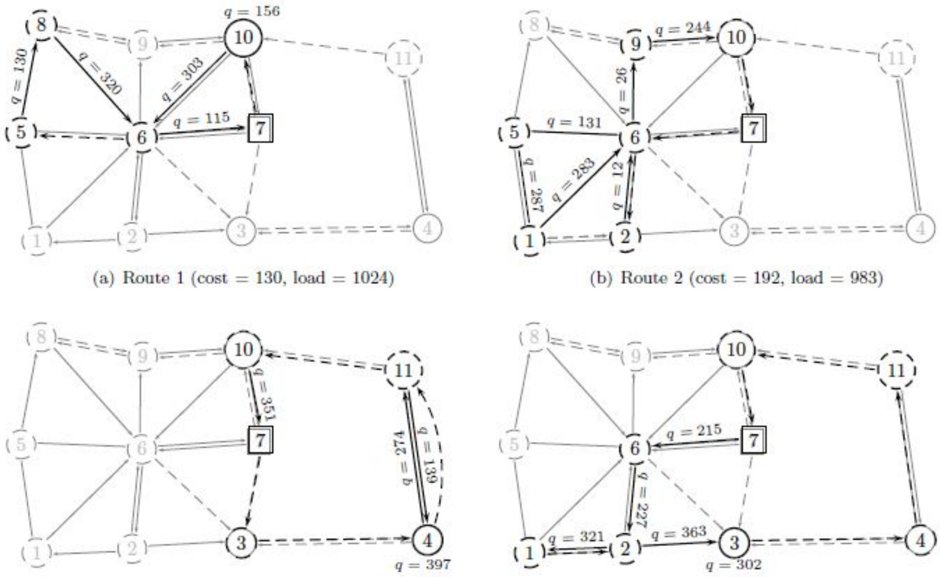
A26278- Unrestricted

Report

An Adaptive Iterated Local Search for the Mixed Capacitated General Routing Problem

Working paper

Author(s)
Mauro Dell'Amico, University of Modena and Reggio Emilia
Geir Hasle, SINTEF
José Carlos Díaz Díaz, University of Modena and Reggio Emilia
Manuel Iori, University of Modena and Reggio Emilia



(a) Route 1 (cost = 130, load = 1024)

(b) Route 2 (cost = 192, load = 983)

(c) Route 3 (cost = 227, load = 1161)

(d) Route 4 (cost = 231, load = 1428)

SINTEF ICT
Applied Mathematics
2014-05-16

SINTEF IKT
SINTEF ICT
Address:
Postboks 124 Blindern
NO-0314 Oslo
NORWAY
Telephone:+47
Telefax:+47 22067350

Enterprise /VAT No:

Report

An Adaptive Iterated Local Search for the Mixed Capacitated General Routing Problem

KEYWORDS:
Vehicle Routing;
Arc Routing;
Mixed Capacitated General Routing Problem;
Node, Edge, and Arc Routing Problem;
Metaheuristics

VERSION
1

DATE
2014-05-16

AUTHOR(S)
Mauro Dell'Amico, University of Modena and Reggio Emilia
Geir Hasle, SINTEF
José Carlos Díaz Díaz, University of Modena and Reggio Emilia
Manuel Iori, University of Modena and Reggio Emilia

CLIENT(S)
Norges forskningsråd / eVita

CLIENT'S REF.
205298

PROJECT NO.
102002202

NUMBER OF PAGES/APPENDICES:
32 / 1

ABSTRACT
We study the Mixed Capacitated General Routing Problem (MCGRP) in which a fleet of capacitated vehicles has to serve a set of requests by traversing a mixed weighted graph. The requests may be located on nodes, edges, and arcs. The problem has theoretical interest because it is a generalization of the Capacitated Vehicle Routing Problem (CVRP), the Capacitated Arc Routing Problem (CARP), and the General Routing Problem (GRP). It is also of great practical interest since it is often a more accurate model for real world cases than its widely studied specializations, particularly for so-called street routing applications. Examples are urban-waste collection, snow removal, and newspaper delivery. We propose a new Iterated Local Search metaheuristic for the problem that also includes vital mechanisms from Adaptive Large Neighborhood Search combined with further intensification through local search. The method utilizes selected, tailored, and novel local search and large neighborhood search operators, as well as a new local search strategy. Computational experiments show that the proposed metaheuristic is highly effective on five published benchmarks for the MCGRP. The metaheuristic yields excellent results also on seven standard CARP datasets, and good results on four well-known CVRP benchmarks.

PREPARED BY
Geir Hasle

SIGNATURE
Geir Hasle

CHECKED BY
Tomas Eric Nordlander

SIGNATURE

APPROVED BY
Trond Runar Hagen

SIGNATURE
Trond Hagen

REPORT NO.	ISBN	CLASSIFICATION	CLASSIFICATION THIS PAGE
A26278	.978-82-14-05361-6	Unrestricted	Unrestricted

SINTEF IKT
SINTEF ICT

Address:
Postboks 124 Blindern
NO-0314 Oslo
NORWAY

Telephone:+47
Telefax:+47 22067350

Enterprise /VAT No:

KEYWORDS:
Vehicle Routing;
Arc Routing;
Mixed Capacitated General Routing Problem;
Node, Edge, and Arc Routing Problem;
Metaheuristics

Report

An Adaptive Iterated Local Search for the Mixed Capacitated General Routing Problem

VERSION
1

DATE
2014-05-16

AUTHOR(S)
Mauro Dell'Amico, University of Modena and Reggio Emilia
Geir Hasle, SINTEF
José Carlos Díaz Díaz, University of Modena and Reggio Emilia
Manuel Iori, University of Modena and Reggio Emilia

CLIENT(S)
Norges forskningsråd / eVita

CLIENT'S REF.
205298

PROJECT NO.
102002202

NUMBER OF PAGES/APPENDICES:
32 / 1

ABSTRACT

We study the Mixed Capacitated General Routing Problem (MCGRP) in which a fleet of capacitated vehicles has to serve a set of requests by traversing a mixed weighted graph. The requests may be located on nodes, edges, and arcs. The problem has theoretical interest because it is a generalization of the Capacitated Vehicle Routing Problem (CVRP), the Capacitated Arc Routing Problem (CARP), and the General Routing Problem (GRP). It is also of great practical interest since it is often a more accurate model for real world cases than its widely studied specializations, particularly for so-called street routing applications. Examples are urban-waste collection, snow removal, and newspaper delivery. We propose a new Iterated Local Search metaheuristic for the problem that also includes vital mechanisms from Adaptive Large Neighborhood Search combined with further intensification through local search. The method utilizes selected, tailored, and novel local search and large neighborhood search operators, as well as a new local search strategy. Computational experiments show that the proposed metaheuristic is highly effective on five published benchmarks for the MCGRP. The metaheuristic yields excellent results also on seven standard CARP datasets, and good results on four well-known CVRP benchmarks.

PREPARED BY
Geir Hasle

SIGNATURE

CHECKED BY
Tomas Eric Nordlander

SIGNATURE

APPROVED BY
Trond Runar Hagen

SIGNATURE

REPORT NO.	**ISBN**	**CLASSIFICATION**	**CLASSIFICATION THIS PAGE**
A26278	.978-82-14-05361-6	Unrestricted	Unrestricted

Document history

VERSION	DATE	VERSION DESCRIPTION
1	2014-05-16	First revision of manuscript submitted 2013-06-25.

An Adaptive Iterated Local Search for the Mixed Capacitated General Routing Problem

Mauro Dell'Amico[(1)], José Carlos Díaz Díaz[(1)], Geir Hasle[(2)], Manuel Iori[(1)]

[(1)] Department of Science and Methods for Engineering,

University of Modena and Reggio Emilia,

Via Amendola 2, 42122 Reggio Emilia, Italy

{mauro.dellamico; jose.diaz; manuel.iori}@unimore.it

[(2)] Department of Applied Mathematics,

SINTEF ICT

P.O. Box 124 Blindern, NO-0314 Oslo, Norway

Geir.Hasle@sintef.no

Abstract

We study the Mixed Capacitated General Routing Problem (MCGRP) in which a fleet of capacitated vehicles has to serve a set of requests by traversing a mixed weighted graph. The requests may be located on nodes, edges, and arcs. The problem has theoretical interest because it is a generalization of the Capacitated Vehicle Routing Problem (CVRP), the Capacitated Arc Routing Problem (CARP), and the General Routing Problem (GRP). It is also of great practical interest since it is often a more accurate model for real world cases than its widely studied specializations, particularly for so-called street routing applications. Examples are urban-waste collection, snow removal, and newspaper delivery. We propose a new Iterated Local Search metaheuristic for the problem that also includes vital mechanisms from Adaptive Large Neighborhood Search combined with further intensification through local search. The method utilizes selected, tailored, and novel local search and large neighborhood search operators, as well as a new local search strategy. Computational experiments show that the proposed metaheuristic is highly effective on five published benchmarks for the MCGRP. The metaheuristic yields excellent results also on seven standard CARP datasets, and good results on four well-known CVRP benchmarks.

KEYWORDS: Vehicle Routing; Arc Routing; Mixed Capacitated General Routing Problem; Node, Edge, and Arc Routing Problem; Metaheuristics

Introduction

Two of the most important optimization problems in freight transportation and logistics are the *Capacitated Vehicle Routing Problem* (CVRP) and the *Capacitated Arc Routing Problem* (CARP).

In the CVRP, a set of customers with known demands must be served by a fleet of identical capacitated vehicles stationed at a central depot. Requests with given demand size are located on the vertices of a graph, and the aim is to route the vehicles along the graph to satisfy all requests with minimum routing cost, obeying vehicle capacity. The graph may be either directed or undirected, and the costs are assigned to links (edges/arcs). The problem has been widely studied (especially in its undirected version) because of the large number of real-world applications it models, including distribution of gasoline, beverage, and food, and collection of solid waste. We refer the interested reader to the books by Toth and Vigo (2002a) and Golden et al. (2008).

In the CARP we are still given a weighted undirected/directed graph, but in this case, requests of given size are located on a subset of links. A fleet of identical vehicles, all based at a central depot and having a fixed capacity, is available for serving the requests. The problem is to route vehicles along the graph in a capacity feasible way to serve all requests with minimum routing cost. Also the CARP has been widely studied, because it captures the essence of a wide range of real-world applications, including street sweeping, winter gritting, and snow clearing. In many cases the CARP is also a good model for postal delivery, newspaper delivery, and household waste collection. We refer the interested reader to the survey by Wøhlk (2008) and the annotated bibliography by Corberán and Prins (2010).

In the literature, there has been a tendency to categorize applications as being either a case of node routing, or a case of arc routing. There are, however, important real-world problems whose essential characteristics cannot be captured neither by the CVRP nor by the CARP, as there is a mixture of requests located on nodes and requests located on links. Prins and Bouchenoua (2005) argue that in certain cases of urban-waste collection, most requests may be adequately modeled as located on streets, but some large point-based demands, located for example at schools or hospitals, are better modeled by the use of vertices. In subscription newspaper delivery, requests are basically located in points, but in dense urban or suburban residential areas a CARP model based on the street network may be a good abstraction. In general, qualified abstraction and problem reduction for a CVRP instance through aggregation, for instance with heuristics based on the road network, will create an instance with requests located on nodes, edges, and arcs, see, e.g., Hasle et al. (2012).

To answer the challenges that are induced by these complex problems, several researchers have recently focused their attention on the so-called Mixed Capacitated General Routing Problem (MCGRP). In the MCGRP, requests are located on a subset of vertices, edges, and arcs of a given weighted mixed graph, and a fleet of identical capacitated vehicles based at a central depot is used to satisfy requests with minimum routing cost while adhering to capacity constraints. The MCGRP is able to model a continuum of mixed node and arc routing problems, and hence removes the sharp boundary that is often seen in the literature. As alluded to above, the problem has large practical interest, particularly for so-called street routing applications, see Bodin et al. (2003). The MCGRP is also of interest in combinatorial optimization, because it generalizes both the CVRP, the CARP and many other routing problems, as described in Section 2. Its resulting combinatorial complexity is, however, very high, and solving it to optimality is a difficult task even for moderate-size instances, see Bach et al. (2013) and Bosco et al. (2013).

In this paper, we propose a novel, hybrid metaheuristic, called Adaptive Iterated Local Search (AILS) for easy reference, to solve large-size instances of the MCGRP. It utilizes vital mechanisms from two classical trajectory-based metaheuristics: Iterated Local Search (ILS), see Lourenço et al. (2010), and Adaptive Large Neighborhood Search (ALNS), see Pisinger and Røpke (2007). We have combined these mechanisms in a new way, and introduced several new elements. Novel local search and large neighborhood search operators have been designed, and well-known operators have been tailored to the problem at hand. When ALNS finds solutions with a certain quality, they are further intensified by local search (LS). We have designed a new, aggressive LS strategy. In each iteration we explore the union of neighborhoods resulting from five operators, and try to execute all moves with positive savings. In effect, we execute all independent moves before generating a new neighborhood.

Our experimental study shows that the resulting algorithm is highly effective. For five MCGRP benchmarks consisting of 409 instances in total, AILS produces 402 best known solutions, 128 of which are new. 180 of the 402 solutions are proven optimal. One instance was closed for the first time by AILS. Notably, the AILS also achieves high quality computational results for heavily investigated special cases of the MCGRP, viz. four standard benchmarks for the CVRP, and seven standard benchmarks for the CARP.

The remainder of the paper is organized as follows. In Section 1 we formally describe the MCGRP. In Section 2 we give a survey of the most relevant results in the related literature. In Section 3 we propose our AILS metaheuristic for the MCGRP, and describe the key elements that make it computationally effective. In Section 4 we evaluate the algorithm by means of extensive computational tests, and in Section 5 we draw conclusions.

1 Problem Description

The MCGRP is defined on a weighted mixed graph $G = (N, E, A)$, where $N = \{1, 2, \ldots, n\}$ is the set of nodes, E the set of edges and A the set of arcs. Let c_{ij} denote the non-negative *traversal cost* associated with any link $(i, j) \in E \cup A$. The traversal cost, also known as deadheading cost, denotes the cost for traversing the link without servicing it. The traversal cost is 0 for all nodes.

