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 Sciences 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 including improvement of the best known upper bound for one instance.

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, further intensification is performed by local search (LS).

We have designed a new, aggressive LS strategy. In each iteration we explore a large neighborhood consisting of the union of moves from five operators, and investigate all moves with positive savings. The effect is that 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 381 best known solutions, 108 of which are new. For three instances, AILS closes the gap for the first time. This brings the number of proven optimal solutions up to 234. AILS fails to find only ten of these. 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 configure and 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.

A 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 task $t \in N_r \cup E_r \cup A_r$ 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.

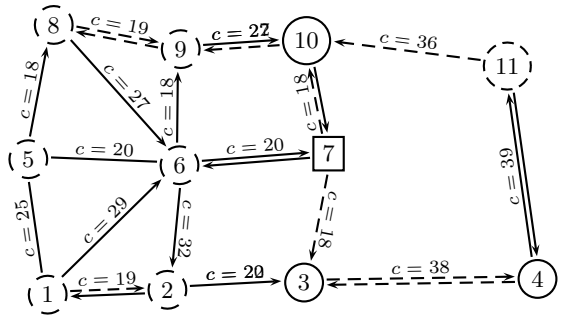
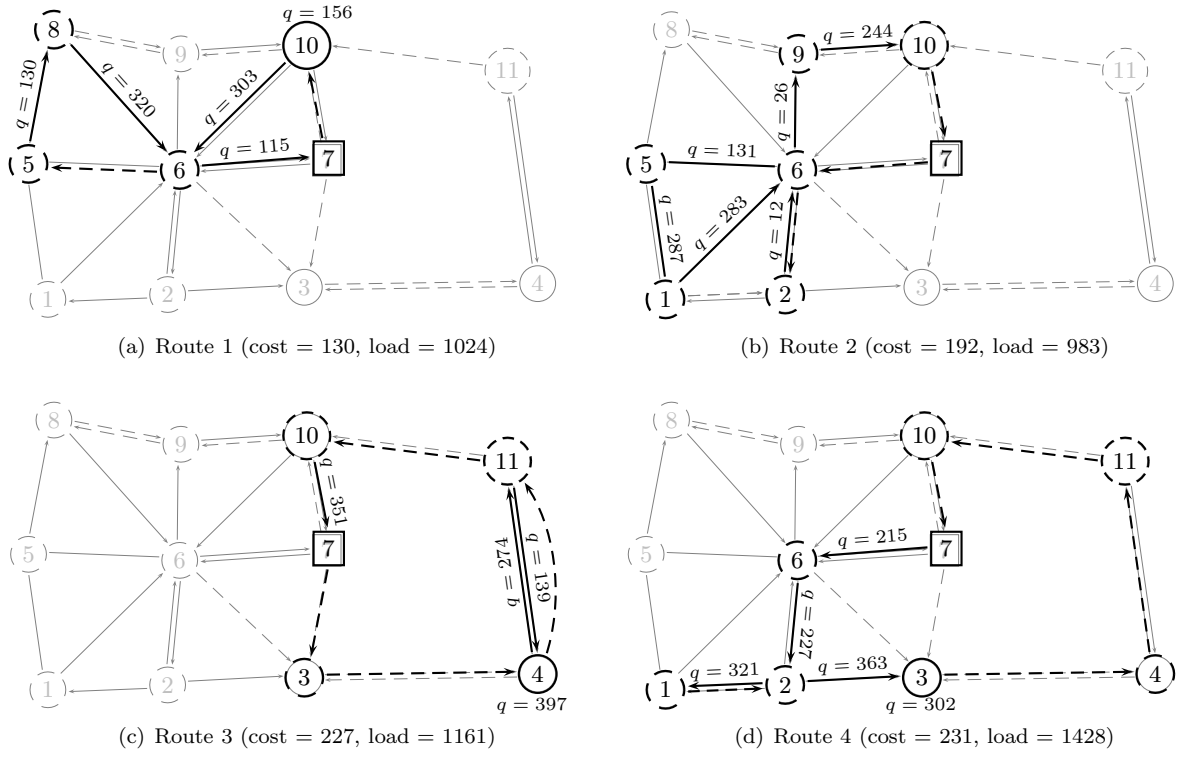


Figure 1: A MCGRP example: Instance `CBMix23`.

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-



(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`.

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 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 proposed by the authors: 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, with 138 instances, was derived from the `gdb` undirected CARP test set. The `mval` mixed CARP dataset is the origin of the `mgval` benchmark that has 204 instances. The authors considered only the 264 instances involving at most seven vehicles in their experiments. For these, the method provided 154 proven optimal solutions. The authors also tested their algorithm on the four smallest-size `CBMix` instances, providing two optimal solutions.

Bach et al. (2014) proposed a branch-and-cut-and-price method for the MCGRP and investigated its performance on `mggdb`, one subset of `mgval`, `CBMix`, and `BHW`. At the time, they proved optimality for 31 new `mggdb` instances. On `CBMix`, they improved the best known upper bound for two instances and the best known lower bound in 21 cases. Optimality was proven for two new `BHW` instances.

With the work reported in Bosco et al. (2013) as basis, Irnich et al. (2015) presented a new two-index mathematical model for the MCGRP, and a two-phase branch-and-cut algorithm that utilizes an aggregate formulation to develop an effective lower bounding procedure. They provided numerical results for three of the six parameter subsets of the `mggdb` and `mgval` benchmarks. 124 of the 171 instances investigated are solved to optimality.

A matheuristic for the MCGRP was proposed by Bosco et al. (2014). The authors provided numerical results for the `mggdb`, `mgval`, and `CBMix` benchmarks. A bi-criteria extension of the MCGRP, where the second criterion is route balance, was studied for the first time by Mandal et al. (In press). A memetic algoritm is proposed, and numerical results are reported for the `CBMix` benchmark.

In Section 4, we provide a comprehensive survey of numerical results for the `CBMix`, `BHW`, `DI-NEARP`, `mggdb`, and `mgval` MCGRP benchmarks. We also refer to the Transportation Optimization Portal (TOP) web site SINTEF that attempts to maintain an updated survey of the best known numerical results for all MCGRP benchmarks.

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 (2002), Corberán et al. (2007) and references therein.

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 these ends, we use the idea of *Iterated Local Search* (ILS) (see, e.g., Lourenço et al. (2010)) that mainly consists of improving

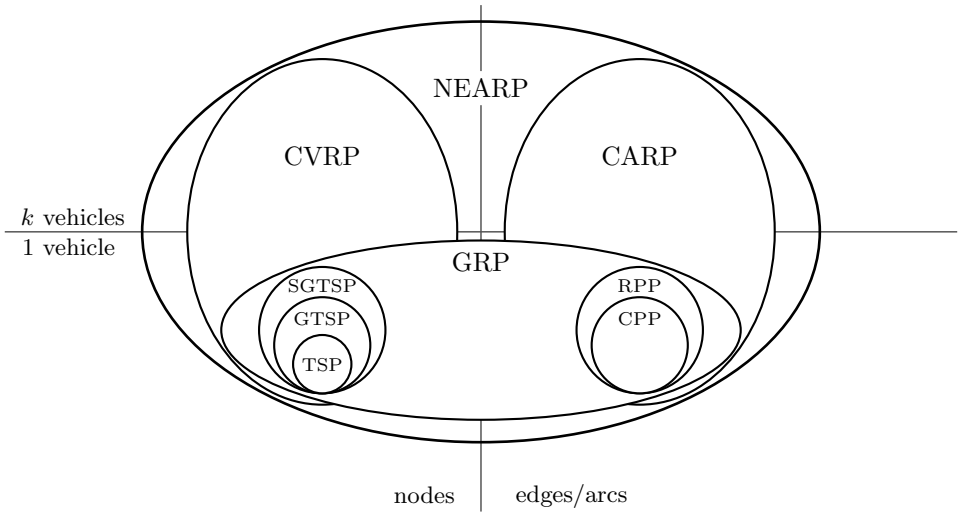


Figure 3: A graphical representation of a problem classification.

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 3.1. It contains several innovations and non-standard mechanisms. If the solution resulting from a destroy/repair operation has good quality, the search is further intensified through local search with five neighborhood operators, and a new, aggressive move selection strategy. Details are given in Section 3.2. The main diversification mechanism of AILS is a major disruption – a *kick* – applied when a certain search effort has been made without acceptance of a new solution.

Pseudocode for the AILS algorithm, the structure of which is quite simple, is given in Algorithms 1 and 2, with explanation below. Parameters, functions, and procedures used by AILS are briefly described, with

final values listed in Table 1. We refer to Section 4.1 for further description of the configuration process.

Algorithm 1 Adaptive Iterative Local Search

```
1: function AILS(Instance)
2:     comment: Initialize global variables
3:     IterPerStage := N_ITER_PER_STAGE
4:     k_max := K_MAX(Instance)
5:     KickCountdown :=ITER_BEFORE_KICK(Instance)
6:     INITIALIZE_ROULETTE_PROBABILITIES( )
7:     comment: Construct first solution and take to deep local optimum
8:     x_init :=CONSTRUCT_INITIAL_SOLUTION(Instance)
9:     x_incumbent :=LS_FULL(x_init)
10:    x_LocalIncumbent := x_incumbent
11:    comment: Main body: iterative phase
12:    repeat
13:        x_current := x_BestThisStage := x_LocalIncumbent
14:        comment: Execute a batch of intensifying iterations – a stage
15:        IterationCounter := 0;
16:        while IterationCounter < IterPerStage do
17:            IterationCounter := IterationCounter + 1
18:            NewStage := COMBINED_ALNS_AND_LS( )
19:            if NewStage then
20:                IterPerStage := N_ITER_PER_STAGE - 1
21:                KickCountdown := iter_BeforeKick
22:                IterationCounter = IterPerStage
23:            end if
24:        end while
25:        comment: Increase number of iterations
26:        IterPerStage := IterPerStage + 1
27:    until TIMEOUT( )
28:    return x_incumbent
29: end function
```

Algorithm 2 Combined ALNS and LS

```
1: function COMBINED_ALNS_AND_LS( )
2:     comment: Returns TRUE if a new stage is required, FALSE otherwise
3:     comment: Check for stagnation
4:     if KickCountdown > 0 then
5:         k :=RANDOM(1, k_max)
6:         x_current :=ROULETTE_DESTROY_AND_REPAIR(k)
7:         if z(x_current) = z(x_BestThisStage) return FALSE
8:         comment: Intensify with LS_FULL, LS1, or LS2 based on random choice and instance size
9:         if RANDOM(0,1)< p_LS_Full then
10:            LS_FULL(x_current)
11:        else
12:            if τ < 200 then LS1(x_current) else LS2(x_current)
13:        end if
14:        KickCountdown := KickCountdown − 1
15:        if z(x_current) < z(x_BestThisStage) then
16:            x_BestThisStage := x_current
17:            comment: Give higher probability to selected Destructor/Constructor pair
18:            UPDATE_ROULETTE_PROBABILITIES( )
19:            UPDATE_INCUMBENTS(x_current)
20:            return TRUE
21:        end if
22:        return FALSE
23:    else
24:        comment: Nothing has happened for a while, make a major, random destroy and repair
25:        k :=RANDOM(τ/2, τ)
26:        x_LocalIncumbent :=RANDOM_DESTROY_AND_REPAIR(k)
27:        x_current := x_LocalIncumbent :=LS_FULL(x_LocalIncumbent)
28:        UPDATE_INCUMBENTS(x_current)
29:        return TRUE
30:    end if
31: end function
```

AILS finds a first feasible solution with simple, fast heuristics. 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 a given number of iterations. Each iteration consists of a destroy and repair cycle, possibly followed by further intensification with local search. 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 *stagnation* occurs, i.e., no new solution has been accepted for a certain number of iterations that depends on the computational complexity of the instance. We refer to Table 1 and Section 4.1 for details. The kick utilizes the random destructor and random constructor operators from the ALNS, see Section 3.1. The resulting solution is taken

to a deep local optimum before a new intensification stage is started.

