjective evolutionary algorithm for the VRP with time windows. In [26] the authors employ global convexity tests to find the types of solution features that are essential for solution quality; these are then used to construct an appropriate distance-preserving recombination operator.

For our problem, we employed a Genetic Algorithm similar to the one used in the top level. The detailed description is given in Table 3.

Table 3. Configuration of the evolutionary algorithm used as VRP solver

Encoding	The chromosome is a permutation of the number of shops visited on the day in question (which is a subset of the total number of shops), where gene i represents the id. of a shop. The routes are constructed with the shops visited in the same order as they appear in the permutation.
Selection	Tournament in 2 steps. To select each parent, we take $tSize$ individuals chosen randomly and select the best. The best 10 individuals of each generation are preserved as the elite.
Evolutionary operators	Partially mapped crossover (PMX), as described in [2]. Two-point swap mutation (swaps two positions in the chromosome).
Termination criterion	Terminate after 50 generations without change in the fitness value, or when the total number of generations (including the initial one) equals 200.
Fixed parameters	Population size, $popSize = 512$ Tournament size, $tSize = 2$ Mutation probability, $pM = 0.2$.

4 Experiments

This section is devoted to present the data we have used in the problem (subsection 4.1) and the experimental procedure we have followed (in the next subsection, 4.2).

4.1 Problem Data

As explained above, we will employ a number of *geographical layouts* taken from Augerat *et al.* [1, 6]. The data is divided in two sets, A and B. The problems from set A correspond to shops scattered more or less uniformly on the map; the problems from set B correspond to shops that are grouped in clusters, see [6]. We chose four problems from each set, with different levels of number of shops and eccentricity, see Table 4. The latter represents the distance between the depot and the geographical centre of the distribution of shops. The coordinates of the geographical centre are calculated as follows:

$$x_{gc}, y_{gc} = \sum_{i=1}^{nShops} x_i, y_i$$

For instance, an instance with low eccentricity (in practise, less than 25) would have the depot centered in the middle of the shops while in another with high eccentricity (above 40) most shops would be located on one side of the depot.

In the original problem the n indices represented the number of shops plus one and the k indices the maximum number of vehicles allowed. In our case we will ignore the latter data, as we are not considering restrictions in the number of vehicles.

Table 4. Problem instances used in the experiments and their characteristics, taken from the bibliography

ID	Instance	Distribution	$nShops$	Eccentricity
A32	A-n32-k5.vrp	uniform	31	47.4
A33	A-n33-k5.vrp	uniform	32	20.2
A69	A-n69-k9.vrp	uniform	68	15.3
A80	A-n80-k10.vrp	uniform	79	63.4
B35	B-n35-k5.vrp	clusters	34	60.5
B45	B-n45-k5.vrp	clusters	44	16.6
B67	B-n67-k10.vrp	clusters	66	19.9
B68	B-n68-k9.vrp	clusters	67	49.2

Table 5. The admissible patterns and their respective frequencies. The checkmark represents that the shop is served on that day, the minus that it is not. As a consequence of the business logic, we will only consider 11 patterns out of the 31 that are possible.

Pattern Id.	Freq	Mon	Tues	Wed	Thu	Fri
5	2	-	-	✓	-	✓
9	2	-	✓	-	-	✓
10	2	-	✓	-	✓	-
11	3	-	✓	-	✓	✓
13	3	-	✓	✓	-	✓
17	2	✓	-	-	-	✓
18	2	✓	-	-	✓	-
21	3	✓	-	✓	-	✓
23	4	✓	-	✓	✓	✓
29	4	✓	✓	✓	-	✓
31	5	✓	✓	✓	✓	✓

It must be noted that we are only using the spatial location and not other restrictions given in the bibliography, such as the number of vehicles (as mentioned above) or the shop demand values. As pointed out earlier, a main characteristic of our problem is that the latter is a function of the delivery frequency, so we had to use our own values for the demands.

We also added a list of *admissible patterns*, which are given in Table 5, and the *inventory costs*, given in Table 6. The table, which contains data for up to 80 shops, is the same across all different geographical instances used. The inventory, demand and admissible patterns data were obtained from Druni SA, a major regional Spanish drugstore chain.

Finally, we have used the vehicle data given in Table 7.

4.2 Experimental Procedure

Our aim is to find an optimal VRP solving algorithm (a *VRP solver*) that the user can successfully apply to a wide class of problems. Note that this does not necessarily correspond to the best performing algorithm for the VRP *in isolation*.

For this purpose, we tested the selected VRP algorithms on each one of eight different geographic layouts selected. We performed a number of runs per VRP algorithm

Table 6. Inventory cost (in Euro), size of the deliveries per shop (expressed in roll containers) depending on the delivery frequency. Missing data corresponds to frequencies that are not admissible for each shop.

Shop	Inventory cost (€)					Delivery size					Shop	Inventory cost (€)					Delivery size				
id.	1	2	3	4	5	1	2	3	4	5	id.	1	2	3	4	5	1	2	3	4	5
1	-	-	-	336	325	-	-	-	2	2	41	-	310	292	285	283	-	3	2	2	1
2	-	-	-	335	325	-	-	-	2	2	42	-	308	290	283	282	-	3	2	2	1
3	-	-	-	334	324	-	-	-	2	2	43	-	307	289	282	280	-	3	2	2	1
4	-	-	-	334	323	-	-	-	2	2	44	-	305	287	280	279	-	3	2	2	1
5	-	-	-	333	322	-	-	-	2	2	45	-	303	286	279	277	-	3	2	2	1
6	-	-	-	332	321	-	-	-	2	2	46	-	302	284	278	276	-	3	2	2	1
7	-	-	-	331	321	-	-	-	2	2	47	-	300	283	276	275	-	3	2	2	1
8	-	-	-	330	320	-	-	-	2	2	48	-	299	281	275	273	-	3	2	2	1
9	-	-	-	329	319	-	-	-	2	2	49	-	297	280	273	272	-	3	2	2	1
10	-	-	-	329	318	-	-	-	2	2	50	-	296	279	272	270	-	3	2	2	1
11	-	-	-	328	317	-	-	-	2	2	51	-	294	277	270	269	-	3	2	2	1
12	-	-	-	327	317	-	-	-	2	2	52	-	293	276	269	267	-	3	2	2	1
13	-	-	-	326	316	-	-	-	2	2	53	-	291	274	268	266	-	3	2	2	1
14	-	-	-	325	315	-	-	-	2	2	54	-	290	273	266	265	-	3	2	2	1
15	-	-	-	324	314	-	-	-	2	2	55	-	288	271	265	263	-	3	2	2	1
16	-	-	-	324	313	-	-	-	2	2	56	-	286	270	263	262	-	3	2	2	1
17	-	-	-	323	313	-	-	-	2	2	57	-	285	268	262	260	-	3	2	1	1
18	-	-	-	322	312	-	-	-	2	2	58	-	283	267	260	259	-	3	2	1	1
19	-	-	-	321	311	-	-	-	2	2	59	-	282	265	259	258	-	3	2	1	1
20	-	-	-	320	310	-	-	-	2	2	60	-	280	264	258	256	-	3	2	1	1
21	-	-	-	320	309	-	-	-	2	1	61	-	279	263	256	255	-	3	2	1	1
22	-	-	-	316	308	-	-	-	2	1	62	-	278	262	256	255	-	3	2	1	1
23	-	-	-	313	308	-	-	-	2	1	63	-	278	262	256	256	-	3	2	1	1
24	-	-	-	310	307	-	-	-	2	1	64	-	278	262	256	256	-	2	2	1	1
25	-	334	315	307	306	-	4	3	2	1	65	-	278	262	256	-	-	2	2	1	-
26	-	333	314	306	304	-	4	3	2	1	66	-	277	261	255	-	-	2	2	1	-
27	-	331	312	305	303	-	4	3	2	1	67	-	276	261	255	-	-	2	2	1	-
28	-	330	311	303	301	-	4	2	2	1	68	-	276	260	254	-	-	2	2	1	-
29	-	328	309	302	300	-	3	2	2	1	69	-	275	259	253	-	-	2	1	1	-
30	-	327	308	300	299	-	3	2	2	1	70	-	274	259	253	-	-	2	1	1	-
31	-	325	306	299	297	-	3	2	2	1	71	-	274	258	252	-	-	2	1	1	-
32	-	324	305	297	296	-	3	2	2	1	72	-	273	257	252	-	-	2	1	1	-
33	-	322	303	296	294	-	3	2	2	1	73	-	272	257	251	-	-	2	1	1	-
34	-	321	302	295	293	-	3	2	2	1	74	-	271	256	250	-	-	2	1	1	-
35	-	319	300	293	291	-	3	2	2	1	75	-	271	255	250	-	-	2	1	1	-
36	-	317	299	292	290	-	3	2	2	1	76	326	270	255	-	-	3	2	1	-	-
37	-	316	298	290	289	-	3	2	2	1	77	325	269	254	-	-	3	2	1	-	-
38	-	314	296	289	287	-	3	2	2	1	78	324	269	254	-	-	3	2	1	-	-
39	-	313	295	287	286	-	3	2	2	1	79	324	268	253	-	-	3	2	1	-	-
40	-	311	293	286	284	-	3	2	2	1	80	323	267	252	-	-	3	2	1	-	-

Table 7. Vehicle data

Capacity	12 roll containers
Transportation cost	0.6 €/Km
Average speed	60 km/h
Unloading time	15 min
Maximum working time	8h

and layout with a termination criterion in all cases of 100 generations. Running times ranged from several minutes to several hours in the computers employed[5].

The results will be evaluated on two fronts: quantitatively for the total costs obtained, and qualitatively for the computational time taken in the runs. The latter is important when considering a possible commercial application of the EVITA methodology.

5 Results and Analysis

We started by running a Mann-Whitney U-test with the total costs of all runs for problem A32. The conclusion was that all results are significantly different with a confidence level of 99%, i.e. all VRP solver algorithms perform significantly differently. Figure 2 shows the boxplots resulting from this comparison.

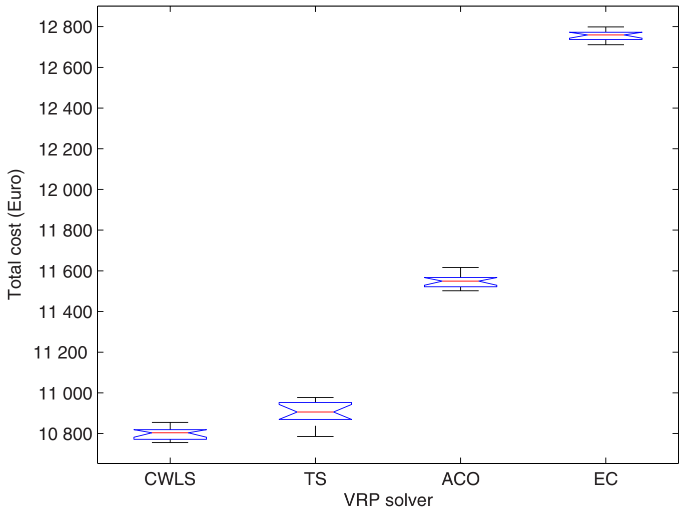


Fig. 2. Box plot for all VRP solvers for problem A32

The boxplot consists of a box and whisker plot for each algorithm. The box has lines at the lower quartile, median, and upper quartile values. The whiskers are lines extending from each end of the box to show the extent of the rest of the data. Outliers are data with values beyond one standard deviation.

In Figure 2 we can see that CWLS obtains the lower costs, followed closely by TS; ACO gives a clearly worse performance, followed by EC, which obtains the worst results of all solvers. Figure 3 shows a detail of the previous figure, with the comparison between only CWLS and TS. The differences between the two are clearly visible here.

In order to be able to compare results between the different problem layouts (or instances) we normalised the fitness values by defining the *relative percentage deviation*, RPD, given by the following expression:

[5] PCs with Intel Celeron processor, between 1 and 3GHz, between 256 and 512 MB RAM.

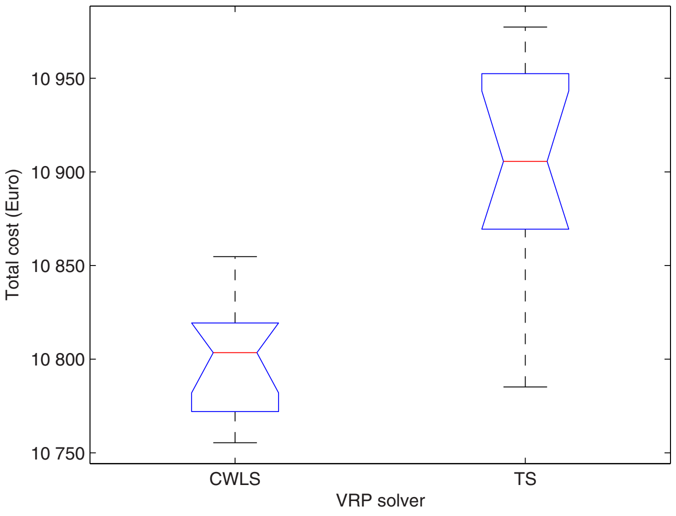


Fig. 3. CWLS vs TS for problem A32

$$RPD = \frac{fitness - fitness_{min}}{fitness_{min}} \times 100$$

where $fitness$ is the fitness value obtained by an algorithm configuration on a given instance. The RPD is, therefore, the average percentage increase over the lower bound for each instance, $fitness_{min}$. In our case, the lower bound is the best result obtained for that instance across all algorithm configurations.

With the RPD results of all the runs for all VRP solvers we ran the tests again; the results are shown in Figure 4. For the inventory costs the conclusions are that CWLS, TS and ACO are significantly different with a confidence interval of over 99.99%. Further, ACO and EC are significantly different with a confidence interval of over 95%. For the transport costs all VRP solvers are significantly different with a confidence interval of over 99.99%.

Again we see that CWLS outperforms all other algorithms, but Figure 4 also offers an interesting conclusion: the choice of VRP algorithm influences not only the performance regarding the total or transport costs, which would be expected, but also the inventory costs. This result warrants further investigation in the subject.

Regarding the computational time, the results clearly favour CWLS over all other VRP solvers. In general, when employing CWLS the time for a whole run took approximately the same as that of a single generation in when using TS, ACO or EC. This is another point in favour of CWLS when considering a potential commercial application.

As a point of interest, Figure 5 shows the evolution of the total, inventory and transportation costs along a run. Each point in the plot is the average value of 30 runs. We can see that although the transport and inventory costs fluctuate, this is not the case for the total cost. We must remember that minimising transport and inventory costs are contradicting objectives and hence the interest of addressing the ITP as a joint problem.