Three subsets $N_r \subseteq N$, $E_r \subseteq E$ and $A_r \subseteq A$ define the *requests*, or *tasks*, i.e., the subsets of, respectively, nodes, edges, and arcs that have demand and must be serviced. Each request has a non-negative *service cost*, s_i for $i \in N_r$ and s_{ij} for $(i, j) \in E_r \cup A_r$, and a non-negative *demand*, q_i for $i \in N_r$ and q_{ij} for $(i, j) \in E_r \cup A_r$. Let $\tau = |N_r| + |E_r| + |A_r|$ be the total number of requests.

An unlimited fleet of identical vehicles, all having capacity Q, is used to service the requests. The fleet is located in a special node, called the *depot*. Each vehicle performs at most one *route*, that is, it starts from the depot, services a number of requests, and then returns to the depot. Deadheading via non-required links is usually necessary to reach the required ones. A route is *feasible* if the sum of serviced demands does not exceed the vehicle capacity.

The aim of the MCGRP is to define a set x of feasible routes for which every request, $i \in N_r$ and $(i, j) \in E_r \cup A_r$, is serviced exactly once, and the total cost $z(x)$ is a minimum. Note that the total service cost is constant over all feasible solutions, hence it is sufficient to minimize the sum of traversal costs.

An example of a MCGRP instance is given in Figure 1. Each node is depicted by a circle, drawn with a solid line if the node is a request, by a dashed line otherwise. Node 7 is the depot and is depicted by a square. Similarly, required links are drawn with a solid line, non-required links with a dashed line. The traversal costs are indicated. In this particular instance the traversal costs are symmetric, hence we give only one cost for each pair of arcs connecting the same two vertices. The vehicle capacity is 1437.

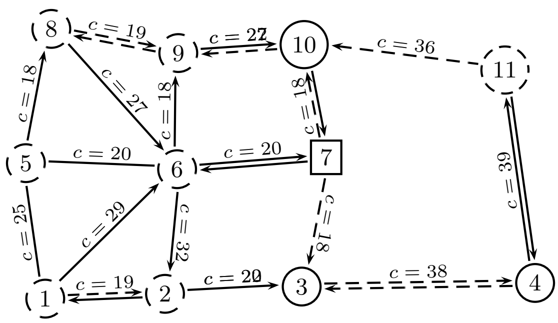
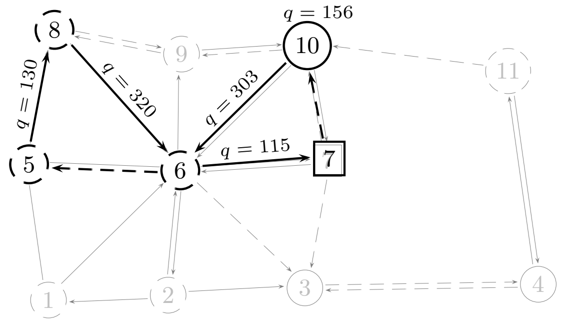


Figure 1: A MCGRP example: Instance `CBMix23`.

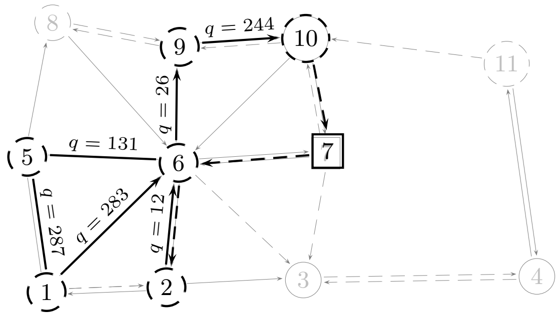
A solution for the instance in Figure 1 is illustrated in Figure 2. It consists of four routes, each presented in a separate sub-figure for the sake of clarity. In each sub-figure, the links with solid lines are serviced by the route, links with dashed lines indicate deadheading. We also indicate the demand for each serviced task. Note that Route 1 starts from the depot and visits, in sequence, nodes 10, 6, 5, and 8, then visits 6 again and returns to the depot. It services five tasks, namely 10, (10,6), (5,8), (8,6) and (6,7), with total demand (denoted *load* for short in the figure) equal to 1024. Note also that Route 2 travels three times through node 6, and Route 3 is forced to travel three times between nodes 4 and 11 to perform the two requests (4,11) and (11,4). The resulting solution value is 780, and its optimality was proven by Bosco et al. (2013).

2 Prior Work in the Area

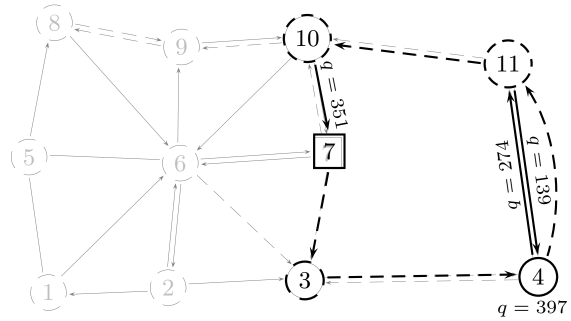
The MCGRP has also been called the Node, Edge, and Arc Routing Problem (NEARP) in the literature. As far as we know, Pandi and Muralidharan (1995) is the first paper that investigates the MCGRP. The authors studied a generalization with a heterogeneous fixed fleet, and a maximum route duration constraint. The resulting problem, denoted the *Capacitated General Routing Problem* (CGRP), was solved with a route-first-cluster-second heuristic. The algorithm was tested on random test instances inspired from curb-side waste collection in residential areas, and on random instances from the Capacitated Chinese Postman Problem



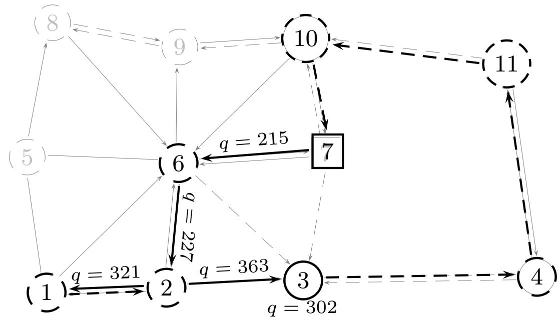
(a) Route 1 (cost = 130, load = 1024)



(b) Route 2 (cost = 192, load = 983)



(c) Route 3 (cost = 227, load = 1161)



(d) Route 4 (cost = 231, load = 1428)

Figure 2: A four-route (optimal) solution for `CBMix23`.

literature.

A few years later, Gutiérrez et al. (2002) studied the homogeneous fixed fleet version of the CGRP, and called it the *Capacitated General Routing Problem on Mixed Graphs* (CGRP-m). In other words, the CGRP-m is a MCGRP with a limited number of vehicles. They proposed an $O(n^3)$ heuristic and compared it with the heuristic by Pandi and Muralidharan (1995), obtaining favorable computational results on a benchmark of 28 instances with the number of vehicles between 2 and 4, and the number of required tasks between 6 and 21.

Prins and Bouchenoua (2005) introduced the Node, Edge, and Arc Routing Problem (NEARP) name for the problem and solved it by means of a memetic algorithm in which a population of solutions is evolved through a genetic process, and each new solution is post-optimized using five local search operators. The resulting algorithm was tested on benchmark instances from the CVRP and CARP literature, and on a new set of MCGRP instances denoted the `CBMix` benchmark. Kokubugata et al. (2007) developed a simulated annealing algorithm that makes use of three local search operators. They tested their algorithm on the CBMix instances and provided several new best known solutions. Recently, upper bounding procedures were discussed by Hasle et al. (2012), who obtained interesting computational results by running the commercial VRP solver Spider.

The first lower bounding procedure for the MCGRP was proposed by Bach et al. (2013). It was obtained by adapting a procedure originally developed for the CARP by Wøhlk (2006), based on the solution of a matching problem. The lower bound was tested on the CBMix benchmark, and on two new sets of MCGRP instances: the BHW benchmark based on well known instances from the CARP literature, and the DI-NEARP benchmark taken from real-world newspaper distribution cases in Norway.

Bosco et al. (2013) proposed the first integer programming formulation for the MCGRP, using three-index variables for nodes and two-index variables for edges and arcs. They extended some valid inequalities originally introduced for the CARP to the MCGRP, and embedded them into a branch-and-cut algorithm. This algorithm was tested on two new benchmarks called mggdb and mgval, each consisting of six sets, and totalling 342 instances. The mggdb benchmark was derived from the gdb undirected CARP instances. The mval mixed CARP dataset is the origin of the mgval benchmark. The authors considered only the instances involving at most seven vehicles in their experiments. They also tested their algorithm on the four smallest-size CBMix instances, providing two optimal solutions.

As mentioned in the introduction, the MCGRP generalizes a large number of optimization problems arising in transportation and logistics. A problem classification is presented in Figure 3. The classification is incomplete, because of the large number of variants addressed in the scientific literature. As depicted by the figure, the MCGRP directly generalizes:

- the *Capacitated Vehicle Routing Problem* (CVRP): $N_r = N$, $E_r = \emptyset$ and $A = \emptyset$;
- the *Capacitated Arc Routing Problem* (CARP): $N_r = \emptyset$ and $A = \emptyset$; and
- the *General Routing Problem* (GRP): one vehicle, $Q = +\infty$ and $A = \emptyset$.

The CVRP is one of the most widely studied problems in the combinatorial optimization literature. Recently, exact algorithms based on branch-and-cut-and-price techniques have been proposed by Baldacci et al. (2008) and Baldacci and Mingozzi (2009). Good-quality heuristic solutions have been obtained in the last years by, among others, Gröer et al. (2011) with local search and integer programming embedded into a parallel algorithm, and Vidal et al. (2012) with a hybrid genetic algorithm that can also take into consideration multiple depots or multiple periods. We also mention that there are works aimed at solving the CVRP on an asymmetric cost matrix. The literature on the problem, known as the *Asymmetric CVRP* (ACVRP), is described, e.g., in Toth and Vigo (2002b). Note that, since the CVRP is strongly $\mathcal{NP}$-hard, so is the MCGRP.

The CARP has also been widely studied in the literature. Recently, branch-and-cut-and-price algorithms have been presented by Bartolini et al. (2011) and Martinelli et al. (2011). Good-quality heuristic solutions have been obtained via Ant Colony Optimization by Santos et al. (2010). Despite the use of the term "arc" in its name, the CARP has been originally defined on an undirected graph. Works aimed at solving arc routing problems on directed graphs, and on more general mixed graphs, are described, e.g., in Corberán et al. (2006).

The GRP was introduced by Orloff (1974), to model the problem of collecting requests on nodes and edges of an undirected graph with a single uncapacitated vehicle. A good cutting plane algorithm was proposed by Corberán et al. (2001). Similar to the CVRP and the CARP, also the GRP has been extended to directed and mixed graphs, see, e.g., Corberán et al. (2005). Notably the GRP generalizes other combinatorial optimization problems, namely:

- the *Rural Postman Problem* (RPP): one uncapacitated vehicle, $A = \emptyset$, $N_r = \emptyset$;
- the *Chinese Postman Problem* (CPP): one uncapacitated vehicle, $A = \emptyset$, $N_r = \emptyset$, $E_r = E$;
- the *Steiner Graphical Travelling Salesman Problem* (SGTSP): one uncapacitated vehicle, $A = \emptyset$, $E_r = \emptyset$;
- the *Graphical Travelling Salesman Problem* (GTSP): one uncapacitated vehicle, $A = \emptyset$, $E_r = \emptyset$, $N_r = N$; and
- the *Travelling Salesman Problem* (TSP): one uncapacitated vehicle, $A = \emptyset$, $E_r = \emptyset$, $N_r = N$.

For the literature on the RPP, CPP, SGTSP, GTSP, TSP and their extensions to directed and mixed graphs, we refer the reader to Corberán et al. (2001), Gutin and Punnen (eds.) (2002), Corberán et al. (2007) and references therein.

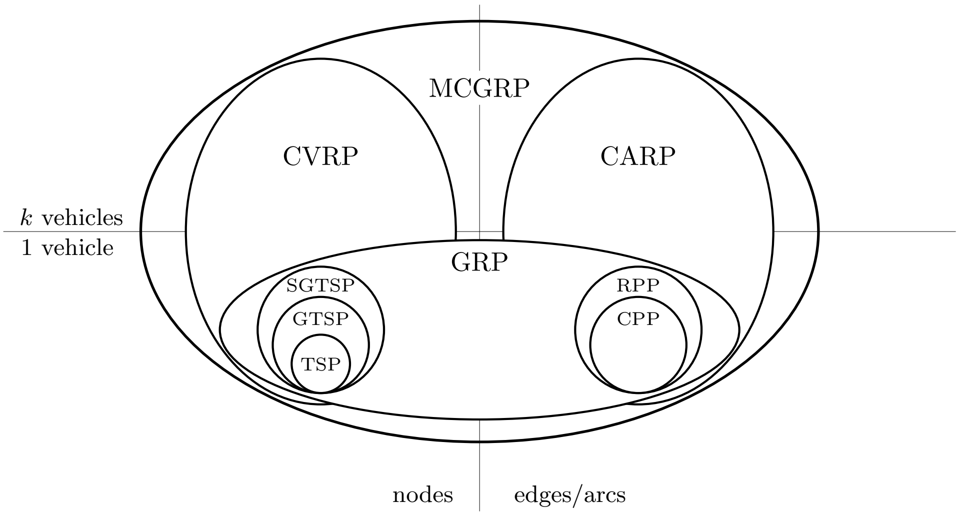


Figure 3: A graphical representation of a problem classification.

3 Adaptive Iterative Local Search

In this section we discuss the novel hybrid metaheuristic that we propose to search for high-quality solutions of MCGRP instances of realistic size in reasonable time. For easy reference, we call the algorithm *Adaptive Iterative Local Search* (AILS). Parts of AILS uses pseudo-random numbers, but we emphasize that it is a deterministic algorithm that will produce the same path in the search space given the same random seed.

First, we give a description of the overall design of AILS. Main goals are to ensure a good balance between intensification and diversification, and to avoid non-productive search efforts. To this end, we use the idea of *Iterated Local Search* (ILS) (see, e.g., Lourenço et al. (2010)) that mainly consists of improving a solution through a trajectory based intensification algorithm, and diversification through a perturbation method when intensification stagnates.