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 (Algorithm 1, 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 savings, 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 insertion 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 for intensification. As acceptance criterion for a new solution, we use simple improvement. 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 a progressive number of iterations was needed to reinforce the intensification.

Table 1: Final parameter setting

Parameter/function	Value
N_ITER_PER_STAGE	10
$k_{MAX}(Instance)$	$min(\tau - 2, 50)$
β	0.75
γ	0.1
$p_{LS_{Full}}$	0.15
ITER_BEFORE_KICK($Instance$)	20.000 * $\max\{1, 20.000/(\tau^2 + \|A\| + 2\|E\| + n^2/5)\}$

Below follows a description of the functions and procedures used in Algorithms 1 and 2.

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

INITIALIZE_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.

The following procedures and functions are used in in COMBINED_ALNS_AND_LS.

RANDOM_DESTROY_AND_REPAIR: line 26, is called to make a *kick* after a certain amount of search effort has been performed without acceptance of a new current solution. It calls the Random-Destructor and then the Random-Constructor for a number of tasks randomly drawn from 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 a randomly drawn k value, see line 6.

UPDATE_ROULETTE_PROBABILITIES: line 18, increases the probability of selecting a successful Destructor/Constructor pair.

UPDATE_INCUMBENTS: line 19, checks whether the current solution is better than the best solution found, 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.

3.1 The Adaptive Large Neighborhood Search Component

The Adaptive Large Neighborhood Search mechanism in AILS is a modified and simplified version of the one proposed by Pisinger and Røpke (2007). 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 4.2) show that it 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 reinforcement learning mechanism in AILS 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. A scores matrix π

contains a measure for the effectiveness of each pair of Destructor and Constructor operators. The score element π_{ij} is associated with the i-th Destructor and the j-th Constructor. AILS uses a very simple score update mechanism. 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. The scores matrix π is never reset in AILS, as focused experiments with different reset models never gave significant improvement.

In each iteration, a number k is used as parameter to the randomly selected pair of operators. As is common in the literature, k is randomly drawn from a uniform distribution over an interval $[k_{min}, k_{max}]$. Early experiments revealed that it is beneficial to have a finite probability of selecting very small k values, so we set k_{min} to 1. Common ALNS insight indicates that k_{max} should be an increasing function of instance size, but with an upper limit to avoid excessive computational burden. Focused experiments confirmed that this model is effective, and we found the best upper limit value for k_{max} to be 50. The final form of the function to determine k_{max} is found in Table 1.

Another important simplification relative to the "canonical" ALNS is that the AILS design uses a simple improvement criterion for acceptance of new solutions, rather than a more complex criterion that is common in the literature.

AILS draws upon 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 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. !!! We need to add more detail. Do we select a task randomly, compute the distance to all other tasks, and then select the k closest ones???

5. **Tree-Destructor**: this is a new operator which is particularly effective for MCGRP instances that contain all three task types. It utilizes the instance graph, as illustrated in Figure 1. First, it randomly selects a task as a root node, and then grows a tree in the instance graph G, 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 are used in 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. When the instances have a fixed number of vehicles, as those in Bosco et al. (2013), we manage this particular case by adding a penalty for each route, so that minimizing the objective function also reduces the number of routes. The feasibility check is then modified by including a control on the number of vehicles used.

- **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 Røpke and Pisinger (2006), to overcome the myopic behavior of greedy repair.

3.2 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 five 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 in Algorithms 1 and 2, 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 three situations call for different LS variants. After initial construction, and after a kick, the goal is to find a deep local optimum fast. Therefore, we utilize the most powerful local search variant called LS_Full that includes all move types. During the ALNS destroy and repair phase, we have seen through experiments that it becomes too expensive to use LS_Full all the time. Therefore, we designed two reduced local search variants: LS1 and LS2. The details of these are given below. LS1 is used for small instances ($\tau < 200$), for larger instances, we use LS2. However, with a small probability $p_{LS_{Full}}$, see Table 1, we invoke the full local search, regardless of instance size. These choices were made after extensive experiments on standard MCGRP, CARP, and CVRP benchmarks.

AILS utilizes the following set of local search operators:

- **Swap**: exchanges the position of two tasks (both intra and inter-route);
- **Or-opt**: breaks a route in three points, then reconnects 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; !!! I propose the following alternative text: relocates a segment of tasks, either within a route, or from one route to another. The length of the segment to be relocated is limited to l tasks; ??? Do we also include reversal of the segment? Probably not, as there are only two subtypes ???
- **2-opt**: breaks a route cycle in two segments and re-connects the segments in the only possible way. We adapted to the MCGRP also the seven additional operator subtypes originally proposed by Santos et al. (2010) for the CARP. They result from breaking two different routes in one point each, and reconnecting the segments in all possible ways when reversals are also considered;
- **3-opt**: breaks a route cycle in three segments A, B and C, and reconnects them in all possible ways, also allowing reversals. There are seven combinations: $AC\bar{B}$, $A\bar{C}B$, $A\bar{C}\bar{B}$, $\bar{A}CB$, $\bar{A}C\bar{B}$, $\bar{A}\bar{C}B$ and $\bar{A}\bar{C}\bar{B}$, where $\bar{X}$ denotes the reversal of X. There are six types of intra-route moves (case $A\bar{C}\bar{B}$ is equal to intra-route 2-opt), and seven types of inter-route moves;
- **Flip**: reverses the direction of all the edge tasks of a route.

The Flip operator was proposed for the MCGRP by Prins (2009).

Hence, AILS uses five operators with a total of 26 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 these operators and subtypes. 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: Swap, 2-opt, 3-opt with $|B| \leq 1$, and Flip. This limits the search to 18 operator subtypes. LS2 is our cheapest local search variant. It consists of Or-opt with $l = 2$, Swap, and 2-opt, covering 12 operator subtypes.

4 Computational Experiments

We coded our algorithm in C++ and ran it on an Intel Xeon E5530 at 2.4 GHz with 23.5 GB of memory, under the Linux Ubuntu 12.04.1 LTS 64-bit operating system. In Section 4.1, we describe AILS parameter

configuration, before we move on to a quantitative investigation of main AILS mechanisms in 4.2. The section finishes with a description of extensive computational experiments on standard benchmarks from the literature, for MCGRP, as well as the CARP and CVRP special cases.

4.1 Configuration of the Algorithm

AILS has six main parameters. Through a combination of analysis, insights from the literature, and extensive computational experiments on `CBMix`, `BHW`, and `DI-NEARP` instances (see Section 2), we determined a good setting. We refer to Table 1 and explanations in Section 3. Further details of the parameter tuning are not given here, except for an account of how we determined the final form of the iteration limit for the kick.

The ITER_BEFORE_KICK function determines the number of iterations without improvement before the kick is invoked. Initially, we performed a set of experiments with a large sample of MCGRP instances to determine the best constant value for ITER_BEFORE_KICK. We tried the following values: 5000, 10.000, 20.000, 30.000, 50.000, and 100.000. Results showed that a good overall choice that balances the potential for further intensification with the potential benefits of diversification is 20.000 iterations. Disabling the kick gave considerably worse average results. For very large instances and reasonable timeout values, the kick will never be invoked within 20.000 iterations, which seems reasonable as the AILS intensification mechanism will not have enough time to stagnate. However, we observed that for small instances, a larger number of iterations before the kick gave better results, indicating that with a 20.000 iteration limit, intensification was interrupted prematurely in many cases.

This observation led us to the conclusion that the iteration limit for the kick should be determined according to the computational complexity of the instance. We identified τ, $|A|$, $|E|$ and n as the main instance metrics of computational complexity. Using a sample of 17 MCGRP benchmark instances (see Section 4.2 for details), we determined a function of these metrics that has a similar behavior to the CPU time required to complete one iteration of AILS.

!!!!! I have removed the paragraph that followed, including Table 2, as proposed by referee 1 !!!!!

Further computational experiments, where small variations of the six main AILS parameters were investigated, revealed only minor changes in performance. For a discussion of AILS robustness with respect to choice of random seed, we refer to Section 4.2.

4.2 Investigation of Main AILS Mechanisms

The special combination of ALNS with conditional LS for further intensification is a main feature of AILS. Further, the Tree-Destructor is an innovation, a new operator in the ALNS component of AILS. This subsection reports from a quantitative assessment of the merit of these two major mechanisms.

For a comprehensive assessment, we selected a sample of 17 MCGRP instances from the `CBMix`, `BHW`, and `DI-NEARP` benchmarks. The sample contains instances of different size and structure, with τ varying from 91 to 833. In general, they are instances that seemed hard from early experiments. The names of the sample instances and their τ values are found in the result tables below. To have a larger sample, also investigating the robustness of AILS with respect to random seed, we ran all experiments with ten different seed values. Thus, the total sample size is 170. For all experiments, a CPU time limit of 3600 seconds was imposed.

4.2.1 Removing the Local Search Component

We disabled the LS component and ran experiments on the 17 sample MCGRP instances. To remove the unpredictable effect of the kick mechanism that would make comparison more difficult, we disabled it and ran two experiments:

- AILS without the kick (*Basic Configuration*)
- AILS without the kick and with LS disabled

The results are found in Table 2. The first two columns give the instance names and their τ values. The following five columns give the results from AILS without the kick, here called the "Basic Configuration". For each instance, the minimum, maximum, and average cost over the ten random seeds for the best solution found within the timeout are given. As usual, the constant sum of service costs is not included. The column marked "RSD%" gives the relative standard deviation for the ten cost values, in percent. The average CPU time (seconds) to find the solutions is given in the following column marked sec_{inc}. The next five columns, marked "No Local Search", give the corresponding results for the Basic Configuration with LS disabled. The rightmost column marked "Δ%" gives the percentage difference of the average results relative to the Basic Configuration.

We make two important observations from these results. First, the Local Search Component contributes substantially to the performance of AILS. The average deterioration for the sample when the LS component is

disabled is more than 5%. There is a consistent deterioration for all instances, and the maximum deterioration is almost 11%.

Second, the RSD values are low for all instances of the sample. For the Basic Configuration, the average RSD is 0.31%, with a maximum of 0.62%. Without the LS component, the average variation is slightly higher at 0.48% with a maximum of 0.94%, which is still low. This is a clear indication that the AILS is robust with respect to selection of random seed.

4.2.2 Removing the Tree-Destructor

We investigated the merit of the novel Tree-Destructor operator in a similar way as for the LS component. Table 3, which has the same layout as Table 2, shows a comparison between the Basic Configuration as defined in Section 4.2.1 above, and the Basic Configuration with the Tree-Destructor disabled. Again, the results show a small variation over ten different seed values. Although the version without the Tree-Destructor only shows a small average deterioration relative to the Basic Configuration, a deterioration is observed for 16 out of 17 instances of the sample. Moreover, when we compare the best cost values found over the ten seeds (in bold), we observe that the Basic Configuration is slightly better.

Based on the results and observations presented we decided to use the Basic Configuration, but with the kick enabled, for our final computational experiments. For all experiments presented below, the timeout sec_{tot} for AILS was set to 3600 seconds. For each instance, we also report the CPU time sec_{inc} to find the solution with the given cost.