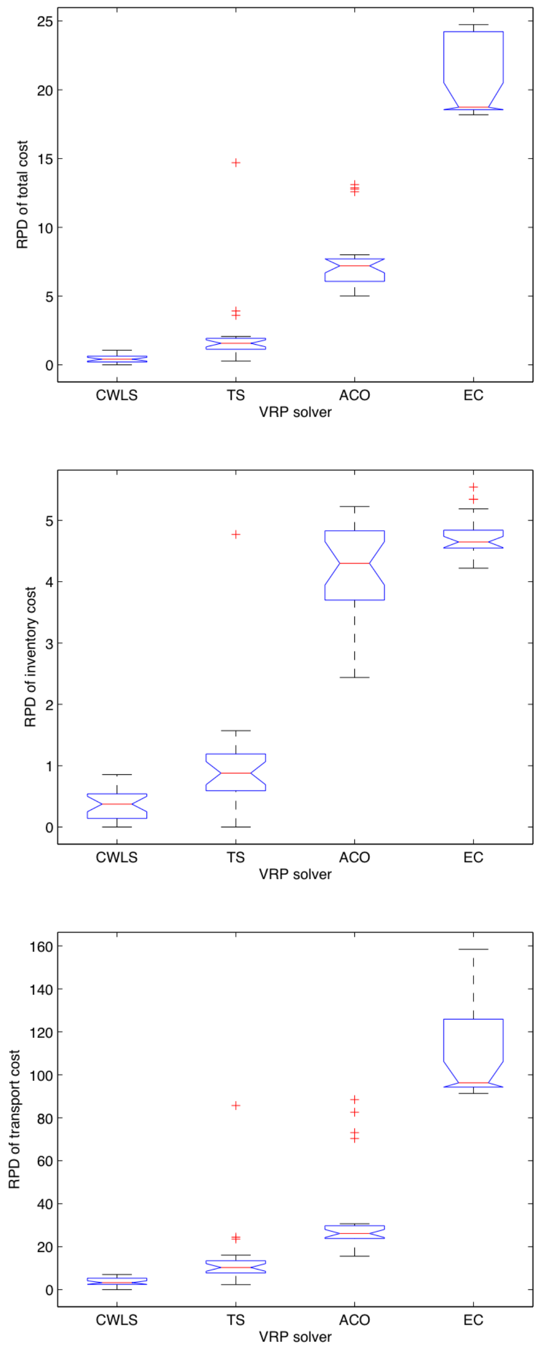


Fig. 4. Boxplots comparing the Relative Percentage Deviation of the total, inventory and transport cost for all VRP solvers

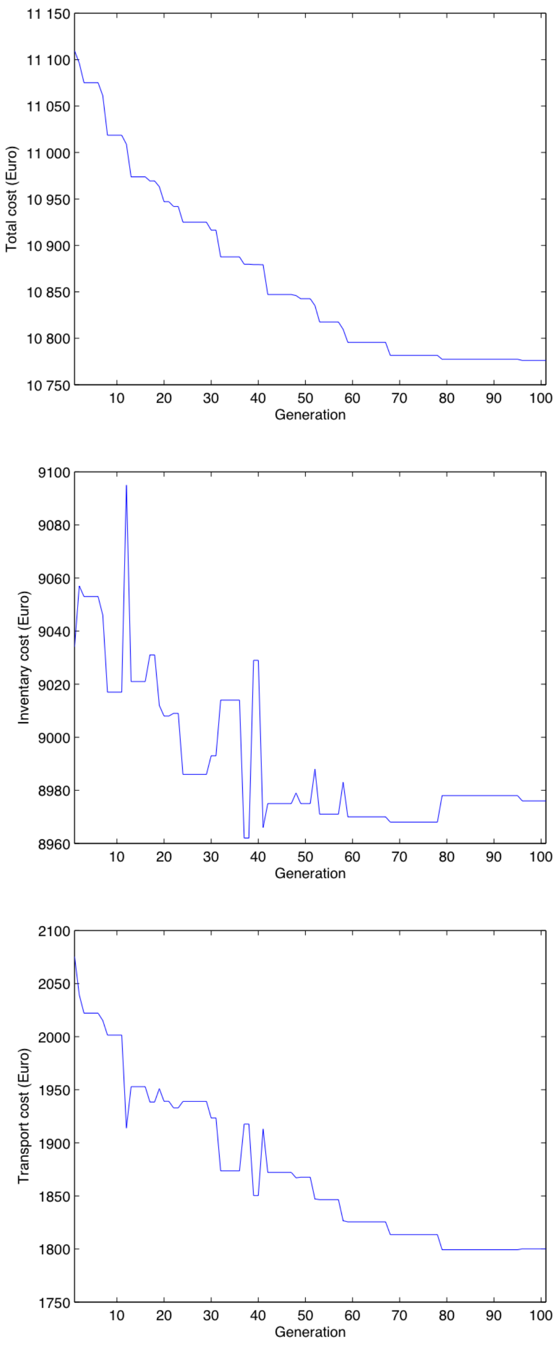


Fig. 5. Evolution of the total, inventory and transport cost when using CWLS as the VRP solver for problem A32

6 Conclusions

We have shown how an evolutionary version of a classical algorithm such as Clarke and Wright's, enhanced with local search, can be the best choice in the context of the Inventory and Transportation Problem, both in terms of the quality of the solutions obtained and the computational time necessary to achieve them. The power of other algorithms known to perform well in the context of VRP, such as ACO and TS, does not grant a good performance for the joint inventory and transportation problem. In general, using a global optimisation algorithm such as evolutionary computation jointly with a heuristic method adapted to the problem at hand, such as CWLS, yields the best results, so this is no surprise.

It could be argued that the three worst-performing algorithms: TS, AC and especially EC, require fine-tuning of their parameters in order to give an adequate performance. This, however, could be interpreted as another disadvantage of their application to a variety of problem configurations and in a commercial context.

It must be noted that this result is subject to the specificities of the problem data, i.e. the fact that in this case the inventory cost greatly outweighs the cost of the transport. As future work we must consider a case in which the products moved are cheaper and hence the inventory cost is more in a par with the transport cost. It would be also interesting to approach the ITP problem with multiobjective optimisation algorithms, so that there is no tradeoff between inventory and transportation costs and both inventory and transportation costs are minimised at the same time.

Acknowledgement

This work was part of projects NABILA (supported by the Instituto de la Pequeña y Mediana Industria Valenciana, IMPIVA, Ref. IMCITA/2007/58, and partially funded by the European Regional Development Fund) and NoHNES (Spanish Ministerio de Educación y Ciencia - TIN2007-68083-C02).

References

1. Augerat, P., Belenguer, J.M., Benavent, E., Corbern, A., Naddef, D., Rinaldi, G.: Computational results with a branch and cut code for the Capacitated Vehicle Routing Problem. Research Report 949-M, Université Joseph Fourier, Grenoble, France (1995)
2. Bäck, T., Fogel, D.B., Michalewicz, Z.: Evolutionary Computation 1: Basic Algorithms and Operators. IOP Publishing Ltd (2000)
3. Baker, B.M., Ayechew, M.A.: A genetic algorithm for the vehicle routing problem. Computers and Operations Research 30(5), 787–800 (2003)
4. Nicolás Barajas. Estado del arte del problema de ruteo de vehículos (VRP). Masters thesis, Universidad Nacional de Colombia (2007)
5. Benjamin, J.: An analysis of inventory and transportation costs in a constrained network. Transportation Science 23(3), 177–183 (1989)
6. Branch and cut.org. Vehicle routing data sets. Website, `http://branchandcut.org/VRP/data/`

7. Burns, L.D., Hall, R.W., Blumenfeld, D.E., Daganzo, C.F.: Distribution strategies that minimize transportation and inventory costs. Operations Research 33(3), 469–490 (1985)
8. Campbell, A.M., Savelsbergh, M.W.P.: A decomposition approach for the Inventory-Routing Problem. Transportation Science 38(4), 488–502 (2004)
9. Cardós, M., García-Sabater, J.P.: Designing a consumer products retail chain inventory replenishment policy with the consideration of transportation costs. International Journal of Production Economics 104(2), 525–535 (2006)
10. Çetinkaya, S., Lee, C.: Stock replenishment and shipment scheduling for vendor-managed inventory systems. Management Science 46(2), 217–232 (2000)
11. Clarke, G., Wright, W.: Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 12, 568–581 (1964)
12. Coltorti, D., Rizzoli, A.E.: Ant Colony Optimization for real-world vehicle routing problems. SIGEVOlution 2(2), 2–9 (2007)
13. Constable, G.K., Whybark, D.C.: The interaction of transportation and inventory decisions. Decision Sciences 9(4), 688–699 (1978)
14. Cordeau, J.-F., Gendreau, M., Laporte, G.: A Tabu Search heuristic for periodic and multidepot Vehicle Routing Problems. Networks 30(2), 105–119 (1997)
15. Dong, L.W., Xiang, C.T.: Ant Colony Optimization for VRP and mail delivery problems. In: IEEE International Conference on Industrial Informatics, pp. 1143–1148. IEEE, Los Alamitos (2006)
16. Dorigo, M., Sttzle, T.: Ant Colony Optimization. MIT Press, Cambridge (2004)
17. dos Santos Coelho, L., Lopes, H.S.: Supply chain optimization using chaotic differential evolution method. In: IEEE International Conference on Systems, Man and Cybernetics, 2006. SMC 2006, October 8-11, vol. 4, pp. 3114–3119 (2006)
18. Esparcia-Alcázar, A.I., Lluch-Revert, L., Cardós, M., Sharman, K., Andrés-Romano, C.: A comparison of routing algorithms in a hybrid evolutionary tool for the Inventory and Transportation Problem. In: Yen, P.B.G., Wang, L., Lucas, S. (eds.) Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2006, Vancouver, Canada, pp. 5605–5611. IEEE, Omnipress (2006) ISBN: 0-7803-9489-5
19. Esparcia-Alcázar, A.I., Lluch-Revert, L., Cardós, M., Sharman, K., Andrés-Romano, C.: Design of a retail chain stocking up policy with a hybrid evolutionary algorithm. In: Gottlieb, J., Raidl, G.R. (eds.) EvoCOP 2006. LNCS, vol. 3906, pp. 49–60. Springer, Heidelberg (2006)
20. Esparcia-Alcázar, A.I., Lluch-Revert, L., Cardós, M., Sharman, K., Merelo, J.J.: Configuring an evolutionary tool for the Inventory and Transportation Problem. In: Keijzer, M., et al. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2007, London, England, vol. II, pp. 1975–1982. ACM Press, New York (2007)
21. Federgruen, A., Zipkin, P.: Combined vehicle routing and inventory allocation problem. Operations Research 32(5), 1019–1037 (1984)
22. Ganeshan, R.: Managing supply chain inventories: A multiple retailer, one warehouse, multiple supplier model. International Journal of Production Economics 59(1-3), 341–354 (1999)
23. Gendreau, M.: An introduction to Tabu Search. In: Glover, F., Kochenberger, G.A. (eds.) Handbook of Metaheuristics, pp. 37–54 (1999)
24. Gendreau, M., Laporte, G., Potvin, J.: Methaheuristics for the Capacitated VRP. In: [32], pp. 144–145 (2002)
25. Glover, F., Kochenberger, G.A.: Handbook of Metaheuristics. Kluwer Academic Publishers, Dordrecht (2002)
26. Jaszkiewicz, A., Kominek, P.: Genetic local search with distance preserving recombination operator for a vehicle routing problem. European Journal of Operational Research 151(2), 352–364 (2003)

27. Qu, W.W., Bookbinder, J.H., Iyogun, P.: An integrated inventory-transportation system with modified periodic policy for multiple products. European Journal of Operational Research 115(2), 254–269 (1999)
28. Sindhuchao, S., Romeijn, H.E., Akçali, E., Boondiskulchok, R.: An integrated inventory-routing system for multi-item joint replenishment with limited vehicle capacity. J. of Global Optimization 32(1), 93–118 (2005)
29. Speranza, M.G., Ukovich, W.: Minimizing transportation and inventory costs for several products on a single link. Operations Research 42(5), 879–894 (1994)
30. Tan, K.C., Chew, Y.H., Lee, L.H.: A hybrid multiobjective evolutionary algorithm for solving vehicle routing problem with time windows. Computational Optimization and Applications 34(1), 115–151 (2006)
31. Tavares, J., Machado, P., Pereira, F.B., Costa, E.: On the influence of GVR in vehicle routing. In: Proceedings of the 2003 ACM Symposium on Applied Computing, SAC, Melbourne, FL, USA, March 9-12, 2003, pp. 753–758. ACM, New York (2003)
32. Toth, P., Vigo, D.: The Vehicle Routing Problem. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, SIAM monography on Discrete Mathematics and Applications (2001)
33. Viswanathan, S., Mathur, K.: Integrating routing and inventory decisions in one-warehouse multiretailer multiproduct distribution systems. Manage. Sci. 43(3), 294–312 (1997)
34. Zhao, Q.H., Wang, S.Y., Lai, K.K., Xia, G.P.: Model and algorithm of an inventory problem with the consideration of transportation cost. Computers and Industrial Engineering 46(2), 397–398 (2004)

A Memetic Algorithm for a Pick-Up and Delivery Problem by Helicopter

Nubia Velasco[1], Philippe Castagliola[2], Pierre Dejax[3], Christelle Guéret[3], and Christian Prins[4]

[1] Universidad de los Andes, Carrera 1 Este No. 19A -40 –Bogotá- Colombia
nvelasco@uniandes.edu.co
[2] IRCCyN – Université de Nantes, 2 avenue du Professeur Jean Rouxel – BP 539 44475 Carquefou Cedex – France
philippe.castagliola@univ-nantes.fr
[3] IRCCyN – École des Mines de Nantes, La Chantrerie – BP 20722 44307 Nantes Cedex 3 – France
{dejax, gueret}@emn.fr
[4] Institut Charles Delaunay – Université de Technologie de Troyes – BP 2060 10010 Troyes Cedex – France
christian.prins@utt.fr

Summary. This paper presents a memetic algorithm for a pick-up and delivery problem. The specific application studied here is the personnel transportation within a set of oil platforms by one helicopter that may have to undertake several routes in sequence. Different versions of the algorithm are presented and tested on randomly generated instances as well as on real instances provided by a petroleum company. The results show that the solutions obtained are 8% better than construction and improvement solutions on randomly generated instances.

1 Introduction

The Pick-up and Delivery Problem (PDP) consists in finding a set of optimal routes for a fleet of vehicles in order to serve a set of transportation requests. Each transportation request is defined by one pick-up location, a corresponding delivery location and a demand to be transported between these locations. The demand could involve goods or persons.

There are several types of PDP: problems comprising one vehicle (1-PDP) or several vehicles (m-PDP), with time windows (PDPTW) or without time windows (PDP). In the majority of these problems, the vehicles are cars (passenger transportation) or trucks (freight transportation) with the same characteristics, i.e. capacity or speed.

The most practical application of PDP involve time constraints, this is why the more general variant of the problem is called PDPTW. Time constraint establishes time intervals to visit a client (time windows constraints) or the maximum ride time restrictions for passengers, these kind of constraints are specified by the clients.

Usual applications of PDP include the transport of disable and elderly persons (Dial-a-ride problem), sealift and airlift of cargo and troupes, urban courier services and emergency vehicle dispatching.

The most common objective functions are the minimization of the cost or the time and the maximization of customer satisfaction, considering the time windows specified by the clients.

The application studied in this chapter is the transport of personnel within a set of oil platforms by helicopter, and does not consider time windows constraint. This chapter presents a memetic algorithm (MA) to solve this problem. In the proposed MA, each chromosome is a sequence of nodes without route delimiters. The split procedure proposed by Prins [10] for the capacitated VRP was adapted to evaluate the chromosome. The algorithm is tested on randomly generated instances as well as on real instances provided by a petroleum company.

The remainder of the chapter is organized as follows. The next section presents a literature review of different PDP's. In section 3 the problem is formally defined. The proposed memetic algorithm is presented in section 4. The algorithm is tested in real and random instances and the results are presented in the section 5. Finally the conclusions and future research are given in section 6.

2 Literature Review

Several approaches have been used to solve the PDPs: two-phase and three-phase heuristics, tabu search, branch and bound algorithms and column generation. Many of these algorithms perform a preliminary grouping phase, in which transportation requests (for example requests having similar time windows) are aggregated. A survey of these methods can be found in Salvelsbergh and Sol [12], Desaulniers *et al.* [2] and Cordeau *et al.* [1]. Some approaches have been applied to the transport of personnel on oil platforms: a column generation approach by Tijssen [14] and insertion and local search methods by Fiala Timlin and Pulleyblank [3].