AILS combines intensification mechanisms that on their own have proven to be highly effective for a variety of discrete optimization problems, including many variants of the VRP, namely *Adaptive Large Neighborhood Search* (ALNS) proposed by Pisinger and Røpke (2007) and partially based on ideas from Shaw (1997), and deep intensification through *Local Search* (LS). Intensification is performed in *stages*, each consisting of a certain number of iterations.

In one iteration, ALNS destroys and repairs the current solution. The pair of *destructor* and *constructor* operators is probabilistically selected among alternative operators. A reinforcement learning technique is used to update the selection probabilities. Further details on our version of the ALNS are given in Section 3.2. It contains several innovations and non-standard mechanisms. If the solution resulting from a destroy/repair operation has good quality, it is further intensified through local search with five local search operators, and a new, aggressive move selection strategy. Details are given in Section 3.3

The main diversification mechanism of AILS is a major disruption – a “kick” – applied when a certain number of iterations have passed without acceptance of a new solution. The kick utilizes the random destructor and random constructor operators from the ALNS, see Section 3.2. It is followed by intensification through local search.

3.1 Structure of AILS

The overall structure of AILS, which is quite simple, is given in Algorithm 1. AILS find a first feasible solution with simple, fast heuristics, as described in Section 3.5. The initial solution is taken to a deep local optimum through an aggressive local search procedure. A main **repeat-until** loop performs Iterated Local Search until timeout. An *intensification stage*, implemented by the inner **for** loop, performs several iterations of ALNS and (possibly) subsequent local search is performed, and a local incumbent for the stage is maintained. When a stage is finished, a new one is started from the local incumbent of the previous one.

A kick is performed whenever a *stagnation* occurs, i.e., no new solution has been accepted for a certain number of iterations. The resulting solution is taken to a local optimum before a new intensification stage is started.

Given the overall design of AILS, we developed alternatives for the most important mechanisms. To select among alternatives, and to assess the contribution of algorithmic components, we performed comparative experiments on a selected subset of MCGRP instances, as described in 3.4. A detailed documentation of the final version of the algorithm is given in Algorithms 2 and 3.

Algorithm 1 Overall structure of Adaptive Iterative Local Search

```
function AILS(Instance)
    x_incumbent :=CONSTRUCT_INITIAL_SOLUTION(Instance)
    comment: Take the initial solution to a local optimum
    x_LocalIncumbent := x_incumbent :=LS(x_incumbent)
    comment: ILS iterative phase: repeats for several stages until a timeout occurs
    repeat
        x_current := x_BestThisStage := x_LocalIncumbent
        comment: Execute a stage of intensifying iterations until no improvement, then kick
        for i := 1 to IterPerStage do
            if NOT Stagnation then
                comment: Reset roulette probabilities regularly and build new tentative solution
                if ResetCriterion then RESET_ROULETTE_PROBABILITIES( )
                comment: Destroy and Repair with randomly selected Destructor and Constructor pair
                x_current :=ROULETTE_DESTROY_AND_REPAIR(x_current)
                comment: Intensify further with local search only if solution is below quality threshold
                if NOT QualityBelowThreshold(x_current) then continue
                comment: Cost not too far from the best solution of the stage, intensify with LS
                x_current := LS(x_current)
                comment: Check if new solution should be accepted
                if NOT Acceptable(x_current) then continue
                if z(x_current) < z(x_BestThisStage) then  UPDATE_BEST_AND_INCUMBENTS(x_current)
                if NewBest then break
            else
                comment: No progress for many iterations, make a major disruption
                x_LocalIncumbent :=RANDOM_DESTROY_AND_REPAIR(x_LocalIncumbent)
                x_current := x_LocalIncumbent :=LS(x_LocalIncumbent)
                if z(x_current) < z(x_BestThisStage) then  UPDATE_BEST_AND_INCUMBENTS(x_current)
                break
            end if
        end for
    until TIMEOUT( )
    return x_incumbent
end function
```

As will become evident in Section 4 below, AILS is competitive not only for MCGRP, but also for the CARP and CVRP special cases. In our view, the good, robust performance of AILS is due to:

- a careful choice of the best-performing LS and ALNS operators from the literature;
- an adaptation of high-performing operators to the MCGRP model;
- the introduction of a novel type of Destructor that utilizes the structure of the instance;
- a new combination of metaheuristic mechanisms that balances intensification and diversification, and

reduces futile search efforts;

- an aggressive LS strategy that fully utilizes information on promising moves
- a selection of main algorithmic components based on empirical investigation.

3.2 The Adaptive Large Neighborhood Search Component

To perform well, ALNS must utilize a varied repertoire of destructor and constructor operators, and a qualified mechanism for selecting the operators to employ in a given cycle. Our ALNS design introduces a novel *tree-destructor* that utilizes the graph structure of the instance at hand. Experiments (see Section 3.4) shows that the proposed tree-destructor is effective.

Self-adaptation in ALNS is typically achieved through a reinforcement learning mechanism that rewards operators that have been successful in past iterations. The Adaptive Large Neighborhood Search mechanism in AILS is a simplified version of the one proposed by Pisinger and Røpke (2007). An important difference is that the reinforcement learning mechanism is based on operator pairs rather than on single operators. Any time the destroy and repair mechanism is invoked, a destructor/constructor operator pair is randomly drawn, using roulette wheel selection. These operators are then used to remove and re-insert a randomly drawn number of tasks k in the interval $[1, k_{max}]$ in the current solution. A scores matrix π contains a measure for the effectiveness of each pair of Destructor and Constructor operators. It is used by the roulette wheel selection procedure. The score element π_{ij} is associated with the i-th Destructor and the j-th Constructor. The initial value for all elements is 1. The update procedure increments the value π_{ij} by 1 unit, whenever the i-th Destructor and the $j-$th Constructor have lead to a new current solution.

Another novel feature of our ALNS concerns the resetting of scores at regular intervals. Through experiments (see Section 3.4 for details) we observed that resetting deteriorated the performance of AILS for large size instances. For such instances, a small number of stages are performed. Thus, resetting will take place prematurely, and the learning effect will suffer. Hence, we introduced a problem size value beyond which we do not invoke resetting of ALNS operator scores.

For the AILS, we designed a repertoire of seven destructor operators, all parameterized by the number of tasks to remove:

1. **Random-Destructor**: k random tasks are selected and removed from the solution;

2. **Task-Destructors**: these are our extensions of the analogous operators used for CVRP and CARP.

   2.a **Node-Destructor**: if $k \leq |N_r|$, k random node tasks are removed from the solution, otherwise the **Random-Destructor** is used;

   2.b **Edge-Destructor**: if $k \leq |E_r|$, k random edge tasks are removed from the solution, otherwise the **Random-Destructor** is used;

   2.c **Arc-Destructor**: if $k \leq |A_r|$, k random arc tasks are removed from the solution, otherwise the **Random-Destructor** is used;

3. **Worst-Destructor**: we define the cost of removing a task t from the current solution x as $\Gamma(t, x) = z(x) - z_{-t}(x)$, where $z_{-t}(x)$ is the cost of the solution without task t. The operator removes the k tasks having the highest values of $\Gamma(t, x)$;

4. **Related-Destructor**: this operator was proposed by Shaw (1997, 1998). Its aim is to remove tasks that are somehow close one to one another. For the MCGRP, extending the original idea, we define the contiguity of two tasks r and t as:

$$\rho(r, t) = \beta \frac{c'_{rt}}{\max_{su} c'_{su}} + \gamma \frac{|q(r) - q(t)|}{\max_s q(s)} + \delta(r, t), \tag{1}$$

   where $\beta = 0.75$, $\gamma = 0.1$ as recommended in the literature, c'_{rt} is the minimum traversal cost between r and t, $q(t)$ is the demand of task t, and $\delta(r, t)$ takes value 1 if r and t are in the same route in the current solution, 0 otherwise.

5. **Tree-Destructor**: this is a new operator which is particularly effective for MCGRPs where the tasks are not all of the same kind (Node, Edge or Arc). It first randomly selects a root node, and then grows a tree from this root by using a breadth-first strategy. The growth is halted as soon as k tasks (of any kind) are encountered.

Three constructors were designed for AILS, as extensions of operators from the literature. They re-insert k removed tasks in the current solution, one at a time, according to a certain criterion. They iterate until all tasks have been re-inserted. The resulting solution is always feasible, although it may contain additional routes.

- **Random-Constructor**: Insert each task, one at a time, according to the order in which they have been removed from the solution by the Destructor, in a random position in the current set of routes. If no feasible position exists, Create a new route with only this task;

- **Greedy-Constructor**: In each iteration, the task with the minimal best insertion cost is inserted in its best position;

- **Regret-Constructor**: Compute for each task t its cheapest insertion cost, and its second cheapest insertion cost, and define its regret $r(t)$ as the difference between the two costs. Insert the task having maximum regret in its best position, and then re-iterate, by re-computing regrets, until all tasks have been re-inserted.

The Regret-Constructor has been used, among others, by Ropke and Pisinger (2006), to overcome the myopic behavior of greedy repair.

3.3 The Local Search Component

Local search in AILS is based on five operators from the node routing and arc routing literature that will be described in detail below. These operators have been extended to accommodate the MCGRP model. In total, 26 move subtypes have been implemented. However, we have designed a new (as far as we know), more aggressive neighborhood evaluation strategy, as follows. In each iteration, the union of neighborhoods resulting from the operators applied to the current solution is fully explored. All moves with positive savings are considered, in the order of decreasing savings. All independent moves that lead to feasible solutions are executed, before local search proceeds with the next neighborhood exploration from the new current solution.

As is seen from Algorithm 1, AILS performs intensification through local search in three situations:

- after construction of the initial solution

- when a solution with sufficient quality has been produced by ALNS

- after a kick

The different situations call for different LS variants. After initial construction, the goal is to find a high quality solution fast, a deep local optimum. Therefore, we utilize the most powerful local search variant called LS_FULL that includes all move types. In the two other situations, we have seen through experiments that LS_FULL becomes too expensive. We therefore designed two reduced variants: LS1 and LS2. The details of these are given below. For further intensification of a deserving solution after a destroy and repair, we randomly select between LS1 and LS2, with a 70% probability for LS1. After a kick, LS1 is employed. These choices were made after extensive experiments on standard MCGRP, CARP, and CVRP benchmarks.

For acceptance of the new solution, we decided to investigate two criteria:

- simple improvement
- a deterministic annealing variant called *Threshold Accepting* (TA) Dueck and Scheuer (1990)

TA is known to be as effective and more computationally efficient than Simulated Annealing. For the final selection of acceptance criterion, we refer to Section 3.4.

AILS utilizes the following set of local search operators:

- **Swap**: exchange the position of two tasks (both intra- and inter-route);
- **Or-opt**: break a route in three points, then reconnect it in the only possible way. The length of the segment to be relocated is limited to l tasks. Also in this case intra and inter-route optimization are performed;
- **2-opt**: break a route in two points and re-connect the two parts obtained in the only possible way. We adapted to the MCGRP also the eight different operators originally proposed by Santos et al. (2010) for the CARP. They consist in the intra-route operator just described, plus seven inter-route operators obtained by breaking two different routes in one point each, and reconnecting them in all possible ways when segment reversals are also considered;
- **3-opt**: break the route in three points and then reconnect it by also allowing reversing portions of the route. Suppose the current route is made by three portions A, B and C, 3-opt considers seven combinations by reversing one or more portions. Namely, $AC\bar{B}$, $A\bar{C}B$, $A\bar{C}\bar{B}$, $\bar{A}CB$, $\bar{A}C\bar{B}$, $\bar{A}\bar{C}B$ and $\bar{A}\bar{C}\bar{B}$, where $\bar{X}$ denotes the reversing of X. There are six cases for intra-route (case $A\bar{C}\bar{B}$ is equal to 2-opt intra-route) and seven cases for inter-route optimization.
- **Flip**: tries to revert the direction of the visit of an arc or edge task for each modified route.

The Flip is commonly used in genetic algorithms and has been applied to MCGRP by Prins (2009). Hence, AILS uses a total of 26 move subtypes: 13 types of 3-opt, 8 types of 2-opt, 2 types of Or-opt, 2 Swap types, and Flip. LS_Full employs all operators above. The segment length limit l for Or-opt is 3, and for 3-opt, $|B| \leq 3$. The computationally cheaper LS1 consists of the following operators: Or-opt with $l = 2$, Swap, 3-opt with $|B| \leq 3$. This limits the search to 14 operator subtypes. Analogously, LS2 is a different reduced LS that consists of Or-opt with $l = 2$, Swap, 2-opt, 3-opt with $|B| \leq 2$. It corresponds to 16 operator subtypes. The flip operator is used in LS_Full and LS1.

3.4 Configuration of the Algorithm

The final choice of alternative mechanisms and parameter settings in AILS was based on a combination of hypotheses and insights from experimental investigation. First, experiments with all proposed AILS mechanisms in place were performed using three MCGRP benchmarks, namely, `CBMix`, `BHW`, and `DI-NEARP`.

Numerical parameters were tuned by running experiments on all these instances. We have observed that the behavior of AILS mainly depends on the mechanisms introduced, rather than on the values of the numerical parameters. Moreover AILS is robust against small changes of parameters. The settings used are as follows: Initial temperature for Threshold Accepting $T_0 = 0.1$, final temperature $T_f = 0.001$, temperature reduction factor $\alpha = 0.95$. The difference between the cost of the initial solution and the incumbent one, is used as a scaling parameter for Threshold Accepting to make the acceptance criterion instance independent. The number of destroy and repair cycles per temperature was set to 10, and the number of iterations with no improvement before invocation of the kick was set to 20.000. Destructor/constructor scores were reset after 500 iterations. The best range for quasi-random selection of number of tasks using a uniform distribution was determined to be $[1, \min\{20, \tau\}]$, where τ is the number of tasks. The value 1 was selected for incrementing the scores matrix for a successful destructor/constructor pair. These setting are summarized in Table 3.

To empirically assess the merit of proposed components in AILS, we selected a subset of 17 instances from the `CBMix`, `BHW`, and `DI-NEARP` benchmarks (see Section 2). The sample contains instances of different size. In general, the sample represents instances that seemed hard from early experiments. The concrete instances are found in Tables 1 and 2 below.