4.3 Results on MCGRP benchmarks

All five MCGRP benchmarks that currently exist in the literature were used for empirical assessment of AILS, 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 27 to 332 links. 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`

Table 2: Assessment of the Local Search Component of AILS

		Basic Configuration					No Local Search					
Instance	τ	Min cost	Max cost	Avg. cost	RSD%	sec_{inc}	Min cost	Max cost	Avg. cost	RSD%	sec_{inc}	Δ%
CBMix4	98	7450	7569	7507.0	0.50	2366.11	7942	8104	8041.1	0.60	263.46	7.11
CBMix7	168	9398	9601	9491.9	0.55	2005.13	10072	10240	10184.0	0.52	890.30	7.29
CBMix8	177	10305	10474	10370.6	0.49	2782.04	11100	11383	11206.4	0.72	693.50	8.06
CBMix15	91	8214	8271	8240.5	0.24	1567.61	8796	9031	8945.7	0.74	1058.23	8.56
CBMix16	169	8714	8743	8734.4	0.15	1726.36	9374	9699	9532.9	0.94	1725.96	9.14
CBMix19	212	16159	16374	16253.1	0.45	2530.33	17686	18022	17833.3	0.52	747.66	9.72
BHW9	178	875	889	879.8	0.43	2339.45	971	981	976.4	0.33	1351.76	10.98
BHW12	115	10873	10932	10893.9	0.14	1742.89	11264	11492	11387.2	0.58	2041.11	4.53
BHW15	128	15371	15446	15403.7	0.14	2006.11	15804	15950	15867.5	0.29	20.70	3.01
BHW16	410	43837	44118	43982.8	0.24	3343.97	45738	46397	46023.6	0.39	69.87	4.64
BHW20	293	16215	16385	16279.7	0.36	2566.36	17320	17828	17603.4	0.82	1334.55	8.13
DI-NEARP-n240-Q4k	240	18181	18266	18202.6	0.15	2678.55	18858	19091	18947.6	0.32	29.70	4.09
DI-NEARP-n422-Q8k	422	14442	14442	14442.0	0.00	621.47	14653	14705	14677.0	0.10	12.76	1.63
DI-NEARP-n442-Q8k	442	43249	43360	43270.9	0.07	1495.41	44033	44294	44105.2	0.21	70.60	1.93
DI-NEARP-n477-Q2k	477	22880	23104	22973.7	0.28	2069.61	23724	24030	23945.6	0.39	81.17	4.23
DI-NEARP-n699-Q4k	699	39761	40505	40185.7	0.62	2936.83	41618	42326	42118.2	0.46	182.31	4.81
DI-NEARP-n833-Q16k	833	32353	32852	32517.9	0.49	3191.94	34567	34862	34754.5	0.26	342.72	6.88
Total/average		318277	321331	319630.2	0.31	2233.54	333520	338435	336149.6	0.48	642.14	5.17

Table 3: Assessment of the AILS Tree-Destructor

		Basic Configuration					No Tree Destructor					
Instance	τ	Min cost	Max cost	Avg. cost	RSD%	sec_{inc}	Min cost	Max cost	Avg. cost	RSD%	sec_{inc}	Δ%
CBMix4	98	**7450**	7569	7507.0	0.50	2366.11	7475	7572	7529.7	0.37	2780.88	0.30
CBMix7	168	**9398**	9601	9491.9	0.55	2005.13	9436	9564	9494.1	0.45	1936.40	0.02
CBMix8	177	**10305**	10474	10370.6	0.49	2782.04	10326	10482	10390.2	0.46	2255.34	0.19
CBMix15	91	**8214**	8271	8240.5	0.24	1567.61	8221	8285	8263.6	0.21	1490.54	0.28
CBMix16	169	8714	8743	8734.4	0.15	1726.36	**8710**	8755	8738.4	0.20	2282.77	0.05
CBMix19	212	16159	16374	16253.1	0.45	2530.33	**16045**	16390	16260.6	0.69	2761.19	0.05
BHW9	178	875	889	879.8	0.43	2339.45	**873**	889	880.4	0.52	2153.09	0.07
BHW12	115	**10873**	10932	10893.9	0.14	1742.89	10882	10932	10906.3	0.14	1817.34	0.11
BHW15	128	15371	15446	15403.7	0.14	2006.11	**15370**	15488	15423.6	0.22	2086.28	0.13
BHW16	410	43837	44118	43982.8	0.24	3343.97	**43836**	44286	43996.2	0.32	3181.37	0.03
BHW20	293	16215	16385	16279.7	0.36	2566.36	**16197**	16378	16260.4	0.33	3154.39	-0.12
DI-NEARP-n240-Q4k	240	**18181**	18266	18202.6	0.15	2678.55	**18181**	18272	18205.9	0.17	2214.66	0.02
DI-NEARP-n422-Q8k	422	**14442**	14442	14442.0	0.00	621.47	**14442**	14467	14447.0	0.07	1341.39	0.03
DI-NEARP-n442-Q8k	442	**43249**	43360	43270.9	0.07	1495.41	43264	43360	43302.7	0.11	923.48	0.07
DI-NEARP-n477-Q2k	477	**22880**	23104	22973.7	0.28	2069.61	22896	23099	23006.9	0.30	2277.07	0.14
DI-NEARP-n699-Q4k	699	**39761**	40505	40185.7	0.62	2936.83	39795	40592	40258.9	0.64	2735.50	0.18
DI-NEARP-n833-Q16k	833	**32353**	32852	32517.9	0.49	3191.94	32414	32785	32580.7	0.34	3108.61	0.19
Total/average		**318277**	321331	319630.2	0.31	2233.54	**318363**	321596	319945.6	0.32	2264.72	0.10

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. Both consist of six subsets, each corresponding to a specific value of the parameter $\beta \in \{0.25, 0.30, 0.35, 0.40, 0.45, 0.50\}$ that controls the number of required links in the original CARP instance that have been shifted to adjacent vertices. Each of the six `mggdb` subsets has 23 instances with between 18 and 48 tasks, and between 3 and 10 vehicles. For `mgval`, each of the six subsets has 34 instances. The number of tasks is between 38 and 129, and fleet size varies between 2 and 10.

In Table 4 we compare upper bounds for the `CBMix` instances yielded by all five MCGRP approximation methods that we have found in the literature, namely:

- MA: the memetic algorithm by Prins and Bouchenoua (2005), run on an Intel Pentium III at 1.0 GHz;
- SA: the simulated annealing algorithm by Kokubugata et al. (2007), run on an Intel Pentium IV at 1.8 GHz;
- Spider: the commercial VRP solver tested in Hasle et al. (2012), run on an Intel Core i7 at 3.07 GHz;
- MH: a matheuristic proposed in Bosco et al. (2014), run on an Intel Xeon Quad at 3.0 GHz;
- AILS: our proposed metaheuristic, run on an Intel Xeon E5530 at 2.4 GHz.

For MA and SA, the termination condition was the number of iterations without new accepted solutions. Spider, MH, and AILS were stopped after a given CPU time limit. MA, Spider, MH, 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. In each line of Table 4, we give the name of the instance addressed, the τ value, the best known lower bound (column LB) from Bach et al. (2013), Bosco et al. (2013), Bach et al. (2014),

and Irnich et al. (2015), as well as upper bounds and CPU times yielded by the branch-and-cut-and-price exact method of Bach et al. (2014) (columns marked B&C&P). The latter results were obtained with an Intel Core 2 Duo CPU P8700 at 2.53 GHz, and CPLEX 12.4 to solve the LP-relaxation. The time limit sec_{tot} was six hours. Note that for all exact method results, the reported CPU times are equal to the time limit, unless an optimal solution has been proven, as the time to find the best upper bound is not known. For MA, we give the solution value z it obtains, and the 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 sec_{tot} 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}. For MH, the time limit has not been reported. AILS was given a time limit of one CPU hour. For both, we report the cost and the time sec_{inc} in seconds needed to reach the reported solution value. The bottom three lines give, for each method, the number of optimal upper bounds, the number of best known upper bounds (including optimal values), and the number of instances for which no feasible solution has been found within the time limit. The best objective function values obtained are reported in bold. We observe that the AILS is very effective on the `CBMix` benchmark. It produces all best known solutions but one, 19 of them for the first time. The sum of objective values over all instances is 1.8% lower than for the best competitor MH, even though the average CPU time to find the reported solutions is substantially lower on a similar computer.

In Table 5 we present the results we obtained on the `BHW` test set, in a similar way as in Table 4. Here, the lower bounds reported are the best among the two exact methods that have been tested on this benchmark, and we refer to Bach et al. (2013), and Bach et al. (2014). For `BHW`, the only competing approximation method is Spider. We observe that the branch-and-cut-and-price of Bach et al. (2014) yields four proven optimal solutions. For these instances, optimal solutions are also found by both approximation methods. In addition, the Bach et al. (2014) method provides three upper bounds that are not competitive, and no upper bound for the remaining 13 instances. Compared to the competition, AILS provides better or equally good solutions on all `BHW` instances, with 1.8% lower total cost than Spider, and 15 new best solutions. `BHW5` was closed for the first time by AILS.

In Table 6 we compare again the performance of AILS with Spider, this time on the `DI-NEARP` benchmark. The lower bounds are taken from Bach et al. (2013), as no exact method has been tested so far on this set of 24 large-size instances. Also here, the performance of the AILS is good. Indeed, it yields novel best known

solutions to all but two instances, lowering the total cost with 1.4% 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 compare the performance of AILS on the 138 `mggdb` instances with all (to our knowledge) competitors in the literature:

- B&C: the branch-and-cut method by Bosco et al. (2013), run on two Intel Xeon Quad at 3 GHz, using CPLEX 12.2;
- B&C&P: the branch-and-cut-and-price exact method of Bach et al. (2014), run on Intel Core 2 Duo CPU P8700 at 2.53 GHz, and CPLEX 12.4;
- B&C2: the branch-and-cut method by Irnich et al. (2015), with the same hardware and basic software as for B&C;
- MH: a matheuristic proposed in Bosco et al. (2014), run on an Intel Xeon Quad at 3.0 GHz;
- AILS: our proposed metaheuristic, run on an Intel Xeon E5530 at 2.4 GHz.

.

B&C2 has been run only on the first three of the `mggdb` subsets. We report CPU times for B&C2 for these instances, and for B&C&P for the remaining instances. Again, note that the reported CPU times for exact methods are equal to the time limit, unless an optimal solution has been proven. The lower bounds are the best ones among the three exact methods reported in the MCGRP literature. For details, we refer to Bosco et al. (2013), Bach et al. (2014), and Irnich et al. (2015).

For 121 of the 138 `mggdb` instances, optimal solutions have been proven by the three exact methods. For 114 of these, AILS finds solutions with optimal upper bounds, in less than one second for most of the cases. For 128 of the 138 instances, AILS finds a solution with the best known upper bound. The MH matheuristic finds 65 optimal solutions and 67 best known upper bounds. The average CPU time for finding the reported solutions is 18.5 seconds for MH and 7.5 seconds for AILS. Over the `mggdb` benchmark, AILS provides 1.4% better upper bounds than MH. The average gap between the AILS upper bounds and the best known lower bounds is less than 0.7%.