Only a few genetic algorithms (*GA*) have been developed for PDPTW's. Potter and Bossomaier [9] presented a *GA* for a PDPTW in which the grouping and the routing parts of the problem are solved separately. Later, Jih and Hsu [5] developed a hybrid genetic algorithm to solve the 1-PDPTW. A solution consists of a single route encoded as a chromosome containing the list of visited nodes. The initial population contains a set of routes generated by a truncated dynamic programming approach.

Jung and Haghani [6] developed a *GA* for the m-PDPTW with one route for each vehicle and soft time windows. The chromosome has $2n$ genes, where n corresponds to the number of transportation requests. The i^{th} gene, associated to a pick-up or a delivery node i, is a four digit number. The first digit identifies the vehicle number that visits the node. The last three digits form a code associated to the sequence. It is possible to know the order in which a vehicle k visits the different nodes by sorting all genes with a first digit equal to k in increasing order of these codes.

Recently, Pankratz [8] developed a grouping genetic algorithm to solve the m-PDPTW, with one route for each vehicle and hard time windows. In this approach, one gene corresponds to one route. A chromosome is then a list of routes. The initial population is generated using an embedded insertion heuristic.

To our knowledge no metaheuristic nor genetic algorithm have been proposed to solve the PDP without time windows, like the case studied in this chapter.

3 Problem Description

The routing problem studied in this chapter consists in finding a set of vehicles routes $\mathcal{R}$ for a single helicopter in order to transport people within oil platforms. The helicopter has an associated base site, a limited capacity and a maximal ride time. At the end of this time the helicopter must return to the base for refueling. This problem is a PDP without time windows.

Formally the PDP can be defined on a directed graphe $G = (\mathcal{N}, \mathcal{A})$ where $\mathcal{N} = \{0\} \cup \mathcal{N}^+ \cup \mathcal{N}^-$, 0 represents the helicopter base, $\mathcal{N}^+ = \{1, \ldots, n\}$ is the set of all the pick-up nodes and $\mathcal{N}^- = \{n+1, \ldots, 2n\}$ is the set of all the delivery nodes. n is the number of clients or transportation requests to be satisfied. Each request k deals with one single passenger who must be transported by helicopter from a pick-up node $k^+ \equiv k \in \mathcal{N}^+$ to a delivery node $k^- \equiv k+n \in \mathcal{N}^-$. Each node $i \in \mathcal{N}$ has a service time s_i (for picking the passenger up and dropping him/her off) and an approach capacity a_i (maximum number of passengers allowed to land at a time, for security reasons).

In set $\mathcal{N}$, two nodes may represent the same geographical location if two requests have the same departure or arrival site. $\mathcal{A}$ is the set of feasible arcs. For each arc $(i, j) \in \mathcal{A}$ the weight $w(i, j)$ is equal to the distance between i and j. The travel time t_{ij} between any two nodes i and j is pre-computed assuming the distances and the average helicopter speed. All data are integers except the t_{ij} which are real numbers. Due to problem constraints some arcs are infeasible: for example a passenger k cannot go from its delivery node k^- to its pick-up node k^+, it is impossible to visit a delivery node before its associated pick-up node. The helicopter is characterized by a capacity Q (maximum number of passengers) and a maximum route duration T before refueling at the base node. No passenger are allowed in the helicopter when it is returning to the base node for refueling.

A typical helicopter route $r = i_1, \ldots, i_{n_r}$ where $i_l \in \mathcal{N}$ and n_r is the number of nodes in the route r, consists in leaving the base, visiting a sequence of nodes, and returning to the base. The helicopter load is incremented after each pick-up and decremented after each delivery. The duration of a route is the sum of its travel and service times. A feasible solution is a set of routes satisfying the following classical PDP constraints:

- **Pairing:** for any request k, k^+ and k^- must be visited in the same route.
- **Precedence:** for any request k, k^+ must be visited before k^-.
- **Capacity:** the number of passengers on board must never exceed Q.

Additionally, the problem studied here has three specific features:

- **Approach capacity:** no more than a_i passengers are allowed on board when landing at site i.
- **Maximum route duration:** the total duration of any route must not exceed the time T.
- **No time windows:** The passengers are ready to take the helicopter at the pick-up node at any time and there are no time constraint at delivery node.

The objective is to compute a set of feasible helicopter routes satisfying all transportation requests and minimizing the total duration of the routes.

4 Proposed Method

The proposed method is a memetic algorithm *MA*. A memetic algorithm is a genetic algorithm (*GA*) in which the offspring undergoes a local search procedure to replace an existing solution [7]. This additional local search can be viewed as a learning phase. In the MA proposed, each chromosome is a sequence of nodes without route delimiters, evaluated with a splitting procedure.

This section describes the main characteristics of the proposed MA: the solution encoding, the chromosome evaluation, the initial population, the crossover operator, the local search procedure, and the management of the population. Finally the general structure of the algorithm is described.

4.1 Solution Encoding

Like in most *GA*s for the Traveling Salesman Problem (TSP), a chromosome is a sequence (permutation) $\mathcal{S} = \{0, i_1, i_2, \ldots, i_{2n}\}$ of nodes, without route delimiters. Such an encoding was used by Hjorring [4] and Prins [10] for the Vehicle Routing Problem (VRP). This encoding may be interpreted as the order in which a vehicle must visit all customers, if a single route satisfies all requests. The chosen chromosome is encoded as a list of transportation request indices, in which each request k appears twice: once as $k+$ to indicate the pick-up node $k \in \mathcal{N}^+$ and once as $k-$ for the delivery node $n + k \in \mathcal{N}^-$. For example, a chromosome is described by: $(1+, 1-, 2+, 2-, 3+, 4+, 4-, 3-)$. In order to cut $\mathcal{S}$ into routes, an optimal splitting procedure *Split* is used to get the best solution respecting the sequence.

4.2 Chromosome Evaluation

The *Split* procedure is a polynomial procedure that allows cutting the chromosomes into routes that satisfy the problem constraints.

Let $\mathcal{S} = (0, i_1, i_2, \ldots, i_{2n})$ be a sequence of $2n + 1$ nodes, where n is the number of requests. $Split$ works by finding a shortest path in an auxiliary graph $\mathcal{G}' = \{\mathcal{N}', \mathcal{A}'\}$. $\mathcal{N}' = \{0, i_1, i_2, \ldots, i_{2n}\}$ contains $2n + 1$ nodes (a node 0 corresponding to the helicopter base, and two nodes associated to each request). $\mathcal{A}'$ contains one arc (i_k, i_{k+n_r}) if a route r starting and ending at the base node and traversing the sequence of clients nodes from i_{k+1} to i_{k+n_r} is feasible. Such a route is feasible if pairing constraints, capacity constraints, approach capacity constraints and maximal route duration are fulfilled. Note that the precedence constraints are respected in the original sequence $\mathcal{S}$. Finally, each arc (i_k, i_{k+n_r}) has a weight z_r equal to the duration of the route. An optimal solution for a chromosome associated to a permutation $\mathcal{S}$ of nodes corresponds to a min-cost path μ from 0 to $2n$ in this graph $\mathcal{G}'$. μ can be computed in $O(n^2)$ time using Bellman's algorithm for acyclic graphs [13].

Algorithm 1. Split procedure

```
1: $V[0] = 0$
2: for $k := 1$ to $2n$ do
3:     $V[i_k] = +\infty$
4: end for
5: for $k = 1$ to $2n$ do
6:     $D = 0$
7:     $L_b = 0$
8:     $L_a = 0$
9:     $\mathcal{V} = \emptyset$
10:    $l = k$
11:    repeat
12:        // Update load of the helicopter and $\mathcal{V}$
13:        if $i_l \in N^+$ then
14:            $\mathcal{V} = \mathcal{V} \cup \{r(i_l)\}$
15:            $L_a = L_a + 1$
16:        else
17:            $\mathcal{V} = \mathcal{V} \setminus \{r(i_l)\}$
18:            $L_a = L_a - 1$
19:        end if
20:        // Update route duration
21:        if $l = k$ then
22:            $D = D + t_{0,i_l} + t_{i_l,0} + s_{i_l}$
23:        else
24:            $D = D + t_{i_{l-1},i_l} + t_{i_l,0} - t_{i_{l-1},0} + s_{i_l}$
25:        end if
26:        // if all constraints are respected
27:        if $(D \leq T)$ and $(L_a \leq Q)$ and $(L_b \leq a_{S_j})$ and $(\mathcal{V} = \emptyset)$ then
28:            if $V[i_{k-1}] + D < V[k_l]$ then
29:                $V[i_l] = V[i_{k-1}] + D$
30:                $pred[i_l] = k - 1$
31:            end if
32:        end if
33:        $l = l + 1$
34:        $L_b = L_a$
35:    until $(j > 2n)$ or $(D > T)$ or $(L_a > Q)$ or $(L_b > a_{i_l})$
36: end for
```

Algorithm 1 represents the $Split$ procedure. It computes for each node $i_k \in \mathcal{N}'$ a label $V[i_k]$ that represents the duration of the shortest path from 0 to i_k. The main loop (*repeat* loop) enumerates all feasible arcs or sub-sequences $i_k, \ldots i_{k+n_r}$ and updates $V[i_k]$. The algorithm does not generate $\mathcal{G}'$ explicitly and runs in $O(n)$ space and $O(n^2)$ time. The algorithm output is a PDP solution whose trips correspond to the arcs on the shortest path, as illustrated by figure 2. The total duration of the solution is given by $V[i_{2n}]$. This PDP solution is optimal for the sequence defined by the chromosome.

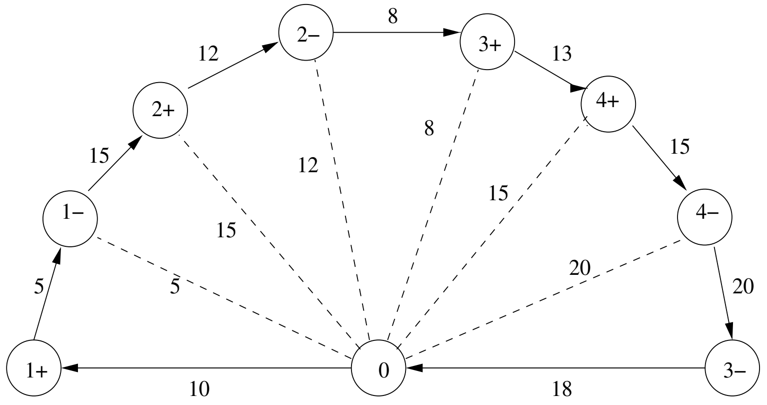


Fig. 1. A sequence of nodes

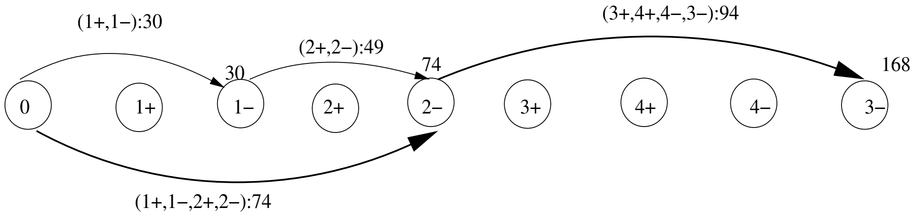


Fig. 2. Graph H

The following variables are used in Algorithm 1:

L_b: number of passengers in the helicopter when it arrives at a node
L_a: number of passengers in the helicopter when it leaves a node
$\mathcal{V}$: set of requests whose pick-up node has been visited but whose delivery node has not been visited
$r(i_k)$: request associated to node i_k
D: route duration
$pred[i_k]$: predecessor of i_k in the shortest path. This variable allows to extract the shortest path solution associated to the chromosome.

In the illustrative example given below, we consider that *a)* there is no approach capacity on the platforms, *b)* the capacity Q of the helicopter is 3, *c)* the service time s_i at each node is 5 minutes, and *d)* the maximal route duration T is 100 minutes. Figure 1 shows a sequence $\mathcal{S} = (1+, 1-, 2+, 2-, 3+, 4+, 4-, 3-)$. The travel duration between nodes is indicated on each arc.

Figure 2 shows the associated graph H which contains, for example, the arc $(0, 2-)$ that models the route $(0, 1+, 1-, 2+, 2-, 0)$ with duration 74. The min-cost path μ (in bold lines) contains two arcs and its cost is 168.

Figure 3 shows the resulting solution. It is composed of two routes. Route 1 is (0,1+,1-, 2+,2-,0) with a duration of 74. Route 2 is (0,3+,4+,4-,3-,0) with a duration of 94.

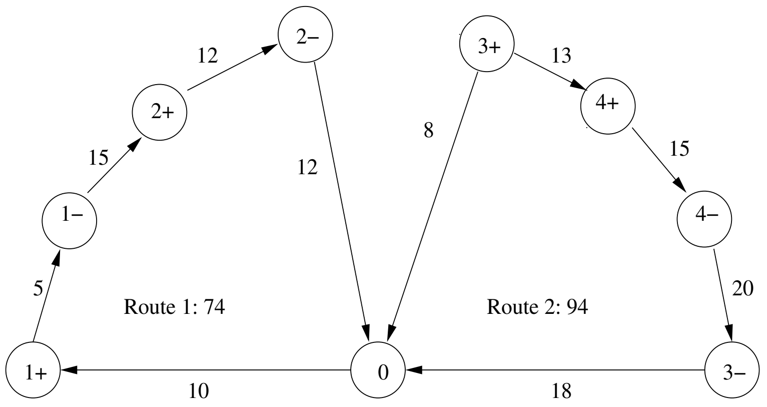


Fig. 3. Resulting solution

4.3 Initial Population

The population is partly initialized using construction and improvement heuristics which find a set of good solutions in a reasonably short time [11]. These heuristics firstly sort the transportation requests according to a given criterion and then insert one new request in the routes at each iteration. Four criteria have been used:

Nearest Request

For each request k to be inserted, the nearest node u of k^+ and the nearest node v of k^- are determined (u and v belonging to the same route r). The request having the minimum sum $t(k^+, u) + t(k^-, v)$ is inserted into route r at the position that minimizes the increase of duration.

Best Insertion

For each request k to be inserted, the route and the position in this route that minimize the increase of duration δ_k are computed. The request inserted is the one having the minimal value of δ_k.

Nearest Center of Gravity

The request inserted at each iteration is the one for which the duration separating its center of gravity from a route node is minimal.

Nearest Global Center of Gravity

Priority for insertion is given to the request having the minimal duration between its center of gravity and a route center of gravity.

In order to diversify the population, the opposite of the previous criteria are also used (Farthest Request, Worst Insertion, Farthest Center of Gravity and Farthest Global Center of Gravity).

The solutions obtained with the construction heuristics are then improved using the local search procedures described in paragraph 4.5. The routes of each solution are concatenated to obtain a chromosome. A proportion of the initial population is generated with these heuristics (proportion of good solutions). The population is completed with solutions in which requests are inserted at random (Random insertion).

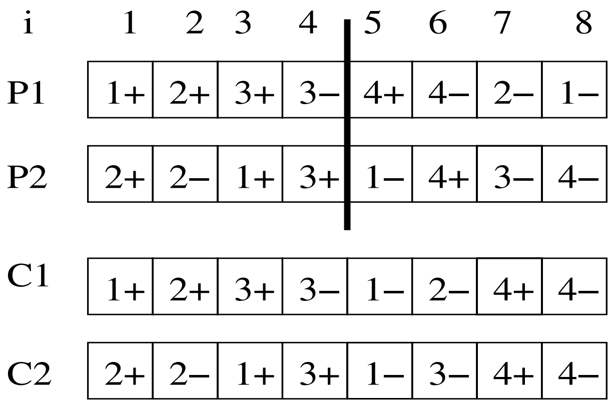
4.4 Crossover

At each iteration, two parents are selected with the binary tournament: to obtain each parent, two solutions are randomly chosen and the least-cost one is kept as parent. A crossover operator is applied to generate two children C_1 and C_2 from the two parents.