3.4.1 Investigation of ALNS Destructors

An early AILS version only had extensions of classical ALNS destructors. Experiments and analyses revealed that these destructors do not work well on MCGRP instances with a strong graph structure, as the selected tasks will be spread all over the graph. This was the motivation for introducing the Tree-Destructor that utilizes the graph structure to select the tasks to be removed. By logging the scores matrix in initial experiments, it became apparent, as expected, that the Tree-Destructor after some iterations received higher scores than any of the three Task-Destructors that pick tasks of a certain kind randomly. We decided to conduct a comparative experiment where an AILS version with only the extended classical destructors (Random-Destructor, the three Task-Destructors, the Worst-Destructor, and the Related-Destructor) was compared to a version including the Tree-Destructor but excluding the Task-Destructors. The choice of excluding the Task-Destructors is also motivated by the fact that each of them can be applied only if the number of tasks to be removed is smaller than the cardinality of the task set addressed by the operator. On the other hand, the use of a combination of the three operators, to manage larger values of k, is equivalent to apply the Random Destructor.

Each run was given a time limit of 3600 CPU seconds (details on the computer and the implementation can be found in Section 4). The results are given in Table 1. The first column shows the name of the instance, the second the number of tasks τ. The third and fourth columns report the best solution value obtained by the two versions. The last column shows the difference between the solution values. Bold numbers identify the best value obtained by the two algorithms. In the last row we report the sum of the values for columns 3-5.

AILS with the Tree-Destructor has the best overall performance with a lower total solution value, by 219 units. Also, it finds two more best solutions than its competitor. Hence, for the final version of AILS, we excluded the **Task-Destructors** and included the **Tree-Destructor**.

3.4.2 Investigation of Other AILS Mechanisms

Having selected the destructor configuration, we proceeded to evaluate other important mechanisms of Algorithm 1. We compared the full AILS version with all proposed mechanisms in place, with five versions in which we disabled, in turn, one important mechanism. The five reduced versions were as follows:

Table 1: Assessment of destructor variants

Instance	τ	Only Classical Destructors	Tree Destructor No Task Destructors	Δ
`CBMix4`	98	**7481**	7497	16
`CBMix7`	168	9515	**9501**	-14
`CBMix8`	177	10367	**10365**	-2
`CBMix15`	91	**8183**	8296	113
`CBMix16`	169	**8714**	8742	28
`CBMix19`	212	16271	**16190**	-81
`BHW9`	178	**875**	**875**	0
`BHW12`	115	**10883**	10898	15
`BHW15`	128	15352	**15347**	-5
`BHW16`	410	43878	**43877**	-1
`BHW20`	293	16262	**16183**	-79
`DI-NEARP-n240-Q4k`	240	18318	**18194**	-124
`DI-NEARP-n422-Q8k`	422	14638	**14469**	-169
`DI-NEARP-n442-Q8k`	442	**43267**	43466	199
`DI-NEARP-n477-Q2k`	477	**23003**	23092	89
`DI-NEARP-n699-Q4k`	699	**39839**	39934	95
`DI-NEARP-n833-Q16k`	833	32966	**32667**	-299
Total		319812	319593	-219

- No LS for further intensification to a local optimum
- No kick diversification (lines 24-28 in Algorithm 1)
- Simple improvement instead of Threshold Accepting in ALNS (line 20)
- No quality discrimination for LS intensification (line 16)
- No reset of scores in the ALNS operator pair matrix (line 12)

Again, each version was given 3600 CPU seconds.

Table 2: Assessing the influence of AILS mechanisms

instance	τ	full	No LS	No Kick	No Thresh. Accepting	No Quality Discrim.	No Score Reset
CBMix4	98	7497	7824	7578	**7487**	7501	7512
CBMix7	168	9501	10007	9463	**9426**	9545	9428
CBMix8	177	10365	10932	10381	10344	10364	**10258**
CBMix15	91	8296	8557	8240	**8225**	8253	8246
CBMix16	169	8742	9263	8742	**8714**	8715	8743
CBMix19	212	16190	17448	**16106**	16252	16196	16176
BHW9	178	875	930	881	881	873	**871**
BHW12	115	10898	11226	10886	**10882**	10917	10893
BHW15	128	**15347**	15802	15413	15413	15422	15373
BHW16	410	43877	46053	43877	43938	44016	**43856**
BHW20	293	**16183**	16940	16242	16223	16268	16253
DI-NEARP-n240-Q4k	240	18194	18514	18399	18194	**18188**	18404
DI-NEARP-n422-Q8k	422	14469	14479	14469	14469	**14442**	**14442**
DI-NEARP-n442-Q8k	442	43466	43367	43466	**43264**	**43264**	43466
DI-NEARP-n477-Q2k	477	23092	23628	23092	23002	23004	**22944**
DI-NEARP-n699-Q4k	699	39934	41053	39934	40415	40389	**39924**
DI-NEARP-n833-Q16k	833	32667	33406	32667	32638	**32572**	32667
Total		319593	329429	319836	319767	319929	319456

Table 2 reports computational results from the six resulting AILS versions. Bold numbers identify the best value, for an instance, among all the runs. The last row contains the sum of the values of each column.

The columns labeled "No LS" refer to the version that does not use local search for further intensification after a destroy/repair cycle. It finds no best solution for any of the instances. Therefore we decided that further intensification should be applied. Similarly, we decided to keep the kick because the "No Kick" version finds just one best solution. We decided to go for simple improvement comparison rather than Threshold Accepting in ALNS due to better results. The "No Quality Discrimination" version showed good performance for large instances. Based on the results we use quality discrimination for local search intensification only for instances with $\tau < 240$. Finally the version without reset of operator pair scores finds good results, in particular for large instances, so we decided to apply the reset only for instances with τ up to 240.

Subsequent computational experiments, whose details are not reported here, were used to define the best choice of the numerical parameters for the selected AILS version. For parameter $k_{\max}$ used as maximum number of tasks to be removed during the application of a Destructor operator. We found that 0.05τ is a good value, for several instances, but that with zero arcs required. In the latter case we fixed $k_{\max} = 0.2\tau$.

The values of the remaining numerical parameters can be found in Table 3 which summarizes our decisions.

Table 3: Final settings of algorithm components and parameters

Component/Parameter	Action/Value
LS	always use LS to optimize the current solution
Kick	always use the Kick strategy
Threshold Accepting	do not use
Quality Acceptance	QUALITY_FACTOR=1.05 if $\tau < 240$, $+\infty$, otherwise
Score Reset	apply when $\tau \leq 240$
$k_{\max}$	0.05τ if $A_r \neq \emptyset$, 0.2τ, otherwise
T_0	0.1
T_f	0.001
α	0.95
β	0.75
γ	0.1
N_ITER_PER_STAGE	10
N_ITER_BEFORE_KICK	20.000
PI_RESET	500

3.5 AILS details

The detailed workings of AILS is documented with pseudo-code in Algorithms 2–3 and explained below.

Before AILS starts, the minimum traversal (deadheading) costs c'_{ij} connecting any pair of vertices i and j are computed with the standard Dijkstra algorithm. Recall that $\tau = |N_r| + |E_r| + |A_r|$ is the total number of requests.

First, the AILS computes an initial solution x_{init} with the function CONSTRUCT_INITIAL_SOLUTION of line 8. Experiments showed that the quality of the initial solution had no significant effect on the final result after a reasonable computing time. Hence, we selected a computationally cheap construction procedure, the *Augment-Merge* heuristic proposed by Golden and Wong (1981). For instances with a given upper bound on the number of vehicles (as the `mggdb` and `mgval` benchmarks) we used a modified version of the Augment-Merge procedure that continues to merge routes, also with negative saving, if the number of routes in the current solution is larger than the upper bound. If this simple procedure fails in finding a feasible solution, we solve a *bin packing problem* where the task demands are objects that have to be packed into bins of capacity equal to the vehicle capacity. We used the powerful variable neighborhood search procedure developed in Dell'Amico et al. (2012), modified so that it stops as soon as the number of bins is not larger than the upper bound. The tasks of each bin are than sequenced with a simple nearest insert procedure.

The initial solution is taken to a deep local optimum, the first version of the incumbent $x_{incumbent}$, by the most powerful LS, called LS_Full.

After initialization, the AILS enters a main loop that is executed until timeout. Within this loop, combined ALNS and LS is executed. We note that the number of iterations per stage is initialized to the parameter N_ITER_PER_STAGE, then increased by one for each stage, as initial experiments indicated that more iterations were needed to reinforce the intensification.

Algorithm 2 Adaptive Iterative Local Search (* Final Detailed Implementation *)

```
function AILS(Instance)
    comment: Initialize global variables
    IterationCounter := 0;
    IterPerStage := N_ITER_PER_STAGE
    KickCountdown := N_ITER_BEFORE_KICK
    Reset_Roulette_Probabilities( )
    comment: Construct first solution and take to deep local optimum
    x_init :=Construct_Initial_Solution(Instance)
    x_incumbent :=LS_Full(x_init)
    x_LocalIncumbent := x_incumbent
    comment: Main body: iterative phase
    repeat
        x_current := x_BestThisStage := x_LocalIncumbent
        comment: Execute a batch of iterations
        for i := 0 to IterPerStage do
            IterationCounter := IterationCounter + 1
            NewBest := Combined_ALNS_and_LS
            if NewBest then
                IterPerStage := N_ITER_PER_STAGE -1
                KickCountdown := N_ITER_BEFORE_KICK
                break
            end if
        end for
        comment: Increase number of iterations
        IterPerStage := IterPerStage + 1
    until Timeout( )
    return x_incumbent
end function
```

Algorithm 3 Combined ALNS and LS

```
 1: function COMBINED_ALNS_AND_LS
 2:     comment: Check for stagnation
 3:     if KickCountdown > 0 then
 4:         comment: Reset roulette probabilities regularly
 5:         if (IterationCounter modulo PI_RESET) = 0 then RESET_ROULETTE_PROBABILITIES( )
 6:         k :=RANDOM(1, k_max)
 7:         x_current :=ROULETTE_DESTROY_AND_REPAIR(k)
 8:         if z(x_current) ≥ QUALITY_FACTOR · z(x_BestThisStage) return FALSE
 9:         comment: Cost acceptable, intensify with LS1 or LS2 based on 70% probability
10:         x_current := if RANDOM(0,100)≤ 70 then LS1(x_current) else LS2(x_current)
11:         KickCountdown := KickCountdown − 1
12:         if z(x_current) < z(x_BestThisStage)  then
13:             x_BestThisStage := x_current
14:             comment: Give higher probability to selected Destructor/Constructor pair
15:             UPDATE_ROULETTE_PROBABILITIES( )
16:             comment: Return true if an update has been performed
17:             return UPDATE_INCUMBENTS(x_current)
18:         end if
19:         return FALSE
20:     else
21:         comment: Nothing has happened for a while, make a major, random destroy and repair
22:         k :=RANDOM(τ/2, τ)
23:         x_LocalIncumbent :=RANDOM_DESTROY_AND_REPAIR(k)
24:         x_current := x_LocalIncumbent :=LS_FULL(x_LocalIncumbent)
25:         UPDATE_INCUMBENTS(x_current)
26:         comment: Return true to exit the for loop of AILS
27:         return TRUE
28:     end if
29: end function
```

Parameters, functions and procedures used by AILS are briefly described hereafter.

TIMEOUT: A function which returns `TRUE` when the given CPU time limit is reached, `FALSE` otherwise.

RESET_ROULETTE_PROBABILITIES: sets all entries in the scores table π for roulette wheel selection to 1. This procedure is invoked in the initialization, line 6), and also periodically after a certain number of iterations given by the parameter PI_RESET, see function COMBINED_ALNS_AND_LS, line 5.

The following functions are used in in COMBINED_ALNS_AND_LS.

RANDOM_DESTROY_AND_REPAIR: line 23 , is called to make a restart (the *kick* component) after a certain number of iterations have passed without acceptance of a new current solution. It calls the Random-Destructor and then the Random-Constructor for a number of tasks in the interval $[\tau/2, \tau)$.

ROULETTE_DESTROY_AND_REPAIR calls the normal, roulette wheel based selection of Destructor and Constructor pair, and the execution of these operators with the randomly drawn k value, see line 7.

UPDATE_ROULETTE_PROBABILITIES: line 15, increases the probability of selecting a successful Destructor/Constructor pair.

UPDATE_INCUMBENTS: line 17, checks whether the current solution is better than the best solution found for the current temperature, and in case, updates the corresponding variable. Similarly, there is a test whether the current solution improves the global incumbent. The function returns `TRUE` if any of the variables were updated, `FALSE` otherwise.

4 Computational Results

We coded our algorithm in C++ and ran it on an Intel Xeon(R) E5530 @2.40GHz with 23.5 GB of memory, under the Linux Ubuntu 12.04.1 LTS 64-bit operating system. We tested the algorithm on a large number of instances from the MCGRP, CARP, and CVRP literature.

4.1 Results on the MCGRP Instances

Five MCGRP benchmarks were used, namely, `CBMix` proposed by Prins and Bouchenoua (2005), `BHW` and `DI-NEARP` proposed by Bach et al. (2013), and `mggdb` and `mgval` by Bosco et al. (2013). The `CBMix` benchmark consists of 23 randomly generated instances on mixed graphs that imitate real street networks. They contain from 11 to 150 nodes, and from 29 to 332 edges and arcs. The instances have a number of requests between 20 and 212, located on a combination of nodes, edges, and arcs. On average, the 50% of the nodes, edges, and arcs have to be serviced.

The `BHW` set has 20 test problems generated by modifying well-known instances from the CARP literature, including `gdb` instances (see Golden et al. (1983)), `val` instances (see Benavent et al. (1992)), and `egl` instances (see Li and Eglese (1996)). Instances contain from 11 to 72 nodes, 0 to 51 edges, and 22 to 380 arcs. The number of requests varies from 20 to 410, and on average, about 62% of the nodes, edges, and arcs have requests.

The `DI-NEARP` benchmark with 24 instances originates from six real-life newspaper carrier routing cases in Norway. There are four different variants corresponding to a reasonable range of capacity values for each case. The instances contain from 563 to 1120 nodes, from 815 to 1450 edges, but no arcs. The number of requests varies from 240 to 833, and roughly 1/3 of the nodes and edges require service.