Similarly, in Tables 13–18, we compare results for a total of 204 `mgval` instances for the same methods as for `mggdb`, and AILS. Again, the B&C2 has been run only on the first three subsets. B&C&P has provided results on the last subset ($\beta = 0.50$) only. For the exact methods, CPU times are reported for B&C2 for the first three subsets, for B&C&P for the last subset, and for B&C for the remaining two subsets. In total, 105 optimal values have been proven for `mgval`. MH finds 84 of these, whereas AILS finds 102. MH produces 103 best known upper bounds versus 189 for AILS. The average gap to the best lower bound is 4.2% for MH and 1.9% for AILS. Note that no non-trivial lower bound is known for 18 of the `mgval-0.40` and `mgval-0.45` instances. The average CPU time for finding the reported solutions is 1099.2 seconds for MH and 93.7 seconds for AILS.

We conclude from the observations presented above that AILS is a highly competitive approximation method for the MCGRP. Over a total of 409 instances for the five MCGRP benchmarks used in the literature, it yields 381 best known upper bounds and 224 out of 234 currently known optimal solution values within a reasonable time limit.

4.4 Results on CVRP and CARP Instances

In tables 19 and 20, we compare AILS with some of the best performing approximation methods for the CARP and the CVRP, according to the recent surveys in Prins (2014) (Section 7.6), and Laporte et al. (2014) (Section 4.7), respectively. Note that the reported gap values are calculated relative to the best known solutions as of 2012, some of which have been improved since then, but relative performance should be clear. Again, we refer to Prins (2014) and Laporte et al. (2014) for updated details on a subset of the benchmarks.

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 six highly competitive CARP metaheuristics:

- 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 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 the method has not been tested on this benchmark. We also compare with MA, the memetic algorithm for the MCGRP by Prins and Bouchenoua (2005), run on an Intel Pentium III at 1.0 GHz. 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 four of 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);

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. AILS finds a new best known solution for the `D151-14c` (`CMT9`) instance proposed in Christofides et al. (1979) in 180.1 CPU seconds. The detailed solution is given in Appendix B.

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, three exact methods, and a matheuristic have been proposed.

In this paper, we report 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 64 of 67 instances of the `CBMix`, `BHW`, and `DI-NEARP` benchmarks in reasonable time, 55 of which are new. For the smaller size `mggdb` and `mgval` benchmarks designed to investigate exact methods, AILS finds 317 of 342 best known upper bounds, and 216 out of 226 proven optimal solutions. 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, in fact improving the best known solution for the much studied `D151-14c` CVRP instance proposed by Christofides et al. (1979).

Acknowledgements

This work has been partly financed by the Research Council of Norway, under contracts 176869/V30, 187293/I40, 205298/V30, and 217108/I40. We thank two anonymous referees whose comments have greatly improved the quality of this paper.

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.

L. Bach, J. Lysgaard, and S. Wøhlk. A Branch-and-Cut-and-Price Algorithm for the Mixed Capacitated General Routing Problem. In *Lukas Bach: Routing and Scheduling Problems - Optimization using Exact and Heuristic Methods*, Ph.D. Thesis, chapter 2, pages 37–75. Aarhus University, 2014.

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

A. Bosco, D. Laganà, R. Musmanno, and F. Vocaturo. A matheuristic algorithm for the mixed capacitated general routing problem. *Networks*, 64(4):262–281, 2014.

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.

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. (eds.) Punnen. *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.

S. Irnich, D. Laganà, C. Schlebusch, and F. Vocaturo. Two-phase branch-and-cut for the mixed capacitated general routing problem. *European Journal of Operational Research*, 243(1):17–29, 2015.

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.

G. Laporte, S. Ropke, and T. Vidal. Heuristics for the Vehicle Routing Problem. In P. Toth and D. Vigo, editors, *Vehicle routing: problems, methods, and applications*, chapter 4, pages 87–116. SIAM, 2014. ISBN 978-1-611973-58-7.

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.

S. Mandal, D. Pacciarelli, A. Løkketangen, and G. Hasle. A memetic NSGA-II for the bi-objective mixed capacitated general routing problem. *Journal of Heuristics*, In press.

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. The Capacitated Arc Routing Problem: Heuristics. In A. Corberan and G. Laporte, editors, *Arc routing: problems, methods, and applications*, chapter 7, pages 131–157. SIAM, 2014. ISBN 978-1-611973-66-2.

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. Røpke 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.

SINTEF. TOP website. URL `http://www.sintef.no/top`.

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.

Table 4: Computational results on the CBMix instances.

			B&C&P		MA		SA		Spider		MH		AILS	
			(sec_{tot}=21600)						(sec_{tot}=7200)				(sec_{tot}=3600)	
Instance	τ	LB	z	sec_{inc}	z	sec_{tot}	$avg\ z$	sec_{tot}	z	sec_{inc}	z	sec_{inc}	z	sec_{inc}
CBMix1	48	2547	**2569**	21600.0	2632	108.3	2617.1	15.1	2589	1231.0	2587	1202.3	2585	1114.3
CBMix2	185	11487	-	21600.0	12336	1078.5	12322.4	661.4	12222	4156.0	12241	4858.4	**11809**	3599.6
CBMix3	79	3514	3684	21600.0	3702	157.0	3695.2	56.0	3767	6612.0	3643	3286.7	**3614**	1590.1
CBMix4	98	7300	7582	21600.0	7583	548.1	7728.5	76.1	7802	6744.0	7583	3165.3	**7511**	431.5
CBMix5	65	4387	5548	21600.0	4562	100.0	4685.3	41.5	4688	1349.0	4531	1179.7	**4459**	876.8
CBMix6	108	6738	7643	21600.0	7087	204.5	7101.4	98.0	7139	6687.0	6968	3587.6	**6969**	1203.7
CBMix7	168	9046	-	21600.0	9974	662.6	9704.8	351.7	9767	3205.0	9859	3805.0	**9461**	2802.5
CBMix8	177	9976	12114	21600.0	10714	767.6	10710.2	263.8	10689	1413.0	10658	3913.3	**10318**	1627.0
CBMix9	50	3837	4044	21600.0	4041	140.8	4132.4	12.5	4147	5517.0	4060	1817.2	**4002**	761.3
CBMix10	107	7343	7614	21600.0	7755	843.2	7763.2	108.3	7931	4665.0	7755	3065.4	**7500**	2908.4
CBMix11	82	4318	-	21600.0	4503	414.7	4599.6	49.8	4525	536.0	4561	2036.8	**4487**	787.9
CBMix12	53	3138	3138	2037.2	3235	71.3	3235.0	21.4	3235	14.0	***3138**	500.6	***3138**	634.0
CBMix13	141	8681	-	21600.0	9339	550.6	9270.6	312.8	9332	1427.0	9110	3877.2	**8984**	2823.1
CBMix14	93	8205	-	21600.0	8615	357.2	8769.3	65.3	8638	6404.0	8671	2726.0	**8443**	735.2
CBMix15	91	8013	8355	21600.0	8359	390.2	8385.3	97.3	8443	3553.0	8359	2489.1	**8249**	2585.2
CBMix16	169	8446	-	21600.0	9389	536.1	9024.3	445.5	9022	6754.0	8933	3924.3	**8714**	3141.2
CBMix17	63	3943	-	21600.0	4165	116.1	4107.6	43.0	4235	1271.0	4037	1476.3	**4034**	966.0
CBMix18	127	6856	7137	21600.0	7411	475.7	7214.6	278.4	7346	1994.0	7254	2593.4	**7044**	3028.9
CBMix19	212	15628	-	21600.0	17036	1273.4	16677.5	469.8	16692	5688.0	16554	5187.4	**16244**	1473.4
CBMix20	73	4647	5068	21600.0	4918	164.6	4902.9	50.7	4859	3501.0	4885	2104.0	**4794**	3322.4
CBMix21	180	17295	18201	21600.0	18509	1370.6	18318.3	530.4	18809	5322.0	18509	4623.4	**17889**	2961.2
CBMix22	42	1905	**1941**	21600.0	**1941**	65.8	1970.5	9.5	**1941**	492.0	**1941**	252.1	**1941**	0.3
CBMix23	20	780	***780**	21600.0	***780**	20.4	***780**	2.7	***780**	0.3	***780**	43.8	***780**	0.0
Sum/Avg.	2431	158030			167806	452.9	166936.0	176.6	167818	3414.6	166028	2683.3	**162969**	1711.9
# Optima			1		1		1		1		2		2	
# Best			3		2		1		2		3		22	
# Unsolved			8		0		0		0		0		0	

Table 5: Computational results on the BHW instances.

			B&C&P (sec_{tot}=21600)		Spider (sec_{tot}=7200)		AILS (sec_{tot}=3600)	
Instance	τ	LB	z	sec_{inc}	z	sec_{inc}	z	sec_{inc}
BHW1	29	337	***337**	987.9	***337**	6.0	***337**	16.5
BHW2	29	470	***470**	0.2	***470**	36.0	***470**	0.4
BHW3	20	415	***415**	29.8	***415**	18.0	***415**	729.1
BHW4	50	240	***240**	15.6	***240**	1.0	***240**	0.0
BHW5	162	502	-	21600.0	506	610.0	***502**	129.5
BHW6	110	388	-	21600.0	***388**	58.0	***388**	4.8
BHW7	229	1047	-	21600.0	1104	6324.0	**1074**	3289.6
BHW8	117	665	-	21600.0	672	1801.0	**668**	2597.9
BHW9	178	858	-	21600.0	920	2431.0	**875**	1396.0
BHW10	142	8310	-	21600.0	8596	6205.0	**8584**	165.3
BHW11	71	4690	-	21600.0	5023	3012.0	**4952**	3473.2
BHW12	115	10605	-	21600.0	11042	6059.0	**10906**	1832.1
BHW13	175	13952	-	21600.0	14510	5723.0	**14428**	300.5
BHW14	221	24377	-	21600.0	25194	4584.0	**24988**	652.8
BHW15	128	15130	16929	21600.0	15564	6728.0	**15354**	1325.7
BHW16	410	42506	-	21600.0	44527	5747.0	**43567**	3545.0
BHW17	240	25570	-	21600.0	26768	6823.0	**26116**	3005.5
BHW18	194	14840	16774	21600.0	15833	5532.0	**15318**	2803.4
BHW19	107	9197	10942	21600.0	9480	3605.0	**9397**	1359.1
BHW20	293	10730	-	21600.0	16625	6769.0	**16162**	1417.9
Sum/Avg.	3020	184829			198214	3603.6	**194741**	1402.2
# Optima			4		5		6	
# Best			4		5		20	
# Unsolved			13		0		0	