Two crossover operators were tested. The first one is an operator with one cutting point and the second has two cutting points. The tests showed that these operators provide similar results. Then, the crossover operator with one cutting point was kept.

This crossover operator consists in selecting one cutting point l. The substring $P_1(i_1) \ldots P_1(i_l)$ is copied from P_1 to C_1 at the same location. Let A be the set of delivery nodes not contained in $P_1(i_1) \ldots P_1(i_l)$ and associated to pick-up nodes in $P_1(i_1) \ldots P_1(i_i)$. Then P_2 is swept circularly from $l + 1$ onwards to complete C_1 first with the delivery nodes of A, then with the other missing nodes. The purpose of A is to reduce the probability of unfeasible solutions when the delivery node is scheduled too far away from its pick-up node. C_2 is produced in the same way by exchanging P_1 and P_2.

Figure 4 gives an example with a cutting point at $l = 4$. C_1 is obtained with the first substring of P_1, that is $(1+, 2+, 3+, 3-)$. Two pick-up nodes (1+ and 2+) appear in this substring without their delivery nodes. Then $A = \{1-, 2-\}$. Thus, P_2 is swept circularly from $l + 1 = 5$ to add the delivery nodes of A first and then the other missing nodes. C_2 is obtained in the same way.



i	1	2	3	4	5	6	7	8
P1	1+	2+	3+	3–	4+	4–	2–	1–
P2	2+	2–	1+	3+	1–	4+	3–	4–
C1	1+	2+	3+	3–	1–	2–	4+	4–
C2	2+	2–	1+	3+	1–	3–	4+	4–

Fig. 4. Crossover operator with one cutting point

4.5 Local Search

After applying the crossover procedure, one child is selected at random. It is transformed into a solution using the *Split* procedure (*see* Paragraph 4.2) and is improved by local search with a fixed probability p_{ls}.

We tested several improvement procedures using the neighborhoods proposed in Prudhomme *et al.* [11]. These procedures sorted by decreasing performance are:

Request transfer

A neighbor *v* of the initial solution *S* is constructed by moving the pick-up and delivery nodes of a request from their initial position to a new one, possibly into another route.

Request permutation

A neighbor *v* of the initial solution S_0 is constructed by permuting the pick-up and delivery nodes of two requests of different routes. For example if d_1 is a request in route r_1 and d_2 is a request in route r_2, the permutation consists in finding the best position for d_1 in r_2 and the best place for d_2 in r_1.

Site transfer

A neighbor *v* of the initial solution S_0 is constructed by moving the successive nodes that represent the same geographical location from their initial position to a new one in the same route. In figure 5, P_4, P_3 and D_1 represent the same geographical site. The site transfer consists in moving nodes P_3 and D_1 right after node P_2 and right before P_4. In this way the helicopter does only one stop in the site represented by nodes D_1, P_3 and P_4.

2-Opt

A neighbor *v* of the initial solution S_0 is constructed by removing two arcs from a route and reconstructing a new different route. This neighborhood achieves the weakest performance because a lot of the neighbors do not respect the precedence constraints.

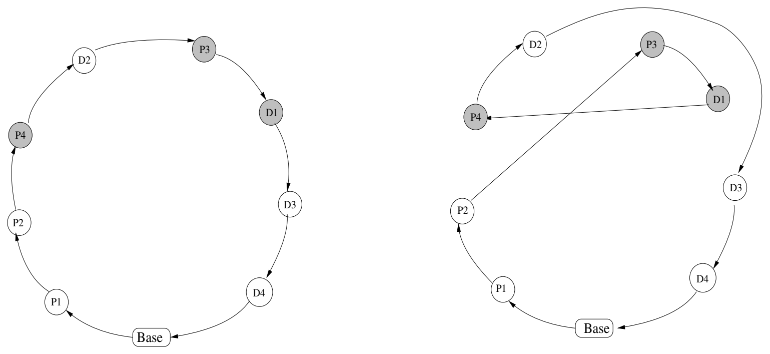


Fig. 5. Site transfer

Algorithm 2. Memetic Algorithm

```
1: $\Pi = \emptyset$ // Initial population
2: $I = 0$ //Iterations
3: $I_imp = 0$ //Iterations without improvement
4: // GENERATION OF INITIAL POPULATION:
5: $|\Pi| = 0$
6: repeat
7:     $A$ = Execute one construction and insertion heuristic (CIH)
8:     if ($A \notin \Pi$) and ($A$ not equivalent to a solution already in $\Pi$) then
9:         $\Pi = \Pi \cup \{A\}$
10:    end if
11: until $|\Pi| = P \times \%gs$ or all CIH executed
12: repeat
13:    $A$ = Execute Random Insertion
14:    if ($A \notin \Pi$) and ($A$ not equivalent to a solution already in $\Pi$) then
15:        $\Pi = \Pi \cup \{A\}$
16:    end if
17: until $|\Pi| = P$
18: // MAIN LOOP:
19: repeat
20:    // Crossover:
21:    Select two parents $P_1$ and $P_2$ by binary tournament
22:    Select one cutting point $i$ at random
23:    Create two children $C1$ and $C2$ by applying crossover to $(P_1, P_2, i)$
24:    Select one child $C$ at random
25:    // Local search:
26:    if $random < p_{ls}$ then
27:        $C$= local search($C$)
28:    end if
29:    // Elimination of a chromosome of the population:
30:    if ($C \notin \Pi$) and ($C$ not equivalent to a chromosome already in $\Pi$) then
31:        Replace the worst chromosome of $\Pi$ by $C$
32:        $I = I + 1$
33:        if $C$ is better than the best chromosome in $\Pi$ then
34:            $I_{imp} = 0$
35:        else
36:            $I_{imp} = I_{imp} + 1$
37:        end if
38:    end if
39: until $I_{imp} > N_{imp}$ or $I > N_{max}$
```

These neighborhoods are searched successively and the current iteration ends at the first improving move. The whole local search stops when all neighborhoods are explored without finding any improvement.

Like the solutions for the initial population, the solution obtained at the end of a local search is converted into a chromosome by concatenating its trips.

4.6 Population Management

Each time that one child is produced, it is added to the population and one chromosome is eliminated. Several rules can be applied to eliminate a chromosome from the population . Two options were tested: replacing the worst chromosome of the population by the produced child, or replacing one chromosome above the median cost. The first version provides the best results.

Furthermore, in the population, clones and equivalent solutions are forbidden to ensure a better variability of solutions and to decrease the risk of premature convergence. We call clones identical chromosomes. Equivalent solutions are different chromosomes that represent the same order of visits of geographical locations. For example, if two requests 1 and 2 have the same pick-up and delivery sites, then the two chromosomes $(1+, 1-, 2+, 2-)$ and $(2+, 2-, 1+, 1-)$ are equivalent solutions since the same geographical locations are visited in the same order. In general, this technique brings enough diversification and a specific mutation operator is not required.

4.7 General Structure

Algorithm 2 summarizes the proposed memetic algorithm. In this algorithm, a population of a given size is first generated. A proportion of this population is generated by the construction and improvement heuristics and completed by random insertion heuristic. The algorithm stops when a maximal number of iterations without improvement (i.e. the child produced is not better than the best chromosome of the population) or a maximal number of iterations is reached. The algorithm uses the following parameters:

1. Population size (P)
2. Local search rate (p_{ls})
3. Proportion of good solution in the initial population ($\%gs$)
4. stopping conditions:
   a) Maximal number of iterations (N_{max})
   b) Maximal number of iterations without improvement (N_{imp})

5 Computational Evaluation

The algorithm was tested on real instances provided by a petroleum company and on randomly generated instances of different sizes. The last ones were created considering that benchmark instances for the PDP without time windows were not found in the literature. Randomly generated problems are composed of instances with 5 sites and 10 requests (5S-10R), 5 sites and 20 requests (5S-20R), 10 sites and 30 requests (10S-30R), 20 sites and 50 requests (20S-50R) and 50 sites and 100 requests (50S-100R). One hundred problems of each size were generated. Real instances are composed of 30 problems from 6 to 12 sites and from 9 to 44 requests. The helicopter has a capacity of 13, the maximal route duration is 70 minutes and the service time is 5 minutes. In all these instances, 5% of the sites have an approach capacity limited to five passengers.

All developments were done in Java (Eclipse) and tested on a AMD Athlon XP 1800 desktop computer clocked at 1.53 GHz under Windows 2000. The algorithm was run once to solve each of the 100 instances of each size. In total the algorithm was run on 530 different problems.

The following sections present the parameters defined in a preliminary testing phase, and the results of experimentations for randomly generated instances and real instances.

5.1 Parameter Tuning

In order to define the parameters that yield the best performance of the algorithm, a testing phase was performed using the 100 instances of 20S-50R problems. After preliminary testing, the values of parameters have been set at:

1. population size (P): 30 and 200.
2. local search rate (p_{ls}): 0 (simple genetic algorithm) and 0.5
3. proportion of good solutions in initial population ($\%gs$): 0 and 30.
4. stopping conditions:
    a) Maximal number of iterations (N_{max}): 10000 and 40000
    b) Maximal number of iterations without improvement (N_{imp}): 1000 and 6000

A $2^5 = 32$ full factorial experimental design was used to find the parameters and the combination of parameters that maximizes the gain obtained by the memetic algorithm (MA), compared to the solutions of the constructive and improvement heuristics (CIH) (see Prud'homme *et al.* [11]). This gain is computed as follows: $(CIH - MA)/CIH \times 100$. This full factorial experimental design allows to estimate the coefficients of a "linear plus interaction" model. Table 1 shows the average results in terms of CPU time and gain for 32 configurations of parameters tested. The gain may be negative when the MA is reduced to a simple GA (MA$_0$: no local search, no good initial solutions), because the construction and improvement heuristics are already quite sophisticated.

The data were analyzed with a multiple regression analysis. Concerning the gain, the results of this analysis are summarized in Table 2. The value of the coefficient of multiple determination $R^2 = 0.9962$ shows an excellent fit for the "linear plus interaction" model. The coefficients corresponding to this model are listed in Table 2 along with their corresponding p-values. The coefficients and the corresponding parameters or combination of parameters that have a p-value less than 0.05 (in bold) are considered to significantly contribute to the gain, i.e. in our case the population size P, the local search rate p_{ls} and the proportion of good solutions in the initial population $\%gs$ while the maximal number of iterations N_{max} and the maximal number of iterations without improvement N_{imp} seems to have no significant influence over the gain.

Concerning the CPU Time, the results of the multiple regression analysis are summarized in Table 3. The value of the coefficient of multiple determination $R^2 = 0.989$ shows again an excellent fit for the "linear plus interarction" model. Based on the values of the p-values, the parameters that significantly contribute to the CPU Time are the population size P, the local search rate p_{ls}, the maximal number of iterations N_{max} and the maximal number of iterations without improvement N_{imp} while the local search rate p_{ls} seems to have no significant influence over the CPU Time.

Indeed, the addition of the local search procedure allows to increase the gain from -13% to 9% in average. The addition of good solutions in the initial population also has a positive effect on the gain increasing it from -13% to 6% in average.

Table 1. Full factorial design for two levels and five factors with $2^5 = 32$ tests: the average results of gain and CPU time

Nmax	Nimp	P	%gs	pls	Gain	CPU
10000	1000	30	0	0.0	-13.23	0.74
				0.5	8.89	53.88
10000	1000	30	30	0.0	5.71	2.35
				0.5	9.22	49.56
10000	1000	200	0	0.0	-13.14	0.77
				0.5	9.97	136.60
10000	1000	200	30	0.0	8.48	12.74
				0.5	9.97	121.18
10000	6000	30	0	0.0	-13.21	4.37
				0.5	9.14	207.11
10000	6000	30	30	0.0	5.70	5.36
				0.5	9.07	203.08
10000	6000	200	0	0.0	-13.02	4.88
				0.5	10.30	294.23
10000	6000	200	30	0.0	8.61	15.43
				0.5	10.48	288.81
40000	1000	30	0	0.0	-13.23	0.74
				0.5	8.99	53.99
40000	1000	30	30	0.0	5.72	2.35
				0.5	9.13	50.37
40000	1000	200	0	0.0	-13.05	0.78
				0.5	9.92	145.40
40000	1000	200	30	0.0	8.46	12.75
				0.5	9.95	123.07
40000	6000	30	0	0.0	-13.25	4.34
				0.5	9.07	214.17
40000	6000	30	30	0.0	5.71	5.38
				0.5	9.07	203.05
40000	6000	200	0	0.0	-12.99	4.87
				0.5	10.43	355.52
40000	6000	200	30	0.0	8.62	15.49
				0.5	10.60	338.69

Considering that the objective is to maximize the gain, the multiple regression analysis indicates that the level of population size, the local search rate and the proportion of good solutions in the initial population should be 200, 0.5 and 30%, respectively. To minimize the CPU time the algorithm is stopped after 1000 iterations without improvement or after 10000 iterations in total.

5.2 Results for Randomly Generated Instances

Table 4 shows the results for the randomly generated instances. This table provides for each problem size:

Table 2. "Linear plus interarction" coefficients and p-values for the gain

$R^2 = 0.9962$		
Factors and Interactions	Coefficients	p-value
1	**3.3153**	0.0000
P	**0.6591**	0.0000
p_{ls}	**6.3229**	0.0000
$\%gs$	**5.0902**	0.0000
N_{max}	0.0077	0.9080
N_{imp}	0.0805	0.2306
$P \times p_{ls}$	-0.0931	0.1661
$P \times \%gs$	**0.3309**	0.0000
$P \times N_{max}$	0.0113	0.8656
$P \times N_{imp}$	0.0737	0.2717
$p_{ls} \times \%gs$	**-5.0430**	0.0000
$p_{ls} \times N_{max}$	0.0004	0.9949
$p_{ls} \times N_{imp}$	0.0519	0.4385
$\%gs \times N_{max}$	-0.0055	0.9347
$\%gs \times N_{imp}$	-0.0037	0.9561
$N_{max} \times N_{imp}$	0.0044	0.9474

Table 3. "Linear plus interarction" coefficients and p-values for the CPU Time

$R^2 = 0.989$		
Factors and Interactions	Coefficients	p-value
1	**91626**	0.0000
P	**25324**	0.0000
p_{ls}	**85793**	0.0000
$\%gs$	-1022	0.4365
N_{max}	**4059.2**	0.0026
N_{imp}	**43673**	0.0000
$P \times p_{ls}$	**22694**	0.0000
$P \times \%gs$	92.72	0.9436
$P \times N_{max}$	**3562**	0.0079
$P \times N_{imp}$	**4117.9**	0.0023
$p_{ls} \times \%gs$	**-4170.6**	0.0020
$p_{ls} \times N_{max}$	**4053.9**	0.0027
$p_{ls} \times N_{imp}$	**41991**	0.0000
$\%gs \times N_{max}$	-768.03	0.5584
$\%gs \times N_{imp}$	135.45	0.9177
$N_{max} \times N_{imp}$	**3332.1**	0.0127

- The CPU time
- The rejection rate: proportion of clones and equivalent solutions generated
- The last iteration with improvement

Table 4. Results for randomly generated instances

	5S-10R	5S-20R	10S-30R	20S-50R	50S-100R
CPU Time (s)					
Average	1.36	5.47	21.71	121.18	906.97
Minimal	0.72	2.88	9.98	52.30	554.41
Maximal	7.88	21.66	92.64	284.58	2087.22
Rejection rate (%)					
Average	38.08	16.80	10.15	5.64	2.06
Minimal	17.97	7.83	4.67	2.15	0.60
Maximal	64.40	31.75	22.97	16.42	7.25
Last iteration with improvement					
Average	169	544	852	1395	773
Minimal	0	0	0	0	0
Maximal	8145	6231	4583	8636	4033
Gain between MA solutions and CIH solutions (%)					
Average	5.92	7.63	8.34	9.97	7.91
Minimal	0.00	0.00	0.41	1.48	2.10
Maximal	26.57	27.49	19.90	18.17	14.25
Gain between MA solutions and MA$_0$ solutions (%)					
Average	6.57	14.62	18.64	20.21	21.41
Minimal	0.00	0.33	4.75	10.21	12.91
Maximal	20.38	33.49	31.01	29.77	32.09
Gap between MA and the lower bound of [15] (%)					
♯ problems	100	52	-	-	-
Average	8.14	5.56	-	-	-
Minimal	0.00	0.00	-	-	-
Maximal	28.66	12.78	-	-	-

- The gain between the CIH solutions and the MA solutions computed as $(CIH - MA)/CIH \times 100$
- The gain between the solutions obtained with the memetic algorithm without good solutions in the initial population and without local search procedure (GA). This gain is computed as $(GA - MA)/GA \times 100$
- The gap between the solutions and a lower bound computed by solving the relaxation of a set covering formulation of the problem solved with the column generation method [15].