In contrast with `CBMix`, `BHW`, and `DI-NEARP`, `mggdb` and `mgval` include a fleet size constraint. Each of the six `mggdb` sets has 23 instances with between 18 and 48 tasks, and between 3 and 10 vehicles. For `mgval`, each of the six sets has 34 instances. The number of tasks is between 38 and 129, and fleet size varies between 2 and 10.

In Table 4 we give the results we obtained on the `CBMix` instances. We compare our AILS with

- MA: the memetic algorithm by Prins and Bouchenoua (2005), run on a Pentium III at 1.0 GHz;
- SA: the simulated annealing algorithm by Kokubugata et al. (2007), run on a Pentium IV at 1.8 GHz;
- Spider: the commercial VRP solver tested in Hasle et al. (2012), run on an Intel(R) Core(TM) i7 at 3.07 GHz.

For MA and SA, the termination condition was the number of iterations without new accepted solutions. Spider and AILS were stopped after a given CPU time limit. MA, Spider, and AILS were run just once on each instance, whereas SA was run ten times by varying the random seed generator. It is worth noting that Spider has been implemented to solve a large variety of routing problems; it is not specifically designed for the MCGRP as is the case for the other three solvers in the table.

In each line of Table 4, we give the name of the instance addressed, the number of tasks τ, and the best known lower bound LB, obtained as the maximum value among those presented by Bach et al. (2013) and Bosco et al. (2013). For MA, we give the solution value z it obtains, and the number of CPU seconds required to run to completion, sec_{tot}. For SA, we give the average solution value $avg\ z$ over the 10 attempts and the average CPU time in seconds to run to completion. For Spider, that was run a single time for two hours on each instance, we provide the solution value z and the CPU time in seconds in which the incumbent solution was found, sec_{inc}.

To obtain a fair comparison with the previous algorithms, we report the results obtained by our AILS with a limited CPU time of 200 seconds (MA needs 1078 seconds in the worst case on a slower computer, whereas SA needs 661 seconds in the worst case, also on a slower computer). We also report the results obtained by the AILS with a timeout of one CPU hour, which corresponds to our best configuration. For both AILS configurations we report the objective function value and the number of seconds needed to reach the incumbent solution.

The best objective function values obtained are reported in bold. We observe that the AILS is very effective, and outperforms on average the competition even within a CPU time limit of 200 seconds. It achieves an total objective value of 163877, 1.8% lower than the best competitor, SA. When the CPU time limit is increased to one hour, AILS finds the best solution values for all instances, consistently lowering the cost to become 2.5% lower than the best competitor. The optimal costs of `CBMix12` and `CBMix23` ($z = 3138$ $z = 780$, respectively), were proven by Bosco et al. (2013). AILS confirms the `CBMix12` value for the first time, whereas the `CBMix23` instance is very easy to solve by all four heuristics (the optimal solution is illustrated in Figure 2).

In Table 5 we present the results we obtained on the `BHW` set. The lower bound values are taken from Bach et al. (2013). On this benchmark we can compare only with Spider, hence we report only the results we obtained with a one CPU hour timeout. AILS provides better or equally good solutions on all `BHW` instances, with 1.5% lower total cost. Four `BHW` results are proven optimal by comparison with the lower bound. Three were obtained by both Spider and AILS, and one by AILS only.

In Table 6 we compare again the performance of AILS with Spider and with the lower bounds by Bach et al. (2013), this time on the `DI-NEARP` benchmark. Also on this set of 24 larger-size instances, the performance of the AILS is good. Indeed, it yields novel best known solutions to all instances, lowering the total cost with 1.5% relative to Spider. For some instances, both algorithms find the incumbent solution close to timeout. This is an indication of the complexity of this test bed, and we believe further improvements are possible for many of these instances.

Tables 7-12 show the performance of AILS on the `mggdb` instances, compared to the lower and upper bounds from the branch-and-cut method reported in Bosco et al. (2013). Of the 114 instances that were investigated in Bosco et al. (2013), 84 solutions were proven optimal. AILS finds these solutions in less than 5 seconds. For the 30 remaining instances, the sum of objective values is equal for both contestants. Of these, Bosco et al. (2013) has found two best known solutions, and AILS has found one. With one exception, AILS finds its best solution within 35 seconds. For the remaining 24 instances that were not investigated empirically by Bosco et al. (2013) (with more than 7 vehicles), AILS has generated the first upper bounds. The time until the best solution is found varies a lot within the given time limit of one hour.

Similarly, in tables 13-18, we present results for a total of 204 `mgval` instances, 150 of which has a fleet size of less than 8 vehicles so they were investigated in Bosco et al. (2013). Of these 150, 70 proven optimal solutions were provided. AILS finds all of these, generally long before the timeout of one hour. For the 80 solutions where the optimal value is not known, the sum of costs is nine units lower for AILS. Bosco et al. (2013) has found five best known upper bounds, AILS seven, and the rest are ties. Again, AILS provides the first feasible solutions for the instances with more than seven vehicles.

4.2 Results on CVRP and CARP Instances

For the CARP, we tested seven well-known benchmarks, 23 `gdb` instances proposed in Golden et al. (1983), 34 `val` instances proposed in Benavent et al. (1992), 24 `egl` instances proposed in Li and Eglese (1996), and 100 `bmcv` instances proposed in Beullens et al. (2003) in four datasets (`C`, `D`, `E` and `F`). In Table 19, we compare our approach against the six best performing CARP metaheuristics (to our knowledge). These are:

- GLS: proposed by Beullens et al. (2003), based on guided local search (run on a Pentium II at 500 MHz);
- MA-CARP: proposed by Lacomme et al. (2004a), based on genetic algorithms (run on a Pentium III at 1 GHz);
- BACO: proposed by Lacomme et al. (2004b), based on ant colony optimization (run on a Pentium III at 800 MHz);
- VNS: proposed by Polacek et al. (2008), based on variable neighborhood search (run on a Pentium IV at 3.0 GHz). Polacek et al. (2008) reported two sets of results, here we only report the "3.0 GHz" solutions;
- TSA: proposed by Brandão and Eglese (2008), based on tabu search (run on a Pentium Mobile at 1.4 GHz). Brandão and Eglese (2008) report results of two versions of TSA, here we show the best configuration, i.e., the second one;
- Ant-CARP: proposed by Santos et al. (2010), based on ant colony optimization (run on an Intel Pentium III at 1 GHz). Santos et al. (2010) report results of two versions of the Ant-CARP, the median of the best one is reported here (Ant-CARP_12);

Note that '–' means that this method has not been tested on this benchmark. We observe that AILS is among the very best competitors on the CARP.

For the CVRP, four heavily investigated benchmarks were used: 14 instances proposed in Christofides and Eilon (1969) and Christofides et al. (1979), 13 instances proposed in Taillard (1993), 20 instances from Golden et al. (1998), and 12 instances from Li et al. (2005).

The four first lines in Table 20 shows the average gap for, to our knowledge, the best performing metaheuristics for the CVRP. The last two rows show results for two MCGRP metaheuristics, namely the memetic algorithm of Prins and Bouchenoua (2005) and AILS. The CVRP metaheuristics are the following:

- GRASP: proposed by Prins (2009), based on GRASP and evolutionary local search (run on a 2.8 GHz Pentium 4);
- MB: proposed by Mester and Bräysy (2007), based on active-guided evolution strategies (run on a 2.8 GHz Pentium 4);
- MA-CVRP: proposed by Nagata and Bräysy (2009), based on memetic algorithm (run on a 3.2 GHz Xeon);
- PARALLEL: proposed by Gröer et al. (2011), based on parallel algorithm (run on 50 computers, each dual-core 2.3 GHz Xeon);

Note that the gaps are calculated relative to the best known solutions from 2012, a few of which have been improved lately, but the relative performance should be clear. We see that AILS is highly competitive also for the CVRP, except for the Golden et al. (1998) instances where the quality is still reasonable. As far as we know, AILS has found a new best known solution for the `TAILLARD100D` instance, with cost of 1580.46 versus 1581.24 reported in Gambardella et al. (1999).

5 Conclusions

The Mixed Capacitated General Routing Problem, also called the Node, Edge, and Arc Routing Problem, provides a capacitated multi-vehicle VRP variant that captures an arbitrary mixture of requests on links and nodes. As far as we know, the problem was first studied by Pandi and Muralidharan (1995). Despite the fact that the MCGRP is scientifically interesting and has considerable practical value, it has received limited attention. Recently, however, several metaheuristics, a lower bound procedure, an ILP formulation, and a Branch & Cut exact method have been proposed.

In this paper, we report from the design and investigation of a new hybrid metaheuristic, called AILS, containing several innovations and non-standard mechanisms, for solving MCGRP instances also of industrial size. Computational experiments on five MCGRP benchmarks show excellent performance, with best known solutions to all instances of the `CBMix`, `BHW`, and `DI-NEARP` benchmarks in reasonable time. For the smaller size `mggdb` and `mgval` benchmarks previously investigated by a branch-and-cut method, AILS finds all proven optimal solutions in a short time, and provides eight new best known upper bounds. A comparative assessment of the AILS metaheuristic on 181 CARP and 59 CVRP instances proves that our metaheuristic is also among the best for special cases of the MCGRP.

References

L. Bach, G. Hasle, and S. Wøhlk. A lower bound for the node, edge, and arc routing problem. *Computers & Operations Research*, 40(4):943–952, 2013. ISSN 0305-0548.

R. Baldacci and A. Mingozzi. A unified exact method for solving different classes of vehicle routing problems. *Mathematical Programming*, 120:347–380, 2009.

R. Baldacci, N. Christofides, and A. Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115:351–385, 2008.

E. Bartolini, J.-F. Cordeau, and G. Laporte. Improved lower bounds and exact algorithm for the capacitated arc routing problem. *Mathematical Programming*, 137(1–2):1–44, 2011.

E. Benavent, V. Campos, A. Corberan, and M. Mota. The Capacitated Arc Routing Problem. Lower Bounds. *Networks*, 22:669–690, 1992.

P. Beullens, L. Muyldermans, D. Cattrysse, and D. Van Oudheusden. A Guided Local Search Heuristic for the Capacitated Arc Routing Problem. *European Journal of Operational Research*, 147(3):629–643, 2003.

L. Bodin, V. Maniezzo, and A. Mingozzi. Street routing and scheduling problems. In Randolph W. Hall and Frederick S. Hillier, editors, *Handbook of Transportation Science*, volume 56 of *International Series in Operations Research & Management Science*, pages 413–449. Springer US, 2003. ISBN 978-0-306-48058-4.

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

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

N. Christofides and S. Eilon. An algorithm for the vehicle-dispatching problem. *Oper. Res. Quart.*, 20(3): 309–318, 1969.

N. Christofides, A. Mingozzi, and P. Toth. The vehicle routing problem. In N. Christofides, A. Mingozzi, P. Toth, and C. Sandi, editors, *Combinatorial Optimization*, pages 315–338. Wiley, Chichester, 1979.

A. Corberán and C. Prins. Recent results on arc routing problems: An annotated bibliography. *Networks*, 23:50–69, 2010.

A. Corberán, A.N. Letchford, and J.M. Sanchis. A cutting plane algorithm for the general routing problem. *Mathematical Programming, Series A*, 90:291–316, 2001.

A. Corberán, G. Mejia, and J.M. Sanchis. New results on the mixed general routing problem. *Operations Research*, 53:363–376, 2005.

A. Corberán, E. Mota, and J.M. Sanchis. A comparison of two different formulations for arc routing problems on mixed graphs. *Computers & Operations Research*, 33:3384–3402, 2006.

A. Corberán, I. Plana, and J.M. Sanchis. A branch & cut algorithm for the windy general routing problem and special cases. *Networks*, 49:245–257, 2007.

M. Dell'Amico, J.C. Díaz Díaz, and M. Iori. The bin packing problem with precedence constraints. *Operations Research*, 60:1491–1504, 2012.

G. Dueck and T. Scheuer. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90(1):161–175, 1990.

L. M. Gambardella, E. Taillard, and G. Agazzi. MACS-VRPTW: A multiple colony system for vehicle routing problems with time windows. In *New Ideas in Optimization*, pages 63–76. McGraw-Hill, 1999.

B. Golden, S. Raghavan, and E. Wasil (eds.). *The Vehicle Routing Problem: Latest Advances And New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces Series*. Springer, Berlin, 2008.

B.L. Golden and R.T. Wong. Capacitated Arc Routing Problems. *Networks*, 11(3):305–315, 1981.

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

B.L. Golden, E.A. Wasil, J.P. Kelly, and I.M. Chao. Metaheuristics in Vehicle Routing. In T. Crainic and G. Laporte, editors, *Fleet management and logistics*, pages 33–56. Boston, MA: Kluwer, 1998.

C. Gröer, B. Golden, and E. Wasil. A Parallel Algorithm for the Vehicle Routing Problem. *INFORMS Journal on Computing*, 23:315–330, 2011.

J.C.A. Gutiérrez, D. Soler, and A. Hérvas. The capacitated general routing problem on mixed graphs. *Revista Investigacion Operacional*, 23:15–26, 2002.

G. Gutin and A.P. Punnen (eds.). *The Traveling Salesman and its Variations*. Kluwer, Dordrecht, 2002.

G. Hasle, O. Kloster, M. Smedsrud, and K. Gaze. Experiments on the node, edge, and arc routing problem. Technical Report A23265, ISBN 978-82-14-05288-6, SINTEF, 2012.

H. Kokubugata, A. Moriyama, and H. Kawashima. A practical solution using simulated annealing for general routing problems with nodes, edges, and arcs. In *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics*, volume 4638, pages 136–149. Springer Berlin/Heidelberg, 2007.

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

P. Lacomme, C. Prins, and A. Tanguy. First competitive ant colony scheme for the carp. In M. Dorigo, M. Birattari, C. Blum, L.M. Gambardella, F. Mondada, and T. Stützle, editors, *Ant Colony Optimization* *and Swarm Intelligence*, volume 3172 of *Lecture Notes in Computer Science*, pages 426–427. Springer Berlin Heidelberg, 2004b. ISBN 978-3-540-22672-7.

F. Li, B. Golden, and E. Wasil. Very large-scale vehicle routing: New test problems, algorithms, and results. *Computers & Operations Research*, 32(5):1165–1179, 2005.

