Computers & Operations Research 68 (2016) 30–62

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Constructive heuristics for the Mixed Capacity Arc Routing Problem under Time Restrictions with Intermediate Facilities

Elias J. Willemse *, Johan W. Joubert [1]

Department of Industrial and Systems Engineering, University of Pretoria, 0002, South Africa

ARTICLE INFO

Available online 27 October 2015

Keywords:
Waste collection
Capacitated Arc Routing Problem
Mixed network
Intermediate Facilities
Time restrictions
Constructive heuristics
Minimise fleet size
Benchmark instances

ABSTRACT

The Mixed Capacity Arc Routing Problem under Time Restrictions with Intermediate Facilities (MCARPTIF) is an extension of the Arc Routing Problem under Capacity and Length Restrictions with Intermediate Facilities (CLARPIF) with application in municipal waste collection. This paper evaluates four constructive heuristics capable of computing feasible solutions for the MCARPTIF with a primary objective to either minimise total cost or to minimise the fleet size. The heuristics were adapted from Path-Scanning and Improved-Merge for the Mixed Capacitated Arc Routing Problem, and compared against two Route-First-Cluster-Second heuristics for the MCARPTIF. The objective was to identify the best performing heuristic for application purposes. In practice, the CARP is often solved for real-time or near real-time decision support. Computational time required by the heuristics was thus also evaluated. Identifying the best heuristic proved difficult due to a lack of realistic MCARPTIF benchmark sets, with the two CLARPIF sets predominantly solved in the literature not resembling actual waste collection instances. Route-First-Cluster-Second heuristics, linked with a new vehicle reduction heuristic performed the worst on the two CLARPIF sets, yet performed the best on new waste collection sets taken from the literature and introduced in this paper. Improved-Merge performed the best on two existing CLARPIF sets and on a realistic set with Intermediate-Facilities incident with the vehicle depot, but struggled on all other sets and in minimising fleet size. Path-Scanning was the most robust heuristic, performing reasonably well on all benchmark sets and both objectives. Results further show that due to the high computational time of one of the Route-First-Cluster-Second heuristics, which was only exposed on realistically sized sets, the slightly worse version is the best alternative when real-time support is required for waste collection applications.

1. Introduction

The Capacitated Arc Routing Problem (CARP), first proposed by Golden and Wong [21], is a classical routing problem where a subset of required edges have to be serviced by a fleet of homogeneous vehicles. The CARP has a wide range of applications such as security guard routing [44] and railway maintenance [24], and its well-known applications for winter gritting, street sweeping and waste collection. The objective of the problem is to determine routes of minimal total cost for a fleet of homogeneous vehicles so that each route starts and ends at the depot, each road segment with demand is serviced exactly once by a vehicle, and the sum of demand serviced by a route does not exceed vehicle capacity. The fleet size can be either limited, unlimited or left as a decision variable. Total cost can also be measured in total distance travelled or the sum of the time taken to complete all routes. We refer the reader to [10,11,15] for a comprehensive review of the problem.

Real residential waste collection problems cannot be approached exactly as a CARP due to specific operational conditions and constraints [39]. The CARP considers only undirected networks, whereas road networks may consist of one-way streets that can be traversed or serviced in only one direction, busy two-way streets that require each side to be serviced separately, and two-way streets that can be serviced in either direction. The Mixed CARP (MCARP), studied in [4,12,22,25], among others, and by Bautista et al. [2] for a waste collection application, accounts for these features but does not consider vehicle offloads at dumpsites. Often in practice, vehicles may unload their collected waste

* Corresponding author. Tel.: +27 71 890 2714.
E-mail addresses: ejwillemse@gmail.com (E.J. Willemse), johan.joubert@up.ac.za (J.W. Joubert).
[1] Tel.: +27 12 420 2843; fax: +27 12 362 5103.

http://dx.doi.org/10.1016/j.cor.2015.10.010
0305-0548/

at one of multiple Intermediate Facilities (IFs) such as dumpsites, return to the service area and continue collecting waste. The sum of demand collected on a subtrip between IF visits may then not exceed vehicle capacity and the route must include a final IF visit before returning to the depot. Furthermore, the total time of a vehicle route may not exceed a time restriction, typically equal to the available working hours per shift. This extension, originally termed the Arc Routing Problem under Capacity and Length Restrictions with Intermediate Facilities (CLARPIF), was introduced by Ghiani et al. [17] and is extended in this paper to the Mixed Capacitated Arc Routing Problem under Time Restrictions with Intermediate Facilities (MCARPTIF), thus accounting for mixed road networks.

Despite what we consider to be a common practical application, the problem, as defined in this paper, has only been formally studied in [45,46], and there has been no formal investigation and comparison on solution approaches to the problem. Work on the CLARPIF is limited to the contributions of Ghiani et al. [17,19] and Polacek et al. [35]. A close variant of the MCARPTIF is studied by Santos et al. [39] which deals with a single IF, a heterogeneous fleet and the demand at intersections as well road segments. Ghiani et al. [16] deal with an MCARPTIF with additional requirements such as a heterogeneous fleet where only certain vehicles can service certain street types. An extended MCARP is studied in [30,31] that also considers one IF but without a route time limit. Instead, each available vehicle in the fleet must be used and consist of at least two subtrips.

The aim of this paper is to contribute to research on the MCARPTIF by developing and testing constructive heuristics for the problem with a primary objective to either minimise the total cost, or to minimise the fleet size. Our focus on constructive heuristics was inspired by a practical waste management project. The four heuristics tested were adapted from *Path-Scanning* and *Improved-Merge* developed by Belenguer et al. [4] for the MCARP, and *Efficient-Route-Cluster* and *Route-Cluster* developed by Willemse and Joubert [46] for the MCARPTIF, which we linked with *Path-Scanning* giant-route generators. Since heuristics for mixed cases can be used as-is on their undirected versions, computational tests are also performed on the CLARPIF. The aim of the tests was to identify the best performing heuristics, which can then be used in waste collection routing applications and future studies on arc routing problems. In line with this objective, we analysed how the treatment of the vehicle fleet size influenced the performance of the heuristics, and we proposed a vehicle reduction procedure that allowed heuristics to better deal with cases where the fleet size has to be minimised. Lastly, benchmark tests were performed on a set of new realistic waste collection instances, thus improving the practical significance of our results. The problem instances are available as on-line supplementary files and from https://sites.google.com/site/wasteoptimisation, and open to researchers working on similar problems.

The following is an outline of the remainder of the paper. In Section 2 we review constructive heuristics for CARPs, discuss the current treatment of vehicle fleet size, and review heuristic evaluation criteria and existing benchmark sets used in similar studies. Section 3 presents formal MCARPTIF definitions and notations used in the paper, followed by algorithm descriptions of the constructive and vehicle reduction heuristics for the problem. In Section 4, we introduce new MCARPTIF benchmark instances, and compare the instances to existing sets. Also in Section 4, the heuristic evaluation method is described, and we report computational experiments on the heuristics, focusing on fleet size and solution cost minimisation as well as computation times. The paper concludes in Section 5 with a summary of our main findings and suggestions for future research.

2. Research motivation and related work

Since the CARP and thus all its extensions are $\mathcal{NP}$-Hard, the most effective methods for dealing with the problems are based on heuristic and metaheuristic solution techniques [11]. Lower-bounding and exact approaches have been successfully applied to the CARP [1,6] and MCARP [4,22], but as a result of their computational complexity, these approaches are not yet capable of dealing with realistically sized instances [27]. In the literature, the standard approach when dealing with a new CARP variant is to simultaneously develop lower and upper bounds. Classical benchmark sets are then modified to account for new problem characteristics, and upper bounds are compared against good lower bounds and against optimal solution values for smaller test instances. This approach can also be followed for the MCARPTIF; however, as motivated in this section, constructive heuristics on their own are still an important area of research. By focusing on constructive heuristics, and leaving lower bounds for future work, results on the MCARPTIF are of immediate value to practitioners if the heuristics are tested on realistic instances. Furthermore, there are two critical research gaps in literature, also discussed in this section, that first need to be recognised and addressed in order to improve the practical significance of heuristic and metaheuristic studies on CARPs.

2.1. Practical waste collection routing requirements

The motivation for our research stems from a project conducted for a metropolitan municipality in South Africa. The municipality's Waste Collection Department investigated the merits of a proposed recycling programme aimed at job creation. The goal was to increase recycling in the areas and outsource the collection and transportation process to members of the community. The Department would provide transportation equipment and the community would collect and sell collected material. As further motivation it was argued that recycling would reduce general municipal waste in the areas. We were subsequently tasked with investigating the potential impact of the programme on the Department's municipal waste collection routes. Operational constraints and considerations of the Department were consistent with the MCARPTIF, and since possible changes in routes were only one component of the programme, the problem had to be solved in a collaborative real-time planning environment involving stakeholders with different objectives. Scenarios were provided by stakeholders who would use our MCARPTIF solutions as an input for evaluating the scenarios, and proposing new ones. Stakeholders would often propose “small” changes to scenarios and expect immediate results. Such changes included using smaller vehicles with less capacity, extending working hours, and including night shifts. The MCARPTIF thus needed to be solved quickly, and on a continuous basis. These requirements are not unique in waste collection planning, see for example [39], and can be met using constructive heuristics. For the project, we identified and adapted the best MCARP heuristics to the MCARPTIF, and it was during our initial review on existing heuristics that we identified the research gaps addressed in this paper.

2.2. Constructive heuristics

Constructive heuristics for CARPs remain an important area of research. As motivated by Santos et al. [40], these heuristics generally provide good solutions in acceptable CPU time, which is an important criterion in many real waste collection applications. Solution times for metaheuristics can become large for even modestly sized instances, making them impractical for certain applications such as our case study. Owing to their simplicity, constructive heuristics are flexible and can be more easily modified to extensions of the CARP, which makes them easier to implement, and they do not require the determination and fine tuning of parameters. They also form the starting point for metaheuristics, see for instance [8,4,17,33,35]. Polacek et al. [35] show that for their CARP metaheuristic, linking it with better quality initial solutions leads to better convergence and higher quality final solutions. Lastly, constructive heuristics are used to solve sub-problems of certain CARPs such as those studied in [23,32].

Numerous constructive heuristics, which follow simply greedy rules to progressively build approximate feasible solutions, have been proposed for the CARP. Coutinho-Rodrigues et al. [12] compare the performance of classical implementations on the CARP and conclude that *Path-Scanning*, proposed in [21] and one of the most commonly used and studied heuristics, performs the best when CPU time is critical. Beginning with [34], *Path-Scanning* has been modified by including randomised construct mechanisms. This allows the heuristic to generate and evaluate a number of different solutions for a problem instance, from which the best is returned. Tests by Belenguer et al. [4] show that on average such versions need to evaluate between 4 and 21 solutions to match their deterministic versions. A *Path-Scanning* version for the CARP that makes use of an Ellipse-Rule is proposed by Santos et al. [40]; their tests have demonstrated that it performs better than the implementations of Golden and Wong [21] and Pearn [34], but the said *Path-Scanning* version requires fine-tuning of a heuristic parameter. Belenguer et al. [4] compare their *Path-Scanning* versions against their improved version of *Augment-Merge*, first proposed by Golden et al. [20], and *Ulusoy-Partitioning*, a *Route-First-Cluster-Second* heuristic, introduced by Ulusoy [42]. For CARP and MCARP benchmark instances, their *Augment-Merge* version, termed *Improved-Merge*, performed the best, followed by *Ulusoy-Partitioning* and then *Path-Scanning*. To demonstrate the robustness of their metaheuristic on the CLARPIF, Polacek et al. [35] use a random giant route generator and a *next-fit* bin-packing heuristic to purposely construct poor starting solutions, which are then improved using the meta-heuristic. Ghiani et al. [17] developed a *Route-First-Cluster-Second* heuristic to solve the CLARPIF, which can also be applied to the MCARPTIF. More efficient *Route-First-Cluster-Second* heuristics for the MCARPTIF are studied in [46], which can be used for large problem instances.

In this paper we extended the *Improved-Merge* and the *Path-Scanning* versions of Belenguer et al. [4] to the MCARPTIF. The heuristics are tested against the *Route-First-Cluster-Second* heuristics of Willemse and Joubert [46]. The aim of the analysis, which is our first research contribution, was to identify the best performing constructive heuristics based on their computing time, and ability to find minimum cost and minimum fleet size solutions.

2.3. Treatment of vehicle fleet size

The second contribution of this paper is the investigation of the vehicle fleet size, K, and how its treatment affects heuristic performance. In CARP literature, the fleet size is either fixed, assigned an upper bound value, left as a decision variable, or treated as unlimited.

In most waste collection applications, see for example [2,13,16,30,31,38,39], and consistent with the original formulation of the CARP [21], K is an input value, usually corresponding to the current fleet size and it is treated as an upper bound; a solution with more than K routes is considered infeasible. In contrast, studies on CARP heuristics typically treat K as either unlimited [1,7] or as a decision variable [4,16,19,26]. In fact, Lacomme et al. [25] consider a limited fleet size to be an extension of the CARP. In studies where K is a decision variable, it is not minimised. The only objective is to minimise total route cost, and the resulting number of routes determines the fleet size requirements. Exceptions include the work of Chu et al. [8] who deal with a Periodic CARP in which K can be fixed or minimised, and the work of Grandinetti et al. [23] on a multi-objective CARP where one of the three objectives is to minimise K. Lacomme et al. [25] and Willemse and Joubert [46] also propose versions of Ulusoy-Partitioning which minimises K as a primary or secondary objective.

It can be argued that the objectives of minimising cost and the fleet size are not in conflict; thus solution approaches need only focus on the former. Additional routes result in additional dead-heading time to and from depots and IFs, as well as offload time. Also, the improving search mechanisms of metaheuristics for the CLARPIF allow for the creation of new routes and combining existing routes [16,19,35]. Since the CLARPIF studies do not report on the required K for minimum cost solutions, it is unknown if these mechanisms are sufficient when dealing with cases where K is an upper bound. This issue was identified by Belenguer and Benavent [3] for the CARP, who found through their experiments that the number of vehicles used in corresponding CARP heuristic solutions is not always equal to the possible minimum.

In this paper, K is analysed when minimising K is treated as either the primary or the secondary objective. Our tests showed that when setting total cost minimisation as the primary objective, the resulting heuristic solutions require additional vehicles compared to best K solutions produced by the same heuristics. We also show that the performance ranking of constructive heuristics changes based on the treatment of K, and should be taken into account when choosing heuristics for application purposes. Lastly, to improve heuristic performance in situations where K should be minimised or is limited, we made small adjustments to the heuristics and developed a route reduction procedure, which we linked with the constructive heuristics, and compared the linked and unlinked versions.

2.4. Heuristic evaluation criteria

In CARP studies, heuristic performance is evaluated on two criteria: solution quality, usually in the form of total route costs and algorithm efficiency. When possible, solution quality is measured by comparing heuristic results to optimal values found on small instances and on lower bounds for larger instances. Tight bounds have been developed for the CARP [1,6], and recently for the MCARP [4,22]. Ghiani et al. [16] calculate weak lower bounds for the CLARPIF with one IF by using the algorithm of Rosa et al. [37]. None exist for the CLARPIF with more IFs, nor for the MCARPTIF, and their development falls beyond the scope of this paper.

When lower bounds are not available, an option is to substitute them with the best solutions found during all the computational tests [36,41], which serves our test purpose as it gives a general quality measure between heuristics. For our paper, we also measured solution quality in terms of the required number of vehicles, K. A tight lower bound on the optimal number of vehicles, K_{LB}, for a CARP solution can

be calculated using the following equation:

$$K_{LB} = \left\lceil \frac{\zeta}{Q} \right\rceil, \tag{1}$$

where ζ is the total demand of the required arcs and edges of the problem instance, and Q is vehicle capacity. For the MCARPTIF, Eq. (1) only gives a lower bound on the number of subtrips with IF visits, since routes are constrained by a time limit, not capacity. The K_{LB} can instead be calculated using the following equation:

$$K_{LB} = \left\lceil \frac{O + \lambda \left\lceil \frac{\zeta}{Q} \right\rceil}{L} \right\rceil, \tag{2}$$

where O is the total service time of the required arcs and edges of the problem, λ is the IF offloading time and L is a route limit. The dividend of Eq. (2) is a weak lower bound on solution cost as it ignores dead-heading times. As a result, the fleet size lower bound is also weak. Finding tight lower bounds for the MCARPTIF is not trivial and one option would be to divide good lower bound cost values by L. Otherwise, similar to total cost, solutions can be compared against best solutions found during all computational tests, which is the approach used in this paper.

Two factors that influence algorithm performance are heuristic parameters and choice of test instances, which is discussed in the next subsection. Constructive heuristics generally have few if any parameters, allowing them to be implemented quickly with little parameter tuning. A key input parameter of randomised heuristics is the number of runs the versions are allowed. A sufficient number of runs allows them to outperform their deterministic counterparts. Allowing more runs results in better final solutions, but requires more computational time. A balance is thus sought between the number of runs and solution quality. The investigation of this balance becomes critical for real applications. However, in literature, heuristics are either evaluated over 10–50 runs [4,12,34] or on very long runs, in the case of Santos et al. [40] in excess of 1000. Furthermore, heuristics produce random solutions, but except for Coutinho-Rodrigues et al. [12], results are only reported for one experiment per problem instance.

In this paper, our third research contribution is the statistical evaluation of algorithm performance. For this purpose, we modelled randomised heuristic runs as a series of Bernoulli trials. For each heuristic, we calculated the statistical cost and K intervals per problem instance, where each interval consists of a minimum observed value, an expected value and a 99th percentile value. The method relies on capturing cost and K values for each heuristic run, from which intervals can be calculated for any run level using binomial distribution functions. The number of expected runs required by random heuristics to break even with their deterministic version was also statistically calculated.

2.5. Benchmark instances

The last contribution of the paper is the development of new benchmark instance for the MCARPTIF, based on real-life instances that can be used in future studies. When evaluating and comparing heuristics, the choice of test instances is critical as their structure can influence heuristics performance. Real-life instances, such as those solved in [2,13,16,38,39], constitute good benchmarks to carry out performance evaluations, but are not openly available. As a result, studies on CARPs have relied on randomly constructed instances. Two benchmark sets, which are adapted from the *gdb* and *bccm*[2] sets introduced by Golden et al. [20] and [5] respectively, have been proposed for the CLARPIF by Ghiani et al. [17]. These instances are solved in Ghiani et al. [17,19] and in Polacek et al. [35]. As cautioned by Rardin and Uzsoy [36], classical benchmark sets introduce subtle biases that need to be recognised. The *gdb* and *bccm* benchmarks were mostly generated by hand [3] and do not closely model any real environment. Their purpose was to demonstrate that heuristics were capable of producing feasible solutions for the (then) new problems. Since all *gdb* instances have been solved to optimality, Lacomme et al. [25] and Corberán and Prins [11] state that they should no longer be used to compare CARP metaheuristics, and between [1,6], the *val* instances are also solved to optimality. Because of their small size, heuristics and metaheuristics scalability issues may also go unnoticed on these instances [29]. Despite their short-comings, both sets are still used to test and rank heuristics and metaheuristics; see for instance [26,28,35,40,43]. In terms of CLARPIF benchmarks, the original CLARPIF problem instances used by Ghiani et al. [17], and solved in Ghiani et al. [19] are no longer available, and applying the transformation process documented by the authors on the *gdb* and *val* sets results in inconsistencies. The only CLARPIF results available in the literature that can be used for validation and comparison of new approaches are those reported by Polacek et al. [35] on the adapted *bccm* instances, which have the same shortcomings of the original set.

Recent CARP studies have relied more on *eglese*, *egl-large* and *bmcv* instances [1,27,29], which are all based on actual applications. Eighty-six out of the 100 *bmcv* instances have been solved to optimality [1,6], limiting their usefulness, but only 11 of the 24 *egl* instances and none of the 10 *egl-large* instances have been optimally solved, thus rendering them good candidates for evaluating heuristics. Both, however, are undirected and based on snow removal. Belenguer et al. [4] address this issue by generating fifteen random *lpr* instances on mixed graphs that mimic waste collection. The authors also create 34 *mval* instances with mixed graphs derived from the *bccm* set. Corberán and Prins [11] note that in spite of the *lpr* set containing some of the largest CARP based instances, they are actually easier to solve. The *lpr* and *mval* sets are currently the only available MCARP benchmarks.

For the MCARPTIF, Willemse and Joubert [46] introduce three very large benchmark instances that are between 1.5 and 3.5 times larger than the biggest *lpr* instances. The authors also modify *lpr* instances into MCARPTIFs by including IFs and time limits. This paper introduces three new MCARPTIF benchmark instances based on an actual waste collection application. We also modify the *mval* instances to develop additional MCARPTIF benchmark sets. Solving benchmark instances is an important area of heuristic testing and therefore a portion of this paper is dedicated to the analyses of CLARPIF and MCARPTIF sets using a few basic metrics. Results for realistic sets should be prioritised to increase the practical significance of the research on CARPs

The objective of this paper was to adapt the best existing MCARP heuristics to the MCARPTIF, and evaluate them against two existing MCARPTIF heuristics to find the best performing heuristic for application purposes. For results to be practically significant, three research gaps in current CARP literature are addressed in this paper. First, heuristics are evaluated on their ability to minimise the fleet size. Second, heuristic performance of randomised versions is statistically evaluated. And third, realistic waste collection benchmark instances are introduced and solved.

[2] The *bccm* set is also referred to as the *val* set.

3. Constructive heuristics for the MCARPTIF

Consistent with the MCARP notation of Belenguer et al. [4], the MCARPTIF considers a mixed graph, $\boldsymbol{G}=(\boldsymbol{V},\boldsymbol{E}\cup\boldsymbol{A})$, where $\boldsymbol{V}$ represents the set of vertices, $\boldsymbol{E}$ represents the set of undirected edges where an edge links two vertices and may be traversed in both directions, and $\boldsymbol{A}$ represents the set of arcs where an arc also links two vertices but can only be traversed in one direction. Edges requiring each side to be serviced separately are modelled as two opposing arcs. A subset of required edges and arcs, $\boldsymbol{E}_r\subseteq\boldsymbol{E}$ and $\boldsymbol{A}_r\subseteq\boldsymbol{A}$, must be serviced by a fleet of K homogeneous vehicles with limited capacity, Q, that are based at the depot vertex, v_1. Each vehicle route must start and end at the depot, and the sum of demand on a route may not exceed Q. The number of routes, K, must be less than the number of available vehicles, K_{UB}, and $K_{UB}\to\infty$ when an unlimited number of vehicles is available. For the MCARPTIF, vehicles are allowed to unload their waste at any IF at a time of λ, and resume their collection routes. IFs are modelled in $\boldsymbol{G}$ as $\boldsymbol{\Gamma}$, where $\boldsymbol{\Gamma}\subset\boldsymbol{V}$. The sum of demand on each subtrip between IF visits may not exceed Q, and unless $v_1\in\boldsymbol{\Gamma}$, each vehicle has to visit an IF before returning to the depot. Lastly, a time restriction, L, is imposed on each vehicle route. We study two versions of the problem. The objective of the first is to find a set of K feasible vehicle routes, servicing all required edges and arcs, that pre-emptively minimises total cost then the vehicle fleet size. The objective of the second version, which is seldom addressed in the literature, is to find a set of K feasible vehicle routes that service all required edges and arcs, and pre-emptively minimises the vehicle fleet size then total cost. When $\boldsymbol{A}$ is an empty set, denoted $\boldsymbol{A}=\varnothing$, the problem reduces to the CLARPIF; when $v_1\in\boldsymbol{\Gamma}$ and $L\to\infty$, the problem reduces to the MCARP; and when both sets of conditions hold, the problem reduces to the CARP. MCARPTIF heuristics can thus be used to solve all four problems.

3.1. Mixed network transformation and solution representation

Consistent with the work of Belenguer et al. [4], Lacomme et al. [25], and Mourão et al. [32], the mixed graph $\boldsymbol{G}$ can be transformed into a fully directed graph, $\boldsymbol{G}^*=(\boldsymbol{V},\boldsymbol{A}^*)$, by replacing each edge, $(v_i,v_j)\in\boldsymbol{E}$, with two opposite arcs, $\{(v_i,v_j),(v_j,v_i)\}\in\boldsymbol{A}^*$. Arcs in $\boldsymbol{A}^*$ are identified by indices from 1 to β, where $\beta=|\boldsymbol{A}^*|$. Each arc u has a dead-heading time, $c(u)$, denoting the time of traversing the arc without servicing it, and the total cost of any route cannot exceed the maximum allowed route time, L.