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

			Spider (sec_{tot}=7200)		AILS (sec_{tot}=3600)	
Instance	τ	LB	z	sec_{inc}	z	sec_{inc}
DI-NEARP-n240-Q2k	240	16376	24371	4569.0	**23947**	303.0
DI-NEARP-n240-Q4k	240	14362	18352	4495.0	**18184**	1505.3
DI-NEARP-n240-Q8k	240	13442	15937	6421.0	**15907**	152.5
DI-NEARP-n240-Q16k	240	13116	14953	5274.0	**14776**	1125.7
DI-NEARP-n422-Q2k	422	11623	19133	6629.0	**18943**	3471.4
DI-NEARP-n422-Q4k	422	11284	15987	4524.0	**15863**	698.8
DI-NEARP-n422-Q8k	422	11220	14627	2925.0	**14469**	3169.2
DI-NEARP-n422-Q16k	422	11198	**14357**	4661.0	14366	2487.0
DI-NEARP-n442-Q2k	442	35068	52062	7091.0	**50573**	3328.6
DI-NEARP-n442-Q4k	442	33585	45906	6308.0	**45297**	672.0
DI-NEARP-n442-Q8k	442	32985	45395	5964.0	**43422**	743.3
DI-NEARP-n442-Q16k	442	32713	**42797**	6480.0	42883	841.9
DI-NEARP-n477-Q2k	477	19722	23124	5996.0	**22956**	1795.8
DI-NEARP-n477-Q4k	477	18031	20198	7006.0	**19991**	1592.7
DI-NEARP-n477-Q8k	477	17193	18561	2999.0	**18490**	910.9
DI-NEARP-n477-Q16k	477	16873	18105	4079.0	**18078**	3528.2
DI-NEARP-n699-Q2k	699	34101	59817	6993.0	**58882**	2956.4
DI-NEARP-n699-Q4k	699	26891	40473	7178.0	**40384**	1853.4
DI-NEARP-n699-Q8k	699	23302	30992	6095.0	**30566**	2973.5
DI-NEARP-n699-Q16k	699	21967	27028	3173.0	**26872**	2955.2
DI-NEARP-n833-Q2k	833	32435	56877	7135.0	**56307**	3555.9
DI-NEARP-n833-Q4k	833	29381	42407	6861.0	**41162**	3143.8
DI-NEARP-n833-Q8k	833	28453	35267	6940.0	**34626**	3488.3
DI-NEARP-n833-Q16k	833	28233	33013	4046.0	**32644**	3045.7
Sum/Avg.	12452	533554	729739	5576.8	**719588**	2095.8
# Optima			0		0	
# Best			2		22	

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

			B&C	B&C&P	B&C2 (sec_{tot}=21600)		MH		AILS (sec_{tot}=3600)	
Instance	τ	LB	z	z	z	sec_{inc}	z	sec_{inc}	z	sec_{inc}
mggdb-0.25-1	21	280	***280**	***280**	***280**	30.4	***280**	14.6	***280**	0.0
mggdb-0.25-2	25	349	***349**	352	***349**	15.9	359	18.1	***349**	12.5
mggdb-0.25-3	22	278	***278**	***278**	***278**	23.5	286	15.1	***278**	0.0
mggdb-0.25-4	18	289	***289**	***289**	***289**	6.1	***289**	14.6	***289**	0.0
mggdb-0.25-5	24	394	***394**	***394**	***394**	62.5	410	24.4	***394**	0.0
mggdb-0.25-6	21	292	***292**	***292**	***292**	2.7	295	15.1	***292**	0.0
mggdb-0.25-7	20	290	***290**	***290**	***290**	3.2	302	6.3	***290**	0.0
mggdb-0.25-8	45	333	-	**336**	358	21600.0	351	90.4	**336**	17.0
mggdb-0.25-9	47	308	-	**309**	349	21600.0	316	82.6	310	27.6
mggdb-0.25-10	22	265	***265**	***265**	***265**	2.7	***265**	5.7	***265**	0.1
mggdb-0.25-11	41	356	***356**	***356**	***356**	230.6	369	36.8	***356**	0.3
mggdb-0.25-12	22	459	***459**	***459**	***459**	616.0	465	18.7	***459**	0.1
mggdb-0.25-13	26	388	***388**	***388**	***388**	5716.6	392	30.9	392	44.4
mggdb-0.25-14	20	107	***107**	***107**	***107**	3.1	***107**	3.7	***107**	0.0
mggdb-0.25-15	20	55	***55**	***55**	***55**	1.2	***55**	11.0	***55**	0.0
mggdb-0.25-16	25	98	***98**	***98**	***98**	7.3	***98**	7.6	***98**	0.0
mggdb-0.25-17	25	71	***71**	***71**	***71**	1.8	***71**	12.0	***71**	0.0
mggdb-0.25-18	32	144	***144**	***144**	***144**	6.1	***144**	45.7	***144**	0.0
mggdb-0.25-19	10	53	***53**	***53**	***53**	1.4	***53**	9.3	***53**	0.0
mggdb-0.25-20	20	116	***116**	***116**	***116**	11.7	117	4.9	***116**	0.9
mggdb-0.25-21	31	146	***146**	***146**	***146**	27.9	***146**	9.4	***146**	0.1
mggdb-0.25-22	38	160	-	***160**	***160**	282.6	168	18.2	***160**	27.6
mggdb-0.25-23	48	181	-	***181**	***181**	19325.6	186	27.6	***181**	178.9
Sum/Average no "-"	445	4430	***4430**	4433	***4430**	356.4	4503	16.0	4434	2.5
Sum/Average all	623	5412	-	**5419**	5478	3025.2	5524	22.7	5421	13.4
# Optima			19	16	21		10		20	
# Best			19	18	21		10		21	
# Unsolved			4	0	0		0		0	

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

			B&C	B&C&P	B&C2 (sec_{tot}=21600)		MH		AILS (sec_{tot}=3600)	
Instance	τ	LB	z	z	z	sec_{inc}	z	sec_{inc}	z	sec_{inc}
mggdb-0.30-1	21	273	***273**	***273**	***273**	77.9	276	21.4	***273**	0.0
mggdb-0.30-2	24	301	***301**	***301**	***301**	11631.5	314	20.0	***301**	0.3
mggdb-0.30-3	19	270	***270**	***270**	***270**	17.2	278	15.9	***270**	0.0
mggdb-0.30-4	18	260	***260**	***260**	***260**	2.7	***260**	25.5	***260**	0.0
mggdb-0.30-5	25	388	***388**	***388**	***388**	37.7	399	18.5	***388**	0.1
mggdb-0.30-6	22	276	***276**	***276**	***276**	56.8	***276**	19.3	***276**	0.0
mggdb-0.30-7	20	273	***273**	***273**	***273**	810.1	277	25.0	***273**	0.0
mggdb-0.30-8	46	329	-	**331**	364	21600.0	338	78.1	**331**	2.1
mggdb-0.30-9	46	281	-	***281**	317	21600.0	284	82.7	***281**	12.7
mggdb-0.30-10	22	242	***242**	***242**	***242**	8.9	***242**	20.8	***242**	0.0
mggdb-0.30-11	43	387	***387**	-	***387**	250.8	399	31.1	***387**	0.3
mggdb-0.30-12	21	467	***467**	***467**	***467**	1586.1	472	11.2	***467**	0.1
mggdb-0.30-13	24	483	486	***483**	***483**	20676.0	***483**	33.0	***483**	3.3
mggdb-0.30-14	17	101	***101**	***101**	***101**	2.5	***101**	9.3	***101**	0.0
mggdb-0.30-15	19	44	***44**	***44**	***44**	0.8	***44**	3.8	***44**	0.0
mggdb-0.30-16	24	105	***105**	***105**	***105**	30.6	107	15.5	***105**	1.4
mggdb-0.30-17	22	65	***65**	***65**	***65**	1.4	67	4.3	***65**	0.0
mggdb-0.30-18	30	144	***144**	***144**	***144**	7.0	***144**	9.4	***144**	0.0
mggdb-0.30-19	10	51	***51**	***51**	***51**	1.0	***51**	6.2	***51**	0.0
mggdb-0.30-20	18	94	***94**	***94**	***94**	24.3	97	13.7	***94**	0.0
mggdb-0.30-21	28	121	***121**	***121**	***121**	24.9	122	20.8	***121**	0.0
mggdb-0.30-22	37	153	-	***153**	***153**	13595.8	156	24.5	***153**	0.2
mggdb-0.30-23	47	167	-	***167**	***167**	20184.6	171	9.0	171	84.1
Sum/Average no "-"	384	3958	3961	***3958**	***3958**	1941.6	4010	16.3	***3958**	0.2
Sum/Average all	603	5275	-	-	5346	4877.4	5358	22.6	**5281**	4.6
# Optima			18	21	21		8		21	
# Best			18	22	21		8		22	
# Unsolved			4	1	0		0		0	

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

			B&C	B&C&P	B&C2 (sec_{tot}=21600)		MH		AILS (sec_{tot}=3600)	
Instance	τ	LB	z	z	z	sec_{inc}	z	sec_{inc}	z	sec_{inc}
mggdb-0.35-1	21	252	***252**	***252**	***252**	41.9	***252**	21.3	***252**	0.0
mggdb-0.35-2	22	284	***284**	***284**	***284**	19.1	***284**	16.2	***284**	0.0
mggdb-0.35-3	20	243	***243**	***243**	***243**	109.0	***243**	18.1	***243**	0.0
mggdb-0.35-4	17	242	***242**	***242**	***242**	14.1	***242**	25.3	***242**	0.0
mggdb-0.35-5	23	309	***309**	***309**	***309**	1056.1	317	26.2	***309**	0.3
mggdb-0.35-6	21	262	***262**	***262**	***262**	140.8	***262**	13.6	***262**	0.0
mggdb-0.35-7	22	272	***272**	***272**	***272**	12.9	***272**	19.8	***272**	0.0
mggdb-0.35-8	38	315	-	**316**	337	21600.0	321	24.0	320	7.7
mggdb-0.35-9	45	265	-	**266**	292	21600.0	274	85.7	267	1.2
mggdb-0.35-10	24	268	***268**	***268**	***268**	6.3	***268**	18.9	***268**	0.0
mggdb-0.35-11	41	303	***303**	313	***303**	255.6	313	26.7	***303**	0.1
mggdb-0.35-12	20	461	***461**	***461**	***461**	197.5	***461**	13.4	***461**	0.0
mggdb-0.35-13	24	417	***417**	***417**	***417**	674.1	435	24.6	***417**	33.4
mggdb-0.35-14	18	84	***84**	***84**	***84**	45.2	85	10.1	***84**	0.0
mggdb-0.35-15	18	44	***44**	***44**	***44**	1.2	***44**	7.0	***44**	0.0
mggdb-0.35-16	22	75	***75**	***75**	***75**	9276.6	***75**	27.5	***75**	0.0
mggdb-0.35-17	23	62	***62**	***62**	***62**	2.6	***62**	11.0	***62**	0.0
mggdb-0.35-18	30	135	***135**	***135**	***135**	12.4	137	9.7	***135**	0.0
mggdb-0.35-19	9	51	***51**	***51**	***51**	0.8	***51**	4.4	***51**	0.0
mggdb-0.35-20	20	96	***96**	***96**	***96**	23.7	***96**	3.8	***96**	1.5
mggdb-0.35-21	28	120	***120**	***120**	***120**	26.8	122	6.4	***120**	0.1
mggdb-0.35-22	36	139	-	***139**	***139**	266.9	143	21.4	***139**	0.2
mggdb-0.35-23	44	179	-	***179**	194	21600.0	185	15.0	***179**	71.4
Sum/Average no "-"	423	3980	***3980**	3990	***3980**	627.2	4021	16.0	***3980**	1.5
Sum/Average all	586	4878	-	4890	4942	3347.1	4944	21.3	**4885**	5.0
# Optima			19	20	20		13		21	
# Best			19	22	20		13		21	
# Unsolved			4	0	0		0		0	