For each of these parameters the table contains the average, the maximal and the minimal value.

The CPU time increases with the problem size: it is equal to 1.36 seconds for small problems (5S-10R) and around 15 minutes for the biggest ones (50S-100R). The CPU time for the CIH methods is less than 1 second for all problem sizes.

The average rejection rate is 2.06% for the 50S-100R instances, 5.64% for the 20S-50R instances, 10.15% for the 10S-30R instances, 16.80% for the 5S-20R instances and 38.08% for the 5S-10R instances. These differences are due to the fact that the

Table 5. Results for real instances

Problem	CPU Time (sec)	Rejection rate (%)	Last iteration with improvement	Gain (%) of MA compared to CIH	Gain (%) of MA compared to GA	Gap (%) to LB
7S-9D	1.156	49.29	0	0.00	0.00	0.00
6S-12D	1.797	44.32	0	11.05	11.49	10.09
7S-13D	2.5	50.47	0	0.00	0.00	22.53
7S-15D	2.75	39.83	0	8.44	10.10	-
7S-15D	2.672	32.93	0	2.35	7.62	7.24
8S-16D	3.468	30.30	12	0.00	9.94	-
9S-16D	2.829	14.89	0	3.43	3.90	-
7S-17D	3.281	23.55	0	0.83	4.03	7.90
8S-17D	2.562	19.48	0	6.03	11.95	4.90
9S-17D	2.407	20.32	0	3.82	7.09	4.97
8S-20D	6.062	14.00	395	5.72	8.26	-
9S-20D	3.719	16.04	0	0.00	6.04	-
7S-21D	4.828	20.06	0	15.09	8.42	-
7S-21D	4.594	18.90	0	15.09	15.36	-
10S-25D	16.422	25.29	1532	2.63	1.29	-
9S-26D	9.406	16.48	318	16.97	17.32	-
7S-28D	9.656	18.12	157	1.04	0.56	-
7S-28D	8.531	14.75	0	1.04	0.56	-
9S-28D	7.953	16.40	96	1.61	3.21	-
8S-29D	11.938	15.40	247	3.95	3.95	-
12S-30D	15.656	10.15	0	12.69	15.08	-
10S-31D	12.781	9.67	0	13.15	7.36	-
10S-31D	12.829	8.26	0	13.15	7.36	-
13S-32D	22.765	12.77	550	6.49	8.57	-
10S-33D	18.766	14.89	0	4.57	5.87	-
12S-33D	22.079	9.27	145	14.69	18.53	-
12S-33D	36.375	10.36	1112	1.53	5.77	-
11S-37D	24.828	7.83	0	6.38	10.38	-
11S-38D	63.156	10.73	2518	0.53	6.85	-
11S-44D	30.781	12.51	0	8.73	9.97	-

probability of clones is higher for small instances. This fact is also illustrated with minimal and maximal rejection rates values: the two extreme values are reached on the 50S-100R instances (0.60%) and on the 5S-10R instances (64.40%).

On average, the last improvement occurs after 169 iterations for 5S-10R instances, 544 iterations for 5S-20R instances, 852 iterations for 10S-30R instances, 1395 for 20S-50R instances and 773 for 50S-100R instances.

The memetic algorithm (MA) solutions are better than the CIH solutions for all instances. The gain is between 0% et 27.49% according to the problem size. The average gain varies from 5.92% for 5S-10R instances to 9.97% for 20S-50R instances. Concerning the comparison of the MA solutions with the MA_0 solutions, the first ones are between 6.57% and 21.41% better in average depending on problem size, proving that the inclusion of good solutions in the initial population and the local search procedure allow to improve the final solution.

The average gap between LB and MA solutions is 8.14% for the 5S-10R instances and 5.56% for 52 of the 100 5S-20R instances, showing that the MA solutions are close to the optimal solution. It is reduced by 60% compared to the gap obtained by construction and improvement heuristics (17.35% for 5S-10R problems and 15.08% for 5S-20R problems).

Due to prohibitive computation times for the lower bound calculation, it could not be calculated for 48 of the 100 5S-20R instances, neither for any 10S-30R, 20S-50R nor 50S-100R problems.

5.3 Results for Real Instances

Table 5 contains the results for the real instances. The solutions obtained by the company were not available for comparisons. Like for the randomly generated instances, we compared the results obtained with the memetic algorithm to the CIH solutions. For these instances, the memetic algorithm provides better solutions than the best CIH method for 86% of problems. The CPU time ranges from 1.15s to 63.12s. The rejection rate is between 7% and 50%. The CIH solutions are improved up to 16.97%. The gain between GA solutions and MA solutions goes up to 18%.

Furthermore, like for randomly generated instances, the algorithm converges quickly: the best solutions are found in less than 1000 iterations for 63% of the problems.

For real instances, due to prohibitive computation times, the lower bound can not be calculated for 23 of the 30 problems. The gap for these seven instances varies between 0% and 22.5%.

6 Conclusion

In this chapter a memetic algorithm has been presented to solve a pick-up and delivery problem by a single helicopter and without time windows. The application especially studied is the transportation of personnel within oil platforms. This problem has specific constraints such as approach capacity and maximal travel time for the helicopter and no time windows are associated to the requests.

Experimental results for randomly generated instances and real instances provided by a petroleum company show that the proposed memetic algorithm is efficient. The solutions obtained are around 8% better than the best solutions computed with construction and improvement heuristics for randomly generated instances. Real instances solutions show improvements between 0% and 18% for 70% of the cases.

In this chapter the flexibility of split procedure to be adapted to vehicle routing problems with precedence and pairing constraints was demonstrated. On the other hand, the importance of local search procedure to improve the solution comparing it with the ones obtained by the use of genetic algorithm without good solutions in the initial population and local search procedure was shown.

The memetic algorithm proposed in this chapter can easily be adapted to problems in which requests concern several passengers which have to travel together. Our objective is also to adapt this memetic algorithm to solve problems with time windows and use several helicopters simultaneously.

Moreover, we are working at improving the lower bound based on the column generation procedure in order to measure more precisely the efficiency of the proposed algorithm.

The randomly generated instances used in this paper and their solutions can be downloaded from http://www.emn.fr/gueret/PDP/PDP.html

References

1. Cordeau, J.F., Laporte, G., Potvin, J.-Y., Savelsbergh, M.W.P.: Transportation on demand. In: Barnhart, C., Laporte, G. (eds.) Transportation, Handbooks in Operations Research and Management Science. Elsevier, Amsterdam (2005)
2. Desaulniers, G., Desrosiers, J., Erdmann, A., Solomon, M.M., Soumis, F.: VRP with pickup and delivery. In: Toth, P., Vigo, D. (eds.) The vehicle routing problem, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 225–242 (2001)
3. Fiala-Timlin, M.T., Pulleyblank, W.R.: Precedence constrained routing and helicopter scheduling: Heuristic design. Interfaces 22, 100–111 (1992)
4. Hjorring, C.A.: The vehicle routing problem and local search metaheuristics. PhD thesis, Department of Engineering Science, The University of Auckland (1995)
5. Jih, W., Hsu, J.Y.: Dynamic vehicle routing using hybrid genetic algorithms. In: ICRA - International Conference on Robotics and Automation, vol. 1, Detroit, Michigan, pp. 453–458 (1999)
6. Jung, S., Haghani, A.: Genetic algorithm for a pick-up and delivery problem with time windows. Transportation Research Record, TRB(1733) (2000)
7. Moscato, P.: Memetic algorithms: A short introduction. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization, pp. 219–234. McGraw-Hill, New York (1999)
8. Pankratz, G.: A grouping genetic algorithm for the pick-up and delivery problem with time windows, Department of Business Administration and Economics, Fern Universitat - University of Hagen, Hagen, Germany (2003)
9. Potter, T., Bossomaier, T.: Solving vehicle routing problems with genetic algorithms. In: ICEC International Conference on Evolutionary Computing, Perth, Australia, vol. 2, pp. 788–793 (1995)
10. Prins, C.: A simple and effective evolutionary algorithm for the vehicle routing problem. Computers and Operations Research 31, 1985–2002 (2004)
11. Prud´Homme, C., Dejax, P., Guéret, C., Velasco, N.: Heuristiques pour un problème réel de tournées d'hélicoptères de type pick-up and delivery. Technical report, École des Mines de Nantes, 04/5/AUTO (2004)
12. Sol, M., Savelsbergh, M.W.P.: A branch-and-price algorithm for the pickup and delivery problem with time windows. COSOR Memorandum 94-22, Dept. of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands (1994)
13. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to algorithms. MIT Press, Cambridge (1990)
14. Tijssen, G.A.: Theoretical and practical aspects of linear optimization. PhD thesis, Graduate School/Research Institute, Systems, organization and management. University of Groningen, Netherlands (2000)
15. Velasco, N., Dejax, P., Guéret, C.: Vehicle routing optimization for the pick-up and delivery: Application to the transport of personnel on oil platforms. EURO XX, Rhodes, Greece (2004)

When the Rubber Meets the Road: Bio-inspired Field Service Scheduling in the Real World

Israel Beniaminy[1], Dovi Yellin[2], Uzi Zahavi[3], and Marko Žerdin[4]

[1] ClickSoftware Ltd. Israel
israel.beniaminy@clicksoftware.com
[2] ClickSoftware Ltd. Israel
dovi.yellin@clicksoftware.com
[3] Bar-Ilan University Israel
zahaviu@cs.biu.ac.il
[4] Zany Ants Ltd. UK
marko.zerdin@zanyants.com

Summary. We discuss a class of large-scale real-world field service optimization problems which may be described as generalizations of the Vehicle Routing Problem with Time Windows (VRPTW). We describe our experience in the real-world issues concerned with describing and solving instances of such problems, and adapting the solution to the needs of service organizations using a ”universal framework” for bringing together various problem representations and experimenting with different algorithms. Implementations and results of several bio-inspired approaches are discussed: Genetic Algorithm (GA), Ant Colony Optimization (ACO), and a hybrid of ACO with GRASP (Greedy Randomized Adaptive Search Procedure). We conclude by discussing generation of ”human-friendly” solutions, through introduction of local considerations into the global optimization process.

1 Introduction

We discuss a class of large-scale real-world field service optimization problems which may be described as a variant of the Vehicle Routing Problem with Time Windows (VRPTW) with more elaborate instance description, a variety of additional constraints, and a variety of components for the value function which we seek to maximize.

In this section, we describe the problem and its business relevance, its variants and complexities, and lay a foundation for a subsequent discussion of practical methods for solving the problem. In section 2 we describe a universal architecture capable of accommodating a large variety of problem variants and providing a framework that can be utilized by various search algorithms. In sections 3 - 5 we address a few selected bio-inspired algorithms that we tested and present their corresponding results. In the conclusion of this chapter, we introduce some additional ideas and present conclusions.

1.1 Background and Problem Classification

Field service organizations exist in a dynamic, ever-changing world which combines long-range planning with emergency responses. While this could be said to apply to most types of businesses, there are two features of field service that make it particularly challenging. First, the main “raw material” used in “producing” field service are work hours which must be applied at the right time – when the service is provided. Unlike tangible raw materials, and unlike work hours used to produce tangibles that can be stored, a supply of work hours cannot be stored until a time when there is a large demand for it. Field service managers describe this fact as “work hours have zero shelf life”, or even more briefly as “use it or lose it”. The second characteristic of field service is that resources and demands all have physical locations (e.g. location of the service engineer, location where the task needs to be performed). This brings travel times to the forefront of any attempt at optimization.

Previous research on similar problems is reported in [6, 9, 10, 13, 15], and studies of such problems in commercial settings in [16]. Our own experience with these problems comes from our work with a commercial software vendor in the field service optimization space, which gives us the opportunity to discover the real-world issues concerning describing a problem, solving instances of the problem, and adapting the solution to the needs of service organizations.

Generally speaking, field service optimization requires finding a matching of resources (employees, tools, vehicles etc.), demands (jobs, tasks, outages, emergencies) and times. As in all optimization problems, such matchings need to be valid – they need to meet specified constraints; and they need to produce a “good quality” solution – that is, achieve a high value for the goal function. Service organizations differ in their needs, and therefore they differ in the required representation for resources, demands, constraints and goal functions.

These four components of problem representation are not independent. In our experience, they are dictated by four categories of business drivers:

1. *Customer Satisfaction:* The matching of resources, demands and times should meet customer expectations regarding the performance of service: sending the right workers (e.g. with the right skills) to the right location, with the right tools and parts, at the right time (e.g. promised appointment window). Failing to do this has at least two negative consequences: if customers are dissatisfied with the service, it can lead to reduction of business; and tasks that were not performed properly (or at all) may have to be repeated, adding to the workload and to the costs.
2. *Employee Satisfaction:* The service must conform to the employment policies: it has to be limited to working hours and a reasonable amount of overtime, respect times when employees are unavailable, and allow for lunch breaks during the day. Failing to meet these constraints may lead to losing the organization’s most important assets – its workforce – as well as involving it in disputes with labor unions and employee representatives.
3. *Finance:* The service must be performed in a cost-effective manner, considering factors such as the cost of travel, the cost of using different types of workers, the cost of overtime etc. Obviously, workforce utilization is a major component of cost; both “idle time” and excessive travel represent work time that must be paid for but does not contribute to meeting the demand for service and does not bring any revenue.
4. *Regulations:* In many cases, there are regulations governing the service process - e.g. a statutory limit to response time for a call, depending on its severity. Failing

to meet regulatory requirements may lead to substantial fines or even to a loss of operating licenses.

Since different field service organizations have different sets of policies and goals, this general description gives rise to a large and varied class of problems. In the following section, we present a formal way of classifying and describing constraints that may be included in each instance, depending on whether the constraint in question controls the validity of scheduling a demand, assigning to a specific resource, scheduling at a specific time, or any combination thereof.

The same classification may be used for components of the goal function. However, in the real world the distinction between constraints and goals is not always clear-cut. For example, time windows are typically considered to be a constraint, but many service organizations feel that it is better to arrive late than to fail to arrive at all. Furthermore, "being late" is itself a fuzzy concept – e.g. being late by ten minutes may be acceptable, missing by thirty minutes may cause some customer dissatisfaction, while keeping the customer waiting for an extra hour may cause the customer to switch to a different service organization. Another negative consequence could be a wasted trip, because the customer will not wait for the technician. Real-life considerations of this sort sometimes call into question the classification of problem terms into constraints and goals.

1.2 Problem Attributes

Taking VRPTW as the baseline problem description, we find that even the simplest types of field service problems have at least one additional characteristic that makes them fundamentally different from VRPTW. Each such characteristic typically applies to two or more of the parts of problem specification mentioned above: resources, demand, time, constraints and components of the goal function.