The required arcs, $\boldsymbol{A}_r$, and edges, $\boldsymbol{E}_r$, in $\boldsymbol{G}$ correspond to a subset $\boldsymbol{R}\subseteq\boldsymbol{A}^*$ of required arcs such that $|\boldsymbol{R}|=2|\boldsymbol{E}_r|+|\boldsymbol{A}_r|$. Each arc, $u\in\boldsymbol{R}$, has a demand, $q(u)$, a collection time, $w(u)$, and a pointer, $inv(u)$. Each required arc in the original graph, $\boldsymbol{G}$, is coded in $\boldsymbol{R}$ by one arc, u, with $inv(u)=0$, while each required edge is encoded as two opposite arcs, u and v, such that $inv(u)=v$ and $inv(v)=u$. An arc task, u, represents an edge if $inv(u)\neq 0$. Furthermore, $q(u)=q(v)$, $c(u)=c(v)$ and $w(u)=w(v)$ for an edge's two opposite arcs, u and v. The depot is modelled by including in $\boldsymbol{A}^*$ a dummy arc, $\sigma=(v_1,v_1)$. IFs are also modelled in $\boldsymbol{A}^*$ as a set of dummy arcs $\boldsymbol{\Phi}_i\in\boldsymbol{I}$, and $\boldsymbol{\Phi}_i=(v_i,v_i)\forall v_i\in\boldsymbol{\Gamma}\subset\boldsymbol{A}^*$. All dummy arcs, including σ, have the following properties:

$$\left.\begin{array}{l} inv(u)=u \\ c(u)=0 \\ w(u)=0 \\ q(u)=0 \end{array}\right\}\ \forall\ u\in\boldsymbol{I}\cup\{\sigma\}. \tag{3}$$

An MCARPTIF solution, $\boldsymbol{T}$, is a list, $[\boldsymbol{T}_1,\ldots,\boldsymbol{T}_K]$, of K vehicle routes. Each route, $\boldsymbol{T}_i$, is a list of subtrips $[\boldsymbol{T}_{i,1},\ldots,\boldsymbol{T}_{i,|\boldsymbol{T}_i|}]$, and each subtrip, $\boldsymbol{T}_{i,j}$, consists of a list of arc tasks $[T_{i,j,1},\ldots,T_{i,j,|T_{i,j}|}]$. The travel time for the shortest path from arc u to arc v, which excludes the time of dead-heading u and v, is given by $D(u,v)$, which is pre-calculated for all arcs in $\boldsymbol{A}^*$. Shortest paths can be efficiently calculated using a modified version of Dijkstra's algorithm which can incorporate forbidden turns and turn-penalties; we refer the reader to [25,39] for implementation details.

The first subtrip, $\boldsymbol{T}_{i,1}$, in a route starts at the depot, and the last subtrip, $\boldsymbol{T}_{i,|\boldsymbol{T}_i|}$, ends with an IF and depot visit. All other subtrips start and end with IF visits while taking care that the starting IF of a subtrip coincides with the end IF of the previous subtrip. It is assumed that the shortest path is always followed between consecutive tasks. For the MCARPTIF, the best IF to visit, $\Phi^*(u,v)$, after servicing arc u and before servicing arc v can be pre-calculated using Eq. (4), and the duration of the visit, $\mu^*(u,v)$, excluding offloading time, is given by Eq. (5):

$$\Phi^*(u,v)=\arg\min\{D(u,x)+D(x,v):x\in\boldsymbol{I}\}, \tag{4}$$

$$\mu^*(u,v)=D\big(u,\Phi^*(u,v)\big)+D\big(\Phi^*(u,v),v\big). \tag{5}$$

The best IF-visit calculation differs from the calculation of Belenguer et al. [4] as it excludes offload time, which instead we include when calculating subtrip and route costs. It is assumed that the best IF is always visited between two arcs and between an arc and the depot.

For the MCARPTIF the load of a subtrip, $load(\boldsymbol{T}_{i,j})$, which may not exceed Q, is calculated as follows:

$$load(\boldsymbol{T}_{i,j})=\sum_{n=1}^{|\boldsymbol{T}_{i,j}|}q(T_{i,j,n}). \tag{6}$$

Table 1
Glossary of mathematical symbols.

Problem symbols		Arcs (u and v) related symbols	
$\boldsymbol{R}$	Set of required arcs	$c(u)$	Dead-heading time of u
K_{UB}	Fleet size limit	$q(u)$	Demand of u
Q	Vehicle capacity	$w(u)$	Service time of u
L	Route time limit	$inv(u)$	Pointer to opposite arc of u
σ	Depot dummy arc	D(u,v)	Shortest dead-heading path time from u to v
$\boldsymbol{\Gamma}$	Set of IF vertices	Φ*(u,v)	Best IF to visit between u and v (Eq. (4))
$\boldsymbol{I}$	Set of IF dummy arcs	μ*(u,v)	Duration of best IF visit between u and v (Eq. (5))
λ	Unload cost		

Table 2
MCARPTIF solution representation symbols.

$\boldsymbol{T}$	MCARPTIF solution
$K = \lvert \boldsymbol{T} \rvert$	Number of vehicles required for $\boldsymbol{T}$
$\boldsymbol{T}_i$	Route i
$\boldsymbol{T}_{i,j}$	Subtrip j of route $\boldsymbol{T}_i$
$T_{i,j,n}$	nth arc task in subtrip $\boldsymbol{T}_{i,j}$
$load(\boldsymbol{T}_{i,j})$	Demand of subtrip $\boldsymbol{T}_{i,j}$ (Eq. (6))
$Z_s(\boldsymbol{T}_{i,j})$	Cost of subtrip $\boldsymbol{T}_{i,j}$ (Eq. (7))
$Z_P(\boldsymbol{T}_i)$	Partial cost of route $\boldsymbol{T}_i$ (Eq. (8))
$Z(\boldsymbol{T}_i)$	Cost of route $\boldsymbol{T}_i$ (Eq. (9))

The cost of subtrip $\boldsymbol{T}_{ij}$ is calculated using Eq. (7), and the partial cost of the route $\boldsymbol{T}_i$ using Eq. (8):

$$Z_s(\boldsymbol{T}_{i,j}) = \sum_{n=1}^{|\boldsymbol{T}_{i,j}|-1} \left(D(T_{i,j,n}, T_{i,j,n+1})\right) + \sum_{n=1}^{|\boldsymbol{T}_{i,j}|} w(T_{i,j,n}) + \lambda, \tag{7}$$

$$Z_P(\boldsymbol{T}_i) = \sum_{j=1}^{|\boldsymbol{T}_i|} Z_s(\boldsymbol{T}_{i,j}). \tag{8}$$

Even though routes contain dummy arcs, Eqs. (6) and (7) remain accurate as a result of Eq. (3). In certain cases, and similar to the notation of Belenguer et al. [4], it is convenient to only include required arcs in $\boldsymbol{T}_i$, which we refer to as a partial route. When this is the case, Eqs. (6) and (7) will still be used, and the route's cost will be calculated using the following equation:

$$Z(\boldsymbol{T}_i) = Z_P(\boldsymbol{T}_i) + D(\sigma, T_{i,1,1}) + \sum_{j=2}^{|\boldsymbol{T}_i|-1} \mu^*(T_{i,j,|\boldsymbol{T}_{i,j}|}, T_{i,j+1,1}) + \mu^*(T_{i,|\boldsymbol{T}_i|,|\boldsymbol{T}_{i,j}|}, \sigma), \tag{9}$$

in which dead-heading times to and from the depot and IFs are automatically added. When referring to an MCARP solution, also denoted by $\boldsymbol{T}$, it consists of a list, $[\boldsymbol{T}_1, \ldots, \boldsymbol{T}_K]$, of K vehicle routes, and each route, $\boldsymbol{T}_i$, consists of a list of tasks $[T_{i,1}, \ldots, T_{i,|\boldsymbol{T}_i|}]$. Eqs. (6) and (7) are also used to calculate the demand and cost of an MCARP route.

The notations introduced in this section are summarised in Tables 1 and 2.

3.2. Path-Scanning

The first MCARP heuristic that we adapted for the MCARPTIF was *Path-Scanning*. For the MCARP the algorithm starts with a single route containing the depot. The algorithm then iteratively adds the closest required unserviced arc, time-wise, to the end of the current route, while ensuring that there is sufficient remaining capacity on the route to service the arc. If there is insufficient capacity to add any unserviced arc the route is closed by adding a final visit back to the depot. A new route is then opened and the process repeats. When all required arcs have been added to a route, the last open route is closed and the process terminates.

The heuristic deals with mixed networks by evaluating all unserviced arcs and both arc orientations of edges when searching for the closest arc to add to a route. When an arc is added to a solution, its inverse (if it exists) is also marked as serviced. This allows the algorithm to deal with both undirected and mixed networks. Let u be the last arc added to $\boldsymbol{T}_i$ and let $\boldsymbol{R}_s$ be the set of required arcs still requiring service. Further let

$$\boldsymbol{P}^{(i)} = \{x \in \boldsymbol{R}_s : load(\boldsymbol{T}_i) + q(x) \leq Q\}, \tag{10}$$

be the set of arcs that can be added without exceeding vehicle capacity. The next arc, v, to be added to $\boldsymbol{T}_i$ is given by

$$v = \arg\min \{D(u,x) : x \in \boldsymbol{P}^{(i)}\}. \tag{11}$$

For the CARP it is common for the heuristic to find multiple unserviced arcs that are at the minimum distance from arc u. These arcs are referred to as the set $\boldsymbol{S}$ of "tied" arcs, given by the following equations:

$$d^* = \min \{D(u,x) : x \in \boldsymbol{P}^{(i)}\}, \tag{12}$$

$$\boldsymbol{S} = \{x \in \boldsymbol{P}^{(i)} : D(u,x) = d^*\}. \tag{13}$$

Different conditions have been proposed to select the arc to add from $\boldsymbol{S}$, the most popular being the five rules proposed by Golden and Wong [21], which are:

$$\text{Rule 1} : v = \arg\max\{D(x,\sigma) : x \in \boldsymbol{S}\} \tag{14}$$

$$\text{Rule 2} : v = \arg\min\{D(x,\sigma) : x \in \boldsymbol{S}\} \tag{15}$$

$$\text{Rule 3} : v = \arg\max\left\{\frac{w(x)}{q(x)} : x \in \boldsymbol{S}\right\} \tag{16}$$

$$\text{Rule 4} : v = \arg\min\left\{\frac{w(x)}{q(x)} : x \in \boldsymbol{S}\right\} \tag{17}$$

$$\text{Rule 5}: v = \begin{cases} \text{use Eq. (14) if } load(\boldsymbol{T}_i) \leq 0.5Q, \\ \text{otherwise use Eq. (15).} \end{cases} \tag{18}$$

Rules 1 and 2 respectively maximise and minimise the time to the depot, and Rules 3 and 4 respectively maximise and minimise the arc yield (service time divided by demand). Rule 5 forces the addition of arcs that are further away from the depot when the vehicle is less than half-empty, and closer to the depot otherwise. The algorithm is executed five times using a different rule for each run to choose v from $\pm bS$, and the best solution from the five runs is chosen. Pearn [34] considers a random tie-break scheme, whereby a rule is uniformly chosen at random whenever $|\boldsymbol{S}| > 1$. The advantage of this approach is that an arbitrary number of solutions can be generated, and the best returned. Belenguer et al. [4] propose a rule-free version, termed *Path-Scanning-Random-Link*, whereby the arc to add is uniformly chosen at random from $\boldsymbol{S}$. This approach also allows for an arbitrary number of solutions to be generated, and the best chosen.

3.2.1. Path-Scanning for the MCARPTIF

The MCARPTIF sees the introduction of subtrips, $\boldsymbol{T}_{i,j}$, and the demand collected on a subtrip cannot exceed Q. The total time of a route cannot exceed L. Let u be the last arc added to subtrip $\boldsymbol{T}_{i,j}$ and let $x \in \boldsymbol{R}_s$ be a candidate arc still requiring service. The algorithm has to check for both the capacity limit on a subtrip and the total time of a route. As a result three sets are defined. The first set,

$$\boldsymbol{P}^{(i)} = \{k \in \boldsymbol{R}_s : Z(\boldsymbol{T}_i) + D(u,x) + w(x) + \mu^*(x,\sigma) \leq L\}, \tag{19}$$

consists of arcs that can be added to the route without exceeding L. If $\boldsymbol{P}^{(i)} = \varnothing$, the route is full as the time of servicing any arc x and returning to the final IF and depot visit exceeds L. The second set,

$$\boldsymbol{P}^{(i,j)} = \{x \in \boldsymbol{P}^{(i)} : load(\boldsymbol{T}_{i,j}) + q(x) \leq Q\}, \tag{20}$$

is a subset of $\boldsymbol{P}^{(i)}$ and consists of arcs that can be added to the current subtrip without exceeding Q. The third set,

$$\boldsymbol{P}^{(i)}_{IF} = \{x \in \boldsymbol{P}^{(i)} : Z(\boldsymbol{T}_i) + \mu^*(u,x) + w(x) + \mu^*(x,\sigma) + \lambda \leq L\}, \tag{21}$$

consists of arcs that can be added to a route after an IF visit, hence in a new subtrip, without exceeding L. If both $\boldsymbol{P}^{(i,j)} = \varnothing$ and $\boldsymbol{P}^{(i)}_{IF} = \varnothing$, the route is full. The arc from $\boldsymbol{S}$ to be added to subtrip $\boldsymbol{T}_{i,j}$ is given by:

$$d^* = \begin{cases} \min\{D(u,x) : x \in \boldsymbol{P}^{(i,j)}\} & \text{if } \boldsymbol{P}^{(i,j)} \neq \varnothing, \\ \infty & \text{otherwise,} \end{cases} \tag{22}$$

$$\boldsymbol{S} = \{x \in \boldsymbol{P}^{(i,j)} : D(u,x) = d^*\}, \tag{23}$$

and the next arc from $\boldsymbol{S}_{IF}$ to be added to a new subtrip $T_{i,j+1}$ is given by:

$$d^*_{IF} = \begin{cases} \min\{\mu^*(u,x) : x \in \boldsymbol{P}^{(i)}_{IF}\} & \text{if } \boldsymbol{P}^{(i)}_{IF} \neq \varnothing, \\ \infty & \text{otherwise,} \end{cases} \tag{24}$$

$$\boldsymbol{S}_{IF} = \{x \in \boldsymbol{P}^{(i)}_{IF} : \mu^*(u,x) = d^*_{IF}\}. \tag{25}$$

If $\boldsymbol{P}^{(i)}_{IF} \neq \varnothing$ and $d^*_{IF} < d^*$, it is cheaper to visit an IF and add an unserviced arc than to directly add an unserviced arc to the current subtrip. This requires the heuristic to find and compare d^* and d^*_{IF}, which adds to its computational complexity. A more efficient version can be used that only calculates $\boldsymbol{P}^{(i)}_{IF}$ if $\boldsymbol{P}^{(i,j)} = \varnothing$. During our preliminary tests we found both the efficient and full versions to perform similarly and limit our results to the efficient version.

To allow the heuristic to deal with multiple closest arcs, the tie-break rules can be updated since vehicles have to visit IFs. Santos et al. [39] update the five rules by using the travel time from the candidate arc to the IF instead of the travel time to the depot. This straight modification is only possible when the problem considers one IF. For the case with multiple IFs we propose that the travel time from the candidate arc to the nearest IF should be used. Furthermore, the original rules involving depot travel times are still valid for the MCARPTIF and need not be replaced. Instead, rules involving IF travel times are considered as additional rules and formally defined as follows:

$$\text{Rule 6}: v = \arg\max\{D(x,y) : x \in \boldsymbol{S}, y \in \boldsymbol{I}\} \tag{26}$$

$$\text{Rule 7}: v = \arg\min\{D(x,y) : x \in \boldsymbol{S}, y \in \boldsymbol{I}\} \tag{27}$$

An eighth rule that we propose takes into account L, whereby arcs that are closer to the last IF and depot visit are chosen when the route is within 75% of its time limit. Otherwise if the subtrip is more than half full, arcs closer to the nearest IF are chosen:

$$\text{Rule 8}: v = \begin{cases} \operatorname{argmin}\{\mu^*(x,\sigma) : x \in \boldsymbol{S}\} \text{ if } Z(\boldsymbol{T}) \geq 0.75L, \\ \text{otherwise use Eq. (26) if } load(\boldsymbol{T}_{i,j}) \leq 0.5Q, \\ \text{otherwise use Eq. (27).} \end{cases} \tag{28}$$

For the deterministic version, one of the eight rules can be applied per run, and the best solution over the eight runs returned. Preliminary tests found none of the rules to be redundant in that each rule resulted in the best solution for at least a few problem instances. For the randomised version, one of the eight rules or of a subset of the eight rules can be chosen at random. The simplest option is to avoid using any rules and choose a closest arc at random, in the same manner as for *Path-Scanning-Random-Link* for the MCARP.

Since *Path-Scanning* produces different solutions per run it is embedded in a high-level procedure, *nConstruct* (Appendix A.1, Algorithm 1), which generates σ solutions and returns the best fleet size solution, the best route cost solution, and the best route cost solution while adhering to the fleet limits if it is applied. A detailed description of *Path-Scanning* generating an MCARPTIF solution is shown in Algorithm 2, Appendix A.2. In terms of computational complexity, all versions of *Path-Scanning* can generate a solution within $\mathcal{O}(\tau^2)$, where $\tau = |\boldsymbol{R}|$, making it efficient even when dealing with large problem instances.

This paper presents results for two versions of *Path-Scanning*. The first is the deterministic version completing eight runs, each linked with one of the eight tie-break rules and returning the least cost and least fleet size solutions. The second is the randomised rule-free version that chooses a closest arc at random and completes a user-specified number of runs, and returns the best solutions found for the two primary objectives. Tests were also conducted on random-rule based versions, which included a version that chooses from Rules 1 to 5, a version that chooses from Rules 1 and 2 and 6 to 8. Both randomised versions performed similarly to the *Path-Scanning-Random-Link* version. We report only on results for *Path-Scanning-Random-Link* since it is the easiest to implement.

3.3. Merge

The second heuristic that we modified for the MCARPTIF was *Improved-Merge*, developed by Belenguer et al. [4] for the MCARP. Their algorithm is a simplified, yet an improved version of *Augment-Merge* [20]. For the MCARPTIF it consists of two steps:

Step 1 : For each required arc, construct a feasible route servicing the arc, starting at the depot, and ending with the best IF visit followed by the depot. Where two arcs are opposite and represent the same edge task, only include the route servicing the arc in its best orientation.

Step 2 : Subject to subtrip capacity and route time limits, evaluate the merge of any two routes. Merge the two routes which yield the best saving, and repeat Step 2. If no feasible merge can be found, stop.

Using partial routes, that is without dummy arcs, two routes, $\boldsymbol{T}_i$ and $\boldsymbol{T}_j$, can be directly merged by combining the last subtrip of $\boldsymbol{T}_i$ with the first subtrip in $\boldsymbol{T}_j$. We refer to such as merge of the routes $\boldsymbol{T}_i$ and $\boldsymbol{T}_j$ as $i \rightarrow j$. If $\boldsymbol{T}_i = [[a],[b,u]]$ and $\boldsymbol{T}_j = [[v,x]]$, merge would result in $\boldsymbol{T}_{i\rightarrow j} = [[a],[b,u,v,x]]$, subject to the demand of the combined subtrips not exceeding Q, and the time of the merged routes not exceeding L. The merge saving, m, is:

$$m = \mu^*(u,\sigma) + D(\sigma,v) + \lambda - D(u,v), \tag{29}$$

which is the positive travel time saving of going directly from arc u to v, instead of going from u to an IF, incurring offload time, going to the depot, and the next route going from the depot to arc v. Routes can also be merged through best IF visits, resulting in $\boldsymbol{T}_{i\Rightarrow j} = [[a],[b,u],[v,x]]$, which is only subject to the time of the merged routes not exceeding L. We refer to such a merge between routes $\boldsymbol{T}_i$ and $\boldsymbol{T}_j$ as $i \Rightarrow j$. For this case, the positive merge saving is:

$$m_{IF} = \mu^*(u,\sigma) + D(\sigma,v) - \mu^*(u,v). \tag{30}$$

In addition to these two mergers, routes can also be merged as $j \rightarrow i$ and $j \Rightarrow i$. Since the CARP considers an undirected network, CARP routes are symmetrical, meaning route $\boldsymbol{T}_i = [u,v,x]$ has the same cost as $\boldsymbol{T}_{i'} = [inv(x), inv(v), inv(u)]$, and $j' \rightarrow i'$ gives the same saving as $i \rightarrow j$. This results in there being an additional two unique mergers: $i \rightarrow j'$ and $i' \rightarrow j$, where i' is the inverse of route i. These four mergers capture all possible savings between two CARP routes. The same does not apply to the MCARP, since not all arc tasks have inverse tasks, and even if they do, the route is only symmetrical if the dead-heading times between these tasks are the same in both directions. To keep Merge efficient, Belenguer et al. [4] propose that mergers involving i' and j' only be considered if both routes $\boldsymbol{T}_i$ and $\boldsymbol{T}_j$ are symmetrical and the new dead-heading path linking the routes is also symmetrical. If this condition is not enforced, all eight merge orientations will have to be calculated. We propose the same for the MCARPTIF and the CLARPIF, even though the latter is undirected. For the CLARPIF, the first subtrip in a route starts with $[\sigma,u,\ldots]$ and its last subtrip ends with $[\ldots,v,\phi_i,\sigma]$, so unless $D(\sigma,inv(v)) = \mu^*(inv(u),\sigma)$ the route is asymmetrical. When merging two routes, this condition must also hold between the first arc in route $\boldsymbol{T}_i$ and the last arc in route $\boldsymbol{T}_j$, otherwise mergers involving their inverse will have different savings.

Similar to *Path-Scanning*, it may happen that different route mergers have equal savings. For the MCARP, Belenguer et al. [4] propose that a route demand discrepancy rule be used to break ties, referred to as *Improved-Merge*, whereby $\boldsymbol{T}_i$ and $\boldsymbol{T}_j$ are chosen to maximise $|load(\boldsymbol{T}_i) - load(\boldsymbol{T}_j)|$. For the MCARPTIF this rule can be used when breaking ties between direct mergers such that $|load(\boldsymbol{T}_{i,|\boldsymbol{T}_i|}) - load(\boldsymbol{T}_{j,1})|$ is maximised. A cost discrepancy rule, maximising $|Z(\boldsymbol{T}_i) - Z(\boldsymbol{T}_j)|$, can be used to break ties between mergers through IFs. An alternative is to only use the cost discrepancy rule; and we found this version to dominate the hybrid load and cost discrepancy rule. Similar to *Path-Scanning*, another option for the MCARPTIF is to break-ties by randomly choosing a merger. We refer to this version as *Randomised-Merge*. It has the ability to produce different solutions over multiple runs, from which the best can be returned. Multiple runs do require more computational time, and more efficient implementations become critical, especially when dealing with realistically sized instances. Belenguer et al. [4], who implemented an $\mathcal{O}(\tau^3)$ version where $\tau = |\boldsymbol{R}|$, following the structure just described, note that *Improved-Merge* can be implemented in a non-trivial way in $\mathcal{O}(\tau^2 \log \tau)$. In this paper we present such an implementation.

3.3.1. Efficient merge implementation

Referring to Eqs. (29) and (30), the merge savings of $i \rightarrow j$ and $i \Rightarrow j$ only depend on the last arc in subtrip $\boldsymbol{T}_{i,|\boldsymbol{T}_i|}$ and the first arc in subtrip $\boldsymbol{T}_{j,1}$. Merge savings between all arcs are calculated in the first merge phase, and the savings remain the same in subsequent phases. Merge savings thus only have to be calculated once between all arcs and stored in a merge savings list, $\boldsymbol{M} = [M_1,\ldots,M_{|\boldsymbol{M}|}]$. The savings list is then sorted in nonincreasing order, and starting with the first entry, $m = M_1$, the merge can be implemented if it is feasible, otherwise the next merge in the list is evaluated. This repeats until the entire list has been scanned. For the efficient implementation, we again use partial routes without dummy arcs, and define mergers between arcs instead of routes, meaning $u \rightarrow v$ is the direct merge of arcs u and v, and $u \Rightarrow v$ is the merge of arcs u and v via an IF visit.