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

			B&C	B&C&P (sec_{tot}=10800)		B&C2	MH		AILS (sec_{tot}=3600)	
Instance	τ	LB	z	z	sec_{inc}	z	z	sec_{inc}	z	sec_{inc}
mggdb-0.40-1	19	279	***279**	***279**	93.0	-	***279**	11.9	***279**	0.0
mggdb-0.40-2	22	308	***308**	***308**	570.7	-	320	28.2	***308**	0.4
mggdb-0.40-3	20	225	***225**	***225**	3.7	-	229	11.1	***225**	0.1
mggdb-0.40-4	17	238	***238**	***238**	0.2	-	***238**	20.4	***238**	0.0
mggdb-0.40-5	22	344	***344**	***344**	2.8	-	346	18.8	***344**	0.1
mggdb-0.40-6	19	270	***270**	***270**	10.4	-	281	10.3	***270**	0.0
mggdb-0.40-7	19	282	***282**	***282**	10800.0	-	283	15.9	***282**	0.0
mggdb-0.40-8	40	326	-	333	10800.0	-	340	73.9	**331**	0.4
mggdb-0.40-9	45	273	-	**275**	10800.0	-	285	34.5	**275**	1.1
mggdb-0.40-10	22	191	***191**	***191**	137.1	-	***191**	14.5	***191**	0.0
mggdb-0.40-11	38	277	**283**	294	10800.0	-	287	34.1	**283**	0.3
mggdb-0.40-12	19	412	***412**	***412**	3.6	-	***412**	10.5	***412**	0.0
mggdb-0.40-13	23	405	***405**	***405**	177.6	-	***405**	9.6	***405**	76.4
mggdb-0.40-14	18	62	***62**	***62**	0.5	-	***62**	2.6	***62**	0.0
mggdb-0.40-15	18	37	***37**	***37**	2.2	-	***37**	12.2	***37**	0.0
mggdb-0.40-16	21	84	***84**	***84**	25.9	-	***84**	8.4	***84**	0.0
mggdb-0.40-17	21	65	***65**	***65**	3.2	-	***65**	1.6	***65**	0.0
mggdb-0.40-18	27	119	***119**	***119**	10800.0	-	122	10.8	***119**	0.0
mggdb-0.40-19	10	38	***38**	***38**	0.1	-	***38**	7.3	***38**	0.0
mggdb-0.40-20	19	94	***94**	***94**	5.4	-	***94**	21.9	***94**	0.1
mggdb-0.40-21	28	104	***104**	***104**	40.9	-	106	11.5	***104**	0.0
mggdb-0.40-22	33	129	-	***129**	116.5	-	132	12.8	***129**	0.8
mggdb-0.40-23	42	160	-	***160**	198.3	-	165	18.7	163	5.2
Sum/Average no "-"	402	3834	**3840**	3851	1762.0	-	3879	13.8	**3840**	4.1
Sum/Average all	562	4722		4748	2408.4	-	4801	17.5	**4738**	3.7
# Optima			18	20		-	11		19	
# Best			19	21		-	11		22	
# Unsolved			4	0		-	0		0	

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

			B&C	B&C&P (sec_{tot}=10800)		B&C2	MH		AILS (sec_{tot}=3600)	
Instance	τ	LB	z	z	sec_{inc}	z	z	sec_{inc}	z	sec_{inc}
mggdb-0.45-1	17	259	***259**	***259**	2.0	-	***259**	21.8	***259**	0.0
mggdb-0.45-2	21	298	***298**	***298**	403.9	-	302	15.7	***298**	0.0
mggdb-0.45-3	19	237	***237**	***237**	0.7	-	245	15.6	***237**	0.0
mggdb-0.45-4	17	228	***228**	***228**	1.3	-	***228**	25.6	***228**	0.0
mggdb-0.45-5	21	350	***350**	***350**	4.0	-	357	23.5	***350**	0.0
mggdb-0.45-6	18	218	***218**	***218**	1.5	-	225	15.4	***218**	0.0
mggdb-0.45-7	20	243	***243**	***243**	968.3	-	***243**	26.6	***243**	0.0
mggdb-0.45-8	41	296	-	***296**	3.5	-	312	23.2	***296**	1.2
mggdb-0.45-9	41	277	-	***277**	7207.2	-	287	40.9	***277**	99.3
mggdb-0.45-10	22	214	***214**	***214**	30.6	-	***214**	14.4	***214**	0.0
mggdb-0.45-11	39	289	**297**	301	10800.0	-	310	29.3	**297**	0.7
mggdb-0.45-12	21	393	***393**	***393**	6.3	-	406	8.3	***393**	0.0
mggdb-0.45-13	21	423	***423**	***423**	2.2	-	***423**	16.9	429	8.7
mggdb-0.45-14	16	66	***66**	***66**	2.4	-	67	20.4	***66**	0.0
mggdb-0.45-15	16	34	***34**	***34**	1.7	-	36	2.6	***34**	0.0
mggdb-0.45-16	20	70	***70**	***70**	12.0	-	***70**	1.1	***70**	0.0
mggdb-0.45-17	21	53	***53**	***53**	5.5	-	***53**	2.7	***53**	0.0
mggdb-0.45-18	25	121	**123**	**123**	10800.0	-	**123**	30.7	**123**	0.0
mggdb-0.45-19	8	48	***48**	***48**	0.1	-	***48**	4.6	***48**	0.0
mggdb-0.45-20	16	78	***78**	***78**	2.3	-	***78**	13.0	***78**	0.0
mggdb-0.45-21	24	122	***122**	***122**	5.9	-	128	22.2	***122**	1.2
mggdb-0.45-22	33	136	-	***136**	121.5	-	139	17.7	***136**	0.1
mggdb-0.45-23	39	144	-	***144**	354.0	-	147	17.4	145	116.9
Sum/Average no "-"	382	3744	**3754**	3758	1213.2	-	3815	16.3	3760	0.5
Sum/Average all	536	4597		**4611**	1336.4	-	4700	17.8	4614	9.9
# Optima			17	17		-	9		16	
# Best			19	22		-	10		21	
# Unsolved			4	0		-	0		0	

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

			B&C	B&C&P (sec_{tot}=10800)		B&C2	MH		AILS (sec_{tot}=3600)	
Instance	τ	LB	z	z	sec_{inc}	z	z	sec_{inc}	z	sec_{inc}
mggdb-0.50-1	18	214	***214**	***214**	7.5	-	***214**	3.6	***214**	0.0
mggdb-0.50-2	19	269	***269**	***269**	40.7	-	281	12.0	***269**	0.0
mggdb-0.50-3	19	218	***218**	***218**	22.4	-	***218**	5.8	***218**	0.0
mggdb-0.50-4	15	219	***219**	***219**	0.5	-	***219**	9.6	***219**	0.0
mggdb-0.50-5	20	292	***292**	***292**	1.7	-	***292**	3.1	***292**	0.0
mggdb-0.50-6	17	276	***276**	***276**	9.4	-	***276**	4.0	***276**	0.0
mggdb-0.50-7	19	265	***265**	***265**	15.7	-	274	5.6	***265**	0.0
mggdb-0.50-8	37	309	-	**310**	10800.0	-	**310**	29.3	**310**	4.0
mggdb-0.50-9	41	260	-	**265**	10800.0	-	270	42.9	**265**	2.1
mggdb-0.50-10	19	194	***194**	***194**	0.9	-	***194**	2.0	***194**	0.0
mggdb-0.50-11	38	267	**275**	-	10800.0	-	278	16.7	**275**	0.7
mggdb-0.50-12	19	445	***445**	***445**	8.5	-	***445**	8.1	***445**	0.0
mggdb-0.50-13	21	259	***259**	***259**	43.1	-	***259**	3.8	261	10.8
mggdb-0.50-14	16	75	***75**	***75**	2.0	-	76	5.1	***75**	0.0
mggdb-0.50-15	15	37	***37**	***37**	0.5	-	***37**	1.2	***37**	0.0
mggdb-0.50-16	19	66	***66**	***66**	14.7	-	***66**	14.7	***66**	0.2
mggdb-0.50-17	20	53	***53**	***53**	5.1	-	***53**	1.0	***53**	0.0
mggdb-0.50-18	25	117	**121**	**121**	10800.0	-	122	12.8	**121**	0.0
mggdb-0.50-19	8	44	***44**	***44**	0.3	-	***44**	1.0	***44**	0.0
mggdb-0.50-20	15	81	***81**	***81**	11.0	-	***81**	3.5	***81**	0.0
mggdb-0.50-21	24	86	***86**	***86**	54.3	-	88	4.7	***86**	0.3
mggdb-0.50-22	31	123	-	***123**	54.6	-	127	5.8	***123**	12.1
mggdb-0.50-23	34	125	-	***125**	203.9	-	***125**	8.4	126	165.0
Sum/Average no "-"	328	3105	**3214**	**3214**	613.2	-	3239	5.6	3216	0.5
Sum/Average all	509	4189				-	4349	8.9	**4315**	8.5
# Optima			17	19		-	14		17	
# Best			19	22		-	15		21	
# Unsolved			4	1		-	0		0	

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

			B&C	B&C&P	B&C2 (sec_{tot}=21600)		MH		AILS (sec_{tot}=3600)	
Instance	τ	LB	z	z	z	sec_{inc}	z	sec_{inc}	z	sec_{inc}
mgval-0.25-1A	54	177	***177**	-	***177**	17.2	***177**	5.8	***177**	0.5
mgval-0.25-1B	47	217	***217**	-	***217**	41.9	***217**	10.2	***217**	18.0
mgval-0.25-1C	51	278	-	-	323	21600.0	335	72.1	**292**	948.9
mgval-0.25-2A	40	259	***259**	-	***259**	18.2	***259**	6.7	***259**	0.7
mgval-0.25-2B	48	336	***336**	-	***336**	53.0	***336**	1672.9	***336**	0.0
mgval-0.25-2C	48	479	-	-	512	21600.0	528	107.8	**480**	110.2
mgval-0.25-3A	44	89	***89**	-	***89**	16.0	***89**	5.2	***89**	0.0
mgval-0.25-3B	41	125	***125**	-	***125**	17.7	***125**	797.4	***125**	0.0
mgval-0.25-3C	41	153	***153**	-	***153**	106.1	161	54.8	***153**	0.7
mgval-0.25-4A	89	514	***514**	-	***514**	582.5	***514**	3684.5	***514**	0.5
mgval-0.25-4B	96	537	***537**	-	***537**	1328.1	541	4286.7	***537**	0.6
mgval-0.25-4C	100	525	***525**	-	***525**	14925.2	549	2466.2	***525**	113.1
mgval-0.25-4D	96	675	-	-	778	21600.0	724	141.4	**685**	543.7
mgval-0.25-5A	92	485	***485**	-	***485**	482.3	***485**	551.2	***485**	1.1
mgval-0.25-5B	86	493	***493**	-	***493**	918.2	500	5591.1	***493**	3.6
mgval-0.25-5C	93	584	***584**	-	***584**	722.3	599	994.6	***584**	11.2
mgval-0.25-5D	85	635	-	-	741	21600.0	681	128.9	**645**	661.1
mgval-0.25-6A	67	274	***274**	-	***274**	47.9	***274**	7.8	***274**	0.3
mgval-0.25-6B	63	263	***263**	-	***263**	11665.0	***263**	18.5	***263**	6.1
mgval-0.25-6C	66	316	-	-	378	21600.0	337	21.0	**324**	19.5
mgval-0.25-7A	84	297	***297**	-	***297**	407.9	***297**	38.9	***297**	2.2
mgval-0.25-7B	85	355	***355**	-	***355**	2153.2	***355**	7.7	***355**	0.9
mgval-0.25-7C	85	374	-	-	437	21600.0	407	53.6	**380**	65.6
mgval-0.25-8A	88	510	***510**	-	***510**	384.1	***510**	4348.8	***510**	1.8
mgval-0.25-8B	84	423	***423**	-	***423**	1910.6	***423**	3410.2	***423**	1.0
mgval-0.25-8C	78	538	-	-	625	21600.0	591	1083.7	**544**	483.1
mgval-0.25-9A	122	371	***371**	-	***371**	669.0	***371**	2405.3	***371**	14.2
mgval-0.25-9B	112	358	***358**	-	***358**	20123.6	363	1587.6	***358**	3.4
mgval-0.25-9C	119	361	**365**	-	**365**	21600.0	369	2495.2	368	130.4
mgval-0.25-9D	121	418	-	-	498	21600.0	478	291.7	**427**	263.2
mgval-0.25-10A	129	492	***492**	-	***492**	8200.5	***492**	5793.9	***492**	28.2
mgval-0.25-10B	123	528	***528**	-	***528**	19187.6	***528**	4346.8	***528**	63.0
mgval-0.25-10C	125	483	***483**	-	***483**	20129.2	501	3335.7	***483**	18.5
mgval-0.25-10D	119	565.5	-	-	655	21600.0	616	698.0	**568**	23.2
Sum/Average no "-"	2072	9209	**9213**	-	**9213**	5045.5	9298	1917.0	9216	28.1
Sum/Average all	2669	13487.5		-	14160	9427.6	13995	1485.9	**13561**	104.1
# Optima			24	-	24		17		24	
# Best			25	-	25		17		33	
# Unsolved			9	-	0		0		0	