Here, we systematically classify some of the more common characteristics of this large variety of problems. The letters `D`, `R`, `T` stand for demand, resource and time, respectively.

- Resource characteristics
  - Resources usually have their own *individuality* which is expressed by attributes such as skills, certifications, efficiency (e.g. some resources consistently finish 10 home-base location, and "calendar" (times at which the resource is available).
  - In light of this individuality, resource capacity is usually limited in a much more rigid way than with the classical VRPTW. Just throwing "another vehicle" at the problem, a common method of achieving feasibility with VRPTW, is usually out of the question.
- Demand characteristics
  - Demand attributes typically include location, allowed time window, expected duration, and required skills. Other attributes that we often see include the revenue generated by performing the task (or a cost of not performing it), priority, and dependency of demand duration on other parts of the solution (e.g. if a task is scheduled immediately after another demand performed by the same resource at the same location, its duration may be reduced).

  - Demands without a uniquely-specified location: Some demands may be fulfilled remotely, from wherever the resource happens to be. Other demands may be fulfilled in one of a number of locations, so that the schedule needs to select one of the allowable locations when assigning the demand to a specific resource and time.
  - Number of required resources: Some demands require more than one resource, specifying the required attributes for each resource.
  - Requirement for parts: Some demands need parts – consumables, spare parts or tools. Vehicles assigned to service engineers may have their own "in-van" inventories, in which case vehicles which already have the required parts are a better choice. When the required parts are not immediately available, a separate "pick-up" sub-task needs to be added for getting the part before driving to the service site.
- `DR` interactions - affecting validity or value of assigning `D` to `R`
  - Matching resource and demand attributes: Demands almost invariably require certain skills and only resources that possess those skills can be assigned to them. Another example is resource "work zones", in which case each matching depends on geographical relationship between demand's location and resource's home-base.
  - Matching specific resources:
    Some demands specify a "required engineer" (who is the only one allowed to perform the specific task) or a "preferred engineer" (so that assigning this engineer raises the goal function's value).
- `RT` interactions - affecting validity or value of scheduling `R` at `T`
  - Resource availability: The resource's calendar specifies when the resource is available to be assigned to demands.
  - Resource time-dependent costs: Resource calendars often specify varying costs depending on time of day, as in overtime work.
- Interactions between assignments
  - Interdependence between demands: Some demands are actually parts of a larger job, and in these cases it is common to see that the decisions made for one demand affect the feasible options of another. Such dependencies usually apply either to timing (e.g. when demand B must not start before demand A was completed) or to resources (e.g. when the resource assigned to demand B must be the same as the one assigned to demand A).
  - Non-local constraints: Often, there are constraints on some aspects of the solution that can't be expressed locally (that is, for one specific assignment or for a single arc between consecutive assignments), but instead depend on the solution as a whole or on a non-trivial part of the solution. For example, even when each resource is limited to a maximum of two hours of overtime per day, there may be a fixed maximum for the amount of overtime any single resource can do throughout a work week (possibly due to union rules), or a maximum on the total amount of overtime in a region (due to budget limitations).

The above list, of course, is just a small sample of what we see in the industry. We could give many additional common (and unique) examples for constraints and value-function components, some of which our formalization places in additional categories such as `DT` and `DRT`.

From a different point of view, we further classify field service scheduling problems into on-line and off-line problems. The scope of this chapter does not allow a detailed discussion of on-line field service optimization, but we note that the on-line variants are very important in real-world applications. One of the goals of on-line variants is to take a given solution, change the problem instance (e.g. by modeling the fact that a demand has extended beyond its expected duration), and find a solution for the new instance that meets the constraints and optimization goals while minimizing the number of changes[1] compared to the original solution. Another important goal of on-line optimization is to provide a prompt response when necessary – the customer won't happily stay on the phone for a long time while waiting for a list of appointments to choose from.

1.3 Problem Complexity

Even small service organizations regularly encounter large problems. For example, take a 10-engineer service organization which performs, on the average, five tasks per engineer per day. If this organization wishes to plan a week ahead (five work days), there are 250 demands in the problem. Our experience extends to workforces with tens of thousands of resources and hundreds of thousands of demands. Such problems are, to some extent, separable, since resources located hundreds of kilometers apart usually won't be considered for the same demand, but factors such as overlapping regions make strict separation difficult. Furthermore, even geographically-localized subproblems can easily reach 30,000 demands.

The intractability of VRPTW for any but the smallest problem instances is well-known. It is worthwhile mentioning that intractability remains even when travel times are all zero ([12, 13]), and even when further requiring all demands to select from a non-overlapping set of time windows (leading to variants of General Assignment Problem - see [5]). Recently, it was found that even when the time window for each task is only slightly longer than its duration, and when all travel times are zero, the problem is still (weakly) NP-hard [1]. This observation is relevant to problem instances we encountered where many travel times are indeed zero (work within facilities) and when most time windows are set by fixed appointment slots (e.g. morning or afternoon appointments).

2 Solution Architecture

2.1 "Universal" Model for Representing Scheduling Problems

The need to support varying business policies of different types of field service organizations has led us to define the **W-6** model, whose name stands for "Who does What, Where, When, With what and for Whom". This is a generic model allowing us to describe the full richness of real-life field service scheduling problems. To make this

[1] The requirement for considering the number of changes as one of the optimization criteria has several reasons, one of which is the propagation of each change to other enterprise software and to the human resources being scheduled.

concrete, we note that the simplest solution description is a set of triplets $\langle D, R, T \rangle$, each triplet specifying that demand D (What) is assigned to resource R (Who) at time T (When). More complex descriptions add additional dimensions, leading to or beyond the six dimensions implied by the name W-6. The W-6 terminology can be viewed as a language in which words are demands, resources and other problem building blocks, the grammar is composed of scheduling policies and logic, and the resulting sentences are the assignments.

2.2 "Universal" Field Service Scheduling Framework

With the **W-6** model as our guide, we defined a framework for representing and solving field service scheduling problems. The design goals for this framework were:

- Accommodate steps, roles and concepts that are a part of most common field service scheduling problems.
- Allow the framework's building blocks to be replaced or extended by customization, so that the framework would support the features that were not part of the original design or were not even anticipated at design time.
- Support a wide variety of optimization problems encountered in real-life field service situations: In order to achieve that, we need to break up the relevant logic into small modules, each responsible for implementing a certain business constraint or goal. These modules are then used together, describing a specific field service problem like adjectives describe a noun.
- Allow the decoupling of complexities associated with creating valid, constraint-satisfying schedules from algorithms that can be used for optimizing those schedules: In order to achieve that, we need to expose the interfaces that allow the schedule-building logic (and its parts) to be used as building blocks for the assembly of working algorithms.

The functionality of the framework needs to support the goals of schedule optimization. We define these goals as: *To create and maintain a collection of valid, constraint-satisfying assignments (a schedule) while providing the services needed to optimize the quality of this schedule.* Figure 1 gives the overall structure of the framework, where a set of logic services encapsulates access to W-6 objects and to computations (such as constraint propagation) performed on these objects. The framework allows additional components (such as custom constraints) to be "plugged-in" and enables any such components to seamlessly participate in the performance of the required services.

From the point of view of scheduling algorithms, the framework is represented by the logic services that it provides. In the remainder of this section we describe these services.

Creating and maintaining W-6 objects

The lowest level on figure 1 – the object server – abstracts the data-manipulation of W-6 objects and frees the higher layers to concentrate on the logic of building

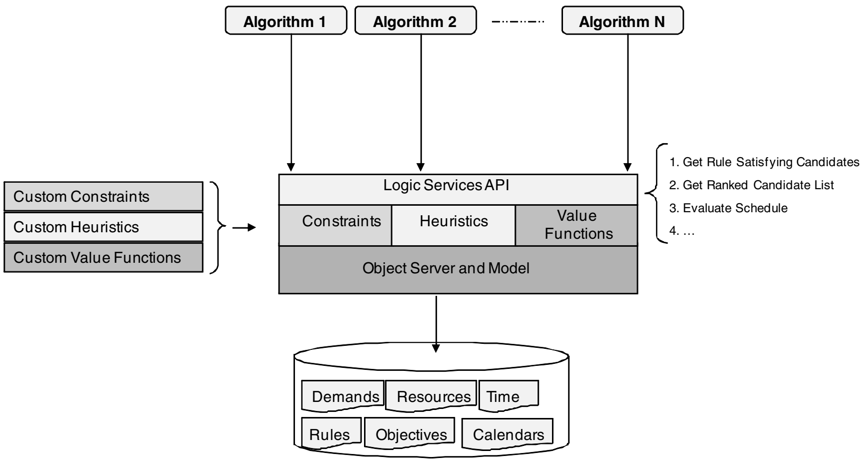


Fig. 1. A schematic view of the universal scheduling framework

constraint-satisfying schedules as well as optimizing these schedules. A few features of the object server deserve to be specifically mentioned:

- Creating, setting property values of, storing, querying and destroying any W-6 object.
- Maintaining certain structural invariants of W-6 objects, e.g. the difference between assignment's start and finish always needs to equal its duration.
- Triggering events before or after the methods are executed, in order to allow the implementation of custom business rules, dynamic property calculation and communication with other information systems.

Constraint-propagation engine

Constraints, both built-in and externally defined, are implemented in a constraint-propagation system tailored specifically for field service optimization problems. This system also includes propagation of temporal constraints. Constraint propagation generally works by successively eliminating possibilities that have become invalidated by the scheduling decisions that were made (usually by the currently active algorithm). For example, if a demand was assigned to a resource, other demands that can't fit into the remaining availability will be eliminated as candidates for that resource.

By providing a generic constraint-propagation engine and by allowing custom constraints to be incorporated, the framework essentially allows any field service problem to be modeled, no matter how exotic its constraints. Furthermore, the algorithms described in this chapter are generic and independent of the specifics of the constraints for each problem instance. Without this decoupling, separate algorithms would need to be developed for different combinations of problem characteristics, leading to an exponential explosion in number of required algorithms.

Ranking and heuristics

Once constraint-satisfying assignment candidates have been produced, they become subject to the scheduling decisions by the algorithm. Most algorithms include basic steps in which they need to construct or modify a solution by creating a new assignment and binding some or all of its slots to concrete values – e.g. selecting a specific resource for the `R` part of an assignment. Due to constraint propagation, the allowable values for selection depend on static problem data as well as on the solution to which the new assignment is being added. These selections are almost invariably guided by heuristics that are used to sort, weigh or otherwise rank the candidates. The framework provides a number of built-in heuristics that can be combined to model the scheduling decisions employed by dispatchers. It also allows the introduction of custom heuristics. These capabilities allow the algorithms deployed on top of the framework to transparently and consistently (across the algorithm stages) order the candidates in any order they need for their optimal performance. In the simplest case, a greedy schedule building algorithm is implemented by always selecting the highest ranked candidate, and the framework will automatically do the rest. More sophisticated algorithms have the same heuristics available to them, leading to consistent and fast design and testing of algorithmic ideas over any type of problem which may be modeled in W-6.

Evaluating the schedule

Once the schedule has been built, the algorithm usually has to calculate its value in order to know how to proceed. The framework allows the value to be composed of any number of business measures defined per assignment, assignment pair or globally. Again, if built-in measures do not provide sufficient expressiveness, the framework allows the incorporation of custom measures of any complexity. The introductory section describes a number of components of schedule value which are often encountered in practice.

Evaluation also provides information about specific parts of the problem which may indicate opportunities for improvements, e.g. demands which aren't served by the current solution or unused resource availability.

3 Genetic Algorithm

In this section we describe how a genetic algorithm (GA) was developed and applied on top of the framework described above. The framework enabled a drastic reduction in development effort, and the resulting GA was more flexible and able to efficiently handle large, complex, real-life problems.

3.1 Implementation

The general structure of genetic algorithms is well known (see [7], [2]), so we won't repeat it here. Instead, we will devote this space to implementation details specific to the field service optimization problems.

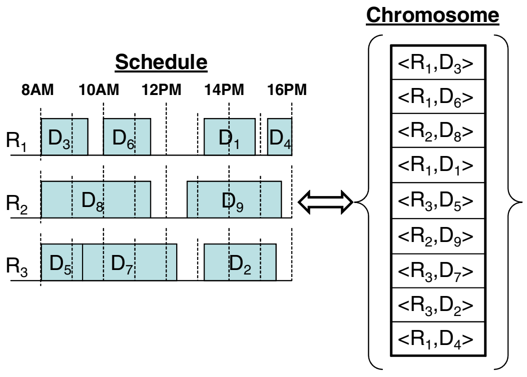


Fig. 2. Relationship of the chromosome structure to the schedule (the gaps between demands correspond to travel times and lunch breaks)

Chromosome structure

Chromosomes are the lifeblood of genetic algorithms. They don't just represent solutions, they also need to have a structure that allows evolutionary mechanisms – mutation, recombination, selection – to work effectively.

Choosing a chromosome representation which is equivalent to the solution representation (e.g. triplets specifying which resource performs which demand at what time) leads to an inefficient algorithm because of the low probability that a mutation or crossover will "squeeze" demands so that they are separated by the exact time required to travel between them. Instead, we chose to represent a chromosome as a "recipe" for generating a solution. This recipe takes the form of an ordered set of $\langle R_i, D_j \rangle$ pairs, specifying that resource i is assigned to demand j. We refer to these pairs as genes. The number of genes in a chromosome is equal to the number of demands in the problem instance. All genetic operators maintain the invariant that each demand appears exactly once in the chromosome. During the evaluation of the fitness function, a solution-generation engine scans the genes in the specified order, and attempts to schedule each demand at the earliest possible time after the other demands already assigned to that resource. This is done be relying on the framework, which considers travel time from this resource's previous assignment as well as any other constraints defined for this problem. This chromosome structure, and the process of generating a schedule from it, is shown in Figure 2.

The order of pairs in the chromosome is significant, since demands are appended only to the end of their resource's current route. For this reason, and for reasons related to the random nature of mutation and crossover operators, not all pairs correspond to permissible assignments. The scheduling framework detects this situation and reports it to the solution-generation engine. When such a situation occurs, the engine triggers a method attempting to locate another candidate allocation from the list supplied by the framework, and use it instead. This approach can be classified as belonging to the family of "repair strategy methods" (see [2, 11]). Note that in many problem instances it is difficult or impossible to find a schedule in which all demands are included. Therefore, it is common to find genes towards the end of the chromosome which can't be translated into valid assignments even after applying the repair strategy.

It is tempting to "fix" the chromosome by changing the gene which generated the infeasible assignment so that it reflects the assignment found by the repair method. However, our tests show this Lamarckian mechanism to exhibit lower performance, and we don't use it in the GA. This finding is in line with reports such as [17], where Lamarckian inheritance was found to have a higher tendency to converge to a local optimum, while utilizing the Baldwin effect had a better chance of finding the global optimum. We note that the repair method may be viewed as a limited Baldwin-inspired search, activated only when the genotype fails to create a valid phenotype.

The chosen chromosome representation has some redundancy. It is clear that many permutations of genes are equivalent – causing the engine to generate the same schedule – as long as the permutations preserve the ordering of the genes having the same resource. [2] We could remove this redundancy by always sorting the genes in some predefined order of resources without changing the schedule which the engine builds from each chromosome. We allow the redundancy, even though it increases the size of the search space, since this representation increases the probability for the operators that we use to find one-step moves which increase the solution quality.