L.Y.O. Li and R.W. Eglese. An Interactive Algorithm for Vehicle Routing for Winter-Gritting. *Journal of the Operational Research Society*, 47:217–228, 1996.

H.R. Lourenço, O.C. Martin, and T. Stützle. Iterated local search: Framework and applications. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics, second edition*, volume 146 of *International Series in Operations Research & Management Science*, pages 363–398. Springer, Berlin, 2010.

R. Martinelli, D. Pecin, M. Poggi, and H. Longo. A branch-cut-and-price algorithm for the capacitated arc routing problem. In M.P. Pardalos and S. Rebennack, editors, *Experimental Algorithms*, volume 6630 of *Lecture Notes in Computer Science*, pages 315–326. Springer Berlin Heidelberg, 2011.

D. Mester and O. Bräysy. Active-guided evolution strategies for large-scale Vehicle Routing Problems. *Computers & Operations Research*, 34:2964–2975, 2007.

Y. Nagata and O. Bräysy. Edge Assembly based Memetic Algorithm for the Capacitated Vehicle Routing Problem. *Networks*, 54:205–215, 2009.

C.S. Orloff. A fundamental problem in vehicle routing. *Networks*, 4(1):35–64, 1974.

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

D. Pisinger and S. Røpke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435, 2007.

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

C. Prins. A grasp * Evolutionary Local Search Hybrid for the Vehicle Routing Problem. In F.B. Pereira and J. Tavares, editors, *Bio-inspired Algorithms for the Vehicle Routing Problem*, volume 161 of *Studies in Computational Intelligence*, pages 35–53. Springer, Berlin, 2009.

C. Prins and S. Bouchenoua. A memetic algorithm solving the vrp, the carp and general routing problems with nodes, edges and arcs. In *Recent Advances in Memetic Algorithms*, volume 166, pages 65–85. Springer Berlin / Heidelberg, 2005.

S. Ropke and D. Pisinger. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, 40(4):455–472, 2006.

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

P. Shaw. A New Local Search Algorithm Providing High Quality Solutions to Vehicle Routing Problems. Technical report, University of Strathclyde, 1997.

P. Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In *Proceedings CP-98 (Fourth International Conference on Principles and Practice of Constraint Programming)*, 1998.

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

P. Toth and D. Vigo. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 2002a.

P. Toth and D. Vigo. An overview of vehicle routing problems. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, pages 1–26. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 2002b.

T. Vidal, T.G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60:611–624, 2012.

S. Wøhlk. New lower bound for the capacitated arc routing problem. *Computers & Operations Research*, 33:3458–3472, 2006.

S. Wøhlk. A decade of capacitated arc routing. In B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances And New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces Series*, pages 29–48. Springer, Berlin, 2008.

Appendix. Detailed experimental results

Table 4: Computational results on the `CBMix` instances.

			MA		SA		Spider		AILS			
							(sec_{tot}=7200)		(sec_{tot}=200)		(sec_{tot}=3600)	
Instance	τ	LB	z	sec_{tot}	$avg\ z$	sec_{tot}	z	sec_{inc}	z	sec_{inc}	z	sec_{inc}
CBMix1	48	2409	2632	108.3	2617.1	15.1	2589	1231.0	**2585**	26.4	**2585**	26.4
CBMix2	185	9742	12336	1078.5	12322.4	661.4	12222	4156.0	11876	192.4	**11749**	1869.2
CBMix3	79	3014	3702	157.0	3695.2	56.0	3767	6612.0	3619	195.8	**3614**	418.3
CBMix4	98	5302	7583	548.1	7728.5	76.1	7802	6744.0	7550	68.4	**7483**	3384.7
CBMix5	65	3789	4562	100.0	4685.3	41.5	4688	1349.0	4508	118.6	**4459**	593.1
CBMix6	108	5201	7087	204.5	7101.4	98.0	7139	6687.0	7043	106.3	**6969**	1619.0
CBMix7	168	7296	9974	662.6	9704.8	351.7	9767	3205.0	9444	139.7	**9428**	2516.7
CBMix8	177	7956	10714	767.6	10710.2	263.8	10689	1413.0	10405	156.1	**10338**	2143.9
CBMix9	50	3460	4041	140.8	4132.4	12.5	4147	5517.0	4002	53.3	**3991**	430.7
CBMix10	107	6432	7755	843.2	7763.2	108.3	7931	4665.0	7538	150.9	**7525**	1399.5
CBMix11	82	3031	4503	414.7	4599.6	49.8	4525	536.0	4494	75.3	**4484**	543.8
CBMix12	53	3138	3235	71.3	3235	21.4	3235	14.0	***3138**	178.9	***3138**	178.9
CBMix13	141	6524	9339	550.6	9270.6	312.8	9332	1427.0	9079	168.4	**9037**	2840.4
CBMix14	93	5731	8615	357.2	8769.3	65.3	8638	6404.0	8511	26.3	**8473**	608.7
CBMix15	91	6318	8359	390.2	8385.3	97.3	8443	3553.0	8269	59.0	**8221**	2962.2
CBMix16	169	7416	9389	536.1	9024.3	445.5	9022	6754.0	8743	185.5	**8742**	844.6
CBMix17	63	3654	4165	116.1	4107.6	43.0	4235	1271.0	**4034**	62.2	**4034**	62.2
CBMix18	127	6089	7411	475.7	7214.6	278.4	7346	1994.0	7130	150.8	**7052**	2556.7
CBMix19	212	11143	17036	1273.4	16677.5	469.8	16692	5688.0	16322	189.6	**16155**	451.8
CBMix20	73	3452	4918	164.6	4902.9	50.7	4859	3501.0	4806	126.7	**4738**	577.9
CBMix21	180	12474	18509	1370.6	18318.3	530.4	18809	5322.0	18060	140.8	**17875**	1012.9
CBMix22	42	1825	**1941**	65.8	1970.5	9.5	**1941**	492.0	**1941**	8.8	**1941**	8.8
CBMix23	20	780	***780**	20.4	***780**	2.7	***780**	0.3	***780**	0.0	***780**	0.0
Sum/Average	2431	126176	167806	452.9	166936.0	176.6	167818	3414.6	163877	112.2	**162811**	1176.1

Table 5: Computational results on the `BHW` instances.

			Spider (sec_{tot}=7200)		AILS (sec_{tot}=3600)	
Instance	τ	LB	z	sec_{inc}	z	sec_{inc}
BHW1	29	324	**337**	6.0	**337**	0.2
BHW2	29	470	***470**	36.0	***470**	0.1
BHW3	20	326	**415**	18.0	**415**	23.2
BHW4	50	240	***240**	1.0	***240**	0.0
BHW5	162	502	506	610.0	***502**	119.2
BHW6	110	388	***388**	58.0	***388**	10.7
BHW7	229	930	1104	6324.0	**1070**	2895.1
BHW8	117	644	672	1801.0	**668**	1273.2
BHW9	178	791	920	2431.0	**875**	212.3
BHW10	142	6810	8596	6205.0	**8524**	152.9
BHW11	71	3986	5023	3012.0	**4914**	2139.6
BHW12	115	6346	11042	6059.0	**10887**	19.0
BHW13	175	8746	14510	5723.0	**14346**	2962.6
BHW14	221	17762	25194	4584.0	**24833**	2812.6
BHW15	128	12193	15564	6728.0	**15354**	2223.0
BHW16	410	26014	44527	5747.0	**43948**	3352.9
BHW17	240	15396	26768	6823.0	**26235**	3548.7
BHW18	194	11202	15833	5532.0	**15170**	1551.1
BHW19	107	7080	9480	3605.0	**9388**	677.2
BHW20	293	10730	16625	6769.0	**16291**	1748.1
Sum/Average	3020	130880	198214	3603.6	**194855**	1286.1

Table 6: Computational results on the DI-NEARP instances.

Instance	τ	LB	Spider (sec_{tot}=7200) z	Spider (sec_{tot}=7200) sec_{inc}	AILS (sec_{tot}=3600) z	AILS (sec_{tot}=3600) sec_{inc}
DI-NEARP-n240-Q2k	240	16376	24371	4569.0	**23807**	3003.0
DI-NEARP-n240-Q4k	240	14362	18352	4495.0	**18197**	2374.2
DI-NEARP-n240-Q8k	240	13442	15937	6421.0	**15884**	2001.8
DI-NEARP-n240-Q16k	240	13116	14953	5274.0	**14717**	1058.4
DI-NEARP-n422-Q2k	422	11623	19133	6629.0	**18943**	2486.2
DI-NEARP-n422-Q4k	422	11284	15987	4524.0	**15869**	940.0
DI-NEARP-n422-Q8k	422	11220	14627	2925.0	**14442**	813.7
DI-NEARP-n422-Q16k	422	11198	14357	4661.0	**14339**	240.5
DI-NEARP-n442-Q2k	442	35068	52062	7091.0	**51052**	1303.4
DI-NEARP-n442-Q4k	442	33585	45906	6308.0	**44952**	3339.7
DI-NEARP-n442-Q8k	442	32985	45395	5964.0	**43264**	1029.5
DI-NEARP-n442-Q16k	442	32713	42797	6480.0	**42683**	1452.8
DI-NEARP-n477-Q2k	477	19722	23124	5996.0	**22896**	3218.8
DI-NEARP-n477-Q4k	477	18031	20198	7006.0	**20035**	1923.5
DI-NEARP-n477-Q8k	477	17193	18561	2999.0	**18490**	1546.5
DI-NEARP-n477-Q16k	477	16873	18105	4079.0	**18040**	2410.3
DI-NEARP-n699-Q2k	699	34101	59817	6993.0	**58948**	1776.7
DI-NEARP-n699-Q4k	699	26891	40473	7178.0	**40124**	2101.8
DI-NEARP-n699-Q8k	699	23302	30992	6095.0	**30799**	2871.4
DI-NEARP-n699-Q16k	699	21967	27028	3173.0	**26999**	3370.4
DI-NEARP-n833-Q2k	833	32435	56877	7135.0	**56102**	3556.7
DI-NEARP-n833-Q4k	833	29381	42407	6861.0	**41192**	3383.6
DI-NEARP-n833-Q8k	833	28453	35267	6940.0	**34812**	2688.3
DI-NEARP-n833-Q16k	833	28233	33013	4046.0	**32567**	3407.6
Sum/Average	12452	533554	729739	5576.8	**719153**	2179.1

Table 7: Computational results on the `mggdb-0.25` instances.

		Bosco et al. (2013)			AILS (sec_{tot}=3600)	
Instance	τ	LB	z	sec_{tot}	z	sec_{inc}
`mggdb-0.25-1`	21	280	***280**	2079.4	***280**	0.0
`mggdb-0.25-2`	25	317	**349**	21600.0	**349**	1.8
`mggdb-0.25-3`	22	278	***278**	923.9	***278**	0.0
`mggdb-0.25-4`	18	289	***289**	22.7	***289**	0.0
`mggdb-0.25-5`	24	364	**394**	21600.0	**394**	28.7
`mggdb-0.25-6`	21	292	***292**	14.9	***292**	0.0
`mggdb-0.25-7`	20	290	***290**	40.4	***290**	0.1
`mggdb-0.25-8`	45	-	-	-	**336**	28.4
`mggdb-0.25-9`	47	-	-	-	**309**	2.5
`mggdb-0.25-10`	22	265	***265**	2.8	***265**	0.0
`mggdb-0.25-11`	41	345	**356**	21600.0	**356**	0.0
`mggdb-0.25-12`	22	400	**459**	21600.0	**459**	0.1
`mggdb-0.25-13`	26	374	**388**	21600.0	**388**	1.0
`mggdb-0.25-14`	20	107	***107**	1.4	***107**	0.0
`mggdb-0.25-15`	20	55	***55**	0.5	***55**	0.0
`mggdb-0.25-16`	25	98	***98**	5.0	***98**	0.0
`mggdb-0.25-17`	25	71	***71**	1.0	***71**	0.0
`mggdb-0.25-18`	32	139	**144**	21600.0	**144**	0.0
`mggdb-0.25-19`	10	53	***53**	1.1	***53**	0.1
`mggdb-0.25-20`	20	116	***116**	7.8	***116**	0.0
`mggdb-0.25-21`	31	145	**146**	21600.0	**146**	0.3
`mggdb-0.25-22`	38	-	-	-	**160**	0.8
`mggdb-0.25-23`	48	-	-	-	**181**	44.0
Sum/Average no "-"	445	4278	**4430**	8121.1	**4430**	1.7
Sum/Average all	623				5416	4.7

Table 8: Computational results on the `mggdb-0.30` instances.

		Bosco et al. (2013)			AILS (sec_{tot}=3600)	
Instance	τ	LB	z	sec_{tot}	z	sec_{inc}
`mggdb-0.30-1`	21	273	***273**	4279.5	***273**	0.0
`mggdb-0.30-2`	24	270	**301**	21600.0	**301**	1.2
`mggdb-0.30-3`	19	270	***270**	4447.9	***270**	0.0
`mggdb-0.30-4`	18	260	***260**	39.7	***260**	0.0
`mggdb-0.30-5`	25	369	**388**	21600.0	**388**	0.0
`mggdb-0.30-6`	22	276	***276**	443.4	***276**	0.4
`mggdb-0.30-7`	20	273	***273**	2870.3	***273**	0.0
`mggdb-0.30-8`	46	-	-	-	**331**	12.9
`mggdb-0.30-9`	46	-	-	-	**281**	20.5
`mggdb-0.30-10`	22	242	***242**	12.1	***242**	0.0
`mggdb-0.30-11`	43	381	**387**	21600.0	**387**	0.4
`mggdb-0.30-12`	21	395	**467**	21600.0	**467**	0.6
`mggdb-0.30-13`	24	447	486	21600.0	**483**	14.6
`mggdb-0.30-14`	17	101	***101**	184.7	***101**	0.0
`mggdb-0.30-15`	19	44	***44**	0.5	***44**	0.0
`mggdb-0.30-16`	24	105	***105**	2.8	***105**	4.5
`mggdb-0.30-17`	22	65	***65**	0.6	***65**	0.0
`mggdb-0.30-18`	30	144	***144**	1.6	***144**	0.0
`mggdb-0.30-19`	10	51	***51**	0.4	***51**	0.0
`mggdb-0.30-20`	18	94	***94**	27.3	***94**	0.1
`mggdb-0.30-21`	28	120	**121**	21600.0	**121**	0.0
`mggdb-0.30-22`	37	-	-	-	**153**	0.1
`mggdb-0.30-23`	47	-	-	-	**167**	591.9
Sum/Average no "-"	427	4180	4348	7469.0	**4345**	1.1
Sum/Average all	603				5277	28.1

Table 9: Computational results on the `mggdb-0.35` instances.