Step 1: *Initialisation*: For the first step, a route with one subtrip is constructed for each required arc:

$$\boldsymbol{T} = \Big[[[u]] \quad \forall \quad u \in \boldsymbol{R}\Big]. \tag{31}$$

We define $T^{-1}(u)$ as a dynamic pointer function, mapping arc u to its route's index i, which is updated after each merge:

$$T^{-1}(u)=\begin{cases} i\in\{1,\ldots,|\boldsymbol{T}|\}: u\in\{T_{i,1,1}\}\cup\{\boldsymbol{T}_{i,|\boldsymbol{T}_i|},\boldsymbol{T}_{i,|\boldsymbol{T}_i|}\}, \\ 0 \qquad\qquad\qquad\qquad\qquad\qquad \text{otherwise}, \end{cases} \tag{32}$$

A merge can only involve the last arc in $\boldsymbol{T}_i$ and the first arc in $\boldsymbol{T}_j$. The rest of the arcs in the route are "locked-in". When $inv(u)=0$, both u and $inv(u)$ are included in $\boldsymbol{T}$ by Eq. (31), instead of including only the best orientation. The reason is that mergers involving bad orientations may still be good enough to overcome extra costs incurred by the orientation. To check if such mergers are indeed better, a penalty function is defined:

$$z(u)=Z\left(\boldsymbol{T}_{T^{-1}(u)}\right), \tag{33}$$

$$pen(u)=\begin{cases} z(u)-z(inv(u)) & : inv(u)\neq 0, z(u)>z(inv(u)) \\ 0 & \text{otherwise}, \end{cases} \tag{34}$$

$$p(u,v)=pen(u)+pen(v), \tag{35}$$

For convenience, Eq. (33) defines a cost function, $z(u)$, that gives the cost of an arc's route. The penalty $p(u,v)$, given by Eqs. (34) and (35), is to be subtracted from the merge saving between arcs u and v. The merge savings list is calculated using Eqs. (36)–(38), which consist of mergers resulting from $u\rightarrow v$ and from $u\Rightarrow v$:

$$\begin{aligned}&\boldsymbol{M}_{\rightarrow}=\big[\mu^*(u,\sigma)+D(\sigma,v)+\lambda-D(u,v)-p(u,v)\\&\forall\ (u,v): u,v\in\boldsymbol{R}, v\notin\{u,inv(u)\}\big],\end{aligned} \tag{36}$$

$$\begin{aligned}&\boldsymbol{M}_{\Rightarrow}=\big[\mu^*(u,\sigma)+D(\sigma,v)-\mu^*(u,v)-p(u,v)\\&\forall\ (u,v): u,v\in\boldsymbol{R}, v\notin\{u,inv(u)\}\big],\end{aligned} \tag{37}$$

$$\boldsymbol{M}=\boldsymbol{M}_{\rightarrow}\cup\boldsymbol{M}_{\Rightarrow}. \tag{38}$$

Two pointer functions, $\boldsymbol{M}_{\rightarrow}^{-1}(m)$ and $\boldsymbol{M}_{\Rightarrow}^{-1}(m)$, are used to map the merge saving, m, to the set of arc pairs, (u,v), that results in the saving:

$$\boldsymbol{M}_{\rightarrow}^{-1}(m)=\begin{cases}\{(u,v): \mu^*(u,\sigma)+D(\sigma,v)+\lambda-D(u,v)-p(u,v)=m\}\\ \varnothing \text{ otherwise},\end{cases} \tag{39}$$

$$\boldsymbol{M}_{\Rightarrow}^{-1}(m)=\begin{cases}\{(u,v): \mu^*(u,\sigma)+D(\sigma,v)-\mu^*(u,v)-p(u,v)=m\}\\ \varnothing \text{ otherwise}.\end{cases} \tag{40}$$

After populating $\boldsymbol{M}$ with all the merge savings, the duplicate savings are removed, the list is sorted in nonincreasing order, and $n=1$ so that $m=M_1$ is the first merge evaluated in Step 2 of the heuristic. The mappings return all the arc pairs in the case of tied-mergers with equal savings; hence why duplicates can be removed from $\boldsymbol{M}$.

Step 2: *Feasible Merge-Identification*: For the second step, let $m=M_n$. A merge between any two arcs u and v involving routes $i=T^{-1}(u)$ and $j=T^{-1}(v)$ is only feasible if the arc is still in the solution (Eq. (41)), arc u is the last arc in its route and arc v the first (Eq. (42)), the merge does not involve the same routes (Eqs. (43) and (44)), and the cost of the merged routes does not exceed the route time limit (Eq. (45)):

$$i\neq 0 \text{ and } j\neq 0, \tag{41}$$

$$u=\boldsymbol{T}_{i,|\boldsymbol{T}|,\boldsymbol{T}_{i,|\boldsymbol{T}_i|}} \text{ and } v=\boldsymbol{T}_{j,1,1}, \tag{42}$$

$$i\neq j, \tag{43}$$

$$i\neq T^{-1}(inv(v)): inv(v)\neq 0 \text{ and } T^{-1}(inv(v))\neq 0, \tag{44}$$

$$z(u)+z(v)-m-p(u,v)\leq L. \tag{45}$$

Mergers may be disqualified by Eq. (41) if their opposite arcs are in asymmetrical routes which are the product of previous mergers. Eq. (42) checks that arcs u and v are not already "locked-in". If mergers are allowed with inverted symmetrical routes, Eq. (44) ensures that a symmetrical route is not merged with its inverted self. In Eq. (45) the orientation penalties are subtracted since they are not actually incurred in a route. If a merge is to be implemented as $u\rightarrow v$, it is only feasible if the demand of the merged subtrips does not exceed vehicle capacity:

$$load(\boldsymbol{T}_{i,|\boldsymbol{T}_i|})+load(\boldsymbol{T}_{j,1})\leq Q. \tag{46}$$

The sets of feasible mergers, $\boldsymbol{M}_{\rightarrow}^{-1}(m)$ and $\boldsymbol{M}_{\Rightarrow}^{-1}(m)$, are thus reduced to $\boldsymbol{F}_{\rightarrow}$ and $\boldsymbol{F}_{\Rightarrow}$ that contain only feasible mergers:

$$\boldsymbol{F}_{\rightarrow}=\{(u,v)\in\boldsymbol{M}_{\rightarrow}^{-1}(m): \text{Eqs. (43) to (46)}\}, \tag{47}$$

$$\boldsymbol{F}_{\Rightarrow}=\{(u,v)\in\boldsymbol{M}_{\Rightarrow}^{-1}(m): \text{Eqs. (43) to (45)}\}. \tag{48}$$

Step 3: *Merge-Implementation*: If $\boldsymbol{F}_{\rightarrow}=\varnothing$ and $\boldsymbol{F}_{\Rightarrow}=\varnothing$, there are no feasible mergers to implement with savings equal to M_n, in which case $n=n+1$ and the heuristic returns to *Step* 2, unless $n>|\boldsymbol{M}|$, which means that all mergers have been evaluated and the heuristic terminates.

Table 3
CARP and CLARPIF benchmark features.

Benchmark sets and instances	$\lvert A' \cup E' \rvert$ [a]	$\lvert A_r \cup E_r \rvert$ [b]	Subtrips		Routes	
			min # [c]	#Arcs/subtrip [d]	K_{LB} [e]	#Arcs/route [f]
Waste collection problem instances						
Case study of [2]	540	679	5	136	NA	NA
Case study of [16]	223	153	3	51	NA	NA
Act-IF-a	509	151	6	67	2	201
Act-IF-b	259	401	2	76	1	151
Act-IF-c	410	250	4	63	2	125
Cen-IF-a [46]	441	1012	17	60	7	145
Cen-IF-c [46]	486	2519	37	68	15	168
Cen-IF-b [46]	360	2755	39	71	16	172
Lpr–IF-a-05 [46]	250	806	18	45	4	76
Lpr–IF-b-05 [46]	75	801	18	45	6	84
Lpr–IF-c-05 [46]	38	803	23	35	9	89
Mean values over benchmark set						
Lpr–IF [46]	71	352	7	28	3	57
mval-IF	0	88	5	21	3	32
gdb-IF	0	29	6	5	6	5
gdb-IF-3L	0	29	6	5	2	14
bccm-IF [35]	0	63	5	15	6	10
bccm-IF-3L	0	63	5	15	2	31

NA: not enough information available to calculate (5) and (6).

[a] Number of arcs and edges not requiring service.
[b] Number of arcs and edges requiring service.
[c] Lower bound on minimum number of subtrips.
[d] (2)–(3).
[e] Weak lower bound on fleet size.
[f] (2)–(5).

If there are feasible mergers to implement, different tie break rules can then be used to choose pair (u,v) from $\boldsymbol{F}_{\rightarrow}$ or $\boldsymbol{F}_{\Rightarrow}$, such as the route cost discrepancy rule:

$$(u,v) = \arg\max \{ |z(u) - z(v)| : (u,v) \in \boldsymbol{F}_{\rightarrow} \cup \boldsymbol{F}_{\Rightarrow} \}, \tag{49}$$

or a pair can be chosen at random. If (u,v) is chosen from $\boldsymbol{M}_{\rightarrow}^{-1}(m)$, let $i = T^{-1}(u)$ and $j = T^{-1}(v)$. The merge is then implemented using Eqs. (50)–(52):

$$\boldsymbol{T}_{i,|\boldsymbol{T}_i|} = \boldsymbol{T}_{i,|\boldsymbol{T}_i|} \cup \boldsymbol{T}_{j,1}, \tag{50}$$

$$\boldsymbol{T}_i = \boldsymbol{T}_i \cup \boldsymbol{T}_j \backslash \boldsymbol{T}_{j,1}, \tag{51}$$

$$\boldsymbol{T}_j = \varnothing. \tag{52}$$

In Eq. (50), the last subtrip in $\boldsymbol{T}_i$ is combined with first subtrip of $\boldsymbol{T}_j$, and in Eq. (51), $\boldsymbol{T}_i$ is combined with $\boldsymbol{T}_j$, excluding its first subtrip whose arcs have already been incorporated in $\boldsymbol{T}_i$. Since all arcs in $\boldsymbol{T}_j$ are now serviced in $\boldsymbol{T}_i$, Eq. (52) removes $\boldsymbol{T}_j$ from solution $\boldsymbol{T}$. If (u,v) is chosen from $\boldsymbol{M}_{\Rightarrow}^{-1}(m)$, the merge is implemented as:

$$\boldsymbol{T}_i = \boldsymbol{T}_i \cup \boldsymbol{T}_j. \tag{53}$$

In Eq. (53), the route $\boldsymbol{T}_i$ is combined with $\boldsymbol{T}_j$, without merging any subtrips, and Eq. (52) is applied thereafter. After implementing a merge, arcs u and v are "locked-in", and $T^{-1}(u) = 0$ and $T^{-1}(v) = 0$. Additional steps are required depending on whether inverting symmetrical routes are allowed, as discussed next. After a merge is implemented the heuristic returns to Step 3.

Inverting symmetrical routes Improved-Merge and *Randomised-Merge* allow for inverted symmetrical routes to also be considered for a merge. As part of Step 1, $\boldsymbol{S}'$ is defined as the set that initially contains all arcs that are part of symmetrical routes, such that:

$$\boldsymbol{S}' = \{u \in \boldsymbol{R} : inv(u) \neq 0, pen(u) = 0\}. \tag{54}$$

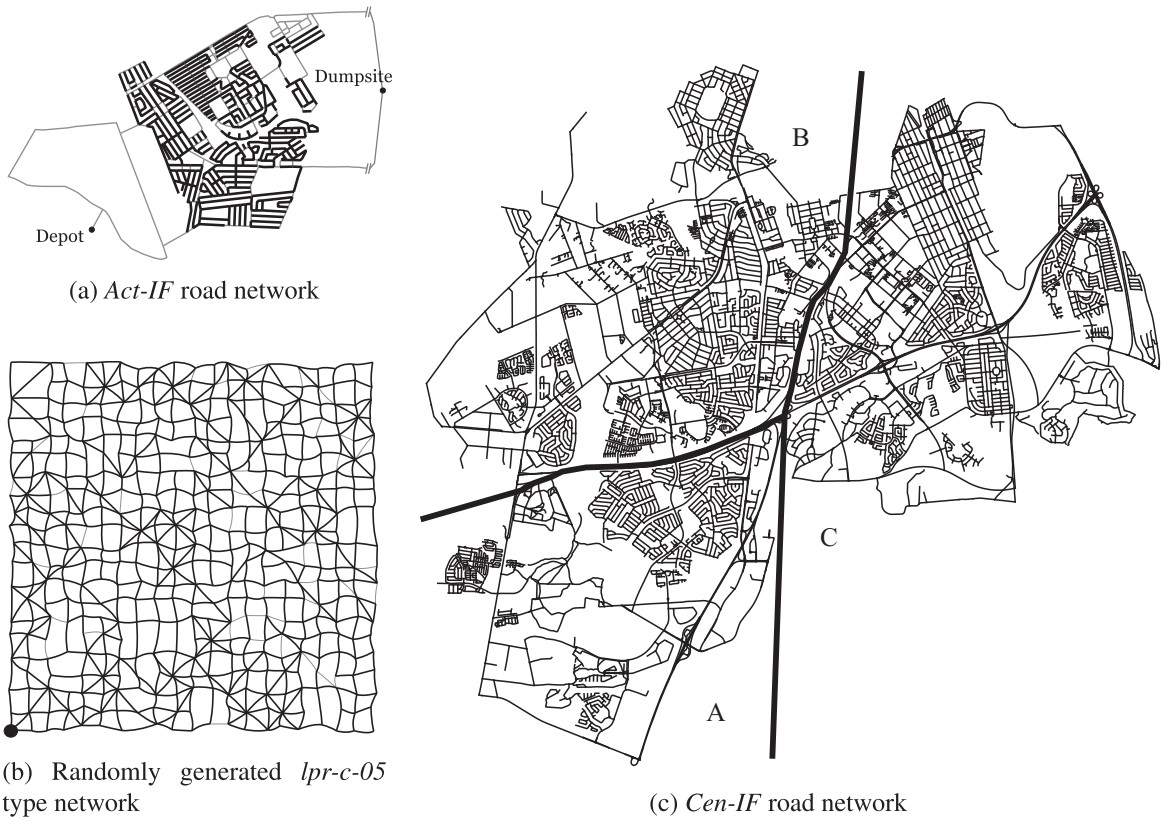
In Step 3, the route resulting from $u \rightarrow v$ or from $u \Rightarrow v$ is symmetrical if Eqs. (55) and (56), or Eqs. (55) and (58) hold, respectively:

$$u, v \in \boldsymbol{S}', \tag{55}$$

$$(inv(v), inv(u)) \in \boldsymbol{M}_{\rightarrow}^{-1}(m), \tag{56}$$

$$(inv(v), inv(u)) \in \boldsymbol{M}_{\Rightarrow}^{-1}(m). \tag{57}$$

After implementing a merge, should these conditions hold, $inv(v) \rightarrow inv(u)$ will also be implemented using Eqs. (50)–(52), or $inv(v) \Rightarrow inv(u)$ using Eqs. (51)–(53). If these conditions do not hold, and subject to $inv(u) \neq 0$ then:



(a) *Act-IF* road network

(b) Randomly generated *lpr-c-05* type network

(c) *Cen-IF* road network

Fig. 1. Comparison between real and random waste collection road networks.

$$\boldsymbol{T}_{T^{-1}(inv(u))} = \varnothing : inv(u) \neq 0, T^{-1}(inv(u)) \neq 0, \tag{58}$$

$$\boldsymbol{T}_{T^{-1}(inv(v))} = \varnothing : inv(v) \neq 0, T^{-1}(inv(v)) \neq 0, \tag{59}$$

$$\boldsymbol{S}' = \boldsymbol{S}' \setminus \{u, v\}, \tag{60}$$

and $T^{-1}(inv(u)) = 0$ and $T^{-1}(inv(v)) = 0$, should arcs u and v have apposing arcs. After all mergers have been evaluated and a final route is symmetrical, both its orientations will be in $\boldsymbol{T}$ and any one can be removed. Also, if $u \in \pm bS'$ could not be merged with any route, there will be two single arc routes in $\boldsymbol{T}$ for u and $inv(u)$, and the worst of the two routes should be removed. If mergers involving inverted symmetrical routes are not allowed, Eqs. (58)–(60) will be automatically applied, regardless of the outcomes of Eqs. (55)–(57). A detailed algorithm description of the efficient version of *Randomised-Merge*, with inverted symmetrical route mergers not allowed, is presented in Appendix A.3, Algorithms 3 and 4.

In Step 1, calculating all possible mergers takes $\mathcal{O}(2\tau^2)$, and sorting $\boldsymbol{M}$ takes $\mathcal{O}(\tau \log(\tau))$. In Steps 2 and 3, evaluating all possible mergers takes $\mathcal{O}(\tau^2)$. The worst case performance of *Improved-Merge* and *Randomised-Merge* for the MCARPTIF is thus $\mathcal{O}(\tau^2)$, same as *Path-Scanning*. An advantage of the efficient *Merge* versions is that $\boldsymbol{M}$ need only be calculated once. Thereafter Steps 2-4 will produce a new solution each time they are applied. Steps 2 and 3 are therefore linked with *nConstruct* (Appendix A.3, Algorithm 1), and the sorted $\boldsymbol{M}$ list is supplied as input data. For our computational tests we used the efficient *Merge* implementation that allowed for symmetrical routes to be inverted. We tested *Improved-Merge* using the cost discrepancy rule as well as *Randomised-Merge*.

3.4. Route-First-Cluster-Second heuristics

A *Route-First-Cluster-Second* heuristic was first proposed for the CARP by Ulusoy [42], and was extended to the CARP with IFs by Ghiani et al. [18], to the CLARPIF by Ghiani et al. [17], to the MCARP by Lacomme et al. [25], and to the MCARPTIF by Willemse and Joubert [46]. For the CARP, the heuristic generates a giant route that services all the required arcs and edges, and partitions the route via a splitting procedure into feasible smaller routes so as to adhere to the capacity limit. For the CLARPIF and MCARPTIF, the giant route is optimally partitioned into feasible routes so as to adhere to a maximum route time restriction, and also into subtrips with Intermediate Facility visits to adhere to the capacity limit. Finding and partitioning the least cost giant route, which is $\mathcal{NP}$-Hard since it involves the Rural Postman Problem, does not guarantee an optimal final solution. As a result, heuristics are used for the construction of the giant route. Lacomme et al. [25] and Belenguer et al. [4] relax vehicle capacity and route time limits of different *Path-Scanning* versions to generate giant routes. *Improved-Merge* and *Randomised-Merge* can also be used in this fashion. As proposed by Belenguer et al. [4], randomised giant route constructors can be used to enable the heuristic to partitioning a different solution with each run, from which the best is returned.

An efficient $\mathcal{O}(\tau^2)$ *Route-First-Cluster-Second* implementation, where $\tau = |\boldsymbol{R}|$, for the MCARP can be found in [25]. The implementation further deals with two objectives, minimising total cost as the primary objective and secondly the number of vehicles, even though it is straightforward to set vehicle minimisation as the primary objective. For the CLARPIF the splitting procedure used for the Cluster-Second phase, as described in [17], requires $\mathcal{O}(\tau^3)$, making it impractical for large scale instances. Willemse and Joubert [46] develop a more efficient

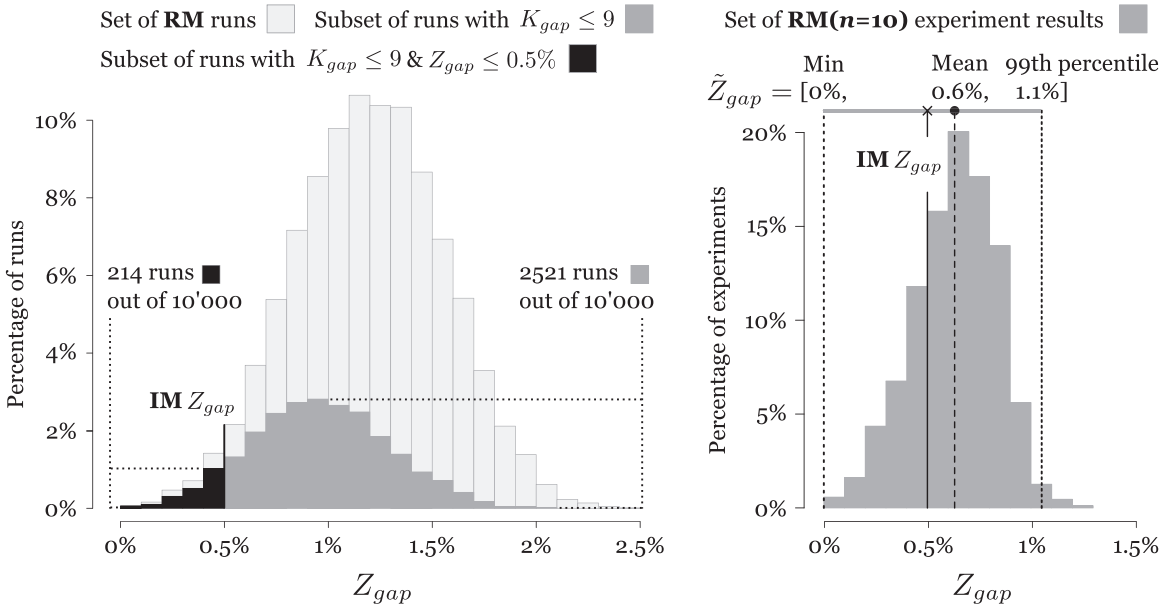


Fig. 2. Histogram of cost gaps for IM and RM on the *Cen-IF-b* problem instance.

optimal splitting procedure, also requiring $\mathcal{O}(\tau^3)$, for the MCARPTIF as well as a $\mathcal{O}(\tau^2)$ version that are on heuristic instead of optimal splitting procedures. Results show that the heuristic partitions are close to optimal partitions. The versions in [46] minimise the number of vehicles as the primary objective, and cost as secondary. For implementation details of the heuristics we refer the reader to [25,46].

For our computational tests we used *Path-Scanning* with six rules and *Path-Scanning-Random-Link* to generate initial giant routes. The *Path-Scanning* rules involving subtrip load and route time limits are redundant since the limits are relaxed by setting $Q \leftarrow \infty$ and $L \leftarrow \infty$. After generating a giant route it was partitioned using the optimal and efficient splitting procedures of Willemse and Joubert [46]. The implementations are referred to as *Efficient-Route-Cluster* and *Route-Cluster*. The heuristics were further adapted to either minimise the total cost or number of required vehicles, depending on the main and secondary problem objectives. We also tested the splitting procedures linked with a *Random-Merge* giant route generator, but found that it was always dominated by other heuristics and do not report on its performance.

3.5. Vehicle reduction heuristic

To enable constructive heuristics to better cope with the minimise fleet size objective, an efficient vehicle reduction heuristic was developed, termed *Reduce-Vehicles*. The heuristic takes a solution as an input, removes a route and using remove-insert operators, attempts to reinsert its arcs in the remaining routes in an effort to reduce K. The heuristic starts by sorting the solution's routes according to their cost. The lowest cost route is removed and its arcs sorted according to individual demand. Starting with the highest demand arc, the algorithm finds the least cost insert position of the arc into one of the remaining route's subtrips, subject to time and capacity limits. If a feasible insert position is found, the arc is added to the route and the next arc evaluated. If an insert position cannot be found, the algorithm reverts back to the starting solution. The process is again applied to the original solution, but starting with the second lowest cost route. If a route is successfully removed, the whole process is repeated on the new reduced solution. The process terminates when none of the routes could be successfully removed. Removal and insertion procedures used by *Reduce-Vehicles* are similar to a remove-insert neighbourhood search [4], which has a computational complexity of $\mathcal{O}(\tau^2)$, but since it stops reinserting a route's arcs when a feasible position cannot be found, and arcs are sorted in increasing order, its observed complexity is usually less. A detailed description of *Reduce-Vehicles* can be found in Appendix A.4, Algorithms 5 and 6.

4. Computational results

The aim of this paper was to develop and evaluate heuristics for the MCARPTIF in an effort to identify the best heuristic for implementation purposes, to measure the impact of minimising fleet size on heuristic performance, and to determine how performance is further influenced by the choice of benchmark sets. Computational tests were performed on the modified MCARPTIF versions of *Path-Scanning* (PS), *Path-Scanning-Random-Link* (PSRL), *Improved-Merge* (IM), *Randomised-Merge* (RM), as well as *Efficient-Route-Cluster* (ERC) linked with PS and PSRL giant route generators, and the optimal partitioning version of *Route-Cluster* (RC) linked with the same giant route generators. In this section, ERC and RC refer to the deterministic versions linked with PS, and the versions linked with PSRL are labelled RCRL and ERCRL. Each heuristic was evaluated in terms of minimising solution cost and minimising fleet size. For the latter we evaluated heuristics when linked with *Reduce-Vehicles*. In an effort to improve the practical value of our results, three new realistic waste collection benchmark instances are introduced and solved in this section. With only one CLARPIF and two MCARPTIF benchmark sets currently available we also introduce and solved a new MCARPTIF set and three new CLARPIF sets, developed by extending classical sets.

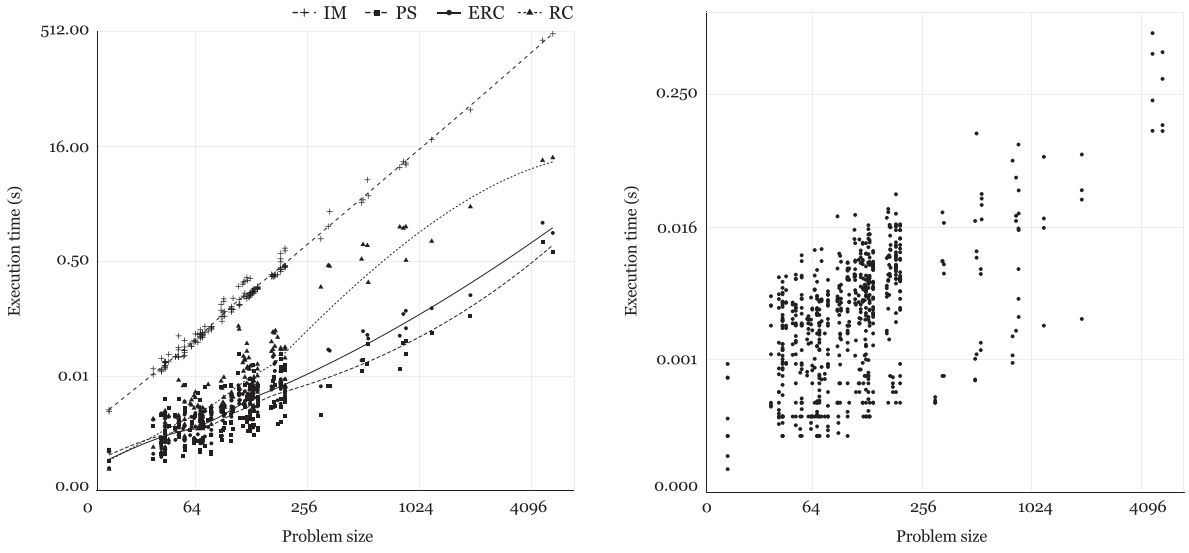
Constructive heuristics were programmed in Python version 2.7, with critical procedures optimised using Cython version 0.17.1. Computational experiments were run on a Dell PowerEdge R910 4U Rack Server with 128GB RAM with four Intel Xeon E7540 processors each having 6 cores, and 12 threads and with a 2 GHz base frequency. Experiments were run without using programmatic multi-threading or multiple processors.