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

			B&C	B&C&P	B&C2 (sec_{tot}=21600)		MH		AILS (sec_{tot}=3600)	
Instance	τ	LB	z	z	z	sec_{inc}	z	sec_{inc}	z	sec_{inc}
mgval-0.30-1A	53	170	***170**	-	***170**	15.0	***170**	5.5	***170**	0.2
mgval-0.30-1B	47	194	***194**	-	***194**	146.6	***194**	25.0	***194**	18.3
mgval-0.30-1C	48	255	-	-	310	21600.0	**280**	17.9	**280**	816.3
mgval-0.30-2A	42	233	***233**	-	***233**	33.5	***233**	2.9	***233**	0.0
mgval-0.30-2B	49	347	***347**	-	***347**	140.7	***347**	4080.4	***347**	4.6
mgval-0.30-2C	45	489	-	-	534	21600.0	542	79.5	**498**	8.0
mgval-0.30-3A	46	105	***105**	-	***105**	73.9	***105**	6.7	***105**	0.6
mgval-0.30-3B	41	115	***115**	-	***115**	51.1	***115**	49.1	***115**	0.0
mgval-0.30-3C	41	149	**153**	-	**153**	21600.0	156	106.9	**153**	18.1
mgval-0.30-4A	87	477	***477**	-	***477**	2098.6	***477**	5383.0	***477**	453.2
mgval-0.30-4B	98	531	**533**	-	**533**	21600.0	537	3983.5	**533**	235.6
mgval-0.30-4C	98	492	**498**	-	**498**	21600.0	513	401.5	**498**	718.2
mgval-0.30-4D	94	652	-	-	765	21600.0	718	176.5	**653**	775.5
mgval-0.30-5A	86	445	***445**	-	***445**	509.5	***445**	362.5	***445**	3.1
mgval-0.30-5B	83	484	**490**	-	**490**	21600.0	492	3372.8	492	0.4
mgval-0.30-5C	87	549	**551**	-	**551**	21600.0	568	510.2	553	0.9
mgval-0.30-5D	86	612	-	-	736	21600.0	675	278.5	**618**	227.4
mgval-0.30-6A	66	252	***252**	-	***252**	139.8	***252**	288.9	***252**	0.3
mgval-0.30-6B	64	262	***262**	-	***262**	1425.9	268	34.7	263	0.4
mgval-0.30-6C	64	307	-	-	364	21600.0	339	24.3	**322**	27.6
mgval-0.30-7A	77	324	***324**	-	***324**	3197.5	***324**	54.3	***324**	4.5
mgval-0.30-7B	82	344	***344**	-	***344**	632.3	***344**	18.8	***344**	1.4
mgval-0.30-7C	85	347	-	-	388	21600.0	380	55.3	**354**	25.0
mgval-0.30-8A	88	431	***431**	-	***431**	1108.1	**431**	3482.7	***431**	0.7
mgval-0.30-8B	83	400	***400**	-	***400**	9578.2	408	1229.5	***400**	7.2
mgval-0.30-8C	75	510	-	-	590	21600.0	570	963.9	**522**	94.5
mgval-0.30-9A	118	357	***357**	-	***357**	21600.0	***357**	65.3	***357**	19.8
mgval-0.30-9B	110	348	***348**	-	***348**	12667.8	356	1985.0	***348**	2.3
mgval-0.30-9C	112	335	***335**	-	***335**	20185.9	347	524.1	***335**	8.4
mgval-0.30-9D	122	421.13	-	-	501	21600.0	475	674.0	**434**	148.9
mgval-0.30-10A	127	484	***484**	-	***484**	1986.7	***484**	534.5	***484**	282.2
mgval-0.30-10B	123	441	***441**	-	***441**	14180.1	***441**	3577.3	***441**	32.2
mgval-0.30-10C	125	475	478	-	**476**	21600.0	483	1711.8	**476**	12.1
mgval-0.30-10D	121	530.8	-	-	640	21600.0	575	309.2	**539**	996.9
Sum/Average no "-"	2033	8744	8767	-	**8765**	8774.8	8847	1271.9	8770	53.7
Sum/Average all	2625	12897.93		-	13593	12169.7	13401	1011.1	**12990**	145.4
# Optima			19	-	19		14		18	
# Best			24	-	25		15		31	
# Unsolved			9	-	0		0		0	

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

			B&C	B&C&P	B&C2 (sec_{tot}=21600)		MH		AILS (sec_{tot}=3600)	
Instance	τ	LB	z	z	z	sec_{inc}	z	sec_{inc}	z	sec_{inc}
mgval-0.35-1A	47	158	***158**	-	***158**	7.5	***158**	1.1	***158**	0.2
mgval-0.35-1B	48	192	***192**	-	***192**	313.0	***192**	111.0	***192**	2.7
mgval-0.35-1C	48	272	-	-	312	21600.0	**284**	98.2	289	733.3
mgval-0.35-2A	40	286	***286**	-	***286**	9.5	***286**	9.0	***286**	0.1
mgval-0.35-2B	46	326	***326**	-	***326**	577.0	***326**	1300.6	***326**	11.9
mgval-0.35-2C	45	482	-	-	520	21600.0	523	63.0	**485**	3.6
mgval-0.35-3A	43	84	***84**	-	***84**	14.7	***84**	5.2	***84**	0.1
mgval-0.35-3B	41	113	***113**	-	***113**	33.0	***113**	365.4	***113**	0.1
mgval-0.35-3C	40	150	***150**	-	***150**	21600.0	159	21.5	***150**	0.6
mgval-0.35-4A	84	430	***430**	-	***430**	449.4	***430**	4379.0	***430**	0.2
mgval-0.35-4B	90	529	**531**	-	**531**	21600.0	**531**	2465.9	**531**	0.4
mgval-0.35-4C	93	516	***516**	-	***516**	13985.6	553	116.8	***516**	151.0
mgval-0.35-4D	96	640.5	-	-	729	21600.0	661	107.7	**643**	653.4
mgval-0.35-5A	82	454	***454**	-	***454**	2396.5	***454**	3566.1	455	1.9
mgval-0.35-5B	81	467	***467**	-	***467**	21600.0	468	2464.5	***467**	908.3
mgval-0.35-5C	82	586	***586**	-	***586**	21600.0	595	3801.8	***586**	508.2
mgval-0.35-5D	80	568	-	-	658	21600.0	648	99.4	**588**	576.4
mgval-0.35-6A	64	248	***248**	-	***248**	126.4	***248**	25.1	***248**	0.8
mgval-0.35-6B	62	250	***250**	-	***250**	717.6	***250**	10.2	***250**	0.3
mgval-0.35-6C	60	303	-	-	372	21600.0	326	16.9	**312**	140.2
mgval-0.35-7A	78	264	***264**	-	***264**	134.3	***264**	7.1	***264**	1.1
mgval-0.35-7B	79	325	***325**	-	***325**	1756.4	***325**	19.0	***325**	1.2
mgval-0.35-7C	82	336	-	-	390	21600.0	351	63.5	***336**	31.4
mgval-0.35-8A	84	415	***415**	-	***415**	296.0	***415**	2834.7	***415**	0.1
mgval-0.35-8B	78	385	***385**	-	***385**	999.0	***385**	420.5	***385**	3.7
mgval-0.35-8C	75	487	-	-	602	21600.0	547	142.0	**499**	281.6
mgval-0.35-9A	116	324	***324**	-	***324**	21600.0	***324**	397.2	***324**	6.9
mgval-0.35-9B	106	331	332	-	***331**	12717.9	332	2998.6	***331**	662.3
mgval-0.35-9C	115	328	329	-	329	21600.0	338	524.4	***328**	172.0
mgval-0.35-9D	115	422	-	-	491	21600.0	473	528.9	**430**	485.7
mgval-0.35-10A	122	475	***475**	-	***475**	7015.3	***475**	5105.4	479	32.1
mgval-0.35-10B	118	461	***461**	-	***461**	9490.7	463	3345.8	***461**	3.9
mgval-0.35-10C	122	428	431	-	431	21600.0	448	3135.5	**430**	198.2
mgval-0.35-10D	114	519	-	-	594	21600.0	566	935.2	**524**	251.0
Sum/Average no "-"	1961	8525	8532	-	**8531**	8089.6	8616	1497.3	8534	78.5
Sum/Average all	2533	12554.5		-	13109	11665.9	12995	1161.4	**12640**	171.3
# Optima			21	-	22		16		22	
# Best			22	-	23		18		31	
# Unsolved			9	-	0		0		0	