		Bosco et al. (2013)			AILS (sec_{tot}=3600)	
Instance	τ	LB	z	sec_{tot}	z	sec_{inc}
`mggdb-0.35-1`	21	252	***252**	601.0	***252**	0.0
`mggdb-0.35-2`	22	284	***284**	1435.2	***284**	0.0
`mggdb-0.35-3`	20	243	***243**	1701.0	***243**	0.0
`mggdb-0.35-4`	17	242	***242**	27.6	***242**	0.0
`mggdb-0.35-5`	23	282	**309**	21600.0	**309**	1.0
`mggdb-0.35-6`	21	262	***262**	1552.4	***262**	0.0
`mggdb-0.35-7`	22	272	***272**	412.7	***272**	0.0
`mggdb-0.35-8`	38	-	-	-	**316**	2835.9
`mggdb-0.35-9`	45	-	-	-	**266**	7.7
`mggdb-0.35-10`	24	268	***268**	813.6	***268**	0.1
`mggdb-0.35-11`	41	303	***303**	3875.7	***303**	0.3
`mggdb-0.35-12`	20	461	***461**	15229.9	***461**	0.1
`mggdb-0.35-13`	24	402	**417**	21600.0	**417**	33.7
`mggdb-0.35-14`	18	84	***84**	369.3	***84**	0.3
`mggdb-0.35-15`	18	44	***44**	0.7	***44**	0.0
`mggdb-0.35-16`	22	75	***75**	2271.4	***75**	0.0
`mggdb-0.35-17`	23	62	***62**	0.9	***62**	0.0
`mggdb-0.35-18`	30	135	***135**	0.4	***135**	0.0
`mggdb-0.35-19`	9	51	***51**	0.2	***51**	0.0
`mggdb-0.35-20`	20	96	***96**	75.8	***96**	1.6
`mggdb-0.35-21`	28	118	**120**	21600.0	**120**	0.0
`mggdb-0.35-22`	36	-	-	-	**139**	0.2
`mggdb-0.35-23`	44	-	-	-	**179**	156.2
Sum/Average no "-"	423	3936	**3980**	4903.6	**3980**	2.0
Sum/Average all	586				4880	132.0

Table 10: Computational results on the `mggdb-0.40` instances.

		Bosco et al. (2013)			AILS (sec_{tot}=3600)	
Instance	τ	LB	z	sec_{tot}	z	sec_{inc}
`mggdb-0.40-1`	19	279	***279**	392.9	***279**	0.1
`mggdb-0.40-2`	22	281	**308**	21600.0	**308**	10.6
`mggdb-0.40-3`	20	225	***225**	88.2	***225**	0.0
`mggdb-0.40-4`	17	238	***238**	9.8	***238**	0.0
`mggdb-0.40-5`	22	289	**344**	21600.0	**344**	0.0
`mggdb-0.40-6`	19	270	***270**	418.9	***270**	0.0
`mggdb-0.40-7`	19	282	***282**	7631.9	***282**	0.4
`mggdb-0.40-8`	40	-	-	-	**331**	1.9
`mggdb-0.40-9`	45	-	-	-	**275**	2.3
`mggdb-0.40-10`	22	191	***191**	3.5	***191**	0.0
`mggdb-0.40-11`	38	270	**283**	21600.0	**283**	0.3
`mggdb-0.40-12`	19	412	***412**	10608.8	***412**	0.0
`mggdb-0.40-13`	23	373	**405**	21600.0	406	20.8
`mggdb-0.40-14`	18	62	***62**	76.0	***62**	0.1
`mggdb-0.40-15`	18	37	***37**	0.2	***37**	0.4
`mggdb-0.40-16`	21	84	***84**	0.7	***84**	0.0
`mggdb-0.40-17`	21	65	***65**	0.5	***65**	0.0
`mggdb-0.40-18`	27	119	***119**	45.3	***119**	0.2
`mggdb-0.40-19`	10	38	***38**	0.6	***38**	0.2
`mggdb-0.40-20`	19	94	***94**	15.9	***94**	0.1
`mggdb-0.40-21`	28	104	***104**	207.6	***104**	0.1
`mggdb-0.40-22`	33	-	-	-	**129**	0.1
`mggdb-0.40-23`	42	-	-	-	**160**	1279.6
Sum/Average no "-"	402	3713	**3840**	5573.7	3841	1.8
Sum/Average	562				4736	57.3

Table 11: Computational results on the `mggdb-0.45` instances.

		Bosco et al. (2013)			AILS (sec_{tot}=3600)	
Instance	τ	LB	z	sec_{tot}	z	sec_{inc}
`mggdb-0.45-1`	17	259	***259**	283.4	***259**	0.0
`mggdb-0.45-2`	21	279	**298**	21600.0	**298**	0.0
`mggdb-0.45-3`	19	237	***237**	246.1	***237**	0.0
`mggdb-0.45-4`	17	228	***228**	10.3	***228**	0.0
`mggdb-0.45-5`	21	309	**350**	21600.0	**350**	0.0
`mggdb-0.45-6`	18	218	***218**	109.0	***218**	0.0
`mggdb-0.45-7`	20	243	***243**	2121.2	***243**	0.0
`mggdb-0.45-8`	41	-	-	-	**296**	0.8
`mggdb-0.45-9`	41	-	-	-	**277**	47.9
`mggdb-0.45-10`	22	214	***214**	84.9	***214**	0.0
`mggdb-0.45-11`	39	278	**297**	21600.0	**297**	0.1
`mggdb-0.45-12`	21	321	**393**	21600.0	**393**	0.9
`mggdb-0.45-13`	21	371	**423**	21600.0	**423**	773.9
`mggdb-0.45-14`	16	66	***66**	102.7	***66**	0.0
`mggdb-0.45-15`	16	34	***34**	0.5	***34**	0.2
`mggdb-0.45-16`	20	70	***70**	238.2	***70**	0.0
`mggdb-0.45-17`	21	53	***53**	1.7	***53**	0.0
`mggdb-0.45-18`	25	112	**123**	21600.0	**123**	0.0
`mggdb-0.45-19`	8	48	***48**	0.9	***48**	0.0
`mggdb-0.45-20`	16	78	***78**	7.2	***78**	0.0
`mggdb-0.45-21`	24	122	***122**	10530.8	***122**	0.1
`mggdb-0.45-22`	33	-	-	-	**136**	0.0
`mggdb-0.45-23`	39	-	-	-	**145**	490.0
Sum/Average no "-"	382	3540	**3754**	7544.0	**3754**	40.8
Sum/Average all	536				5608	57.1

Table 12: Computational results on the `mggdb-0.50` instances.

		Bosco et al. (2013)			AILS (sec_{tot}=3600)	
Instance	τ	LB	z	sec_{tot}	z	sec_{inc}
`mggdb-0.50-1`	18	214	***214**	86.8	***214**	0.0
`mggdb-0.50-2`	19	233	**269**	21600.0	**269**	0.0
`mggdb-0.50-3`	19	218	***218**	399.7	***218**	0.0
`mggdb-0.50-4`	15	219	***219**	16.4	***219**	0.0
`mggdb-0.50-5`	20	292	***292**	2889.3	***292**	0.0
`mggdb-0.50-6`	17	276	***276**	1171.8	***276**	0.0
`mggdb-0.50-7`	19	265	***265**	379.0	***265**	0.1
`mggdb-0.50-8`	37	-	-	-	**310**	83.8
`mggdb-0.50-9`	41	-	-	-	**265**	11.2
`mggdb-0.50-10`	19	194	***194**	3.7	***194**	0.1
`mggdb-0.50-11`	38	249	**275**	21600.0	**275**	28.5
`mggdb-0.50-12`	19	445	***445**	13492.7	***445**	0.0
`mggdb-0.50-13`	21	214	**259**	21600.0	261	20.1
`mggdb-0.50-14`	16	75	***75**	20.6	***75**	0.1
`mggdb-0.50-15`	15	37	***37**	0.4	***37**	0.0
`mggdb-0.50-16`	19	66	***66**	36.3	***66**	0.0
`mggdb-0.50-17`	20	53	***53**	0.9	***53**	0.0
`mggdb-0.50-18`	25	111	**121**	21600.0	**121**	0.0
`mggdb-0.50-19`	8	44	***44**	0.3	***44**	0.0
`mggdb-0.50-20`	15	81	***81**	20.7	***81**	0.0
`mggdb-0.50-21`	24	86	***86**	15691.3	***86**	0.2
`mggdb-0.50-22`	31	-	-	-	**123**	0.8
`mggdb-0.50-23`	34	-	-	-	**126**	513.0
Sum/Average no "-"	366	3372	**3489**	6347.9	3491	2.6
Sum/Average all	509				4315	28.6

Table 13: Computational results on the `mgval-0.25` instances.

		Bosco et al. (2013)			AILS (sec_{tot}=3600)	
Instance	τ	LB	z	sec_{tot}	z	sec_{inc}
`mgval-0.25-1A`	54	177	***177**	0.3	***177**	3.4
`mgval-0.25-1B`	47	217	***217**	0.1	***217**	63.2
`mgval-0.25-1C`	51	-	-	-	**279**	2604.0
`mgval-0.25-2A`	40	259	***259**	6.1	***259**	8.6
`mgval-0.25-2B`	48	336	***336**	941.5	***336**	0.2
`mgval-0.25-2C`	48	-	-	-	**480**	109.1
`mgval-0.25-3A`	44	89	***89**	3.2	***89**	2.2
`mgval-0.25-3B`	41	125	***125**	7217.3	***125**	0.1
`mgval-0.25-3C`	41	130	**153**	21600.0	**153**	0.1
`mgval-0.25-4A`	89	504	**514**	21600.0	**514**	1.0
`mgval-0.25-4B`	96	507	**537**	21600.0	**537**	0.8
`mgval-0.25-4C`	100	504	**525**	21600.0	**525**	130.8
`mgval-0.25-4D`	96	-	-	-	**683**	569.8
`mgval-0.25-5A`	92	485	***485**	2418.2	***485**	3.3
`mgval-0.25-5B`	86	472	**493**	21600.0	**493**	2.7
`mgval-0.25-5C`	93	561	**584**	21600.0	**584**	1.6
`mgval-0.25-5D`	85	-	-	-	**644**	583.3
`mgval-0.25-6A`	67	274	***274**	11.8	***274**	0.0
`mgval-0.25-6B`	63	254	**263**	21600.0	**263**	9.7
`mgval-0.25-6C`	66	-	-	-	**324**	46.4
`mgval-0.25-7A`	84	297	***297**	121.3	***297**	8.4
`mgval-0.25-7B`	85	355	***355**	1859.9	***355**	1.8
`mgval-0.25-7C`	85	-	-	-	**378**	20.4
`mgval-0.25-8A`	88	507	**510**	21600.0	**510**	3.9
`mgval-0.25-8B`	84	405	**423**	21600.0	**423**	3.4
`mgval-0.25-8C`	78	-	-	-	**545**	191.6
`mgval-0.25-9A`	122	367	**371**	21600.0	**371**	9.4
`mgval-0.25-9B`	112	354	**358**	21600.0	**358**	40.7
`mgval-0.25-9C`	119	347	**365**	21600.0	**365**	146.6
`mgval-0.25-9D`	121	-	-	-	**429**	69.7
`mgval-0.25-10A`	129	492	***492**	5.1	***492**	7.1
`mgval-0.25-10B`	123	528	***528**	1.6	***528**	119.8
`mgval-0.25-10C`	125	480	**483**	21600.0	**483**	700.1
`mgval-0.25-10D`	119	-	-	-	**567**	2864.5
Sum/Average no "-"	2072	9026	**9213**	11735.5	**9213**	50.7
Sum/Average all	2669				12869	244.9

Table 14: Computational results on the `mgval-0.30` instances.

Instance	τ	Bosco et al. (2013)			AILS (sec_{tot}=3600)	
		LB	z	sec_{tot}	z	sec_{inc}
`mgval-0.30-1A`	53	170	***170**	4.4	***170**	12.0
`mgval-0.30-1B`	47	194	***194**	24.5	***194**	21.1
`mgval-0.30-1C`	48	-	-	-	**270**	347.8
`mgval-0.30-2A`	42	233	***233**	1.6	***233**	0.2
`mgval-0.30-2B`	49	347	***347**	3310.0	***347**	3.2
`mgval-0.30-2C`	45	-	-	-	**495**	416.8
`mgval-0.30-3A`	46	105	***105**	6.3	***105**	0.1
`mgval-0.30-3B`	41	115	***115**	31.3	***115**	0.0
`mgval-0.30-3C`	41	127	**153**	21600.0	**153**	2.0
`mgval-0.30-4A`	87	466	**477**	21600.0	**477**	44.8
`mgval-0.30-4B`	98	491	**533**	21600.0	**533**	19.2
`mgval-0.30-4C`	98	469	**498**	21600.0	500	34.6
`mgval-0.30-4D`	94	-	-	-	**653**	2112.2
`mgval-0.30-5A`	86	445	***445**	96.3	***445**	1.0
`mgval-0.30-5B`	83	465	**490**	21600.0	**490**	289.1
`mgval-0.30-5C`	87	513	**551**	21600.0	553	0.6
`mgval-0.30-5D`	86	-	-	-	**621**	896.8
`mgval-0.30-6A`	66	240	**252**	21600.0	**252**	0.3
`mgval-0.30-6B`	64	262	***262**	6331.6	***262**	525.0
`mgval-0.30-6C`	64	-	-	-	**320**	116.7
`mgval-0.30-7A`	77	324	***324**	1.7	***324**	2.1
`mgval-0.30-7B`	82	336	**344**	21600.0	**344**	0.8
`mgval-0.30-7C`	85	-	-	-	**354**	6.7
`mgval-0.30-8A`	88	428	**431**	21600.0	**431**	0.6
`mgval-0.30-8B`	83	391	**400**	21600.0	**400**	61.8
`mgval-0.30-8C`	75	-	-	-	**522**	9.8
`mgval-0.30-9A`	118	356	**357**	21600.0	**357**	3.7
`mgval-0.30-9B`	110	344	**348**	21600.0	**348**	1.6
`mgval-0.30-9C`	112	330	**335**	21600.0	**335**	8.5
`mgval-0.30-9D`	122	-	-	-	**430**	1807.7
`mgval-0.30-10A`	127	481	**484**	21600.0	**484**	464.2
`mgval-0.30-10B`	123	435	**441**	21600.0	**441**	2.7
`mgval-0.30-10C`	125	464	478	21600.0	**475**	1534.0
`mgval-0.30-10D`	121	-	-	-	**539**	1144.9
Sum/Average no "-"	2033	8531	**8767**	14216.3	8768	121.3
Sum/Average all	2625				12338	291.0

Table 15: Computational results on the `mgval-0.35` instances.