4.1. Benchmark instances

For our computational tests we used four MCARPTIF and four CLARPIF benchmark sets. Four of these are new sets introduced in this paper. Sets taken from the literature include the *Cen-IF* and modified *lpr* MCARPTIF benchmarks of Willemse and Joubert [46], and the modified *gdb* and *bccm* CLARPIF sets proposed by Ghiani et al. [17]. To separate them from their original CARP and MCARP versions, we refer to the modified sets as *lpr-IF*, *gdb-IF* and *bccm-IF*. Two more realistic CLARPIF sets, *gdb-IF-3L* and *bccm-IF-3L*, are introduced in this paper and developed by increasing the route time limit of the *gdb-IF* and *bccm-IF* by a factor of three. We also developed a new *Act-IF* set, which consists of three MCARPTIF instances taken from the recycling project discussed in Section 2. Lastly, a new MCARPTIF set, *mval-IF-3L*, was developed by transforming the *mval* set of [4] by using the same cost limits as *bccm-IF-3L*. The original CARP sets are available from http://logistik.bwl.uni-mainz.de/benchmarks.php and MCARP sets from http://www.uv.es/belengue/mcarp/. The new CLARPIF and MCARPTIF benchmark sets are available as on-line supplementary files and from https://sites.google.com/site/wasteoptimisation/.

Key features of specific waste collection benchmark instances and all benchmarks sets are shown in Table 3. For comparison, the table shows features of the six *Act-IF* and *Cen-IF* instances, the three largest *lpr* instances, and the two case study problem instances of Bautista et al. [2] and Ghiani et al. [16]. The study area, shown in Fig. 1a, was used to develop three *Act-IF* instances. To generate the instances, the known total amount of weekly waste generated in the study area was evenly distributed among its households. Edge demands for the network were then calculated using the number of households on each road segment multiplied by the waste generated per household. Dead-heading travel speed was taken as 28 km/h, and service time per edge was calculated using a collection time of 1 s per kilogram of waste plus a travel speed between bins of 14 km/h. We assumed that it takes 5 min for a vehicle to offload its waste at an IF, and vehicle capacity was set to 10,000 kg. Waste collection crews operated from 08:00 AM to 17:00 PM, including breaks totalling 1 h, typically taken when the vehicle visits the IF. The route time limit was accordingly set to 8 hours. All instances have one IF situated outside the study area and away from the vehicle depot. The instances are relatively small, ranging in size from 151 to 400 required edges. Table 3 shows key features of the *Act-IF* set, which are consistent with the case study of Ghiani et al. [16].

The three *Cen-IF* instances of [46] are based on actual road networks of Centurion, which forms part of Pretoria, South Africa, and their key features are shown in Table 3. The benchmark set was created by dividing the Centurion area as shown in Fig. 1c into three sub-areas, and the instances range in size from 1012 to 2755 required arcs and edges, making them the largest benchmarks currently available. Two of the instances, *Cen-IF-a* and *Cen-IF-c*, have one IF that coincides with the vehicle depot, and are thus not strictly MCARPTIFs but Mixed Capacitated Arc Routing Problems under Time restrictions. *Cen-IF-b* has two IFs, one coinciding with the vehicle depot. The *Cen-IF* instances are much larger than the two case study problem instances in [2,16], which may detract from the realism of *Cen-IF*. In practice, routes may only have to be developed for smaller



(a) CPU time of constructive heuristics. (b) CPU time of *Reduce-Vehicles*.

Fig. 3. Problem instance size ($|\boldsymbol{R}|$) versus CPU time (in seconds), on log-*xy* axis, of deterministic heuristics to produce one solution and for *Reduce-Vehicles* to improve the solution.

Table 4
Average heuristic CPU times (in seconds) to generate one solution per benchmark set.

Set	IM	PS	ERC	RC	*Reduce-Vehicles*
Cen-IF	306.71	0.55	0.99	8.16	0.37
Act-IF	4.14	0.01	0.03	0.73	0.01
Lpr-IF	4.39	0.02	0.04	0.48	0.01
mval-IF-3L	0.19	0.01	0.01	0.02	<0.01
bccm-IF-3L	0.21	<0.01	0.01	0.01	<0.01
gdb-IF-3L	0.04	<0.01	<0.01	<0.01	<0.01
bccm-IF	0.24	0.01	0.01	0.01	0.01
gdb-IF	0.05	0.01	<0.01	0.01	<0.01
Global mean	39.50	0.10	0.187	1.35	0.10

Table 5
Average number of runs, α_e, required by randomised heuristics to break-even with their deterministic versions.

Set	min *Z* primary objective				min *K* primary objective			
	RM	PSRL	ERCRL	RCRL	RM	PSRL	ERCRL	RCRL
Cen-IF	14	48	–	–	5	48	502	–
Act-IF	13	24	115	24	69	24	115	24
Lpr-IF	1024	111	24	16	946	117	31	48
mval-IF-3L	8	58	10	11	7	53	11	13
bccm-IF-3L	7	114	12	8	4	111	15	47
gdb-IF-3L	41	35	7	6	23	178	6	6
bccm-IF	11	68	32	16	21	73	45	21
gdb-IF	368	78	51	27	908	52	60	49
Global mean	186	67	36	15	248	82	98	30

Table 6
Number of problem instances on which randomised heuristics failed to break-even over 10,000 runs.

Set	# Instances in set	min Z_{gap} Primary objective				min K_{gap} Primary objective			
		RM	PSRL	ERCRL	RCRL	RM	PSRL	ERCRL	RCRL
Cen-If	3	0	0	3	3	0	0	2	3
Act-If	3	2	0	0	0	2	0	0	0
Lpr-IF	15	5	1	0	0	2	1	1	0
mval-IF-3L	34	0	0	0	0	0	0	0	0
bccm-IF-3L	34	0	0	0	0	0	0	0	0
gdb-IF-3L	23	0	0	0	0	0	0	0	0
bccm-IF	34	0	0	0	0	0	0	0	1
gdb-IF	23	4	0	0	0	0	0	3	1
Total	169	11	1	3	3	4	1	6	5

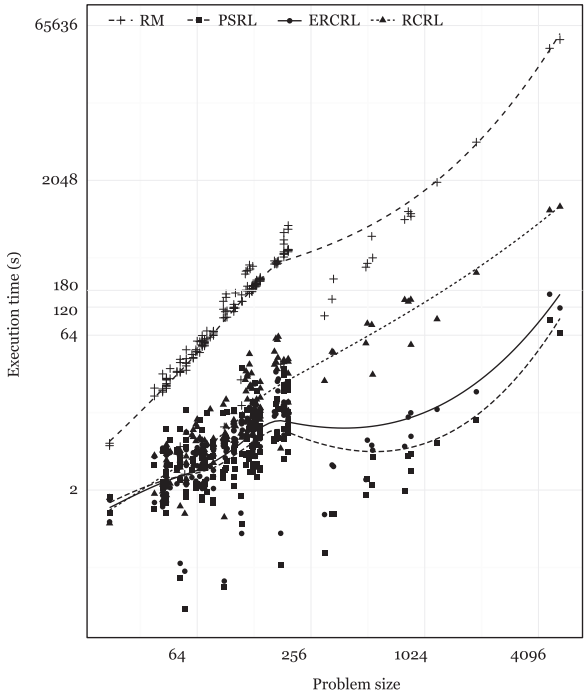


Fig. 4. Problem instance size ($|\boldsymbol{R}|$) versus execution time of randomised heuristic implementations.

sub-areas, or sectors, linked with collection days. This makes the set ideal for tests on MCARP sectoring type problems studied in [9,14,32].

The *Lpr-IF* set, used in [46], was generated using existing *lpr* instances by including IFs at vertices $\lfloor|\boldsymbol{V}|/2\rfloor$ and $2\lfloor|\boldsymbol{V}|/2\rfloor$, and setting the max route time limit to 28,800 s. Fig. 1b shows the resulting road network when using the approach of Belenguer et al. [4] to create a problem instance similar to *lpr-c-05*. The *Lpr-IF* and *lpr* sets are especially useful during early heuristic development, or when studying new CARP versions, since it contains a wide range of instances. As shown in Table 3, features of the larger instances are consistent with the case study problem instance of [2], and features on smaller instances are consistent with *Act-IF*.

The *gdb-IF* and *bccm-IF* sets, first proposed by Ghiani et al. [17], are modified versions of *gdb* [20] and *bccm* [5]. The *bccm-IF* instances are also solved by Polacek et al. [35]. The *bccm* CARP instances originally had separate dead-heading and service times per arc. Since the service times are fixed for any feasible CARP solution, Belenguer and Benavent [3] removed it from the instances and worked with only the dead-heading times. The reduced versions have been predominantly used in literature since. For the CLARPIF, service time plays a role due to the route time limit. Results

Table 7
Number of problem instances on which *Reduce-Vehicles* was able to reduce the fleet size.

Set	# Instances	IM	RM	PS	PSRL	ERC	ERCRL	RC	RCRL
Cen-IF	3	2	2	2	2	3	2	3	1
Act-IF	3	2	1	0	0	0	0	0	0
Lpr-IF	15	10	9	2	2	11	2	6	1
mval-IF-3L	34	29	21	13	11	22	17	19	15
bccm-IF-3L	34	30	17	3	3	16	7	11	4
gdb-IF-3L	23	20	1	3	2	15	5	8	1
bccm-IF	34	34	26	32	31	34	34	34	34
gdb-IF	23	19	9	14	9	22	21	22	20
Total	169	146	86	69	60	123	88	103	76
Fraction		0.86	0.51	0.41	0.36	0.73	0.52	0.61	0.45

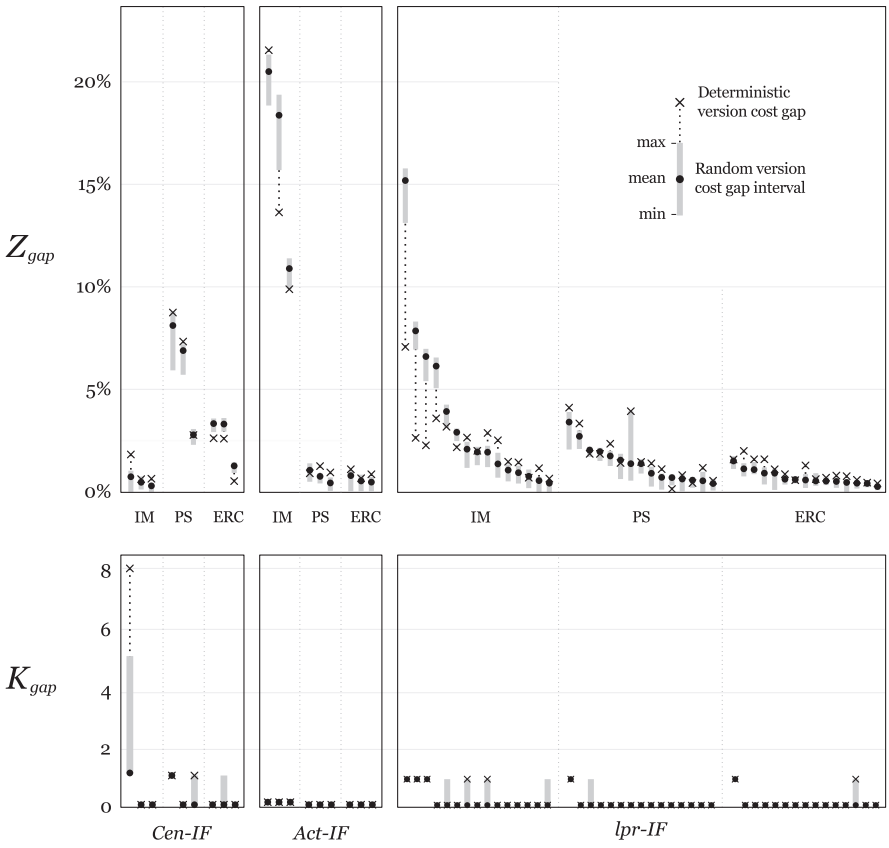


Fig. 5. Results on waste collection benchmark instances when min Z or min K is the primary objective.

reported in [17] indicate that the CLARPIF transformation was performed on the original *bccm* instances, with service times. Polacek et al. [35] performed tests on the transformed *bccm* instances without service times, and the authors incorrectly compare their metaheuristic results directly against those of Ghiani et al. [17]. We also found inconsistencies when transforming and solving *gdb* instances. Specifically, our constructive heuristics found solutions with costs better than the lower bound values reported in [17,19]. Since the original and transformed instances of Ghiani et al. [17] are no longer available, we treated the *gdb-IF* set generated using their approach as a new CLARPIF set. We also transformed and solved available *bccm* instances, thus rendering our results comparable to Polacek et al. [35]. As shown in Table 3, the *gdb-IF* and *bccm-IF* instances have a low number of arcs per subtrip ratio, and the same or lower number of arcs per route. Furthermore, the fleet size lower bound calculation used is weak, meaning the number of arcs per route is even lower. As a result, and confirmed during our initial testing, the capacity limit is rarely reached on these instances, meaning that most routes will not have inter-route IF visits. The *gdb-IF-3L* and *bccm-IF-3L* were developed specifically to overcome this issue, which is why the new *mval-IF-3L* set, with the same origins as *bccm-IF*, was developed using *bccm-IF-3L* time limits. Compared to waste collection sets, all these sets are small and undirected and all the edges require service. A key question is whether these problem instance characteristics influence heuristic performance. If so, the results on realistic sets have to be prioritised.

4.2. Evaluation criteria

Five criteria were used to evaluate heuristic performance. The first three were solution cost, the required fleet size of a solution, and computational time required by a heuristic to generate a solution. All randomised heuristics are allowed α runs, from which the best is returned. We refer to such a heuristic execution as RM(α), using *Random-Merge* as an example. Two additional criteria were used for

Table 8
Average Z_{gap} values (in %) on benchmark sets when min Z is the primary or secondary objective.

min Z	Set	IM	RM	PS	PSRL	ERC	ERCRL	RC	RCRL
1st Obj	*Cen-IF*	1.0	[0.0, 0.5, 0.7]	6.3	[4.7, 6.0, 6.4]	1.9	[2.3, 2.6, 2.9]	1.7	[2.1, 2.5, 2.7]
	Act-IF	15.0	[14.8, 16.6, 17.4]	0.9	[0.0, 0.6, 0.9]	1.0	[0.3, 0.7, 1.0]	0.8	[0.1, 0.5, 0.7]
	Lpr-IF	2.4	[2.8, 3.6, 3.9]	1.7	[0.8, 1.4, 1.8]	1.0	[0.4, 0.7, 0.9]	0.7	[0.1, 0.4, 0.6]
	Mean	6.1	[5.9, 6.9, 7.3]	3.0	[1.8, 2.7, 3.0]	1.3	[1.0, 1.3, 1.6]	1.1	[0.8, 1.1, 1.3]
	mval-IF-3L	18.0	[8.2, 12.4, 14.6]	13.6	[3.6, 6.2, 9.3]	14.7	[8.0, 8.3, 10.7]	13.0	[6.3, 6.4, 8.4]
	bccm-IF-3L	31.5	[17.1, 17.3, 20.1]	10.1	[1.6, 4.0, 6.4]	14.0	[5.9, 6.1, 8.9]	10.8	[3.9, 4.0, 5.5]
	gdb-IF-3L	11.7	[1.5, 3.0, 4.2]	7.8	[1.2, 2.6, 4.7]	11.3	[4.1, 4.5, 5.9]	8.8	[2.2, 2.5, 3.7]
	bccm-IF	18.1	[7.5, 9.9, 13.0]	25.4	[8.5, 12.3, 16.7]	68.6	[47.6, 48.0, 52.6]	68.5	[47.4, 47.7, 52.2]
	gdb-IF	11.0	[1.4, 2.2, 3.1]	11.2	[1.7, 3.4, 5.2]	34.9	[20.0, 20.2, 24.2]	34.5	[19.8, 20.0, 24.0]
	Mean	18.1	[7.1, 9.0, 11.0]	13.6	[3.3, 5.7, 8.5]	28.7	[17.1, 17.4, 20.5]	27.1	[15.9, 16.1, 18.8]
2nd Obj	*Cen-IF*	2.4	[1.1, 1.3, 2.3]	6.3	[4.7, 5.9, 6.4]	3.0	[3.0, 4.0, 4.5]	2.5	[2.5, 3.1, 3.6]
	Act-IF	10.3	[10.7, 15.7, 16.0]	0.9	[0.0, 0.6, 0.9]	1.0	[0.3, 0.7, 1.0]	0.8	[0.1, 0.5, 0.7]
	Lpr-IF	2.9	[2.6, 3.5, 3.9]	1.5	[0.8, 1.4, 1.8]	1.4	[0.7, 1.1, 1.3]	0.9	[0.2, 0.5, 0.8]
	Mean	5.2	[4.8, 6.8, 7.4]	2.9	[1.8, 2.6, 3.0]	1.8	[1.3, 1.9, 2.3]	1.4	[0.9, 1.4, 1.7]
	mval-IF-3L	16.3	[2.6, 9.1, 10.0]	12.6	[3.0, 5.7, 7.6]	15.2	[4.5, 8.3, 9.0]	13.5	[2.6, 6.4, 7.1]
	bccm-IF-3L	28.4	[8.4, 17.3, 18.0]	9.9	[1.6, 4.0, 6.4]	14.8	[4.1, 6.2, 7.7]	11.2	[1.3, 4.0, 4.6]
	gdb-IF-3L	13.3	[0.4, 2.3, 3.4]	7.8	[1.2, 2.5, 4.7]	12.6	[1.7, 4.4, 5.1]	9.6	[1.4, 2.3, 3.2]
	bccm-IF	15.5	[5.6, 8.6, 9.9]	21.4	[7.8, 11.4, 15.1]	69.0	[41.4, 48.0, 48.6]	68.9	[41.0, 47.7, 48.4]
	gdb-IF	10.4	[0.9, 1.9, 2.8]	9.3	[1.3, 2.9, 4.7]	35.5	[15.6, 20.2, 20.7]	34.6	[15.5, 20.0, 20.5]
	Mean	16.8	[3.6, 7.8, 8.8]	12.2	[3.0, 5.3, 7.7]	29.4	[13.5, 17.4, 18.2]	27.6	[12.4, 16.1, 16.8]
1st Obj	*Global*	13.6	[6.7, 8.0, 9.4]	9.6	[2.8, 4.4, 6.2]	18.4	[11.1, 11.3, 13.2]	17.4	[10.2, 10.4, 12.1]
2nd Obj	*Global*	12.4	[4.0, 7.4, 8.0]	8.7	[2.6, 4.1, 5.7]	19.1	[8.9, 11.4, 12.1]	17.8	[8.1, 10.4, 11.0]

$[Z_{gap}^{min}, \overline{Z}_{gap}, Z_{gap}^{99th}]$: results interval for random heuristics; *Mean*: average over set averages. *Global*: average over all set averages. 1st Obj: min Z was the primary objective and min K secondary; 2nd Obj: min K was the primary objective and min Z secondary.

randomised heuristic versions, namely the number of runs required for randomised heuristics to break-even with their deterministic versions, and lastly, statistical cost and fleet size intervals for heuristics when allowed a certain number of runs.

With no lower bounds currently available for the MCARPTIF, the least cost and least vehicle fleet size solutions found during all computational tests were used as a baseline. Solution quality was measured as the cost gap, Z_{gap}, for a solution $\boldsymbol{T}$, calculated as

$$Z_{gap} = \frac{\sum_{i=1}^{|\boldsymbol{T}|} Z(\boldsymbol{T}_i) - Z_{BF}}{Z_{BF}}, \tag{61}$$

where $Z(\boldsymbol{T}_i)$ is the cost of route i in solution $\boldsymbol{T}$, and Z_{BF} is the cost of the best solution found for a problem instance during all computational tests, except for *bccm-IF* instances, where Z_{BF} was taken from the best metaheuristic solutions reported in [35]. For vehicle fleet size evaluation the fleet size excess, K_{gap}, for a heuristic solution was calculated as:

$$K_{gap} = |\boldsymbol{T}| - K_{BF}, \tag{62}$$

where K_{BF} is the minimum fleet size found for a problem instance during all computational tests. The required number of vehicles of solutions are not reported by Polacek et al. [35] since the authors solve an unlimited fleet size version of the CLARPIF. We refer to the Z_{BF} and K_{BF} values of random heuristics as the random variables $\tilde{Z}_{gap}$ and $\tilde{K}_{gap}$. During our computational tests, heuristics were evaluated over two objectives. One was their ability to find minimum cost solutions and the other to find minimum fleet size solutions. We refer to these objectives as min Z and min K. For each heuristic tested, the best solution for each objective is returned by Algorithm 1 in Appendix A.1.

To compare the deterministic and randomised versions of a heuristic we first calculated the expected number of runs required by a randomised version to produce a better solution than its deterministic version. A randomised heuristic performs a sequence of runs and since each run is independent, the runs can be modelled as a series of Bernoulli trials. A success is then a run that produces a solution that is equal to or better than the deterministic version's best solution. Let Z_{gap}^{DT} and K_{gap}^{DT} be the gaps of the best deterministic solutions for min Z and min K. If p is the probability of a success, which can be empirically calculated, the expected number of runs, α_e, to find the first success is $\frac{1}{p}$. To calculate α_e, each randomised heuristic was executed with 10,000 runs, and Z_{gap} and K_{gap} were calculated for each run before and after applying *Reduce-Vehicles*. The number of runs out of 10,000 that produced solutions with $Z_{gap} \leq Z_{gap}^{DT}$ was then used to calculate p; the same approach was used to calculate break-even points on K_{gap}^{DT}.

As an illustration on the *Cen-IF-b* problem instance, Fig. 2 shows a histogram of Z_{gap} values for 10,000 runs of RM and the one run of IM. On the *Cen-IF-b* problem instance IM had gap values of $Z_{gap} = 0.5\%$ and $K_{gap} = 9$. Out of the 10,000 RM runs 251 had a solution with $Z_{gap} \leq 0.5\%$. By setting $p = 0.0251$, the expected number of runs required by RM to break-even with IM is thus $\alpha_e = 40$ when minimising Z is the only objective. Out of the 10,000 RM runs, 2521 (as shown in the figure) had a solution with $K_{gap} \leq 9$ giving RM a break-even point of $\alpha_e = 4$ when minimising K is the only objective. Lastly, 214 runs (also shown in the figure) gave a solution with $Z_{gap} \leq 0.5\%$ and $K_{gap} \leq 9$, giving RM a break-even point of $\alpha_e = 47$ over both objectives. Break-even points depend on the primary objectives. Referring to our example, if minimising Z is the primary objective, p is calculated using the number of runs where $Z_{gap} \leq Z_{gap}^{DT}$ or $Z_{gap} = Z_{gap}^{DT}$ and $K_{gap} < K_{gap}^{DT}$. Similarly if minimising K is the primary objective, p is calculated using the number of runs where $K_{gap} \leq K_{gap}^{DT}$ or $K_{gap} = K_{gap}^{DT}$ and $Z_{gap} < Z_{gap}^{DT}$.

A concise way of comparing RM and IM is through a statistical $\tilde{Z}_{gap}$ interval, which starts at the min Z_{gap} solution found over the 10,000 runs, and ends at the calculated 99th percentile value. Also included in the interval is the calculated mean of $\tilde{Z}_{gap}$; the interval for $\tilde{K}_{gap}$ is similarly defined. As an illustration, Fig. 2 further shows a histogram of observed Z_{gap} values for RM($\alpha = 10$) over 1000 experiments.[3] The observed $\tilde{Z}_{gap}$ interval for RM($\alpha = 10$) on *Cen-IF-b* is $[0\%, 0.6\%, 1.1\%]$. The mean of RM($\alpha = 10$) is higher than IM's Z_{gap} since 40 runs is required for RM to break

[3] The problem instance was solved 1000 times using RM($\alpha = 10$). For each experiment the Z_{gap} value of the best solution returned over the 10 runs was captured.