Initial population

The creation of schedules that constitute the initial population is based on the framework's capabilities for solution construction. In order to generate a diverse initial population, the level of variation in ordering and selection of demands and resources should be high. We therefore employ a degree of randomization while still aiming at selecting heuristically good candidates. We have found that a good selection of heuristics in the construction of the initial population, combined with a suitable randomized selection of the candidate assignments as ranked by these heuristics, is important for the success of the GA as a method of finding high-quality solutions. In fact, we also developed a GA that searches for good heuristics, but space limitations preclude description of that GA.

Successive generations

Given a population, we use the following method to create the next generation:

1. *Segmentation by operators:* We divide the population into fixed-size segments, each of which is subject to a different operator. One of the segments is the "elite" segment, preserving the top few solutions from the previous generation. While this can lead into a premature "lock-in" of the population into a local optimum, this method has performed better in our studies than methods which allow the maximum quality to decline between generations.
   Each of the other segments is dedicated to one operator out of a variety of mutation and crossover (recombination) operators. We have found that the specific mix of

[2] For simplicity, we ignore the repair method in this paragraph.

operators is less important than the fact that they perform a wide variety of actions in order to more efficiently explore the search space. Some of these operators are described below.

2. *Choice of parents:* Each operator needs to select one or two parent chromosomes from the preceding generation. The operator acts on these parents to create a new chromosome in the current generation. After some experimentation, we decided on uniform probability for selecting any parent from the previous generation. In our experiments this performed better than other methods in the literature, such as competition or biasing the probability of selection towards higher-value solutions.
3. *Applying the operators:* Finally, we apply the relevant operators to the selected parents, which results in the new generation of chromosomes.

Crossover

There are numerous approaches to applying crossover and mutation to various scheduling problems in the literature (e.g. [14]). We find the most appealing choices to be variations of the "edge recombination" and the "order-based" operators (see chapters 21 and 22 in [2]). Both can be implemented for the field service scheduling problem, though they require substantial modification.

Our implementation has focused on order-based combinations, whose purpose is to create child chromosomes which preserve some of the ordering of demands in both parents. In our case, this affects the order in which demands will be sent into the solution-generation engine when a schedule is constructed from a chromosome. The re-ordering may change travel times as well as changing the validity of assignments generated by some genes.

A simplified example of this method (taken from [14]) is shown below. Suppose we want to recombine the following two parent chromosomes (in which each letter symbolizes a demand to be performed):

(Parent 1): `C D E G B A F`
(Parent 2): `F B G D E C A`

As the crossover begins, we randomly select a number of demands. Suppose the selected demands are B, C and G. To create the first child, we take the order of the three selected demands in parent 1 (which is C, G, B) and order those same demands in parent 2 (which currently has the order B, G, C) in this order while keeping them collectively in the same places in the chromosome:

$$\begin{tikzcd}
\text{(Parent 1):} & & \mathtt{C} \arrow[d] & \mathtt{G} \arrow[d] & & & \mathtt{B} \arrow[d] & \\
\text{(Parent 2):} & \mathtt{F} & \mathtt{B} & \mathtt{G} & \mathtt{D} & \mathtt{E} & \mathtt{C} & \mathtt{A}
\end{tikzcd}$$

We developed a few variations of crossover operators based on this idea.

Mutation

We used a mutation operator that randomly changes a number of demands and resources inside a chromosome. The changes are implemented in such a way that the resulting chromosome is valid – each demand appears in the chromosome exactly once.

Fitness function

A method of grading the solutions is part of the scheduling framework. We use the solution-generation engine for each chromosome in order to generate the schedule, and then use the framework's schedule-evaluation service to assign a quality grade to the chromosome.

3.2 Results

We have applied the GA to many real-world problems, often achieving better schedules, and in shorter times, than those achieved by other methods. Figure 3 shows a typical run of the GA. In this case, there are 620 demands and 88 resources with widely varying skills and locations. The service organization knew that it would be impossible to respond to all the demands within the specified SLA (Service Level Agreement), so some demands had to be scheduled later than their time window permits. Each such assignment is an SLA violation and deducts from the total value of the schedule. This is an example of a situation when a hard constraint needs to be converted into a soft one, namely one that may be violated with an accompanying reduction in solution's value. Here, as in many other places, the GA relies on the flexibility of the underlying framework.

Other cost factors which reduce the total value are overtime hours and total travel time. Since this problem instance allowed all demands to be scheduled (as long as we accept that some are scheduled later than their time window – being late is better than not being there at all!), the schedule's value has a large fixed positive component: the total value of all the scheduled demands.

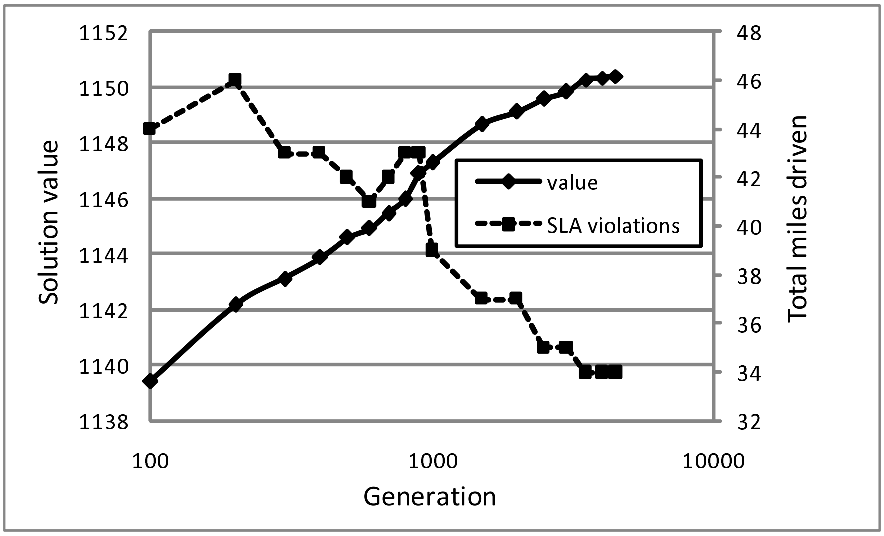


Fig. 3. Typical GA run, showing schedule values and numbers of SLA violations for the best solution in each generation

As we can see in the figure, the semi-logarithmic plot of value against generation number is close to linearity. It is interesting to note that at two points the number of SLA violations increases as the search finds solutions which compensate for this growth by reducing other cost components of the value function, such as mileage and overtime. However, such events are followed both times by a reversal, in which the number of SLA violations drops to its earlier level and even lower. This shows the power of the GA's exploration: having found a better value point, it uses it as an "anchor" from which to look for reductions in costs of SLA violations. After all, it makes sense to expect that as we find solutions which involve less driving there will also be some additional free time. We can then use some of this time to reorder the demands and reach the customers faster. The GA discovers and utilizes such opportunities without being designed with such domain understanding – just using blind, random evolution over suitably designed chromosomes.

4 Ant Colony Optimization

Ant colony optimization (ACO - see [3]) is a family of algorithms inspired by the way ants, which are relatively simple creatures following relatively simple behavioral patterns, exhibit group behavior of great complexity, be it building anthills, constructing live bridges or optimizing the path between a food source and the nest. The mechanism of communication are pheromones deposited by ants according to the circumstances in which they find themselves. These pheromones in turn directly influence the behavior of other ants that happen to find themselves in the same place, which leads to reinforcement of (for the group) positive behaviors and rejection of negative ones. Ant colony optimization algorithms are essentially weighted graph traversal algorithms that take this idea, implement it in software, and optionally enhance it with local search which, as also reported in prior research (see [6] for a VRPTW context), often improves ACO performance. Some local search operators for VRPTW are described in [9], and may be adapted and extended for the field service problem.

4.1 Pheromone Tables

In our pheromone tables, arc (A, B) represents the attraction associated with the scheduling of demand B immediately after demand A. There are also arcs connecting demands to the resource home-base. The pheromone tables are naturally asymmetrical: while $A \rightarrow B$ might be highly preferred, $B \rightarrow A$ might be very undesirable or even illegal due to the combination of time constraints and geography. We use an individual pheromone table for each resource. We found that using a single pheromone table for all resources is only appropriate when the resources are essentially identical (e.g. VRPTW), and that it leads to slower convergence and lower quality solutions if the resources in the problem instance at hand are heterogeneous (different skills, calendars, home-base locations). Let us give a couple of examples why individual pheromone tables are important:

- Let us have two resources with different skills. For the first one, it makes sense to schedule a remote low-value demand B after demand A because demand B is on the way from A to high-value demand C whose appointment starts too late for it to be scheduled straight after demand A without waiting. Therefore, the arc $A \rightarrow B$ will be marked with a strong pheromone trail and be strongly preferred. However, if the second resource doesn't have the skills to perform demand C but can still work on both A and B, then the arc $A \rightarrow B$ should have a very low value because it represents a lot of travel and lost time for very little reward.

- Let us have two resources with the same skills that can both be assigned to demands A, B and C from the previous example. Let the two resources have different working times, so that one of them is unable to fit all three tasks into the schedule because her work time finishes too soon. Then, again, for one the arc $A \rightarrow B$ should be strongly preferred, and for the other it should be discouraged.

4.2 Algorithm

The algorithm has two major components on two hierarchical levels – the "controller", which defines the high-level workflow of the algorithm and corresponds to the colony with its ability to exhibit complex behavior and learning, and the "builder", which corresponds to ants and builds specific solutions given the environment created by the colony. First, we present the high-level workflow, the "controller":

1. Create and initialize the global pheromone tables. This includes preliminary constraint propagation to determine which demands are candidates for which resources, and which demand combinations are possible in a legal schedule for each resource.
2. For each colony (the colonies can run in parallel) do the following:
   a) Create a local copy of the global pheromone tables.
   b) For each ant (ants within a colony run consecutively) do the following:
      i. Create a solution (see below).
      ii. Update the local pheromone table based on the following criteria:
         - the quality of the solution in terms of value; and
         - the number of scheduled demands.
3. Scan generated solutions for quality based on the following two criteria:
   - $v > (1 - c_1)v^*$, where v is the value of the generated solution, v^* is the best value found so far, and c_1 is a small positive constant; or
   - $n > (1 - c_2)n^*$, where n is the number of demands in the schedule, n^* is the number of demands in the best schedule so far, and c_2 is a small positive constant.
4. Use thus identified "high-quality solutions" to update the global pheromone tables using the same procedure as in step 2(b)ii.
5. Go to step 2 unless stop criteria are met.

Several parts of this algorithm represent refinements of the generic formulation of the ACO to the problem at hand:

- We are using more than one ant colony, and differentiate between global and local pheromone tables. By doing this, we achieve two goals that we judged to be desirable for the algorithm:

  - Our algorithm is naturally parallel in a sense that the colonies are completely independent of each other and can run in different processes, on different processors or even on different computers.
  - By allowing only the solutions of high quality to update the global pheromone table, we intensify exploration in the vicinity of good quality solutions and accelerate convergence. Of course, there is a price we pay for this – we quickly narrow down the search and potentially miss other promising areas. Still, when dealing with large problems (or many smaller ones) in a limited time, this is often a price one needs to be prepared to pay.
- We update the local pheromone tables after each ant. This again speeds up the convergence at the expense of the breadth of search.
- We don't only use the objective function to update the pheromone table, we use an additional heuristic measure which is the number of demands in the schedule. It turns out that this is quite a strong heuristic because the correlation between the number of demands in the schedule and the quality of the solution is very strong. This makes sense, since more tasks indicates better packing within available time and less travel.

Each solution represents routes for all resources. Let us present the algorithm for the "builder" part of the algorithm (a single ant traversal):

1. Choose a random resource. If all resources have already been chosen, go to step 3.
2. For the chosen resource, repeat:
   a) With probability p_b (0.15 in our case), stop with the current resource and go to step 1.
   b) Probabilistically choose the next demand (or resource's home-base) to append to the end of the schedule based on the following criteria:
      - pheromone level associated with the transition;
      - proximity to the demand (taking into account the latest time at which the work on the demand can start); and
      - level of deprivation for the demand (see below).

      The candidate demands are pre-selected by the framework so they can be scheduled at this point of schedule building in a constraint-satisfying way.
   c) If home-base was chosen or no candidate demands remain then go to step 1.
3. Randomize the order of resources for the next step.
4. For each resource in turn, repeat:
   a) Stop with the current resource if there are no more legal options or if the resource has already chosen a home-base and thus ended its schedule.
   b) Probabilistically choose the next demand (or resource's home-base) to append to the end of the schedule based on the same criteria as in the first stage.
5. If local search is being used then we use simple insertion – we insert deprived demands into the schedule in a greedy (i.e. most value increasing) way.

The reason that the resources are selected randomly for schedule building and that there is a possibility for schedule building to be interrupted is to somewhat balance the opportunities for all resources. If we kept the same order or built the full route for each resource every time, the probability would be quite high that the resources that

Table 1. Problem instances used in ACO tests

Name	Resources	Demands	Comment
small	12	46	2 skills, mostly same-site resources
large	88	620	several skills, geographically dispersed resources, lunch break

were scheduled first would always skim the top choices, and the rest would have to be satisfied with whatever remained. As a consequence, the colony would quickly get stuck with the local optima associated with good quality schedules for the first few resources and the schedules of very inferior quality for the rest, while huge parts of search space hiding potentially much better and more balanced solutions would remain unexplored.

The level of deprivation for each demand depends on how often a demand is left out of solutions created by the colony. If a demand often remains unscheduled, then the arcs leading towards it will be prioritized during subsequent runs. The levels of deprivation for all demands are reset when a new colony is started.

In this section, we present the results of using an ant colony optimization algorithm on two instances of the field service optimization problem. The summary of characteristics of the two instances are presented in Table 1.

The ACO results for both instances are shown in figure 4 and figure 5. In both cases we tried four variants of the algorithm based on two algorithm elements:

- *Individual or shared pheromone tables.* The first variant has a separate pheromone table for each resource, while the other has a single pheromone table for all resources. We discuss the reasons why individual pheromone tables are likely to lead to better results in section 4.1. On the other hand, shared pheromone tables lead to quicker iterations and lower memory consumption.
- *Local search or no local search.* In our case, a simple insertion algorithm was used in the local search step – the deprived tasks were taken one by one, and inserted in the most improving place in the schedule (or, more commonly, left unscheduled). The variant with local search could achieve better solution quality with the same pheromone table, while the one without had shorter iterations and could therefore perform more of them.

4.3 Results

Small instance

This small instance is one of the most commonly used within our suite of test data for the field service problem. The best result obtained so far for this instance was generated by the ACO algorithm described here.

Let us take a closer look at figure 4. The figure shows results of a single run of all four algorithm variants on the small instance – each point represents the moment in time that a new best solution was found (and its value).

We see two very different kinds of improvements. One is incremental and barely noticeable visually. These are improvements stemming from rerouting, reordering or

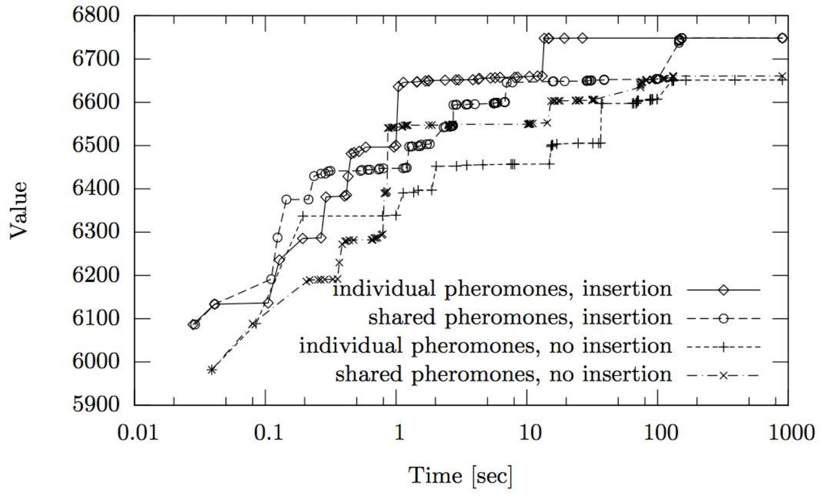


Fig. 4. ACO results for the small instance

replacing one demand with another. The other one is an improvement in which the total number of scheduled demands has been increased. Since typical demand revenue is one to two orders of magnitude greater than typical travel cost, this is expected.