Instance	τ	Bosco et al. (2013) LB	z	sec_{tot}	AILS (sec_{tot}=3600) z	sec_{inc}
`mgval-0.35-1A`	47	158	***158**	0.2	***158**	7.2
`mgval-0.35-1B`	48	192	***192**	39.6	***192**	3.4
`mgval-0.35-1C`	48	-	-	-	**290**	190.8
`mgval-0.35-2A`	40	286	***286**	2.0	***286**	0.1
`mgval-0.35-2B`	46	326	***326**	135.1	***326**	4.6
`mgval-0.35-2C`	45	-	-	-	**485**	9.4
`mgval-0.35-3A`	43	84	***84**	2.9	***84**	0.4
`mgval-0.35-3B`	41	113	***113**	129.0	***113**	0.0
`mgval-0.35-3C`	40	130	**150**	21600.0	**150**	0.1
`mgval-0.35-4A`	84	415	**430**	21600.0	**430**	0.0
`mgval-0.35-4B`	90	488	**531**	21600.0	**531**	0.1
`mgval-0.35-4C`	93	480	**516**	21600.0	**516**	143.6
`mgval-0.35-4D`	96	-	-	-	**643**	501.7
`mgval-0.35-5A`	82	436	**454**	21600.0	**454**	3139.8
`mgval-0.35-5B`	81	439	**467**	21600.0	**467**	42.9
`mgval-0.35-5C`	82	539	**586**	21600.0	**586**	165.9
`mgval-0.35-5D`	80	-	-	-	**578**	2605.5
`mgval-0.35-6A`	64	248	***248**	50.2	***248**	0.2
`mgval-0.35-6B`	62	250	***250**	1715.7	***250**	0.2
`mgval-0.35-6C`	60	-	-	-	**312**	75.4
`mgval-0.35-7A`	78	264	***264**	1.2	***264**	0.8
`mgval-0.35-7B`	79	325	***325**	208.8	***325**	1.1
`mgval-0.35-7C`	82	-	-	-	**336**	59.1
`mgval-0.35-8A`	84	414	**415**	21600.0	**415**	0.2
`mgval-0.35-8B`	78	376	**385**	21600.0	**385**	43.1
`mgval-0.35-8C`	75	-	-	-	**494**	1011.9
`mgval-0.35-9A`	116	324	***324**	1071.1	***324**	4.5
`mgval-0.35-9B`	106	321	332	21600.0	**331**	113.3
`mgval-0.35-9C`	115	316	329	21600.0	**328**	78.4
`mgval-0.35-9D`	115	-	-	-	**430**	554.8
`mgval-0.35-10A`	122	475	***475**	2230.3	***475**	1338.2
`mgval-0.35-10B`	118	457	**461**	21600.0	**461**	40.0
`mgval-0.35-10C`	122	411	431	21600.0	**430**	172.8
`mgval-0.35-10D`	114	-	-	-	**523**	117.5
Sum/Average no "-"	1961	8267	8532	11455.4	**8529**	212.0
Sum/Average all	2533				11980	306.7

Table 16: Computational results on the `mgval-0.40` instances.

Instance	τ	Bosco et al. (2013) LB	z	sec_{tot}	AILS (sec_{tot}=3600) z	sec_{inc}
`mgval-0.40-1A`	48	165	***165**	1.3	***165**	0.4
`mgval-0.40-1B`	43	196	***196**	1415.0	***196**	4.2
`mgval-0.40-1C`	46	-	-	-	**263**	67.2
`mgval-0.40-2A`	38	222	***222**	1.7	***222**	0.0
`mgval-0.40-2B`	49	311	***311**	1506.0	***311**	0.2
`mgval-0.40-2C`	43	-	-	-	**469**	5.6
`mgval-0.40-3A`	41	86	***86**	0.4	***86**	0.5
`mgval-0.40-3B`	40	110	***110**	7.3	***110**	0.0
`mgval-0.40-3C`	38	120	**148**	21600.0	**148**	0.9
`mgval-0.40-4A`	82	400	***400**	11152.6	***400**	2.2
`mgval-0.40-4B`	89	395	**423**	21600.0	**423**	13.3
`mgval-0.40-4C`	89	424	**462**	21600.0	**462**	259.4
`mgval-0.40-4D`	88	-	-	-	**622**	298.8
`mgval-0.40-5A`	82	426	***426**	195.3	***426**	8.9
`mgval-0.40-5B`	77	402	**424**	21600.0	**424**	9.6
`mgval-0.40-5C`	84	488	**524**	21600.0	527	0.2
`mgval-0.40-5D`	79	-	-	-	**608**	1667.0
`mgval-0.40-6A`	61	224	***224**	150.2	***224**	3.3
`mgval-0.40-6B`	58	211	***211**	858.3	***211**	0.1
`mgval-0.40-6C`	62	-	-	-	**312**	3.1
`mgval-0.40-7A`	76	271	***271**	956.9	***271**	0.1
`mgval-0.40-7B`	77	270	***270**	1609.0	***270**	36.6
`mgval-0.40-7C`	80	-	-	-	**332**	92.8
`mgval-0.40-8A`	80	393	***393**	4331.6	***393**	7.9
`mgval-0.40-8B`	77	356	**371**	21600.0	**371**	396.4
`mgval-0.40-8C`	72	-	-	-	**517**	398.1
`mgval-0.40-9A`	114	337	**341**	21600.0	**341**	6.9
`mgval-0.40-9B`	105	319	**327**	21600.0	**327**	41.0
`mgval-0.40-9C`	104	280	**295**	21600.0	**295**	67.5
`mgval-0.40-9D`	116	-	-	-	**382**	119.9
`mgval-0.40-10A`	118	406	***406**	5107.9	***406**	0.5
`mgval-0.40-10B`	113	431	**433**	21600.0	**433**	1.2
`mgval-0.40-10C`	114	417	**432**	21600.0	433	186.3
`mgval-0.40-10D`	112	-	-	-	**482**	77.8
Sum/Average no "-"	1897	7660	**7871**	10595.7	7875	41.9
Sum/Average all	2458				11238	111.1

Table 17: Computational results on the `mgval-0.45` instances.

		Bosco et al. (2013)			AILS (sec_{tot}=3600)	
Instance	τ	LB	z	sec_{tot}	z	sec_{inc}
`mgval-0.45-1A`	47	168	***168**	7.1	***168**	0.2
`mgval-0.45-1B`	41	166	***166**	3.2	***166**	5.5
`mgval-0.45-1C`	44	-	-	-	**258**	559.0
`mgval-0.45-2A`	38	251	***251**	0.5	***251**	0.0
`mgval-0.45-2B`	46	314	***314**	1312.0	***314**	524.0
`mgval-0.45-2C`	44	-	-	-	**462**	0.4
`mgval-0.45-3A`	41	82	***82**	4.8	***82**	0.1
`mgval-0.45-3B`	39	91	***91**	3.2	***91**	2.6
`mgval-0.45-3C`	38	122	**143**	21600.0	**143**	0.9
`mgval-0.45-4A`	80	381	***381**	1198.9	***381**	197.3
`mgval-0.45-4B`	91	423	**471**	21600.0	**471**	0.6
`mgval-0.45-4C`	85	434	481	21600.0	**480**	120.7
`mgval-0.45-4D`	87	-	-	-	**577**	31.1
`mgval-0.45-5A`	78	378	**391**	21600.0	392	0.7
`mgval-0.45-5B`	75	379	**416**	21600.0	**416**	13.6
`mgval-0.45-5C`	79	445	**492**	21600.0	**492**	15.3
`mgval-0.45-5D`	76	-	-	-	**552**	1768.2
`mgval-0.45-6A`	60	213	***213**	528.5	***213**	0.9
`mgval-0.45-6B`	58	210	***210**	892.7	***210**	3.2
`mgval-0.45-6C`	58	-	-	-	**296**	14.2
`mgval-0.45-7A`	73	261	***261**	200.1	***261**	14.1
`mgval-0.45-7B`	77	290	**294**	21600.0	**294**	0.5
`mgval-0.45-7C`	76	-	-	-	**336**	612.8
`mgval-0.45-8A`	76	367	**370**	21600.0	**370**	3.2
`mgval-0.45-8B`	72	341	**360**	21600.0	**360**	10.0
`mgval-0.45-8C`	65	-	-	-	**498**	319.2
`mgval-0.45-9A`	109	299	**306**	21600.0	**306**	6.9
`mgval-0.45-9B`	100	311	**323**	21600.0	**323**	10.9
`mgval-0.45-9C`	102	273	**291**	21600.0	**291**	10.6
`mgval-0.45-9D`	108	-	-	-	**387**	1.6
`mgval-0.45-10A`	115	385	**388**	21600.0	**388**	3.4
`mgval-0.45-10B`	108	390	**399**	21600.0	**399**	1.4
`mgval-0.45-10C`	111	382	**403**	21600.0	**403**	36.4
`mgval-0.45-10D`	105	-	-	-	**487**	2888.8
Sum/Average no "-"	1839	7356	**7665**	13126.0	**7665**	39.3
Sum/Average	2370				10926	211.1

Table 18: Computational results on the `mgval-0.50` instances.

		Bosco et al. (2013)			AILS (sec_{tot}=3600)	
Instance	τ	LB	z	sec_{tot}	z	sec_{inc}
`mgval-0.50-1A`	43	145	***145**	6.28	***145**	0.0
`mgval-0.50-1B`	42	170	***170**	2.92	***170**	0.8
`mgval-0.50-1C`	40	-	-	-	**255**	62.7
`mgval-0.50-2A`	38	248	***248**	0.57	***248**	0.0
`mgval-0.50-2B`	44	284	***284**	278.96	***284**	0.1
`mgval-0.50-2C`	40	-	-	-	**464**	20.2
`mgval-0.50-3A`	40	75	***75**	1.62	***75**	1.4
`mgval-0.50-3B`	37	107	***107**	801.35	***107**	0.0
`mgval-0.50-3C`	36	117	**137**	21600.00	**137**	0.0
`mgval-0.50-4A`	78	350	***350**	144.85	***350**	0.7
`mgval-0.50-4B`	82	361	**413**	21600.00	**413**	98.0
`mgval-0.50-4C`	83	442	**488**	21600.00	**488**	6.0
`mgval-0.50-4D`	83	-	-	-	**580**	1192.1
`mgval-0.50-5A`	75	367	***367**	1296.00	***367**	8.5
`mgval-0.50-5B`	73	348	**378**	21600.00	**378**	24.0
`mgval-0.50-5C`	74	417	459	21600.00	**457**	6.8
`mgval-0.50-5D`	72	-	-	-	**541**	810.2
`mgval-0.50-6A`	53	210	***210**	123.21	***210**	0.6
`mgval-0.50-6B`	56	210	***210**	368.05	***210**	0.0
`mgval-0.50-6C`	55	-	-	-	**293**	4.3
`mgval-0.50-7A`	74	248	***248**	126.78	***248**	3.8
`mgval-0.50-7B`	71	276	***276**	1429.50	***276**	0.3
`mgval-0.50-7C`	74	-	-	-	**320**	9.8
`mgval-0.50-8A`	74	382	**388**	21600.00	**388**	0.4
`mgval-0.50-8B`	68	330	**350**	21600.00	**350**	27.4
`mgval-0.50-8C`	63	-	-	-	**501**	24.7
`mgval-0.50-9A`	105	306	***306**	4.05	***306**	44.6
`mgval-0.50-9B`	97	262	**278**	21600.00	**278**	22.2
`mgval-0.50-9C`	100	279	301	21600.00	**292**	55.8
`mgval-0.50-9D`	103	-	-	-	**358**	1674.6
`mgval-0.50-10A`	109	378	**385**	21600.00	**385**	42.9
`mgval-0.50-10B`	110	364	**369**	21600.00	**369**	1171.2
`mgval-0.50-10C`	108	389	**406**	21600.00	**406**	47.5
`mgval-0.50-10D`	110	-	-	-	**457**	251.8
Sum/Average no "-"	1770	7065	7348	10551.4	**7337**	62.5
Sum/Average all	2285				10536	165.1

Table 19: Average percentage above the BKS for top-performing CARP algorithms in the literature.

Algorithm	Problem set `gdb`	`val`	`egl`	`C`	`D`	`E`	`F`
GLS	0.000	0.032	–	0.047	0.011	0.098	0.000
MA	0.025	0.132	0.805	–	–	–	–
BACO	0.154	0.351	2.348	–	–	–	–
VNS	–	0.056	0.538	–	–	–	–
TSA	0.070	0.100	0.725	0.054	0.164	0.168	0.249
Ant-CARP	0.102	0.083	0.558	0.210	0.083	0.360	0.199
MA	0.285	–	–	–	–	–	–
AILS	0.000	0.053	0.330	0.018	0.000	0.228	0.000

Table 20: Average percentage above the BKS for top-performing CVRP algorithms in the literature.

Algorithm	Problem set Christofides et al. (1969, 1979)	Taillard (1993)	Golden et al. (1998)	Li et al. (2005)
GRASP	0.071	–	0.525	–
MB	0.027	0.236	0.263	0.202
MA-CVRP	0.030	0.096	0.210	–
PARALLEL	0.085	0.131	0.411	0.299
MA	0.389	–	–	–
AILS	0.073	0.180	1.063	0.489

SINTEF

Technology for a better society

www.sintef.no