Table 9
Average K_{gap} values on benchmark sets when min K is the primary or the secondary objective.

min K	Set	IM	RM	PS	PSRL	ERC	ERCRL	RC	RCRL
1st Obj	*Cen-IF*	2.7	[0.3, 1.3, 1.7]	0.7	[0.3, 0.7, 0.7]	0.7	[0.0, 0.3, 0.3]	0.3	[0.0, 0.0, 0.0]
	Act-IF	0.0	[0.0, 0.0, 0.0]	0.0	[0.0, 0.0, 0.0]	0.0	[0.0, 0.0, 0.0]	0.0	[0.0, 0.0, 0.0]
	Lpr-IF	0.3	[0.2, 0.4, 0.5]	0.1	[0.1, 0.1, 0.1]	0.2	[0.1, 0.1, 0.1]	0.1	[0.1, 0.1, 0.1]
	Mean	1.0	[0.2, 0.6, 0.7]	0.3	[0.1, 0.3, 0.3]	0.3	[0.0, 0.1, 0.1]	0.1	[0.0, 0.0, 0.0]
	mval-IF-3L	0.7	[0.2, 0.2, 0.3]	0.3	[0.1, 0.1, 0.2]	0.6	[0.2, 0.2, 0.3]	0.6	[0.1, 0.1, 0.2]
	bccm-IF-3L	1.2	[0.2, 0.2, 0.3]	0.1	[0.0, 0.0, 0.1]	0.4	[0.0, 0.0, 0.1]	0.3	[0.0, 0.0, 0.0]
	gdb-IF-3L	0.7	[0.0, 0.0, 0.0]	0.1	[0.0, 0.0, 0.0]	0.5	[0.0, 0.0, 0.0]	0.3	[0.0, 0.0, 0.0]
	bccm-IF	1.1	[0.1, 0.5, 0.6]	1.2	[0.1, 0.4, 0.7]	6.7	[3.9, 3.9, 4.2]	6.7	[3.9, 3.9, 4.2]
	gdb-IF	0.9	[0.1, 0.2, 0.3]	0.6	[0.0, 0.1, 0.3]	3.4	[1.3, 1.3, 1.7]	3.4	[1.3, 1.3, 1.7]
	Mean	0.9	[0.1, 0.2, 0.3]	0.5	[0.0, 0.1, 0.3]	2.3	[1.1, 1.1, 1.3]	2.3	[1.1, 1.1, 1.2]
2nd Obj	*Cen-IF*	3.3	[3.7, 5.0, 4.0]	0.7	[0.3, 0.7, 0.7]	7.0	[8.3, 9.3, 9.3]	3.7	[4.0, 4.7, 5.3]
	Act-IF	0.7	[0.3, 0.3, 0.3]	0.0	[0.0, 0.0, 0.0]	0.0	[0.0, 0.0, 0.0]	0.0	[0.0, 0.0, 0.0]
	Lpr-IF	0.9	[0.9, 1.1, 1.1]	0.1	[0.1, 0.1, 0.1]	1.1	[0.9, 1.2, 1.3]	0.9	[0.7, 0.8, 0.9]
	Mean	1.6	[1.6, 2.1, 1.8]	0.3	[0.1, 0.3, 0.3]	2.7	[3.1, 3.5, 3.5]	1.5	[1.6, 1.8, 2.1]
	mval-IF-3L	1.3	[0.7, 0.9, 1.1]	0.4	[0.1, 0.2, 0.3]	1.1	[0.6, 0.6, 0.9]	0.7	[0.4, 0.4, 0.6]
	bccm-IF-3L	2.2	[0.6, 0.6, 0.9]	0.1	[0.0, 0.0, 0.1]	0.6	[0.2, 0.2, 0.5]	0.4	[0.1, 0.1, 0.2]
	gdb-IF-3L	1.9	[0.0, 0.3, 0.4]	0.1	[0.0, 0.0, 0.1]	0.8	[0.4, 0.3, 0.7]	0.5	[0.2, 0.1, 0.3]
	bccm-IF	1.9	[0.4, 0.7, 1.2]	1.6	[0.2, 0.5, 0.9]	6.9	[4.6, 4.6, 5.2]	6.9	[4.6, 4.6, 5.1]
	gdb-IF	1.2	[0.3, 0.3, 0.5]	0.8	[0.1, 0.2, 0.4]	3.6	[1.9, 1.9, 2.5]	3.5	[1.9, 1.9, 2.5]
	Mean	1.7	[0.4, 0.6, 0.8]	0.6	[0.1, 0.2, 0.4]	2.6	[1.5, 1.5, 2.0]	2.4	[1.4, 1.4, 1.7]
1st Obj	*Global*	1.0	[0.1, 0.3, 0.4]	0.4	[0.1, 0.1, 0.2]	1.6	[0.7, 0.7, 0.8]	1.5	[0.7, 0.7, 0.8]
2nd Obj	*Global*	1.7	[0.9, 0.9, 1.2]	0.5	[0.1, 0.2, 0.3]	2.6	[2.1, 2.1, 2.6]	2.1	[1.5, 1.6, 1.8]

$[K_{gap}^{min}, \overline{K}_{gap}, K_{gap}^{99th}]$: results interval for random heuristics; *Mean*: average over set averages. *Global*: average over all set averages. 1st Obj: min K was the primary objective and min Z secondary; 2nd Obj: min Z was the primary objective and min K secondary.

even with IM. The $\tilde{Z}_{gap}$ interval can be also be directly calculated using the 10'000 runs, eliminating the need to empirically calculate the interval for different α levels. The interval can be calculated using the probability mass function of a binomial distribution, given in the following equations:

$$P(s) = \binom{\alpha}{s} p^s (1-p)^{\alpha - s}, \tag{63}$$

$$P(s \geq 1) = 1 - P(s = 0). \tag{64}$$

Here s is the number of required successes, α is the number runs, and p the probability of a success. The probability of a success is the probability that the heuristic will return a solution in a single run with Z_{gap} or K_{gap} below a certain threshold. Referring to Fig. 2, the probability that RM will return a solution with $Z_{gap} \leq 0.5\%$ and $K_{gap} \leq 9$ is $\frac{214}{10000}$. Using Eqs. (63) and (64) we calculate that RM($\alpha = 10$) will find a better solution than IM on one or more of its 10 runs with a probability of 19%. By using this method in conjunction with RM run observations, the interval of RM can be calculated at specific α values. For all our computational tests the interval lower limit is taken as the min Z_{gap} solution value over all the runs, and the 99th percentile upper limit is calculated via Eqs. (63) and (64) as the point where $P(k \geq 1) \approx 99\%$. The interval mean, $\overline{Z}_{gap}$, is calculated as the point where $P(\tilde{Z}_{gap} \leq \overline{Z}_{gap}) \approx \frac{1}{\alpha}$. When min K is the primary objective, the same interval can be calculated for K_{gap}, before and after applying *Reduce-Vehicles*. We refer to the $\tilde{Z}_{gap}$ interval as $[Z_{gap}^{min}, \overline{Z}_{gap}, Z_{gap}^{99^{th}}]$ and to the $\tilde{K}_{gap}$ interval as $[K_{gap}^{min}, \overline{K}_{gap}, K_{gap}^{99^{th}}]$. For *Cen-IF-b*, using the 10,000 RM runs the interval of $\tilde{Z}_{gap}$ for RM($\alpha = 10$) was calculated as $[0\%, 0.7\%, 1.1\%]$, which gives a slightly higher $\overline{Z}_{gap}$ than the observed interval shown in Fig. 2. In limited comparisons between the observed and calculated mean we found the calculated mean to always be slightly higher, but by no more than 0.5%. For all randomised heuristics we captured the Z_{gap} and K_{gap} values over 10,000 runs and used the results to calculate break-even points and intervals for the min K and min Z objectives. Intervals were calculated for both the primary and secondary objectives.

4.3. Computational time and break even analysis

The CPU time required by heuristics to produce a single solution, which is the same for the deterministic and randomised versions, plus the CPU time taken by *Reduce-Vehicles* on a solution are shown in Fig. 3a, and averages of benchmark sets are shown in Table 4. The CPU time of *Reduce-Vehicles* per solution is constant for all the heuristics and is shown in Fig. 3b. The CPU time of IM includes its initialisation phase, which took between three and four times longer than its execution phase. CPU times of ERC and RC to produce a single solution also include the CPU time to construct an initial giant route. On the two largest *Cen-IF* instances, IM took 8 min to produce a single feasible solution, whereas PS took less than a second. ERC was also efficient, capable of solving large *Cen-IF* instances in 1.5 s. Results further show that *Reduce-Vehicles* is efficient, taking at most one second on large *Cen-IF* instances. The CPU times of RC were on average under 1 s on all sets except *Cen-IF* where RC was under 10 s. On small instances with $|\boldsymbol{R}| \leq 256$, all randomised heuristics can complete 100 runs in less than 10 s, including RM whose most expensive component is its initialisation phase. The high computational times of IM and RC are only a factor on realistically sized instances with $|\boldsymbol{R}| > 256$. The tested versions of IM, RM, RC and RCRL are thus too slow for near real-time decision support. Additional effort can be invested to improve their implementation code efficiency, depending on how well they perform in minimising Z and K compared to more efficient heuristics.

To compare the deterministic heuristics with their randomised versions, the number of runs, σ_e, required by the randomised versions to break even was calculated for the min Z_{gap} and min K_{gap} objectives. In case of the latter, *Reduce-Vehicles* was applied to both the deterministic and random versions. The number of runs per deterministic heuristic are one run for IM, eight runs for PS, and six runs for ERC and RC. Average break-even values for PSRL against PS, RM against IM, ERCRL against ERC and RCRL against RC per benchmark set are shown in Table 5. Averages were calculated by excluding problem instances for which randomised heuristics failed to find a better solution in 10,000 runs. The number of instances per set for which this was the case is shown in Table 6. On most instances the randomised heuristics are capable of outperforming their deterministic versions if allowed sufficient number of runs. Comparing α_e against the number of runs required by the deterministic versions, α_e was always equal or higher. When min Z was the primary objective, ERCRL and RCRL required the least number of runs to break-even and RM required the most. PSRL broke even on all but 1 of the 159 instances, and its break-even point was on average under 100 runs. When min K was the primary objective, all the heuristics required more runs to break-even, indicating that the deterministic rules result in solutions requiring fewer vehicles. On small problem instances, computation time is not a factor and randomised heuristics can be allowed high-run levels to outperform deterministic versions. The break-even points were also lower on these instances. This was not the case on more realistic instances. Randomised heuristics required more runs to break-even which, in turn, also require more computation time. Contrary to tests on small instances, the choice between randomised and deterministic versions is not clear for realistic instances and will depend on available computational time and the solution quality of heuristics. These factors are analysed next.

4.4. Performance of constructive heuristics

The aim of our computational tests was to identify the best performing heuristic, subject to computation time limits encountered in practical applications. The second aim was to determine if heuristic performance was different between the primary objectives of minimising Z or K, and if it differed between different benchmark sets, particularly between more realistic sets and those proposed in literature. Based on the computational times and break-even points of heuristics (Tables 4 and 5), the randomised heuristics were evaluated as follows. On small *gdb* and *bccm* related benchmark sets as well as *mval-IF-3L*, the Z_{gap} and K_{gap} primary and secondary objective intervals were calculated at $\alpha = 1000$, and on *Lpr-IF*, *Act-IF* and *Cen-IF* at $\alpha = 100$. Computational times per problem instance for these setups are shown in Fig. 4. With $\alpha = 1000$, the execution times of PSRL and ERCRL fell below 1 min on small instances, and with $\alpha = 100$, both heuristics have total computation times of less than 2 min on the largest *Cen-IF* instances, except for one problem instance where ERCRL required 3 min, which we assume to be acceptable for real-time decision support. The same criteria would disqualify IM and RM from implementation, and limit RCRL to less than 10 runs; but as mentioned, their implementation code efficiency can be improved. ERCRL and RCRL are compared in [46] who found RCRL to only marginally perform better. Accordingly, we only report on aggregated results of our tests on RCRL.

To measure the impact of *Reduce-Vehicles*, the fleet size before and after its application was measured on deterministic heuristic solutions. The $\tilde{K}_{gap}$ interval calculations were also completed without and with its application and the resulting intervals compared. Table 7 shows the number of problem instances per set on which *Reduce-Vehicles* decreased the number of required vehicles for deterministic solutions, and on which it decreased any of the three interval values on randomised heuristics. Despite its simplicity, the application of *Reduce-Vehicles* improved the performance of heuristics, and because of its computational efficiency (Fig. 3) it did so without significantly increasing the computation times. It can thus be applied to any heuristics when real-time decision support is needed and min K is an objective, or if the fleet sizes of solutions exceed the fleet limit. In the remainder of this paper, results reported on heuristic costs and fleet sizes are with the application of *Reduce-Vehicles* when min K is the primary objective, and without it when min Z is the primary objective.

Results for deterministic and randomised heuristics on the waste collection benchmark sets, *Lpr-IF*, *Act-IF* and *Cen-IF*, are shown in Fig 5 and includes Z_{gap} values when min Z was the primary objective, and K_{gap} values when min K was the primary objective. On these sets, the best performing heuristic differed per set and per primary objective. RM and IM performed the best on *Cen-IF* in minimising Z but they performed poorly in minimising K with IM exceeding the minimum fleet size solution by eight vehicles on *Cen-IF-b*. RM had a $\tilde{K}$ interval of [1.0, 1.0, 5.0] for the same problem instance. IM and RM also performed the worst for *lpr-IF* in minimising K. ERCRL performed the best for *Act-IF* and *lpr-IF* whereas ERC performed better for all three *Cen-IF* instances. Both versions performed the best for all three instances in minimising K, followed closely by PS and PSRL. Despite the application of *Reduce-Vehicles*, all heuristics failed to find min fleet size solutions for a few instances. ERCRL was the most consistent of the randomised heuristics, with small $\tilde{Z}$ intervals spanning less than 1%. RM had the largest intervals in excess of 5% for some of the problem instances. The intervals on $\tilde{K}$ were small for all heuristics, except RM and IM for *Cen-IF-b*. ERC and ERCRL were always within 5% of the minimum Z solution found, and PS and PSRL within 10% on *Cen-IF* and within 5% on *lpr-IF*. IM and RM were also within 10% on most instances, except one *Lpr-IF* problem instance and the *Act-IF* set where it performed very poorly compared to other heuristics.

Summary Z_{gap} results for the heuristics on all the benchmark sets are shown in Table 8. Full results are included in Appendix B. Results are shown when min Z is either the primary or the secondary objective, and for the five smaller benchmark sets the randomised heuristics were analysed at $\alpha = 1000$. Results for the smaller sets further highlight the impact of benchmark sets on heuristic performance. The increase in α resulted in the random heuristics dominating their deterministic versions, whereas the performance of the deterministic versions were much closer for the waste collection sets, and even better in a few cases. In addition to *Cen-IF*, RM performed the best for *bccm-IF* and *gdb-IF*, but it performed the worst for all other sets. Routes for *bccm-IF* and *gbd-IF* typically consist of a single subtrip, indicating that RM and IM performs better when this is the case, and possibly when the depot coincides with an IF, as is the case for *Cen-IF*. ERCRL and RCRL struggled with *bccm-IF* and *gbd-IF*, producing solutions with $\overline{Z}_{gap}$ values that were on average 48% and 20% from Z_{BF}, respectively. This implies that the expected costs, $\overline{Z}$, of the heuristics were on average 1.5 and 1.2 times more than Z_{BF}. These were the largest gaps observed during all the computational tests. On all the other sets except *Cen-IF*, ERCRL performed the best. As expected, RC and RCRL performed better than ERC and ERCRL, more so on smaller sets. The marginally better solutions of RC and RCRL for waste collection sets comes at a significant increase in computational time, as shown in Fig. 4. For time critical applications, the more efficient ERC and ERCRL heuristics can be used instead of RC and RCRL as their solutions will still be better than those of other heuristics. For all the benchmarks, PS and PSRL were the most consistent with the best global Z_{gap} mean, despite not being the best performing heuristic on any specific set. The $\tilde{Z}_{gap}$ intervals of PSRL were however the largest, whereas ERCRL and RCRL had the smallest intervals, even for benchmarks where they performed poorly. When min Z was the secondary objective, Z_{gap} values were higher in a few cases as expected; but for a number of heuristics, particularly on smaller sets, the values were lower. This can be attributed to *Reduce-Vehicles* which was able to

reduce total costs through its route removal and arc insertion procedures. The cost reduction was most prominent for deterministic solutions and for Z_{gap}^{min} values. Treating min Z as a secondary objective did not have a significant impact on the solution costs of heuristics.

Summary K_{gap} results for heuristics on all benchmark sets together with min K as primary and secondary objectives are shown in Table 9. Full results are included in Appendix B including computational times for the different setups. The intervals were calculated at $\alpha = 1000$ for randomised heuristics for small benchmark sets, and at $\alpha = 100$ for *Cen-IF*, *Act-IF* and *Lpr-IF*. Similar to Z_{gap} analysis, results were different between waste collection and small benchmark sets, and between heuristics. All heuristics were able to find min fleet solutions for *Act-IF*. On the other waste collection sets, RCRL performed the best followed by RC, ERCRL, PSRL, ERC and PS. IM and RM performed the worst. On small *bccm-IF* and *gdb-IF* benchmarks ERC and RC again struggled, exceeding the minimum fleet size on average by 3.5 and 7 vehicles. Both performed better than RM and IM on *mval-IF-3L* and *bccm-IF-3L*, and all the randomised heuristics performed similar for *gdb-IF-3L*. PS and PSRL performed the best on small benchmark sets and were again the most consistent over all sets with the best global Z_{gap} means. In all the cases the randomised heuristics performed better than their deterministic versions, and their intervals were also small. When min K was treated as a secondary instead of primary objective, the increase in K_{gap} values was in excess of one vehicle on ERC and ERCRL, and close to one vehicle for IM and RM. PS and PSRL results were similar when min K was the primary and secondary objective.

Compared to other heuristics, PS and PSRL were expected to do well in meeting fleet limits. During their construction process arcs are added to a route until it reaches its limit. As subtrips become full, low demand arcs can still be included in subtrips; these are added by PS regardless of how far they are from the current route position. The construct mechanisms of IM and RM result in them struggling with fleet limits. Routes progressively became longer as more mergers took place; however, the mergers were made only according to cost savings. Directly merging two routes results in better savings than merging them through an IF visit. IM and RM thus have a tendency to produce long routes that cannot be merged through IF visit without exceeding the route time limit, when one route can be easily split and incorporated into the other routes. *Reduce-Vehicles* is effective in reducing the fleet size in these situations. ERC, RC, ERCRL and RCRL take giant routes as their input, which are constructed without taking arc demand or service time into consideration. If a giant route consists of successive high demand arcs, the heuristics will be forced to partition the route between these arcs, and the first portion will not be close to its demand limit. The same also occurs in high service time arc-sequences. *Reduce-Vehicles* then attempts to reinsert small demand and service time arcs in these routes, which is why the heuristics benefit from its application.

Results show that the choice of benchmark sets influence heuristic performance, as does the treatment of K. If tests were limited to the *gdb-IF* and *bccm-IF* benchmark sets proposed in the literature, and the only objective is to minimise Z, RM would have performed the best, followed by PSRL. ERCRL and RCRL would have been discarded based on their very poor performance on the sets. However, as shown in our benchmark set analysis (Table 3) these problem instances have features that are significantly different from waste collection instances. The assumption of an unlimited fleet size also rarely holds in practice. RCRL and ERCRL were the best performing heuristics in terms of minimising K and Z, but the high computational time of RCRL limits its practical use. For computational tests to be of greater practical value, results on realistic instances should be prioritised. The limited tests performed in this paper indicate that ERCRL linked with *Reduce-Vehicles* is best suited for application purposes where near real-time decision support is required and the primary objective is to either minimise Z or K. The deterministic version, ERC, can be used when available computational time is very limited. If small instances are to be solved, PSRL is best suited for application purposes. Similarly, if instances are to be solved with unique characteristics, inconsistent with those of all benchmark sets solved in this paper, PSRL would also be best suited as it is seems to be more robust for different types of instances.

A limitation of the analysis, and by extension the evaluations, is that tests were performed on only six realistic waste collection instances, three of each from the *Act-IF* and *Cen-IF* sets. Results on *lpr-IF* may also be misleading since gap measurements from lower bounds or best solutions found are generally small for the benchmark set. Another limitation is that Z_{gap} and K_{gap} values were taken from the best solutions found during all our computational tests, but are still limited to the solutions produced by heuristics. The gaps will thus be higher when taken from lower bounds and optimal solution values. The Z_{gaps} were high for *bccm-IF* instances, which were calculated from the metaheuristic solutions reported in [35]. The heuristics were able to match metaheuristic solutions for three out of 19 instances solved in [35], but on average the best performing heuristic, RM, had cost gaps of 7.5% when allowed 10,000 runs. Optimal solution gaps on other sets may also be in excess of 7.5% if one were to use *bccm-IF* as a baseline. The set did prove more difficult for the heuristics, with the best performing heuristic differing per problem instance. The same did not occur on realistic waste collection sets. Whether the poor performance of constructive heuristics on *bccm-IF* can be generalised to realistic waste collection instances remains an open question. In terms of K_{gap} analysis, which is seldom presented in CARP studies, all heuristics had $K_{gap} > 0$ for some of the problem instances. Since K_{gap} values were taken from best solutions found, the gaps from the minimum fleet size will be equal or higher. If K was treated as being limited, instead of an objective, and the fleet limit was close to minimum, the heuristics would have failed to find feasible solutions on a number of instances, even with the application of *Reduce-Vehicles*.

5. Conclusion

In this paper, we evaluated constructive heuristics for the MCARPTIF. Constructive heuristics that can quickly generate feasible solutions for the MCARPTIF is an important area of research. In many practical applications such as the one that inspired this paper, the problem has to be solved in near real-time. When faced with a new problem such as the MCARPTIF, which accurately models waste collection, constructive heuristics are used as a first step to solve the problem. Thereafter, more advanced solution approaches can be investigated for which constructive heuristics provide input solutions. Results presented in this paper for constructive heuristics are still limited, and a natural extension for future work is to develop lower bounds and metaheuristics for the MCARPTIF. This paper has highlighted and addressed two main research gaps that can be prioritised for such a study, as well as for CARP and MCARP studies in general. These studies would be for the development of more realistic benchmark instances, and to focus on problems where the fleet size is limited or must be minimised.

In contrast to its original formulation, studies on CARP heuristics treat the vehicle fleet as unlimited. In practical applications, the fleet is generally limited and our first research contribution was the evaluation of MCARPTIF constructive heuristics on their ability to minimise the fleet size. To improve heuristic performance an efficient fleet size reduction procedure was developed and linked with the tested heuristics. This allowed heuristics to better deal with the objective, but despite its application, none of the heuristics could find minimum fleet solutions on all CLARPIF and MCARPTIF benchmark instances. More advanced procedures for minimising the number of required routes should yield better results, thus warranting future research on the topic. Secondly, results showed that problem instance characteristics influence heuristic performance, but the lack of realistic waste collection benchmark sets makes this difficult to investigate. In this paper, we have addressed this gap, admittedly to a limited extent, by the introduction of new realistic waste collection instances.

Results indicate that problem instance characteristics such as IF locations that are incident to vehicle depots and low route time limits in relation to vehicle capacity, play a critical role in heuristic performance. Execution times were also high for new problem instances, pointing to heuristic inefficiencies that would have gone unnoticed for smaller instances.

The method that we employed to evaluate randomised heuristics is also of value to future studies on CARPs. The heuristics were statistically analysed by modelling runs as a series of Bernoulli trials, and calculating total cost and vehicle fleet size intervals for specific run levels. An advantage of the analysis is that a single set of experiments can be used to calculate intervals for different run levels. The analyses were used to more accurately compare heuristic performance, and can be applied to any multi-start heuristic or metaheuristic where runs are independent.

An often cited advantage of CARP constructive heuristics is that they can be easily modified to extensions of the problem. The existing CARP and MCARP heuristics were indeed easy to modify for the MCARPTIF, but their performance on benchmark instances were inconsistent. In particular, the best performing MCARP heuristic, *Improved-Merge*, adapted to the MCARPTIF performed the best on two CLARPIF benchmarks proposed in the literature, yet struggled with waste collection benchmark sets. The weakest MCARP heuristic, *Path-Scanning-Random-Link*, was the most robust overall benchmarks sets. On waste collection applications, *Efficient-Route-Cluster-Random-Link* linked with *Reduce-Vehicles* performed the best, yet it produced solutions with total costs 1.2–1.5 times more than that of other heuristics on benchmarks currently proposed in the literature. Given the lack of realistic MCARPTIF benchmark sets, identifying the best performing heuristic for waste collection applications is difficult. Results indicate that of the tested heuristics, *Efficient-Route-Cluster-Random-Link* is best suited when waste collection routing problems have to be solved in real-time or near real-time.

Acknowledgements

This work is based on the research supported in part by the National Research Foundation of South Africa (Grant no. 87749). Lastly, we acknowledge the work of the anonymous referees for their valuable contribution to the improvement of the quality of the paper.

Appendix A. Detailed algorithm descriptions

A.1. Multiple run solution constructor

Algorithm 1. *nConstruct.*

Input: number of solutions to be generated, α, the *Solution-Constructor*, and fleet size limit, K_{UB}, if applicable.
Output: Min cost solution $\boldsymbol{T}^*_{min\ Z}$, min fleet size solution $\boldsymbol{T}^*_{min\ K}$, and min cost and limited fleet size solution $\boldsymbol{T}^*_{K_{UB}}$.