As expected, the two algorithms with local search start with a greater number of assignments and therefore have an immediate advantage. This advantage is maintained throughout most of the run, with a few isolated crossings where one of the algorithms without local search briefly takes over. Local search turns out to be much more important than the individual pheromones for this instance.

In fact, with or without local search the final solution's quality produced with individual pheromones is practically identical to the one with shared pheromones (and happens to be even slightly better in the case of no local search). We can explain this by a very small variety among the resources in this instance – two calendars, three skill sets, only three home-bases, so that five resources are identical in all their attributes. The results seem to be inconclusive with regards to whether using individual pheromone tables is actually useful in this case, since they are associated with (mostly) quicker convergence in algorithm variants with local search, and with slower convergence in those without.

Large instance

Figure 5 shows the results of all four algorithm variants on the large instance. Since this is a more realistic and more complex instance of the field service scheduling problem, we also observe different behavior and draw different conclusions about different algorithm variants.

Initially, we notice that the two instances with local search stay close to each other and are doing better than the two instances without local search, which also remain close together at this stage. This is the time during which the pheromone tables are still largely undeveloped, and the difference between individual and shared pheromone tables doesn't matter that much.

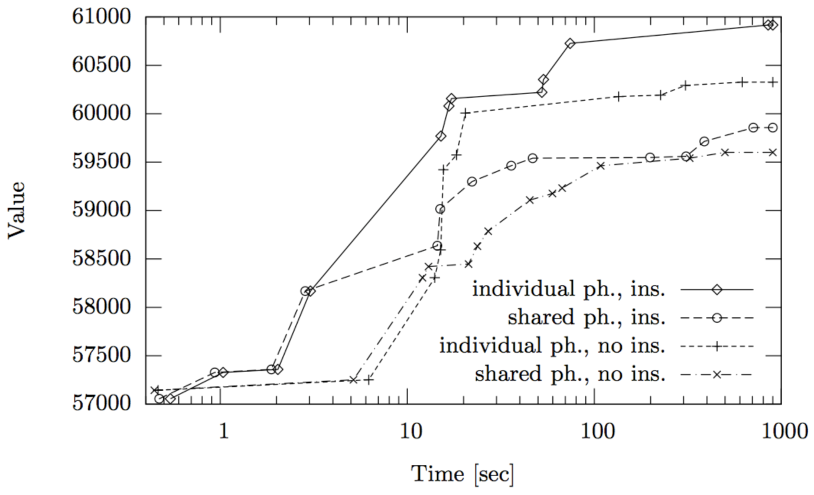


Fig. 5. ACO results for the large instance

However, at between 10 and 11 seconds the situation changes. The ACO with individual pheromone tables but without local search overtakes the one with a shared pheromone table and with local search, and never falls below it again. From that point onwards, the four algorithm variants stay at the same quality ranking. With this more typical instance, the diversity of resources has a dominant role in determining which algorithm variant will do better. Individual pheromone tables perform better than shared, and, given the same pheromone structure, local search is better than no local search.

5 Hybridizing GRASP with ACO

Encouraged by the success achieved by the ACO, we also tried to develop a hybrid algorithm that would enable a GRASP algorithm to take advantage of the learning capabilities of ACO. This is still work in progress, so the presented results are preliminary and should be taken as only indicative of things to come.

The Greedy-Randomized Adaptive Search Procedure (GRASP - see [4]) implementation that we used in this experiment is an optimization method that works in two stages:

1. A solution is constructed using a constructive algorithm in which the decisions are randomized at each stage, with greedier moves (the ones with more positive immediate effect on the objective function) having a greater weight. The specific method in which the moves are weighted is a non-trivial component of the implementation. This stage is called a randomized greedy stage.
2. Local variation operators are used to explore the solutions's local neighborhood in the search space, resulting in ever-increasing changes to the original solution. This stage is called an adaptive stage.

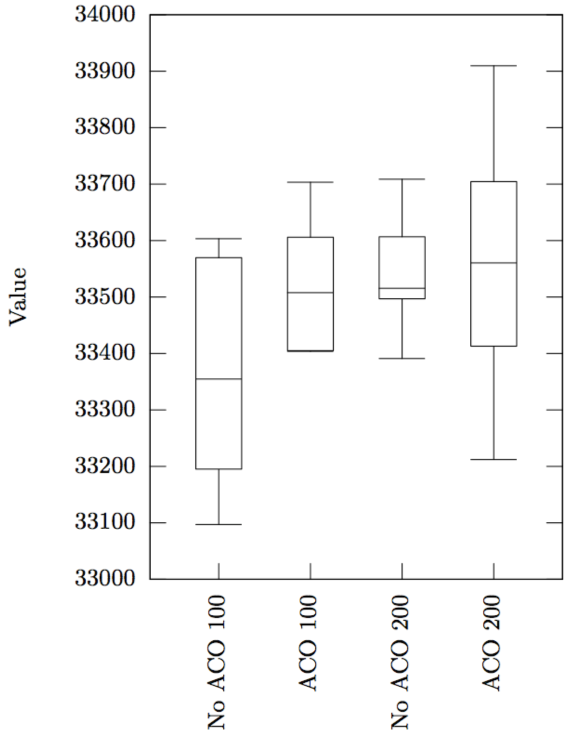


Fig. 6. Box-and-whiskers diagram for results of GRASP and GRASP with ACO after 100 and 200 second runs. Each of the four parts for each algorithm variant and run time represents a quartile over all the runs on the same instance.

Each stage has its own control loop and termination criteria, which need to be adjusted to the problem at hand and fine-tuned during algorithm development.

The main idea of the hybrid algorithm that we are proposing here is to use the ACO pheromones as an additional component of "greediness" during both stages of GRASP. The pheromones are updated after the end of each outer iteration based on the best solution found in each try. Thus, the pheromones collected in previous runs affect the probability distribution over moves in succeeding runs. The intent is that the pheromones will store some global information about the quality of moves, and therefore add a global component to the exclusively local considerations of traditional GRASP. This may be seen as a way of adding learning to a randomized search framework, while controlling its effect by including some forgetting (pheromone evaporation) as well. This algorithm may be compared to the combination of genetic and cultural evolution in human (and some non-human) societies [8].

Our preliminary results are illustrated in figure 6 and certainly show some promise. The diagram depicts all four quartiles over a number of runs on the same problem instance using both two algorithm variants (pure GRASP and GRASP with ACO) over

100[3] and 200[4] seconds. In the shorter run, ACO shows a definite improvement in all four quartiles over pure GRASP. The results over the longer run are a bit more ambiguous: while we notice a quite dramatic improvement in best and a bit less dramatic improvement in median result, we also notice a general increase in spread and therefore a decrease in consistency. Still, given that these are still preliminary results and that a lot of work remains to be done to make the hybrid algorithm more robust, we are greatly encouraged by what we saw and think the approach definitely shows some promise.

6 Human-oriented Schedule Generation

People who use schedule optimization systems include the dispatchers – who view and manipulate parts of the optimized schedule, and technicians – who perform their tasks according to the final schedule. For the application to succeed, these users must accept and trust the software's decisions. Yet, there are several reasons why a purely mathematical approach, striving to get as close as possible to the global optimum, can be perceived by users as failing to deliver a satisfying solution.

One reason for such perceptions is the fact that not all people in the organization agree on the components of the value function, and on the weights which should be assigned to each component. As pointed out above, real-world schedule optimization involves balancing many types of objectives. Some of these objectives may be ranked highly by the regional manager, while the technicians would prefer other objectives to rank more prominently, and the financial staff request raising the weight of still other objectives. A discussion of methods to establish the components and weights of a value function which everyone can agree upon is out of the scope of this chapter, but in this section we present an idea that, in our opinion, is one way to address the collisions of different interests.

Another reason for perceiving the schedule as sub-optimal is the tension between local and global views. A typical example occurs when a technician is driving from area A to area B, and sees another technician driving in the opposite direction - from area B to area A. Both technicians may think that a better schedule would have been to exchange tasks between them, so that the technician who has just completed a task in area A would remain there to start the next task in that area, and similarly both tasks in area B should be performed by the other technician. According to this view, the schedule which was assigned to them leads to longer and more expensive travel while reducing productivity and "wrench time" (time spent on the actual work). In practice, we almost always find that such a "strange" schedule is actually better: for example, even if both technicians have the required skills for these tasks, there may be yet another task waiting in area B which requires the first technician, so this technician would have had to drive to area B anyway, and it's just a matter of finding the best time to do it. There are many other possible reasons that justify this perceived "waste of time", but they all have something in common – they are invisible to the technicians that do not see the whole picture.

[3] With statistical significance of the observed difference at $p < 0.05$ level.
[4] With statistical significance of the observed variance at $p < 0.01$ level.

We have found that while it is possible to explain the rationale behind such scheduling decisions so that the technicians at least outwardly accept that they were given a good schedule, it is also possible to significantly reduce such complaints by adding a few mechanisms into the optimization process. One way is to introduce low-weight components into the value function or the heuristic that slightly reduce the value of solutions which give rise to such complaints. When this is not possible, we can add a "post-processing" phase to the optimization process that identifies such occurrences and uses local search to find friendlier alternatives in the solution's neighborhood in the search space, thus hopefully avoiding the problem while still maintaining most of the solution's value. Even though some service organizations are perfectly willing to accept a modest decrease in the solution value in order to balance such local points of view with global considerations, we have found that these mechanisms typically do not noticeably reduce the value of generated solutions (as expressed by the global value function), and only add a modest increment to the execution time. The same approach can be used to address tensions between solution characteristics given different priorities by different parts of the organization: locally searching for a solution whose value is very close to the best solution found, while avoiding situations where one objective seems to have taken over without any consideration for the other objectives.

7 Conclusions

In this chapter, we describe a wide variety of VRPTW-like optimization problems encountered in field service scheduling, and introduce a conceptual framework for categorizing these problems by the relevant attributes of major entities (e.g. demands, resources, customers, equipment), by the constraints defining acceptable solutions, and by the components of the value function which together define the quality of solutions. We also describe a framework for modeling problems of this family and for separating search algorithms from the services required for efficient constraint propagation and for building and evaluating partial solutions. These conceptual and algorithmic frameworks have enabled us to build, evaluate and hybridize many different search algorithms, including approaches from the fields of mathematical programming, clustering, temporal constraints, local search, and population search such as GA and ACO.

We note that the algorithms presented here are not entirely robust: that is, a few parameters need to be determined according to the problem's characteristics. They include, for example, the size of the population and the number of crossover operations in each iteration for the GA algorithm, and probability and pheromone-update parameter values for the ACO algorithm. However, since most heuristic approaches require some parameter setting as well, this need for fine-tuning is not a limitation that is unique to the approaches described here.

A workable field service optimization system needs quite a few additional capabilities which affect its design and applicability. Among these, we mention **scalability** – the ability to handle problems beyond the abilities of a single computer while maintaining the visibility of the "best-known" solution across all participating computers; **real-time optimization** – the ability to find good solutions immediately after new information is received, and to continue optimization concurrently with other updates and with actions by the human users which change the schedule; and **optimization across multiple time horizons** - the ability to plan ahead for the next week based on a statistical prediction of demand (when no tasks are yet known) and for next month based, additionally, on statistical prediction of capacity (since vacations are not yet known), while allowing the interchange of information and decisions between the different planning processes.

The practice of developing and extending commercial optimization software bears significant similarities with, as well as significant differences from, academic research. We gratefully acknowledge our debt to the academic research which contributed many of the methods we use, and our debt to the cooperation of the service organizations who look to these mathematical and algorithmic techniques as a tool to get the job done – to deliver service effectively and efficiently.

7.1 Remarks

The authors would like to stress that the contents of this chapter are not a description of the algorithms used by ClickSoftware Ltd., the commercial software vendor with which all authors are or have been associated. ClickSoftware's algorithms are proprietary and involve a combination of methods and building blocks. Our intent here is to report some of the research we've performed in order to improve our understanding of the problem and of potential algorithms, leveraging our access to real-world problem instances and to comments made by business organizations that evaluated the solutions generated during these experiments. We gratefully acknowledge the help, support and contributions which our research received from ClickSoftware management and from our colleagues.

References

1. Beniaminy, I., Nutov, Z., Ovadia, M.: Approximating interval scheduling problems with bounded profits. In: Proceeding ESA 2007, Eilat, Israel, pp. 487–497 (2007)
2. Davis, L.: Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York (1991)
3. Dorigo, M., Stutzle, T.: Ant Colony Optimization (Bradford Books). MIT Press, Cambridge (2004)
4. Feo, T., Resende, M.: Greedy randomized adaptive search procedures (1995)
5. Fleischer, L., Goemans, M.X., Mirrokni, V.S., Sviridenko, M.: Tight approximation algorithms for maximum general assignment problems. In: SODA 2006: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 611–620 (2006)
6. Gambardella, L.C., Taillard, E., Agazzi, G.: MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. Technical report, IDSIA, Lugano, Switzerland (1999)
7. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional, Reading (1989)
8. Jablonka, E., Lamb, M.J. (eds.): Evolution in Four Dimensions: Genetic, Epigenetic, Behavioral, and Symbolic Variation in the History of Life (Life and Mind: Philosophical Issues in Biology and Psychology). MIT Press, Cambridge (2006)
9. Larsen, J.: Parallelization of the Vehicle Routing Problem with Time Windows. PhD thesis, Department of Mathematical Modeling, Technical University of Denmark (1999)

10. Louis, S.J., Yin, X., Yuan, Z.Y.: Multiple vehicle routing with time windows using genetic algorithms. In: Angeline, P.J., Michalewicz, Z., Schoenauer, M., Yao, X., Zalzala, A. (eds.) Proceedings of the Congress on Evolutionary Computation, Mayflower Hotel, Washington D.C, vol. 3, pp. 1804–1808. IEEE Press, Los Alamitos (1999)
11. Mitchell, G.G., O'Donoghue, D., Barnes, D., McCarville, M.: GeneRepair - a repair operator for genetic algorithms. In: Rylander, B. (ed.) Genetic and Evolutionary Computation Conference Late Breaking Papers, Chicago, USA, July 12–16, pp. 235–239 (2003)
12. Nutov, Z., Beniaminy, I., Yuster, R.: A (1-1/e)-approximation algorithm for the generalized assignment problem. Oper. Res. Lett. 34(3), 283–288 (2006)
13. Spieksma, F.: On the approximabilty of an interval scheduling problem. Journal of Scheduling 2, 215–227 (1999)
14. Stein, R., Dhar, V.: Satisfying customers: Intelligently scheduling high volume service requests. AI Expert 12, 20–27 (1994)
15. Tan, K.C., Lee, L.H., Zhu, K.Q., Ou, K.: Heuristic methods for vehicle routing problem with time windows. Artificial Intelligence in Engineering 15(3), 281–295 (2001)
16. Weigel, D., Cao, B.: Applying GIS and OR techniques to solve Sears technician-dispatching and home delivery problems. Interfaces 29(1), 113–130 (1999)
17. Whitley, D.L., Gordon, V.S., Mathias, K.E.: Lamarckian evolution, the baldwin effect and function optimization. In: Davidor, Y., Schwefel, H.-P., Männer, R. (eds.) Parallel Problem Solving from Nature – PPSN III, pp. 6–15. Springer, Berlin (1994)

Author Index