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

			B&C (sec_{tot}=10800)		B&C&P	B&C2	MH		AILS (sec_{tot}=3600)	
Instance	τ	LB	z	sec_{inc}	z	z	z	sec_{inc}	z	sec_{inc}
mgval-0.40-1A	48	165	***165**	1.3	-	-	***165**	4.7	***165**	7.3
mgval-0.40-1B	43	196	***196**	1415.0	-	-	***196**	192.4	***196**	0.1
mgval-0.40-1C	46	-	-	-	-	-	272	307.8	**263**	28.2
mgval-0.40-2A	38	222	***222**	1.7	-	-	***222**	1.2	***222**	0.1
mgval-0.40-2B	49	311	***311**	1506.0	-	-	***311**	886.7	***311**	0.0
mgval-0.40-2C	43	-	-	-	-	-	485	63.0	**469**	1.2
mgval-0.40-3A	41	86	***86**	0.4	-	-	***86**	3.4	***86**	0.1
mgval-0.40-3B	40	110	***110**	7.3	-	-	***110**	5.3	***110**	0.3
mgval-0.40-3C	38	120	**148**	21600.0	-	-	157	52.2	**148**	0.9
mgval-0.40-4A	82	400	***400**	11152.6	-	-	***400**	2079.0	***400**	2.2
mgval-0.40-4B	89	395	**423**	21600.0	-	-	**423**	254.3	**423**	0.8
mgval-0.40-4C	89	424	**462**	21600.0	-	-	487	692.4	**462**	65.3
mgval-0.40-4D	88	-	-	-	-	-	669	49.1	**624**	27.8
mgval-0.40-5A	82	426	***426**	195.3	-	-	***426**	592.6	***426**	1.1
mgval-0.40-5B	77	402	**424**	21600.0	-	-	428	997.9	**424**	3.8
mgval-0.40-5C	84	488	**524**	21600.0	-	-	539	879.4	527	3.5
mgval-0.40-5D	79	-	-	-	-	-	665	143.3	**614**	402.9
mgval-0.40-6A	61	224	***224**	150.2	-	-	***224**	23.8	***224**	11.0
mgval-0.40-6B	58	211	***211**	858.3	-	-	***211**	13.0	***211**	2.1
mgval-0.40-6C	62	-	-	-	-	-	316	12.3	**312**	1.9
mgval-0.40-7A	76	271	***271**	956.9	-	-	***271**	26.8	***271**	3.1
mgval-0.40-7B	77	270	***270**	1609.0	-	-	***270**	9.9	***270**	1.0
mgval-0.40-7C	80	-	-	-	-	-	336	29.2	**332**	516.9
mgval-0.40-8A	80	393	***393**	4331.6	-	-	***393**	3340.8	***393**	10.5
mgval-0.40-8B	77	356	**371**	21600.0	-	-	372	204.2	372	1.9
mgval-0.40-8C	72	-	-	-	-	-	573	102.6	**517**	767.9
mgval-0.40-9A	114	337	**341**	21600.0	-	-	**341**	3573.3	**341**	7.8
mgval-0.40-9B	105	319	**327**	21600.0	-	-	331	2582.9	**327**	6.7
mgval-0.40-9C	104	280	**295**	21600.0	-	-	301	308.2	**295**	144.1
mgval-0.40-9D	116	-	-	-	-	-	414	306.9	**383**	283.0
mgval-0.40-10A	118	406	***406**	5107.9	-	-	***406**	5247.4	***406**	10.2
mgval-0.40-10B	113	431	**433**	21600.0	-	-	439	4934.1	**433**	11.5
mgval-0.40-10C	114	417	**432**	21600.0	-	-	435	1482.2	434	21.5
mgval-0.40-10D	112	-	-	-	-	-	521	315.9	**484**	54.4
Sum/Average no "-"	1897	7660	**7871**	10595.7	-	-	7944	1139.5	7877	9.3
Sum/Average all	2458				-	-	12195	875.5	**11875**	70.7
# Optima			14		-	-	14		14	
# Best			25		-	-	16		31	
# Unsolved			9		-	-	0		0	

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

			B&C (sec_{tot}=10800)		B&C&P	B&C2	MH		AILS (sec_{tot}=3600)	
Instance	τ	LB	z	sec_{inc}	z	z	z	sec_{inc}	z	sec_{inc}
mgval-0.45-1A	47	168	***168**	7.1	-	-	***168**	5.5	***168**	0.3
mgval-0.45-1B	41	166	***166**	3.2	-	-	***166**	10.9	***166**	1.4
mgval-0.45-1C	44	-	-	-	-	-	313	61.5	**258**	237.6
mgval-0.45-2A	38	251	***251**	0.5	-	-	***251**	2.1	***251**	0.0
mgval-0.45-2B	46	314	***314**	1312.0	-	-	***314**	595.2	***314**	120.8
mgval-0.45-2C	44	-	-	-	-	-	496	56.2	**462**	341.9
mgval-0.45-3A	41	82	***82**	4.8	-	-	***82**	5.3	***82**	0.0
mgval-0.45-3B	39	91	***91**	3.2	-	-	***91**	24.7	***91**	0.2
mgval-0.45-3C	38	122	**143**	21600.0	-	-	**143**	24.0	**143**	0.3
mgval-0.45-4A	80	381	***381**	1198.9	-	-	***381**	1668.4	***381**	702.6
mgval-0.45-4B	91	423	**471**	21600.0	-	-	481	523.3	**471**	1.4
mgval-0.45-4C	85	434	**481**	21600.0	-	-	508	418.3	**481**	2.8
mgval-0.45-4D	87	-	-	-	-	-	626	87.6	**577**	542.4
mgval-0.45-5A	78	378	**391**	21600.0	-	-	**391**	5679.8	392	1.8
mgval-0.45-5B	75	379	**416**	21600.0	-	-	**416**	1066.7	**416**	2.9
mgval-0.45-5C	79	445	**492**	21600.0	-	-	502	79.2	494	5.6
mgval-0.45-5D	76	-	-	-	-	-	612	121.8	**552**	307.2
mgval-0.45-6A	60	213	***213**	528.5	-	-	***213**	18.3	***213**	0.3
mgval-0.45-6B	58	210	***210**	892.7	-	-	***210**	47.1	***210**	3.5
mgval-0.45-6C	58	-	-	-	-	-	308	22.2	**296**	21.5
mgval-0.45-7A	73	261	***261**	200.1	-	-	***261**	29.5	***261**	1.7
mgval-0.45-7B	77	290	**294**	21600.0	-	-	**294**	39.5	**294**	0.9
mgval-0.45-7C	76	-	-	-	-	-	347	33.5	**337**	14.6
mgval-0.45-8A	76	367	**370**	21600.0	-	-	**370**	1201.3	**370**	3.7
mgval-0.45-8B	72	341	**360**	21600.0	-	-	376	509.8	364	0.8
mgval-0.45-8C	65	-	-	-	-	-	535	70.7	**504**	142.8
mgval-0.45-9A	109	299	**306**	21600.0	-	-	**306**	5755.0	**306**	0.9
mgval-0.45-9B	100	311	**323**	21600.0	-	-	**323**	1517.4	**323**	6.6
mgval-0.45-9C	102	273	**291**	21600.0	-	-	306	1411.0	**291**	44.0
mgval-0.45-9D	108	-	-	-	-	-	403	430.8	**387**	434.1
mgval-0.45-10A	115	385	**388**	21600.0	-	-	**388**	5186.0	**388**	2.7
mgval-0.45-10B	108	390	**399**	21600.0	-	-	**399**	2682.5	**399**	7.2
mgval-0.45-10C	111	382	**403**	21600.0	-	-	418	780.9	**403**	14.5
mgval-0.45-10D	105	-	-	-	-	-	504	179.4	**488**	896.7
Sum/Average no "-"	1839	7356	**7665**	13126.0	-	-	7758	1195.7	7672	27.3
Sum/Average	2370				-	-	11902	892.5	**11533**	113.7
# Optima			10		-	-	10		10	
# Best			25		-	-	19		31	
# Unsolved			9		-	-	0		0	

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

			B&C (sec_{tot}=10800)		B&C&P	B&C2	MH		AILS (sec_{tot}=3600)	
Instance	τ	LB	z	sec_{inc}	z	z	z	sec_{inc}	z	sec_{inc}
mgval-0.50-1A	43	145	***145**	-	21600.0	-	***145**	1.9	***145**	0.0
mgval-0.50-1B	42	170	***170**	***170**	459.5	-	***170**	3.8	***170**	0.3
mgval-0.50-1C	40	253	-	270	21600.0	-	**261**	9.0	270	15.6
mgval-0.50-2A	38	248	***248**	351	21600.0	-	***248**	6.0	***248**	0.0
mgval-0.50-2B	44	284	***284**	-	21600.0	-	***284**	226.1	***284**	0.8
mgval-0.50-2C	40	453	-	473	21600.0	-	488	23.5	**464**	0.3
mgval-0.50-3A	40	75	***75**	-	21600.0	-	***75**	1.2	***75**	0.3
mgval-0.50-3B	37	107	***107**	-	21600.0	-	***107**	66.2	***107**	0.0
mgval-0.50-3C	36	137	***137**	***137**	9.8	-	139	6.8	***137**	0.7
mgval-0.50-4A	78	350	***350**	-	21600.0	-	***350**	70.5	***350**	2.8
mgval-0.50-4B	82	400	**413**	-	21600.0	-	419	361.2	414	71.9
mgval-0.50-4C	83	472	**488**	-	21600.0	-	512	360.9	**488**	96.5
mgval-0.50-4D	83	565	-	-	21600.0	-	613	145.2	**587**	772.3
mgval-0.50-5A	75	367	***367**	-	21600.0	-	***367**	919.8	***367**	3.6
mgval-0.50-5B	73	365	**378**	-	21600.0	-	**378**	386.2	**378**	26.5
mgval-0.50-5C	74	449	459	-	21600.0	-	476	419.9	**457**	39.4
mgval-0.50-5D	72	527	-	551	21600.0	-	570	193.2	**544**	180.0
mgval-0.50-6A	53	210	***210**	-	21600.0	-	***210**	40.1	***210**	0.8
mgval-0.50-6B	56	210	***210**	-	21600.0	-	***210**	21.0	***210**	1.2
mgval-0.50-6C	55	281	-	-	21600.0	-	306	17.9	**293**	2.9
mgval-0.50-7A	74	248	***248**	-	21600.0	-	***248**	68.5	***248**	0.3
mgval-0.50-7B	71	276	***276**	-	21600.0	-	***276**	36.1	***276**	0.3
mgval-0.50-7C	74	306	-	-	21600.0	-	333	120.8	**320**	56.3
mgval-0.50-8A	74	386	**388**	-	21600.0	-	**388**	3230.1	**388**	2.6
mgval-0.50-8B	68	343	**350**	-	21600.0	-	356	573.3	**350**	101.6
mgval-0.50-8C	63	485	-	502	21600.0	-	535	381.4	**501**	199.5
mgval-0.50-9A	105	306	***306**	-	21600.0	-	***306**	10.3	***306**	110.7
mgval-0.50-9B	97	267	**278**	-	21600.0	-	**278**	2599.2	**278**	256.8
mgval-0.50-9C	100	283	301	-	21600.0	-	303	663.7	**293**	388.9
mgval-0.50-9D	103	349	-	-	21600.0	-	408	213.1	**361**	40.0
mgval-0.50-10A	109	378	**385**	-	21600.0	-	**385**	5717.7	**385**	385.1
mgval-0.50-10B	110	364	**369**	-	21600.0	-	371	2474.8	**369**	18.8
mgval-0.50-10C	108	396	**406**	-	21600.0	-	416	675.0	**406**	160.8
mgval-0.50-10D	110	436	-	-	21600.0	-	492	403.8	**456**	632.5
Sum/Average no "-"	1770	7236	7348			-	7417	757.6	**7339**	49.1
Sum/Average all	2285	10891				-	11423	601.4	**11135**	105.0
# Optima			14	2		-	13		14	
# Best			23	2		-	18		32	
# Unsolved			9	28		-	0		0	

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

	Problem set						
Algorithm	gdb	val	egl	C	D	E	F
GLS	0.000	0.032	–	0.047	0.011	0.098	0.000
MA-CARP	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	0.054	0.328	0.024	0.155	0.125	0.017

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

	Problem set			
Algorithm	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	0.159	0.167	1.455	0.433

Appendix B – New best known solution for `D151-14c`

Route 1: 26 113 114 99 43 86 97 98

Route 2: 63 17 147 92 42 93 65 107 44 137 37

Route 3: 68 132 24 96 95 25 58 14 133 110 18

Route 4: 5 10 54 39 89 117 73 106 125 122

Route 5: 11 126 16 127 53 129 29 79 21 118 50 130 78

Route 6: 145 150 64 88 40 136 13 67 134 55

Route 7: 77 119 1 120 80 28 31 82 140 8 60 81

Route 8: 38 62 9 34 74 75 105 30 104 49 76

Route 9: 142 87 148 141 19 94 41 66 111 135 143 109

Route 10: 12 144 4 149 146 56 47 139

Route 11: 103 90 71 123 124 33 72 91 45 15 52 108

Route 12: 32 22 70 116 3 59 20 131 83 2 100

Route 13: 51 101 121 115 36 85 35 84 128

Route 14: 27 138 48 112 61 7 69 23 57 6 102 46

Total distance: 1158.41