$Z^*_{min\ Z} = Z^*_{min\ K} = Z^*_{K_{UB}} = \infty$ // values for min cost solution //;
$K^*_{min\ Z} = K^*_{min\ K}$ // values for min fleet solution //;
$t = 1$;
while $t \le \alpha$ **do**
  $\boldsymbol{T} = Solution - Constructor$ // solution constructed using a constructive heuristic // ;
  $K = |\boldsymbol{T}|$ // number of vehicles used // ;
  $Z = \sum_{j=1}^{K} Z(\boldsymbol{T}_j)$ // solution cost // ;
  if $(Z < Z^*_{min\ Z})$ **or** $(Z = Z^*_{min\ Z}$ **and** $K < K^*_{min\ Z})$ **then** // new least cost incumbent //
    $Z^*_{min\ Z} = Z$;
    $K^*_{min\ Z} = K$;
    $\boldsymbol{T}^*_{min\ Z} = \boldsymbol{T}$;
  if $(K < K^*_{min\ K})$ **or** $(K = K^*_{min\ K}$ and $Z < Z^*_{min\ K})$ **then** // new least number of vehicles incumbent //
    $Z^*_{min\ K} = Z$;
    $K^*_{min\ K} = K$;
    $\boldsymbol{T}^*_{min\ K} = \boldsymbol{T}$
  if $\left(Z < Z^*_{K_{UB}}\right)$ **and** $(K \le K_{UB})$ **then** // new least cost incumbent meeting fleet size limit, if applicable //
    $Z^*_{K_{UB}} = Z$;
    $\boldsymbol{T}^*_{K_{UB}} = \boldsymbol{T}$;
  $t = t + 1$;
return $\{\boldsymbol{T}^*_{min\ Z}, \boldsymbol{T}^*_{min\ K}, \boldsymbol{T}^*_{K_{UB}}\}$

A.2. Path-Scanning

Algorithm 2. *Path-Scanning.*

Output MCARPTIF solution $\boldsymbol{T}$
$\boldsymbol{R}_s = \boldsymbol{R}$ // all required arcs are initially unserviced //;
$\boldsymbol{T} = [\![\sigma]\!]$ // solution initially consists of a single route with single subtrip //;
$u = \sigma, i = 1, j = 1$;
while $\boldsymbol{R}_s \neq \varnothing$ **do** // while there remains unserviced required arcs //

$\boldsymbol{P}^{(i)} = \{x \in \boldsymbol{R}_s : Z(\boldsymbol{T}_i) + D(u,x) + w(x) + \mu^*(x,\sigma) \leq L\}$ // feasible arc additions are first isolated // ;
$\boldsymbol{P}^{(i,j)} = \{x \in \boldsymbol{P}^{(i)} : load(\boldsymbol{T}_{i,j}) + q(x) \leq Q\}$;
$d^* = \begin{cases} \min\{D(u,x) : x \in \boldsymbol{P}^{(i,j)}\} \text{if } \boldsymbol{P}^{(i,j)} \neq \varnothing, \\ \infty \quad \text{otherwise} \end{cases}$;
$\boldsymbol{P}_{IF}^{(i)} = \{x \in \boldsymbol{R}_s : Z(\boldsymbol{T}_i) + \mu^*(u,x) + w(x) + \mu^*(x,\sigma) + \lambda \leq L\}$;
$d_{IF}^* = \begin{cases} \min\{\mu^*(u,x) : x \in \boldsymbol{P}_{IF}^{(i)}\} \text{ if } \boldsymbol{P}_{IF}^{(i)} \neq \varnothing, \\ \infty \quad \text{otherwise} \end{cases}$;
if $\boldsymbol{P}^{(i,j)} \neq \varnothing$ **and** $d^* \leq d_{IF}^*$ **then** // nearest arc is added to current subtrip //
  $\boldsymbol{S} = \{x \in \boldsymbol{P}^{(i,j)} : D(u,x) = d^*\}$;
  Select v randomly from $\boldsymbol{S}$ // can be replaced with other tie − break rules // ;
  $\boldsymbol{T}_{i,j} = \boldsymbol{T}_{i,j} \cup [v]$ // arc added to end of subtrip // ;
  $\boldsymbol{R}_s = \boldsymbol{R}_s \backslash \{v, inv(v)\}$ // arc is removed from set of arcs requiring service // ;
  $u = v$
else if $\boldsymbol{P}_{IF}^{(i)} \neq \varnothing$ **then** // new subtrip is added with nearest arc after IF visit //
  $\boldsymbol{S}_{IF} = \{x \in \boldsymbol{P}_{IF}^{(i)} : \mu^*(u,x) = d_{IF}^*\}$;
  Select v randomly from $\boldsymbol{S}_{IF}$ // can be replaced with other tie − break rules // ;
  $\boldsymbol{T}_{i,j} = \boldsymbol{T}_{i,j} \cup [\Phi_{u,v}^*]$ // IF visit added to end of subtrip // ;
  $\boldsymbol{T}_i = \boldsymbol{T}_i \cup \left[[\Phi_{u,v}^*, v]\right]$ // subtrip added to end of route // ;
  $\boldsymbol{R}_s = \boldsymbol{R}_s \backslash \{v, inv(v)\}$ // arc is removed from set of arcs requiring service // ;
  $j = j+1, u = v$;
else // vehicle returns to the depot and a new route is opened //
  $\boldsymbol{T}_{i,j} = \boldsymbol{T}_{i,j} \cup [\Phi_{u,\sigma}^*, \sigma]$ // IF and depot added to end of subtrip // ;
  $\boldsymbol{T} = \boldsymbol{T} \cup [[\![\sigma]\!]]$ // new route with one subtrip added to solution // ;
  $i = i+1, j = 1, u = \sigma$;
if $\boldsymbol{R}_s = \varnothing$ **then** // all arcs are serviced //
  $\boldsymbol{T}_{i,j} = \boldsymbol{T}_{i,j} \cup [\Phi_{u,\sigma}^*, \sigma]$ // IF and depot added to end of subtrip // ;
return $\{\boldsymbol{T}\}$

A.3. Randomised-Merge

Algorithm 3. *Initialise-Merge.*

Output Initial partial solution $\boldsymbol{T}$ (without dummy-arcs), arc to solution mapping T^{-1}, ordered merge savings list $\boldsymbol{M}$, and merge saving to arc pair mappings, $\boldsymbol{M}_{\rightarrow}$ and $\boldsymbol{M}_{\Rightarrow}$.

$i = 0$;
for $u \in \boldsymbol{R}$ **do**
  $i = i+1$;
  $\boldsymbol{T}_i = [[u]]$ // route is created for each required arc // ;
  $T^{-1}(u) = i$ // pointer function is updated that maps an arc to its route's index // ;
$\boldsymbol{M}_{\rightarrow} = \varnothing, \boldsymbol{M}_{\Rightarrow} = \varnothing, \boldsymbol{M}_{\rightarrow}^{-1} = \varnothing, \boldsymbol{M}_{\Rightarrow}^{-1} = \varnothing$;
for $u \in \boldsymbol{R}$ **do**
for $v \in \boldsymbol{R} \backslash \{u, inv(u)\}$ **do**
  $m' = \mu^*(u,\sigma) + D(\sigma,v) + \lambda - D(u,v) - p(u,v)$ // direct merge savings, including orientation penalty // ;
  if $m' \notin \boldsymbol{M}_{\rightarrow}$ **then**
    $\boldsymbol{M}_{\rightarrow} = \boldsymbol{M}_{\rightarrow} \cup \{m'\}$ // saving added to direct savings list // ;
    $\boldsymbol{M}_{\rightarrow}^{-1}(m') = \{(u,v)\}$ // pointer function is updated that maps savings to arcs to be directly merged // ;
  else // cost saving is equal to that of other mergers //
    $\boldsymbol{M}_{\rightarrow}^{-1}(m_{\rightarrow}) = \boldsymbol{M}_{\rightarrow}^{-1}(m_{\rightarrow}) \cup \{(u,v)\}$;
  $m'' = \mu^*(u,\sigma) + D(\sigma,v) - \mu^*(u,v) - p(u,v)$ // IF merge savings, including orientation penalty // ;
  if $m'' \notin \boldsymbol{M}_{\Rightarrow}$ **then**
    $\boldsymbol{M}_{\Rightarrow} = \boldsymbol{M}_{\Rightarrow} \cup \{m''\}$ // saving added to indirect savings list // ;
    $\boldsymbol{M}_{\Rightarrow}^{-1}(m'') = \{(u,v)\}$ // pointer function is updated that maps savings to arcs to be merged through IFs // ;
  else // cost saving is equal to that of other mergers //
    $\boldsymbol{M}_{\Rightarrow}^{-1}(m'') = \boldsymbol{M}_{\Rightarrow}^{-1}(m'') \cup \{(u,v)\}$;

21 $\boldsymbol{M} = \boldsymbol{M}_{\rightarrow} \cup \boldsymbol{M}_{\Rightarrow}$;
22 Sort $\boldsymbol{M}$ in nonincreasing order $\boldsymbol{M} = [M_1, \ldots, M_{|\boldsymbol{M}|}]$, such that $M_i > M_{i+1}\ \forall\ i \in \{1, \ldots, |\boldsymbol{M}|\}$;
23 return $\{\boldsymbol{T}, T^{-1}, \boldsymbol{M}, \boldsymbol{M}_{\rightarrow}^{-1}, \boldsymbol{M}_{\Rightarrow}^{-1}\}$

Algorithm 4. *Execute-Merge* (without symmetrical route inversion).

Input Initial partial solution $\boldsymbol{T}$ (without dummy-arcs), arc to solution mapping T^{-1}, ordered merge savings list $\boldsymbol{M}$, and merge saving to arc pair mappings, $\boldsymbol{M}_{\rightarrow}$ and $\boldsymbol{M}_{\Rightarrow}$.
Output Partial solution $\boldsymbol{T}$ (without dummy-arcs).
1 for $m \in M$ **do**// merge starts with the greatest merge saving //
  while $\boldsymbol{M}_{\rightarrow}^{-1}(m) \cup \boldsymbol{M}_{\Rightarrow}^{-1}(m) \neq \varnothing$**do** // there are potentially feasible mergers to implement with savings m //
    $\boldsymbol{M}'_{\rightarrow} = \boldsymbol{M}_{\rightarrow}^{-1}(m), \boldsymbol{M}'_{\Rightarrow} = \boldsymbol{M}_{\Rightarrow}^{-1}(m)$ // sets of all potential direct and IF mergers with equal savings m // ;
    for $(u, v) \in \boldsymbol{M}_{\rightarrow}^{-1}(m) \cup \boldsymbol{M}_{\Rightarrow}^{-1}(m)$ **do** // for each potential merger, eliminate infeasible ones //
2 $i = T^{-1}(u), j = T^{-1}(v)$ // find route indices of arc routes // ;
3 **if** $i = 0$ **or** $j = 0$ **then**
4 $merge = False$ // arc is no longer part of solutions, as its inverse is included // ;
5 **else if** $u \neq \boldsymbol{T}_{i,|\boldsymbol{T}_i|}$ **or** $v \neq \boldsymbol{T}_{j,1}$ **then**
6 $merge = False$ // arc u and v are not the last and first arcs in a route // ;
7 **else if** $i = j$ **then**
8 $merge = False$ // arcs belong to the same route //
9 **else if** $z(u) + z(v) - m - p(u, v) > L$ **then**
10 $merge = False$ // cost of merged route exceeds route cost limit // ;
11 **else if** $(u, v) \in \boldsymbol{M}_{\Rightarrow}^{-1}(m)$ **and** $load\ (\boldsymbol{T}_{i,|\boldsymbol{T}_i|}) + load(\boldsymbol{T}_{j,1}) > Q$ **then**
12 $merge = False$ // demand of merged subtrip exceeds vehicle capacity // ;
13 **else**
14 $merge = True$ // merge is feasible // ;
15 **if** $merge = False$ **and** $(u, v) \in \boldsymbol{M}_{\rightarrow}^{-1}(m)$
16 $\boldsymbol{M}_{\rightarrow}^{-1}(m) = \boldsymbol{M}_{\rightarrow}^{-1}(m) \backslash \{(u, v)\}$ // remove merge from set of potential direct mergers // ;
17 **else if** $merge = False$ **and** $(u, v) \in \boldsymbol{M}_{\Rightarrow}^{-1}(m)$ **then**
18 $\boldsymbol{M}_{\Rightarrow}^{-1}(m) = \boldsymbol{M}_{\Rightarrow}^{-1}(m) \backslash \{(u, v)\}$ // remove merge from set of potential IF mergers // ;
19 **if** $\boldsymbol{M}_{\rightarrow}^{-1}(m) \cup \boldsymbol{M}_{\Rightarrow}^{-1}(m) \neq \varnothing$ **then** // there are mergers to implement //
20 Randomly choose (u, v) from $\boldsymbol{M}'_{\rightarrow}(m) \cup \boldsymbol{M}'(m)_{\Rightarrow}$// can be replaced with other tie − break rules // ;
21 $i = T^{-1}(u), j = T^{-1}(v)$ // find route indices of arc routes // ;
22 **if** $(u, v) \in \boldsymbol{M}_{\rightarrow}^{-1}(m)$ **then** // merge belongs to direct merge set //
23 $\boldsymbol{T}_{i,|\boldsymbol{T}_i|} = \boldsymbol{T}_{i,|\boldsymbol{T}_i|} \cup \boldsymbol{T}_{j,1}$ // combine last and first subtrips // ;
24 $\boldsymbol{T}_i = \boldsymbol{T}_i \cup \boldsymbol{T}_j \backslash \boldsymbol{T}_{j,1}$ // combine rest of routes // ;
25 **else if** $(u, v) \in \boldsymbol{M}_{\Rightarrow}^{-1}(m)$ **then** // merge belongs to IF merge set //
26 $\boldsymbol{T}_i = \boldsymbol{T}_i \cup \boldsymbol{T}_j$ // combine two routes // ;
27 $\boldsymbol{T}_j = \varnothing$ // route j is removed from the solution since all its arcs are in route i // ;
28 $T^{-1}(T_{i,|\boldsymbol{T}_i|}, |\boldsymbol{T}_{i,|\boldsymbol{T}_i|}|) = i$ // route pointer function is updated so that the last arc in the new route also points to i // ;
29 **if**$inv(u) \neq 0$ **and** $T^{-1}(inv(u)) \neq 0$ **then**
30 $i' = T^{-1}(inv(u))$;
31 $\boldsymbol{T}_{i'} = \varnothing$ // apposing single − arc route is removed from solution // ;
32 $T^{-1}(inv(u)) = 0$ // apposing arc cannot be merged into other routes // ;
33 **if** $inv(v) \neq 0$ **and** $T^{-1}(inv(v)) \neq 0$ **then**
34 $j' = T^{-1}(inv(v))$;
35 $\boldsymbol{T}_{j'} = \varnothing$ // apposing single − arc route is removed from solution // ;
36 $T^{-1}(inv(v)) = 0$ // apposing arc cannot be merged into other routes // ;
37 **return** $\{\boldsymbol{T}\}$

A.4. Reduce-Vehicles

Algorithm 5. *Insert-Arc.*

Input Partial solution $\boldsymbol{T}$, and arc to insert u.
Output Partial solution $\boldsymbol{T}$ with u possibly inserted.
$i^* = j^* = n^* = 0$;
$m^* = \infty$;
for $i = 1$ **to** $|\boldsymbol{T}|$ **do**
  for $j = 1$ **to** $|\boldsymbol{T}_i|$ **do**
    if $j = |\boldsymbol{T}_i|$ **then**
      $J = |\boldsymbol{T}_{i,j}| - 1$ // IF and depot visits are fixed at end of route // ;
    else
      $J = |\boldsymbol{T}_{i,j}|$ // IF visit is fixed at end of route // ;
    for $k = 2$ **to** J **do**
      $m = D(T_{i,j,n}, u) + w(u) + D(u, T_{i,j,n+1}) - D(T_{i,j,n}, T_{i,j,n+1})$ // cost of arc insert // ;
      if $Z(\boldsymbol{T}_{i,j}) + q(u) \le Q$ **and** $\boldsymbol{Z}(T_i) + m \le L$ **then** // check that arc insert is feasible //
        if $m \le m^*$ **then** // new best insert position found //
          $m^* = m$;
          $i^* = i$;
          $j^* = j$;
          $n^* = n$;
if $i^* \neq 0$ **then** // insert arc into best feasible position //
  $\boldsymbol{T}_{i^*,j^*} = [T_{i^*,j^*,1}, \ldots, T_{i^*,j^*,n^*}] \cup [u] \cup [T_{i^*,j^*,n^*+1}, \ldots, T_{i^*,j^*,|\boldsymbol{T}_{i^*,j^*}|}]$;
else // no feasible insert position could be found //
  $\boldsymbol{T} = \varnothing$;
return $\{\boldsymbol{T}\}$

Algorithm 6. *Reduce-Vehicles.*

Input Solution $\boldsymbol{T}$ with K routes
Output Solution $\boldsymbol{T}^*$ with K possibly reduced
Let $\boldsymbol{T}^* = \boldsymbol{T}$, and sort $\boldsymbol{T}^*$ such that $Z(\boldsymbol{T}^*_i) \le Z(\boldsymbol{T}^*_{i+1})\ \forall i \in \{1, \ldots, K\}$;
$K = |\boldsymbol{T}^*|$;
$i = 1$ // start with least cost route. //;
while $i \le K$ **do**
  $\boldsymbol{S} = \{u \in \boldsymbol{T}^*_{i,1} \cup \boldsymbol{T}^*_{i,2}, \ldots, \boldsymbol{T}^*_{i,|\boldsymbol{T}^*_i|}\}$;
  $\boldsymbol{T} = \boldsymbol{T}^* \backslash \boldsymbol{T}^*_i$ // remove route from solution // ;
  Sort $\boldsymbol{S}$ in decreasing order according the tasks demands;
  $n = 1$ // start with highest demand arc. // ;
  while $n \le |\boldsymbol{S}|$ **do**
    $u = \boldsymbol{S}_n$;
    $\boldsymbol{T} =$ Insert$-$Arc$(\boldsymbol{T}, u)$ // insert arc in the best position using Algorithm 5 // ;
    if $\boldsymbol{T} = \varnothing$ **then**
      $n = |\boldsymbol{S}| + 1$ // no feasible insertion could be found // ;
    else
      $n = n + 1$ // arc was successfully inserted and next lowest demand arc will be evaluated // ;
  if $\boldsymbol{T} \neq \varnothing$ **then** // number of vehicles was successfully reduced //
    $\boldsymbol{T}^* = \boldsymbol{T}$;
    $K = |\boldsymbol{T}^*|$;
    $i = 1$ // process is repeated // ;
  else
    $i = i + 1$ // try next least cost route // ;
return $\{\boldsymbol{T}^*\}$

Appendix B. Computational results

Heuristics tested in this paper were *Improved-Merge* (IM), *Randomised-Merge* (RM), *Path-Scanning* (PS), *Path-Scanning-Random-Link* (PSRL), *Efficient-Route-Cluster* (ERC), *Efficient-Route-Cluster-Random-Link* (ERCRL), *Route-Cluster* (RC) and *Route-Cluster-Random-Link* (RCRL). Results for all heuristics for *Act-IF*, *Cen-IF* and *lpr-IF* problem instances are shown in Table B1. Results for deterministic heuristic versions are shown for the expected values, $\overline{Z}$ and $\overline{K}$, when allowed 100 runs. Results for all heuristics for *mval-IF-3L*, *bccm-IF-3L*, *gdb-IF-3L*, *bccm-IF-3L* and *gdb-IF* problem instances are shown in Tables B2–B6. Results for deterministic heuristic versions are shown for the expected values, $\overline{Z}$ and $\overline{K}$, when allowed 1000 runs. Computational times of the heuristics for all the sets are shown in Tables B7–B12.

Table B1
Heuristic results for *Act-IF*, *Cen-IF* and *lpr-IF* sets.

1st Obj	Instance	Best found		IM		RM(1000)		PS		PSRL(1000)		ERC		ERCRL(1000)		RC		RCRL(1000)	
		Z	K	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$
min Z	Cen-IF-a	231,431	9	235,650	9	233,135	10	251,726	9	250,258	9	237,433	13	239,082	16	23,7433	11	23,9005	12
	Cen-IF-b	594,142	21	597,825	30	596,913	31	610,737	22	610,799	22	597,253	26	601,669	28	59,4142	27	599,486	26
	Cen-IF-c	512,669	19	515,952	20	514,160	23	550,367	20	548,107	20	526,067	31	529,738	33	52,6067	22	529,574	25
	Mean Z_{gap} and K_{gap}			1.0%	3.3	0.5%	5	6.3%	0.7	5.9%	0.7	1.9%	7	2.6%	9.3	1.7%	3.7	2.5%	4.7
	Act-IF-a	22,351	1	25,398	1	26,457	1	22,499	1	22,470	1	22,553	1	22,587	1	22,553	1	22,533	1
	Act-IF-b	72,244	3	87,808	4	87,054	4	73,040	3	72,816	3	73,151	3	72,789	3	72,821	3	72,598	3
	Act-IF-c	49,972	2	54,911	3	55,415	2	50,400	2	50,209	2	50,445	2	50,189	2	50,296	2	50,125	2
	Mean Z_{gap} and K_{gap}			15.0%	0.7	16.6%	0.3	0.9%	0	0.6%	0	1.0%	0	0.8%	0	0.8%	0	0.5%	0
	Lpr-IF-a-01	13,594	1	13,954	1	13,877	1	13,783	1	13,717	1	13,810	2	13,718	1	13713	1	13,639	1
	Lpr-IF-a-02	28,399	1	29,113	2	28,786	1	29,516	2	28,789	1	28,966	2	28,718	1	28,720	1	28,551	1
	Lpr-IF-a-03	77,670	3	78,565	4	78,095	4	80,261	3	79,775	3	78,896	4	78,840	5	78,834	4	78,712	4
	Lpr-IF-a-04	132,824	5	142,214	7	152,995	7	133,006	5	133,746	5	133,840	6	133,432	6	13,3733	6	133,166	6
	Lpr-IF-a-05	211,345	8	216,907	10	227,939	10	212,252	8	212,548	8	212,295	9	212,213	9	21,1840	9	211784	8
	Lpr-IF-b-01	14,835	1	15,261	1	15,122	1	15,009	1	14,915	1	14,927	1	14,914	1	14,927	1	14,875	1
	Lpr-IF-b-02	28,773	1	29,196	2	29,078	2	29,955	2	29,752	2	29,230	2	29,085	2	29,230	2	29,031	2
	Lpr-IF-b-03	79,664	3	80,215	3	80,268	4	80,786	3	80,897	3	80,690	5	80,124	5	80,624	4	79,998	4
	Lpr-IF-b-04	131,493	5	136,204	6	139,558	6	134,580	5	133,794	5	132,940	6	132,689	6	132,387	6	132,267	6
	Lpr-IF-b-05	219,670	8	224,653	10	234,169	10	222,874	8	222,684	8	221,413	9	220,783	10	220,782	9	220,305	9
	Lpr-IF-c-01	18,735	1	19,004	1	18,910	1	18,837	1	18,811	1	18,845	1	18,814	1	18,807	1	18,771	1
	Lpr-IF-c-02	36,605	2	37,338	2	37,317	2	36,903	2	36,834	2	36,858	2	36,797	2	36,737	2	36,700	2
	Lpr-IF-c-03	113,648	4	117,256	5	118,103	6	114,908	4	114,450	4	114,293	5	114,330	5	114,127	5	114,067	5
	Lpr-IF-c-04	172,944	7	174,065	8	173,674	8	176,151	7	176,470	7	174,438	9	174,010	10	174,255	9	173,783	9
	Lpr-IF-c-05	271,558	10	277,458	11	279,447	13	276,587	10	276,887	10	272,676	13	272,257	14	272,271	13	271,981	13
	Mean Z_{gap} and K_{gap}			2.4%	0.9	3.6%	1.1	1.7%	0.1	1.4%	0.1	1.0%	1.1	0.7%	1.2	0.8%	0.9	0.4%	0.8
min K	Cen-IF-a	231,431	9	235,650	9	233,476	9	251,726	9	250,258	9	242,929	9	242,680	9	238,868	9	240,410	9
	Cen-IF-b	594,142	21	605,708	29	609,014	25	610,737	22	610,799	22	599,029	22	605,166	22	596,721	22	601,452	21
	Cen-IF-c	512,669	19	529,887	19	515,652	19	550,367	20	547,883	20	528,713	20	539,099	19	532,141	19	534,676	19
	Mean Z_{gap} and K_{gap}			2.4%	2.7	1.3%	1.3	6.3%	0.7	5.9%	0.7	3.0%	0.7	4.0%	0.3	2.5%	0.3	3.1%	0
	Act-IF-a	22,351	1	25,398	1	26,457	1	22,499	1	22,470	1	22,553	1	22,587	1	22,553	1	22,533	1
	Act-IF-b	72,244	3	79,179	3	85,858	3	73,040	3	72,816	3	73,151	3	72,789	3	72,821	3	72,598	3
	Act-IF-c	49,972	2	53,771	2	54,916	2	50,400	2	50,209	2	50,445	2	50,189	2	50,296	2	50,125	2
	Mean Z_{gap} and $K_{g.ap}$			10.3%	0	15.7%	0	0.9%	0	0.6%	0	1.0%	0	0.8%	0	0.8%	0	0.5%	0
	Lpr-IF-a-01	13,594	1	13,954	1	13,877	1	13,783	1	13,717	1	13,816	1	13,718	1	13,713	1	13,639	1
	Lpr-IF-a-02	28,399	1	28,678	1	28,726	1	28,776	1	28,789	1	28,966	2	28,692	1	28,720	1	28551	1
	Lpr-IF-a-03	77,670	3	797,75	3	78,521	3	80,261	3	79,775	3	79,576	3	79,382	3	79,103	3	78,958	3
	Lpr-IF-a-04	132,824	5	140,752	6	150,247	6	133,006	5	133,746	5	134,452	5	133,838	5	133,838	5	133,234	5
	Lpr-IF-a-05	211,345	8	216,390	9	226,016	9	212,252	8	212,548	8	213,366	8	212,801	8	212,300	8	211,784	8
	Lpr-IF-b-01	14,835	1	15,261	1	151,22	1	15,009	1	14,915	1	14,927	1	14,914	1	14,927	1	14875	1
	Lpr-IF-b-02	28,773	1	29,196	2	29,078	2	29,955	2	29,752	2	29,230	2	29,085	2	29,230	2	29,031	2
	Lpr-IF-b-03	79,664	3	80,215	3	80,992	3	80,786	3	80,897	3	81,338	3	80,818	3	80,872	3	80,131	3
	Lpr-IF-b-04	131,493	5	140,297	5	141,052	5	134,580	5	133,794	5	133,763	5	133,226	5	132,840	5	132,363	5
	Lpr-IF-b-05	219,670	8	230,112	9	231,697	9	222,874	8	222,684	8	222,582	8	222,835	8	220,837	8	220,728	8
	Lpr-IF-c-01	18,735	1	19,004	1	18,910	1	18,837	1	18,811	1	18,845	1	18,814	1	18,807	1	18,771	1
	Lpr-IF-c-02	36,605	2	37,338	2	37,317	2	36,903	2	36,834	2	36,858	2	36,797	2	36,737	2	36,700	2
	Lpr-IF-c-03	113,648	4	117,256	5	117,345	5	114,763	4	114,450	4	114,293	5	114,846	4	114,715	4	114,067	4
	Lpr-IF-c-04	172,944	7	177,241	7	174,195	7	176,151	7	176,470	7	175,302	7	174,765	7	174,590	7	174,034	7
	Lpr-IF-c-05	271,558	10	280,380	10	281,448	11	276,587	10	276,887	10	276,318	10	275,637	10	274,146	10	273,463	10
	Mean Z_{gap} and K_{gap}			2.9%	0.3	3.5%	0.4	1.5%	0.1	1.4%	0.1	1.4%	0.2	1.1%	0.1	0.9%	0.1	0.5%	0.1

Table B2
Heuristic results for *mval-IF-3L* set.

1st Obj	Instance	Best found		IM		RM(1000)		PS		PSRL(1000)		ERC		ERCRL(1000)		RC		RCRL(1000)	
		Z	K	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$
min Z	mval-IF-3L-1A	250	2	312	3	294	3	281	3	250	2	276	3	268	3	276	3	268	3
	mval-IF-3L-1B	309	3	360	4	316	3	349	3	336	3	362	4	336	3	356	3	317	3
	mval-IF-3L-1C	356	3	452	6	387	4	448	4	398	4	426	5	397	4	410	5	377	4
	mval-IF-3L-2A	413	2	545	3	525	3	511	3	442	2	523	3	494	3	523	3	493	3
	mval-IF-3L-2B	429	2	488	3	477	3	517	3	442	2	490	3	483	3	490	3	481	3
	mval-IF-3L-2C	506	3	691	5	582	3	551	3	524	3	619	3	555	3	586	3	528	3
	mval-IF-3L-3A	136	2	166	3	158	3	148	2	139	2	149	2	138	2	149	2	138	2
	mval-IF-3L-3B	149	2	173	3	173	3	179	3	151	2	174	3	154	2	172	3	154	2
	mval-IF-3L-3C	123	2	188	4	147	2	149	2	131	2	150	2	140	2	146	2	131	2
	mval-IF-3L-4A	645	3	749	4	717	4	766	4	698	3	751	4	722	4	741	4	714	4
	mval-IF-3L-4B	757	4	829	4	804	4	825	4	791	4	791	4	777	4	791	4	759	4
	mval-IF-3L-4C	769	4	864	6	867	5	834	4	802	4	856	5	798	4	846	4	788	4
	mval-IF-3L-4D	782	4	886	5	820	4	902	4	832	4	906	5	839	4	865	4	790	4
	mval-IF-3L-5A	790	4	991	6	990	5	813	4	806	4	853	4	819	4	853	4	819	4
	mval-IF-3L-5B	743	4	935	5	963	5	799	4	762	4	815	4	776	4	815	4	773	4
	mval-IF-3L-5C	824	4	976	6	1054	6	885	4	841	4	952	5	937	5	950	5	935	5
	mval-IF-3L-5D	821	4	1020	6	1004	6	890	4	845	4	920	5	898	5	902	5	878	4
	mval-IF-3L-6A	347	3	384	3	357	3	410	3	387	3	397	3	378	3	397	3	369	3
	mval-IF-3L-6B	334	3	389	4	345	3	432	3	393	3	394	4	358	4	376	3	358	3
	mval-IF-3L-6C	419	3	483	4	427	4	549	4	503	4	549	6	482	6	525	4	473	4
	mval-IF-3L-7A	390	4	423	4	394	4	454	4	427	4	438	5	421	4	438	5	421	4
	mval-IF-3L-7B	447	4	518	5	471	5	543	5	512	5	536	7	494	5	536	6	488	5
	mval-IF-3L-7C	494	4	546	5	505	5	614	5	573	5	571	8	536	7	560	6	526	6
	mval-IF-3L-8A	668	4	714	4	687	4	694	4	686	4	698	4	675	4	698	4	675	4
	mval-IF-3L-8B	591	3	692	4	630	4	607	3	610	3	669	4	644	4	663	4	632	3
	mval-IF-3L-8C	660	4	744	6	670	5	732	4	696	4	769	6	735	4	734	5	697	4
	mval-IF-3L-9A	543	4	693	7	654	6	620	5	591	5	655	6	603	5	655	6	603	5
	mval-IF-3L-9B	527	4	639	6	637	6	597	5	574	5	600	5	588	5	600	5	587	5
	mval-IF-3L-9C	528	4	605	5	633	6	585	5	547	4	598	5	571	5	598	5	570	5
	mval-IF-3L-9D	649	5	711	6	696	6	715	6	681	5	738	6	719	7	714	6	686	6
	mval-IF-3L-10A	833	5	896	6	959	7	871	5	850	5	926	6	868	5	926	6	868	5
	mval-IF-3L-10B	820	5	951	7	935	6	855	5	829	5	907	6	849	5	907	6	849	5
	mval-IF-3L-10C	782	5	796	5	871	6	843	5	806	5	832	5	801	5	825	5	794	5
	mval-IF-3L-10D	795	5	892	7	881	6	841	5	809	5	884	7	846	6	863	6	813	5
	Mean Z_{gap} and K_{gap}			18.0%	1.3	12.4%	0.9	13.6%	0.4	6.2%	0.2	14.7%	1.1	8.3%	0.6	13.0%	0.7	6.4%	0.4
min K	mval-IF-3L-1A	250	2	312	3	277	3	281	3	250	2	276	3	268	3	276	3	268	3
	mval-IF-3L-1B	309	3	344	3	315	3	349	3	332	3	366	3	334	3	356	3	317	3
	mval-IF-3L-1C	356	3	434	5	380	4	448	4	398	4	435	4	397	4	413	4	377	3
	mval-IF-3L-2A	413	2	545	3	501	2	433	2	430	2	523	3	494	2	523	3	493	2
	mval-IF-3L-2B	429	2	488	3	475	2	517	3	436	2	490	3	483	3	490	3	481	2
	mval-IF-3L-2C	506	3	632	4	543	3	551	3	524	3	619	3	555	3	586	3	528	3
	mval-IF-3L-3A	136	2	161	2	151	2	144	2	139	2	149	2	138	2	149	2	138	2
	mval-IF-3L-3B	149	2	173	3	171	2	179	3	151	2	174	3	154	2	172	3	154	2
	mval-IF-3L-3C	123	2	205	3	134	2	149	2	131	2	150	2	140	2	146	2	131	2
	mval-IF-3L-4A	645	3	697	3	707	3	718	3	684	3	751	4	722	3	741	4	714	3
	mval-IF-3L-4B	757	4	829	4	780	4	825	4	791	4	791	4	777	4	791	4	759	4
	mval-IF-3L-4C	769	4	884	5	822	4	834	4	802	4	868	4	798	4	846	4	788	4
	mval-IF-3L-4D	782	4	922	4	806	4	898	4	832	4	906	5	839	4	865	4	790	4
	mval-IF-3L-5A	790	4	896	5	948	4	813	4	806	4	853	4	819	4	853	4	819	4
	mval-IF-3L-5B	743	4	825	4	925	4	799	4	762	4	815	4	776	4	815	4	773	4
	mval-IF-3L-5C	824	4	996	5	1017	5	875	4	838	4	952	5	937	4	950	5	935	4
	mval-IF-3L-5D	821	4	987	5	952	5	890	4	845	4	920	5	898	4	902	5	878	4
	mval-IF-3L-6A	347	3	384	3	357	3	410	3	387	3	397	3	378	3	397	3	369	3
	mval-IF-3L-6B	334	3	368	3	345	3	432	3	391	3	400	3	358	3	376	3	358	3
	mval-IF-3L-6C	419	3	483	4	431	3	549	4	503	4	553	4	488	4	525	4	473	4
	mval-IF-3L-7A	390	4	423	4	394	4	454	4	427	4	440	4	421	4	440	4	421	4
	mval-IF-3L-7B	447	4	518	5	458	4	543	5	503	4	557	5	494	4	545	5	488	4
	mval-IF-3L-7C	494	4	546	5	505	5	610	5	573	5	575	6	536	5	619	5	532	5
	mval-IF-3L-8A	668	4	714	4	679	4	694	4	686	4	698	4	675	4	698	4	675	4
	mval-IF-3L-8B	591	3	692	4	618	3	607	3	606	3	669	4	644	3	663	4	632	3
	mval-IF-3L-8C	660	4	741	5	670	4	732	4	696	4	785	4	735	4	742	4	697	4
	mval-IF-3L-9A	543	4	660	5	634	5	620	5	591	5	656	5	603	4	656	5	603	4
	mval-IF-3L-9B	527	4	597	5	622	5	597	5	543	4	600	5	588	5	600	5	587	5
	mval-IF-3L-9C	528	4	605	5	609	5	585	5	539	4	598	5	571	5	598	5	570	4
	mval-IF-3L-9D	649	5	711	6	674	5	715	6	677	5	738	6	717	6	714	6	686	5
	mval-IF-3L-10A	833	5	893	5	913	5	871	5	850	5	926	6	868	5	926	6	868	5
	mval-IF-3L-10B	820	5	902	6	907	5	855	5	829	5	907	6	849	5	907	6	849	5
	mval-IF-3L-10C	782	5	796	5	838	5	843	5	806	5	832	5	801	5	825	5	794	5
	mval-IF-3L-10D	795	5	967	6	851	5	841	5	809	5	889	6	846	5	863	6	813	5
	Mean Z_{gap} and K_{gap}			16.3%	0.7	9.1%	0.2	12.6%	0.3	5.7%	0.1	15.2%	0.6	8.3%	0.2	13.5%	0.6	6.4%	0.1

Table B3
Heuristic results for *bccm-IF-3L* set.

1st Obj	Instance	Best found		IM		RM(1000)		PS		PSRL(1000)		ERC		ERCRL(1000)		RC		RCRL(1000)	
		Z	K	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$
min Z	bccm-IF-3L-1A	189	2	258	3	230	2	201	2	190	2	191	2	190	2	191	2	191	2
	bccm-IF-3L-1B	189	2	259	4	225	2	208	2	196	2	207	2	196	2	199	2	190	2
	bccm-IF-3L-1C	237	2	310	7	251	3	260	3	244	2	285	4	255	3	271	3	248	3
	bccm-IF-3L-2A	264	2	345	2	299	2	293	2	278	2	297	2	282	2	288	2	282	2
	bccm-IF-3L-2B	274	2	361	3	281	2	304	2	286	2	304	2	290	2	288	2	280	2
	bccm-IF-3L-2C	357	2	498	5	375	2	400	2	363	2	430	3	398	2	412	2	379	2
	bccm-IF-3L-3A	92	2	111	2	106	2	97	2	93	2	102	2	93	2	102	2	93	2
	bccm-IF-3L-3B	93	2	113	2	106	2	105	2	94	2	104	2	94	2	102	2	93	2
	bccm-IF-3L-3C	103	2	156	6	110	2	118	2	107	2	124	2	109	2	118	2	106	2
	bccm-IF-3L-4A	418	2	547	4	519	3	474	2	439	2	486	3	438	2	472	2	435	2
	bccm-IF-3L-4B	419	2	553	5	510	3	475	2	445	2	492	3	454	2	492	3	446	2
	bccm-IF-3L-4C	426	2	565	6	516	3	458	2	451	2	526	3	455	2	492	3	440	2
	bccm-IF-3L-4D	505	3	615	7	552	3	577	3	533	3	586	4	536	3	542	3	507	3
	bccm-IF-3L-5A	494	3	744	4	667	4	541	3	509	3	560	3	520	3	542	3	517	3
	bccm-IF-3L-5B	496	3	746	5	656	4	548	3	509	3	558	3	527	3	548	3	517	3
	bccm-IF-3L-5C	510	3	762	6	668	4	560	3	521	3	596	3	530	3	568	3	531	3
	bccm-IF-3L-5D	568	3	790	7	700	4	584	3	579	3	648	3	620	3	638	3	591	3
	bccm-IF-3L-6A	233	2	287	3	245	2	260	2	247	2	266	3	246	2	258	2	244	2
	bccm-IF-3L-6B	249	2	285	2	260	2	286	2	272	2	291	3	263	2	283	3	258	2
	bccm-IF-3L-6C	359	3	401	4	359	3	419	3	397	3	421	4	399	4	411	4	390	4
	bccm-IF-3L-7A	295	3	363	4	318	3	348	3	329	3	332	3	313	3	326	3	310	3
	bccm-IF-3L-7B	309	3	371	4	319	3	355	3	333	3	326	3	322	3	326	3	315	3
	bccm-IF-3L-7C	378	4	438	5	379	4	424	4	412	4	439	6	406	6	428	5	395	4
	bccm-IF-3L-8A	411	2	563	4	514	3	472	3	422	2	460	3	418	2	460	3	419	2
	bccm-IF-3L-8B	410	2	563	5	523	3	445	3	422	2	476	3	452	3	460	3	438	3
	bccm-IF-3L-8C	495	3	614	8	539	3	557	3	516	3	560	4	522	3	529	3	514	3
	bccm-IF-3L-9A	366	3	513	5	481	4	387	3	375	3	419	4	390	3	402	3	390	3
	bccm-IF-3L-9B	378	3	513	5	481	4	397	3	381	3	426	4	406	3	424	4	394	3
	bccm-IF-3L-9C	380	3	527	7	481	5	400	3	390	3	446	4	419	4	430	4	413	3
	bccm-IF-3L-9D	460	4	571	7	524	5	492	4	466	4	518	5	490	4	504	5	478	4
	bccm-IF-3L-10A	476	3	648	5	627	5	511	3	490	3	535	4	499	3	522	3	499	3
	bccm-IF-3L-10B	487	3	648	5	639	4	505	3	497	3	538	4	519	3	538	4	497	3
	bccm-IF-3L-10C	490	3	656	5	631	4	527	3	503	3	565	4	536	3	556	4	519	3
	bccm-IF-3L-10D	586	4	689	7	661	4	606	4	591	4	641	4	616	4	635	4	605	4
	Mean Z_{gap} and K_{gap}			31.6%	2.2	17.3%	0.6	10.1%	0.1	4.0%	0.0	14.0%	0.6	6.1%	0.2	10.8%	0.4	4%	0.1
min K	bccm-IF-3L-1A	189	2	220	2	230	2	201	2	190	2	191	2	190	2	191	2	191	2
	bccm-IF-3L-1B	189	2	264	3	225	2	208	2	196	2	207	2	196	2	199	2	190	2
	bccm-IF-3L-1C	237	2	309	6	251	2	260	3	244	2	298	3	255	3	271	3	247	2
	bccm-IF-3L-2A	264	2	345	2	299	2	293	2	278	2	297	2	282	2	288	2	282	2
	bccm-IF-3L-2B	274	2	372	2	281	2	304	2	286	2	304	2	290	2	288	2	280	2
	bccm-IF-3L-2C	357	2	452	4	375	2	400	2	363	2	430	3	398	2	412	2	379	2
	bccm-IF-3L-3A	92	2	111	2	106	2	97	2	93	2	102	2	93	2	102	2	93	2
	bccm-IF-3L-3B	93	2	113	2	106	2	105	2	94	2	104	2	94	2	102	2	93	2
	bccm-IF-3L-3C	103	2	131	5	110	2	118	2	107	2	124	2	109	2	118	2	106	2
	bccm-IF-3L-4A	418	2	529	3	519	2	474	2	439	2	486	3	438	2	472	2	435	2
	bccm-IF-3L-4B	419	2	542	4	510	2	475	2	445	2	492	3	454	2	492	3	446	2
	bccm-IF-3L-4C	426	2	676	4	516	2	458	2	451	2	526	3	455	2	492	3	440	2
	bccm-IF-3L-4D	505	3	614	6	552	3	577	3	533	3	590	3	536	3	542	3	507	3
	bccm-IF-3L-5A	494	3	744	4	667	3	541	3	509	3	560	3	520	3	542	3	517	3
	bccm-IF-3L-5B	496	3	691	4	656	3	548	3	509	3	558	3	527	3	548	3	517	3
	bccm-IF-3L-5C	510	3	722	4	668	3	560	3	521	3	596	3	530	3	568	3	531	3
	bccm-IF-3L-5D	568	3	818	5	700	3	584	3	579	3	648	3	620	3	638	3	591	3
	bccm-IF-3L-6A	233	2	299	2	245	2	260	2	247	2	272	2	246	2	258	2	244	2
	bccm-IF-3L-6B	249	2	285	2	260	2	286	2	272	2	299	2	263	2	287	2	258	2
	bccm-IF-3L-6C	359	3	437	3	359	3	419	3	397	3	423	3	399	3	417	3	388	3
	bccm-IF-3L-7A	295	3	337	3	318	3	348	3	329	3	332	3	313	3	326	3	310	3
	bccm-IF-3L-7B	309	3	371	4	319	3	355	3	333	3	326	3	322	3	326	3	315	3
	bccm-IF-3L-7C	378	4	490	4	379	4	424	4	412	4	484	4	410	4	454	4	395	4
	bccm-IF-3L-8A	411	2	540	3	514	3	472	3	419	2	460	3	418	2	460	3	419	2
	bccm-IF-3L-8B	410	2	529	4	523	3	421	2	421	2	476	3	452	2	460	3	438	2
	bccm-IF-3L-8C	495	3	624	7	539	3	557	3	516	3	580	3	522	3	529	3	514	3
	bccm-IF-3L-9A	366	3	439	4	481	4	387	3	375	3	419	4	390	3	402	3	390	3
	bccm-IF-3L-9B	378	3	461	4	481	4	397	3	381	3	426	4	406	3	424	4	394	3
	bccm-IF-3L-9C	380	3	485	5	481	4	399	3	390	3	446	4	419	3	430	4	413	3
	bccm-IF-3L-9D	460	4	565	6	524	4	492	4	466	4	518	5	490	4	520	4	478	4
	bccm-IF-3L-10A	476	3	567	4	627	4	511	3	490	3	535	4	499	3	522	3	499	3
	bccm-IF-3L-10B	487	3	628	4	639	4	505	3	497	3	538	4	519	3	538	4	497	3
	bccm-IF-3L-10C	490	3	653	4	631	4	527	3	503	3	565	4	536	3	556	4	519	3
	bccm-IF-3L-10D	586	4	678	5	661	4	606	4	591	4	641	4	613	4	635	4	605	4
	Mean Z_{gap} and K_{gap}			28.4%	1.2	17.3%	0.2	9.9%	0.1	4.0%	0.0	14.8%	0.4	6.2%	0.0	11.2%	0.3	4.0%	0.0

Table B4
Heuristic results for *gdb-IF-3L* set.

1st Obj	Instance	Best found		IM		RM(1000)		PS		PSRL(1000)		ERC		ERCRL(1000)		RC		RCRL(1000)	
		Z	K	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$
min Z	gdb-IF-3L-1	308	2	316	3	308	2	337	2	310	2	338	4	318	2	337	3	308	2
	gdb-IF-3L-2	337	2	369	5	345	2	362	2	341	2	361	3	347	2	353	3	338	2
	gdb-IF-3L-3	275	2	301	4	276	2	281	2	283	2	297	3	284	2	289	2	276	2
	gdb-IF-3L-4	344	2	427	4	383	2	377	2	349	2	424	2	369	2	396	2	369	2
	gdb-IF-3L-5	448	2	520	4	468	2	490	2	458	2	528	3	495	2	516	3	466	2
	gdb-IF-3L-6	293	2	341	4	301	2	313	2	293	2	313	2	303	2	313	2	294	2
	gdb-IF-3L-7	342	2	349	4	343	2	349	2	343	2	387	3	351	2	387	3	342	2
	gdb-IF-3L-8	419	3	449	5	422	4	471	4	448	4	470	4	458	4	466	4	446	4
	gdb-IF-3L-9	343	3	374	6	361	3	388	3	365	3	402	4	372	3	381	4	352	3
	gdb-IF-3L-10	312	2	374	3	323	2	355	2	317	2	336	2	322	2	336	2	316	2
	gdb-IF-3L-11	447	3	532	4	477	4	495	3	465	3	509	4	481	3	505	3	469	3
	gdb-IF-3L-12	624	2	826	3	710	3	662	2	641	2	777	3	682	3	727	3	659	2
	gdb-IF-3L-13	594	2	634	3	600	2	640	2	609	2	640	3	604	2	623	2	602	2
	gdb-IF-3L-14	110	2	134	6	112	3	132	3	114	2	132	3	120	2	125	3	112	2
	gdb-IF-3L-15	58	1	62	3	58	1	62	2	60	1	62	2	58	1	60	1	58	1
	gdb-IF-3L-16	131	4	139	5	131	4	141	4	133	4	139	4	133	4	139	4	133	4
	gdb-IF-3L-17	93	3	97	5	93	3	99	3	95	3	97	3	95	3	97	3	95	3
	gdb-IF-3L-18	178	3	210	6	194	4	186	3	180	3	197	3	188	3	190	3	188	3
	gdb-IF-3L-19	83	1	91	1	83	1	91	1	91	1	93	1	83	1	93	1	83	1
	gdb-IF-3L-20	147	2	162	3	150	2	147	2	147	2	165	3	155	2	160	3	155	2
	gdb-IF-3L-21	164	4	171	6	166	4	170	4	166	4	174	5	166	5	172	5	166	4
	gdb-IF-3L-22	215	5	229	8	219	6	226	5	218	5	227	6	220	6	225	6	220	6
	gdb-IF-3L-23	243	6	258	9	245	7	255	6	250	6	259	8	251	8	255	7	249	7
	Mean Z_{gap} and K_{gap}			11.7%	1.9	3.0%	0.3	7.8%	0.1	2.6%	0.0	11.3%	0.8	4.5%	0.3	8.8%	0.5	2.5%	0.1
min K	gdb-IF-3L-1	308	2	366	2	308	2	337	2	310	2	358	2	318	2	338	2	308	2
	gdb-IF-3L-2	337	2	371	4	345	2	362	2	341	2	361	3	347	2	365	2	338	2
	gdb-IF-3L-3	275	2	336	3	276	2	281	2	283	2	345	2	284	2	289	2	276	2
	gdb-IF-3L-4	344	2	422	3	383	2	377	2	349	2	424	2	369	2	396	2	369	2
	gdb-IF-3L-5	448	2	511	3	464	2	490	2	458	2	528	3	495	2	516	3	466	2
	gdb-IF-3L-6	293	2	375	3	301	2	313	2	293	2	313	2	303	2	313	2	294	2
	gdb-IF-3L-7	342	2	386	3	343	2	349	2	343	2	387	3	351	2	387	3	342	2
	gdb-IF-3L-8	419	3	446	4	422	3	471	4	450	3	470	4	458	3	466	4	446	3
	gdb-IF-3L-9	343	3	386	3	358	3	388	3	365	3	402	4	372	3	402	3	352	3
	gdb-IF-3L-10	312	2	366	2	312	2	355	2	317	2	336	2	322	2	336	2	316	2
	gdb-IF-3L-11	447	3	495	3	465	3	495	3	465	3	519	3	481	3	505	3	469	3
	gdb-IF-3L-12	624	2	769	3	681	2	662	2	641	2	777	3	682	2	727	3	659	2
	gdb-IF-3L-13	594	2	653	2	600	2	640	2	609	2	644	2	604	2	623	2	602	2
	gdb-IF-3L-14	110	2	129	4	112	2	132	3	112	2	132	3	120	2	125	3	112	2
	gdb-IF-3L-15	58	1	64	2	58	1	62	2	60	1	62	2	58	1	60	1	58	1
	gdb-IF-3L-16	131	4	141	4	131	4	141	4	133	4	139	4	133	4	139	4	133	4
	gdb-IF-3L-17	93	3	103	4	93	3	99	3	95	3	97	3	95	3	97	3	95	3
	gdb-IF-3L-18	178	3	196	4	188	3	186	3	180	3	197	3	188	3	190	3	188	3
	gdb-IF-3L-19	83	1	91	1	83	1	91	1	91	1	93	1	83	1	93	1	83	1
	gdb-IF-3L-20	147	2	160	2	148	2	147	2	147	2	165	3	151	2	165	2	151	2
	gdb-IF-3L-21	164	4	176	4	166	4	170	4	166	4	178	4	166	4	174	4	166	4
	gdb-IF-3L-22	215	5	228	7	219	5	226	5	218	5	227	6	219	5	229	5	219	5
	gdb-IF-3L-23	243	6	267	7	247	6	255	6	250	6	261	7	253	6	255	7	249	6
	Mean Z_{gap} and K_{gap}			13.3%	0.7	2.3%	0.0	7.8%	0.1	2.5%	0.0	12.6%	0.5	4.4%	0.0	9.6%	0.3	2.3%	0.0

Table B5
Heuristic results for *bccm-IF* set.

1st Obj	Instance	Best found		IM		RM(1000)		PS		PSRL(1000)		ERC		ERCRL(1000)		RC		RCRL(1000)	
		Z^{a}	K	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$
min Z	bccm-IF-1A	229	6	301	9	261	7	309	8	240	6	365	10	309	8	365	10	309	8
	bccm-IF-1B	229	6	301	9	261	7	309	8	242	6	365	10	309	8	365	10	309	8
	bccm-IF-1C	259	8	326	10	287	8	347	9	303	8	390	11	337	9	390	11	330	9
	bccm-IF-2A	434	6	480	7	434	6	507	7	441	6	743	11	687	10	743	11	687	10
	bccm-IF-2B	434	6	480	7	436	6	507	7	441	6	743	11	681	10	743	11	681	10
	bccm-IF-2C	488	7	595	9	533	8	612	9	558	8	863	13	754	11	863	13	754	11
	bccm-IF-3A	120	5	138	6	129	5	145	6	128	5	174	7	153	6	174	7	153	6
	bccm-IF-3B	120	5	138	6	128	5	134	5	128	5	174	7	157	6	174	7	157	6
	bccm-IF-3C	126	5	164	8	141	6	168	7	143	6	201	8	173	7	199	8	164	7
	bccm-IF-4A	533	8	632	9	617	9	775	10	625	8	939	14	767	10	939	14	767	10
	bccm-IF-4B	533	8	632	9	614	9	770	10	630	8	939	14	812	11	939	14	812	11
	bccm-IF-4C	533	8	632	9	619	9	766	10	625	8	939	14	790	11	939	14	790	11
	bccm-IF-4D	538	9	692	10	640	10	697	9	689	9	974	14	839	12	974	14	842	12
	bccm-IF-5A	879	13	1103	16	990	14	1083	15	1007	14	1848	27	1608	23	1848	27	1608	23

Table B5 (*continued*)

1st Obj	Instance	Best found		IM		RM(1000)		PS		PSRL(1000)		ERC		ERCRL(1000)		RC		RCRL(1000)	
		Z[a]	K	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$
	bccm-IF-5B	879	13	1103	16	991	14	1083	15	1011	14	1848	27	1599	23	1848	27	1599	23
	bccm-IF-5C	879	13	1103	16	994	14	1083	15	1006	14	1848	27	1596	23	1848	27	1596	23
	bccm-IF-5D	879	13	1103	16	996	14	1085	15	1009	14	1848	27	1616	23	1848	27	1616	23
	bccm-IF-6A	343	7	377	8	348	7	410	8	357	7	448	9	387	8	448	9	387	8
	bccm-IF-6B	343	7	377	8	349	7	410	8	361	7	448	9	383	8	448	9	383	8
	bccm-IF-6C	367	8	415	10	389	9	483	10	437	9	486	11	439	10	486	11	441	10
	bccm-IF-7A	444	11	469	12	445	11	525	13	460	11	620	16	527	14	620	16	527	14
	bccm-IF-7B	444	11	469	12	446	11	525	13	477	12	620	16	523	13	620	16	523	13
	bccm-IF-7C	444	11	469	12	452	11	529	13	481	12	620	16	527	14	620	16	527	14
	bccm-IF-8A	545	9	676	11	630	10	745	11	675	10	987	16	917	14	987	16	917	14
	bccm-IF-8B	545	9	676	11	624	10	745	11	673	10	987	16	876	14	987	16	876	14
	bccm-IF-8C	547	10	676	11	659	10	748	11	732	11	1007	16	909	14	1007	16	909	14
	bccm-IF-9A	634	14	699	17	669	15	748	17	667	15	1082	26	976	23	1082	26	976	23
	bccm-IF-9B	546	14	699	17	657	15	748	17	670	15	1082	26	933	22	1082	26	933	22
	bccm-IF-9C	622	14	699	17	653	15	748	17	670	15	1082	26	963	22	1082	26	963	22
	bccm-IF-9D	546	14	684	16	670	15	719	16	670	15	1082	26	955	22	1082	26	955	22
	bccm-IF-10A	731	13	809	15	784	14	801	14	761	13	1179	21	1095	20	1179	21	1095	20
	bccm-IF-10B	732	13	809	15	783	14	801	14	759	13	1179	21	1034	18	1179	21	1034	18
	bccm-IF-10C	740	13	809	15	782	14	801	14	756	13	1179	21	1085	19	1179	21	1085	19
	bccm-IF-10D	732	13	809	15	785	14	802	14	792	14	1179	21	1087	20	1179	21	1087	20
	Mean Z_{gap} and K_{gap}			18.1%	1.9	9.9%	0.7	25.4%	1.6	12.3%	0.5	68.6%	6.9	48.0%	4.6	68.5%	6.9	47.7%	4.6
min K	bccm-IF-1A	229	6	289	8	257	7	280	7	240	6	365	10	309	8	365	10	309	8
	bccm-IF-1B	229	6	289	8	258	7	280	7	240	6	365	10	309	8	365	10	309	8
	bccm-IF-1C	259	8	317	9	279	8	335	9	298	8	390	11	337	9	390	11	330	9
	bccm-IF-2A	434	6	480	7	434	6	507	7	441	6	743	11	687	9	743	11	687	9
	bccm-IF-2B	434	6	480	7	436	6	507	7	441	6	743	11	681	9	743	11	681	9
	bccm-IF-2C	488	7	549	8	533	8	554	8	553	8	863	13	754	10	863	13	754	10
	bccm-IF-3A	120	5	131	5	125	5	145	6	128	5	174	7	153	6	174	7	153	6
	bccm-IF-3B	120	5	131	5	125	5	134	5	127	5	174	7	157	6	174	7	157	6
	bccm-IF-3C	126	5	159	7	139	6	151	6	143	6	201	8	173	7	199	8	164	6
	bccm-IF-4A	533	8	612	8	589	8	769	10	622	8	957	13	767	10	957	13	767	10
	bccm-IF-4B	533	8	612	8	603	8	702	9	626	8	957	13	812	10	957	13	812	10
	bccm-IF-4C	533	8	612	8	592	8	766	10	625	8	957	13	790	10	957	13	790	10
	bccm-IF-4D	538	9	692	10	629	9	697	9	689	9	974	14	839	11	974	14	842	11
	bccm-IF-5A	879	13	1059	15	986	14	1025	14	1007	14	1848	27	1608	22	1848	27	1608	22
	bccm-IF-5B	879	13	1059	15	991	14	1025	14	1006	14	1848	27	1599	22	1848	27	1599	22
	bccm-IF-5C	879	13	1059	15	981	13	1025	14	1001	14	1848	27	1596	22	1848	27	1596	22
	bccm-IF-5D	879	13	1059	15	989	14	1085	15	1009	14	1848	27	1616	22	1848	27	1616	22
	bccm-IF-6A	343	7	377	8	347	7	410	8	357	7	448	9	387	8	448	9	387	8
	bccm-IF-6B	343	7	377	8	345	7	410	8	359	7	448	9	383	8	448	9	383	8
	bccm-IF-6C	367	8	430	9	389	9	473	10	433	9	486	11	439	9	486	11	441	9
	bccm-IF-7A	444	11	469	12	445	11	525	13	457	11	620	16	527	13	620	16	527	13
	bccm-IF-7B	444	11	469	12	446	11	525	13	465	11	620	16	523	13	620	16	523	13
	bccm-IF-7C	444	11	469	12	452	11	526	13	476	12	620	16	527	13	620	16	527	13
	bccm-IF-8A	545	9	666	10	630	9	742	11	675	10	987	16	917	13	987	16	917	13
	bccm-IF-8B	545	9	666	10	624	10	742	11	670	10	987	16	876	13	987	16	876	13
	bccm-IF-8C	547	10	670	10	659	10	688	10	701	10	1007	16	909	13	1007	16	909	13
	bccm-IF-9A	634	14	685	16	669	15	713	16	664	15	1090	25	976	22	1090	25	976	22
	bccm-IF-9B	546	14	685	16	657	15	713	16	667	15	1090	25	933	21	1090	25	933	21
	bccm-IF-9C	622	14	685	16	653	14	713	16	667	15	1090	25	963	21	1090	25	963	21
	bccm-IF-9D	546	14	684	16	670	15	719	16	667	15	1082	26	955	21	1082	26	955	21
	bccm-IF-10A	731	13	786	14	766	14	760	13	755	13	1179	21	1095	19	1179	21	1095	19
	bccm-IF-10B	732	13	786	14	765	14	760	13	753	13	1179	21	1034	17	1179	21	1034	17
	bccm-IF-10C	740	13	786	14	768	14	760	13	751	13	1179	21	1085	18	1179	21	1085	18
	bccm-IF-10D	732	13	786	14	766	14	802	14	763	13	1179	21	1087	19	1179	21	1087	19
	Mean Z_{gap} and K_{gap}			15.5%	1.1	8.6%	0.5	21.4%	1.2	11.4%	0.4	69.0%	6.7	48.0%	3.9	68.9%	6.7	47.7%	3.9

[a] Best solution costs include those reported in [35].

Table B6
Heuristic results for *gdb-IF* set.

1st Obj	Instance	Best found		IM		RM(1000)		PS		PSRL(1000)		ERC		ERCRL(1000)		RC		RCRL(1000)	
		Z	K	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$
min Z	gdb-IF-1	345	6	351	6	345	6	407	7	350	6	441	8	383	7	441	8	383	7
	gdb-IF-2	345	6	399	7	366	7	353	6	354	6	465	9	393	7	457	8	393	7
	gdb-IF-3	312	6	326	6	312	6	332	6	327	6	379	7	337	6	379	7	334	6
	gdb-IF-4	460	6	548	7	462	6	483	6	462	6	756	10	659	8	756	10	659	8

Table B6 (*continued*)

1st Obj	Instance	Best found		IM		RM(1000)		PS		PSRL(1000)		ERC		ERCRL(1000)		RC		RCRL(1000)	
		Z	K	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$	Z	K	$\overline{Z}$	$\overline{K}$
	gdb-IF-5	586	7	645	8	640	8	649	8	593	7	1012	13	841	10	1012	13	841	10
	gdb-IF-6	301	4	341	5	302	4	339	4	310	4	355	6	310	4	355	5	310	4
	gdb-IF-7	371	6	412	7	373	6	457	8	373	6	504	9	424	7	504	9	424	7
	gdb-IF-8	445	10	477	11	456	10	516	11	478	10	630	15	535	12	624	15	529	12
	gdb-IF-9	354	9	386	11	368	10	395	9	390	9	485	13	454	11	471	12	446	11
	gdb-IF-10	423	7	470	8	425	7	569	10	479	8	623	11	577	10	623	11	577	10
	gdb-IF-11	606	10	697	12	628	10	726	12	661	11	939	16	853	14	939	16	853	14
	gdb-IF-12	835	8	1092	10	903	9	941	9	846	8	1419	14	1211	12	1419	14	1211	12
	gdb-IF-13	602	5	634	6	618	6	700	6	644	6	714	7	652	7	712	7	652	7
	gdb-IF-14	138	8	179	10	138	8	141	8	139	8	209	12	178	10	209	12	178	10
	gdb-IF-15	60	3	64	5	60	3	64	4	60	3	66	4	64	4	66	4	64	4
	gdb-IF-16	156	12	157	12	157	12	156	12	156	12	195	16	172	13	195	16	172	13
	gdb-IF-17	99	8	105	9	99	8	105	9	101	8	117	10	109	9	117	10	109	9
	gdb-IF-18	222	10	268	13	241	12	244	11	228	10	330	16	315	15	330	16	315	15
	gdb-IF-19	83	3	91	3	83	3	91	3	83	3	93	3	83	3	93	3	83	3
	gdb-IF-20	179	7	191	8	179	7	191	7	180	7	231	9	203	8	231	9	203	8
	gdb-IF-21	190	12	202	13	192	12	210	13	194	12	232	16	213	14	232	16	213	14
	gdb-IF-22	265	19	285	20	265	19	288	20	276	19	316	23	298	21	316	23	298	21
	gdb-IF-23	282	20	299	23	286	21	302	22	295	21	343	28	317	24	343	28	317	24
	Mean Z_{gap} and K_{gap}			11.0%	1.2	2.2%	0.3	11.2%	0.8	3.4%	0.2	34.8%	3.6	20.2%	1.9	34.5%	3.5	20%	1.9
min K	gdb-IF-1	345	6	351	6	345	6	373	6	347	6	441	8	383	6	441	8	383	6
	gdb-IF-2	345	6	399	7	366	7	353	6	354	6	471	8	393	7	457	8	393	7
	gdb-IF-3	312	6	326	6	312	6	332	6	321	6	379	7	337	6	379	7	334	6
	gdb-IF-4	460	6	495	6	462	6	483	6	462	6	756	10	659	7	756	10	659	7
	gdb-IF-5	586	7	645	8	640	8	646	8	593	7	1012	13	841	10	1012	13	841	10
	gdb-IF-6	301	4	341	5	302	4	339	4	310	4	367	5	310	4	355	5	310	4
	gdb-IF-7	371	6	412	7	373	6	457	8	373	6	504	9	424	7	504	9	424	7
	gdb-IF-8	445	10	472	10	454	10	492	10	474	10	634	14	535	11	628	14	529	11
	gdb-IF-9	354	9	388	10	365	9	395	9	385	9	511	12	454	10	471	12	446	10
	gdb-IF-10	423	7	470	8	425	7	524	9	477	8	623	11	577	10	623	11	577	10
	gdb-IF-11	606	10	694	11	619	10	686	11	641	10	939	16	853	13	939	16	853	13
	gdb-IF-12	835	8	1092	10	903	8	915	9	846	8	1419	14	1211	11	1419	14	1211	11
	gdb-IF-13	602	5	634	6	617	6	700	6	640	6	714	7	652	6	721	6	652	6
	gdb-IF-14	138	8	179	10	138	8	141	8	139	8	209	12	178	10	209	12	178	10
	gdb-IF-15	60	3	64	4	60	3	64	4	60	3	66	4	64	4	66	4	64	4
	gdb-IF-16	156	12	157	12	157	12	156	12	156	12	195	16	172	13	195	16	172	13
	gdb-IF-17	99	8	105	8	99	8	103	8	101	8	117	10	109	8	117	10	109	8
	gdb-IF-18	222	10	268	13	237	11	244	11	227	10	330	16	315	14	330	16	315	14
	gdb-IF-19	83	3	91	3	83	3	91	3	83	3	93	3	83	3	93	3	83	3
	gdb-IF-20	179	7	191	8	179	7	188	7	180	7	231	9	203	7	231	9	203	7
	gdb-IF-21	190	12	202	13	191	12	207	13	193	12	232	16	214	13	232	16	214	13
	gdb-IF-22	265	19	285	20	265	19	286	20	274	19	316	23	298	20	316	23	298	20
	gdb-IF-23	282	20	295	22	285	20	302	22	293	21	343	28	317	23	343	28	317	23
	Mean Z_{gap} and K_{gap}			10.4%	0.9	1.9%	0.2	9.3%	0.6	2.9%	0.1	35.5%	3.4	20.2%	1.3	34.6%	3.4	20%	1.3

Table B7
Computational times, in seconds, of heuristics for *Act-IF*, *Cen-IF* and *lpr-IF* sets.

Instance	α	IM	RM	PS	PSRL	ERC	ERCRL	RC	RCRL
Cen-IF-a	100	48.2	4818.3	0.8	9.5	1.1	18.0	15.6	259.5
Cen-IF-b	100	478.3	47,830.6	5.4	67.3	7.1	117.7	68.3	1138.5
Cen-IF-c	100	393.6	39,364.4	7.1	89.0	9.6	160.2	63.0	1050.0
Mean time		306.7	30,671.1	4.4	55.3	5.9	98.6	49.0	816.0
Act-IF-a	100	1.0	98.8	0.0	0.5	0.1	1.2	1.4	23.0
Act-IF-b	100	8.5	851.1	0.2	1.9	0.3	5.3	8.5	141.3
Act-IF-c	100	2.9	293.4	0.1	1.8	0.2	2.5	3.2	53.5
Mean time		4.1	414.4	0.1	1.4	0.2	3.0	4.4	72.6
Lpr-IF-a-01	100	0.1	5.3	0.0	0.3	0.0	0.4	0.1	1.4
Lpr-IF-a-02	100	0.2	23.6	0.1	1.8	0.1	1.2	0.4	7.1
Lpr-IF-a-03	100	2.3	225.4	0.1	1.2	0.2	3.4	2.6	43.3
Lpr-IF-a-04	100	5.9	585.5	0.2	2.3	0.3	5.4	4.8	80.7
Lpr-IF-a-05	100	9.7	971.5	0.4	4.5	0.7	11.3	8.5	141.9
Lpr-IF-b-01	100	0.0	2.9	0.0	0.1	0.0	0.3	0.1	1.2
Lpr-IF-b-02	100	0.1	13.3	0.1	0.9	0.0	0.8	0.4	6.5
Lpr-IF-b-03	100	1.4	143.4	0.1	1.2	0.2	3.5	2.7	44.3
Lpr-IF-b-04	100	3.2	319.6	0.2	2.6	0.4	6.1	5.0	83.1
Lpr-IF-b-05	100	10.2	1018.8	0.3	4.2	0.6	10.2	8.2	135.9

Table B7 (*continued*)

Instance	α	IM	RM	PS	PSRL	ERC	ERCRL	RC	RCRL
Lpr-IF-c-01	100	0.1	8.2	0.0	0.2	0.0	0.3	0.1	1.1
Lpr-IF-c-02	100	0.3	33.4	0.0	0.4	0.0	0.8	0.3	5.1
Lpr-IF-c-03	100	3.6	361.6	0.3	4.2	0.3	4.9	1.6	26.5
Lpr-IF-c-04	100	9.2	917.2	0.2	3.1	0.4	6.7	3.1	51.7
Lpr-IF-c-05	100	19.6	1961.7	0.5	5.8	0.7	12.2	5.5	91.7
Mean time		4.4	439.4	0.2	2.2	0.3	4.5	2.9	48.1

Table B8
Computational times, in seconds, of heuristics for *mval-IF-3L*.

Instance	α	IM	RM	PS	PSRL	ERC	ERCRL	U	RCRL
mval-IF-3L-1A	1000	0.1	61.2	0.0	4.2	0.0	3.4	0.1	12.2
mval-IF-3L-1B	1000	0.1	56.1	0.0	4.7	0.0	5.2	0.0	8.2
mval-IF-3L-1C	1000	0.1	53.3	0.0	3.8	0.0	3.5	0.0	6.1
mval-IF-3L-2A	1000	0.0	41.6	0.0	3.1	0.0	2.7	0.0	6.8
mval-IF-3L-2B	1000	0.0	44.6	0.0	2.3	0.0	3.4	0.1	9.7
mval-IF-3L-2C	1000	0.0	44.0	0.0	2.3	0.0	4.4	0.0	6.0
mval-IF-3L-3A	1000	0.0	46.3	0.0	2.1	0.0	2.4	0.1	8.7
mval-IF-3L-3B	1000	0.0	38.8	0.0	3.8	0.0	3.4	0.0	7.7
mval-IF-3L-3C	1000	0.0	38.2	0.0	4.0	0.0	4.6	0.0	6.6
mval-IF-3L-4A	1000	0.2	162.5	0.1	7.8	0.1	12.7	0.3	42.1
mval-IF-3L-4B	1000	0.2	175.7	0.0	4.6	0.1	8.6	0.2	33.3
mval-IF-3L-4C	1000	0.2	171.5	0.0	4.3	0.1	9.1	0.2	27.3
mval-IF-3L-4D	1000	0.2	169.4	0.0	5.4	0.1	10.4	0.1	19.3
mval-IF-3L-5A	1000	0.2	191.8	0.0	4.5	0.0	7.5	0.2	26.5
mval-IF-3L-5B	1000	0.3	299.6	0.0	3.5	0.0	7.5	0.1	21.3
mval-IF-3L-5C	1000	0.3	251.5	0.0	4.5	0.1	9.2	0.1	23.6
mval-IF-3L-5D	1000	0.3	308.4	0.0	4.0	0.1	8.5	0.1	20.0
mval-IF-3L-6A	1000	0.1	149.3	0.0	3.3	0.0	4.7	0.1	15.2
mval-IF-3L-6B	1000	0.1	136.7	0.0	2.6	0.0	5.4	0.1	13.5
mval-IF-3L-6C	1000	0.2	161.3	0.1	6.4	0.0	5.1	0.0	8.3
mval-IF-3L-7A	1000	0.3	281.8	0.0	3.0	0.1	9.2	0.1	20.8
mval-IF-3L-7B	1000	0.3	268.6	0.0	3.5	0.0	6.7	0.1	18.2
mval-IF-3L-7C	1000	0.3	273.0	0.1	12.1	0.1	10.0	0.1	15.2
mval-IF-3L-8A	1000	0.2	175.6	0.0	4.4	0.0	6.8	0.2	28.6
mval-IF-3L-8B	1000	0.2	172.3	0.0	6.2	0.0	6.8	0.2	25.3
mval-IF-3L-8C	1000	0.1	136.3	0.0	4.5	0.0	7.7	0.1	10.7
mval-IF-3L-9A	1000	0.3	314.8	0.0	6.2	0.1	11.5	0.3	44.4
mval-IF-3L-9B	1000	0.3	337.9	0.0	6.0	0.1	10.5	0.2	33.1
mval-IF-3L-9C	1000	0.3	327.1	0.1	13.7	0.1	11.0	0.2	35.3
mval-IF-3L-9D	1000	0.3	313.7	0.1	7.2	0.1	14.2	0.2	30.0
mval-IF-3L-10A	1000	0.3	333.1	0.1	15.3	0.1	22.8	0.3	57.7
mval-IF-3L-10B	1000	0.3	316.8	0.0	6.2	0.2	28.2	0.4	59.4
mval-IF-3L-10C	1000	0.3	337.6	0.1	12.2	0.1	24.1	0.4	61.9
mval-IF-3L-10D	1000	0.3	320.1	0.1	6.4	0.1	18.4	0.2	38.7
Mean time		0.2	191.5	0.0	5.5	0.1	9.1	0.1	23.6

Table B9
Computational times, in seconds, of heuristics for *bccm-IF-3L*.

Instance	α	IM	RM	PS	PSRL	ERC	ERCRL	RC	RCRL
bccm-IF-3L-1A	1000	0.1	63.6	0.0	1.6	0.0	2.1	0.0	5.2
bccm-IF-3L-1B	1000	0.1	69.3	0.0	1.5	0.0	2.7	0.0	4.5
bccm-IF-3L-1C	1000	0.1	65.7	0.0	2.7	0.0	3.1	0.0	3.7
bccm-IF-3L-2A	1000	0.0	49.4	0.0	1.7	0.0	2.3	0.0	5.1
bccm-IF-3L-2B	1000	0.1	56.7	0.0	1.7	0.0	2.1	0.0	4.3
bccm-IF-3L-2C	1000	0.1	50.9	0.0	1.6	0.0	4.3	0.0	3.7
bccm-IF-3L-3A	1000	0.1	54.4	0.0	3.6	0.0	2.1	0.0	5.4
bccm-IF-3L-3B	1000	0.1	51.3	0.0	2.3	0.0	2.2	0.0	4.7
bccm-IF-3L-3C	1000	0.1	50.3	0.0	2.4	0.0	2.4	0.0	3.2
bccm-IF-3L-4A	1000	0.2	216.3	0.1	7.7	0.0	8.0	0.1	22.5
bccm-IF-3L-4B	1000	0.2	220.1	0.0	3.1	0.0	8.1	0.1	19.5
bccm-IF-3L-4C	1000	0.2	213.8	0.1	7.1	0.0	6.3	0.1	15.8
bccm-IF-3L-4D	1000	0.2	226.7	0.0	3.3	0.0	5.3	0.1	10.4
bccm-IF-3L-5A	1000	0.2	192.7	0.0	3.6	0.0	3.3	0.1	13.2
bccm-IF-3L-5B	1000	0.2	193.0	0.0	4.3	0.0	6.0	0.1	12.1
bccm-IF-3L-5C	1000	0.2	199.9	0.0	4.3	0.0	4.2	0.1	9.7
bccm-IF-3L-5D	1000	0.2	188.5	0.0	3.4	0.0	6.5	0.1	9.9
bccm-IF-3L-6A	1000	0.1	108.6	0.0	2.2	0.0	6.3	0.1	11.5

Table B9 (*continued*)

Instance	α	IM	RM	PS	PSRL	ERC	ERCRL	RC	RCRL
bccm-IF-3L-6B	1000	0.1	117.0	0.0	3.9	0.0	4.6	0.1	9.2
bccm-IF-3L-6C	1000	0.1	119.8	0.0	2.7	0.0	5.3	0.0	6.0
bccm-IF-3L-7A	1000	0.2	192.5	0.0	2.7	0.0	6.3	0.1	13.7
bccm-IF-3L-7B	1000	0.2	201.6	0.0	5.5	0.0	6.2	0.1	15.2
bccm-IF-3L-7C	1000	0.2	201.1	0.0	3.6	0.0	6.2	0.1	8.4
bccm-IF-3L-8A	1000	0.2	182.0	0.0	5.3	0.0	5.6	0.1	14.5
bccm-IF-3L-8B	1000	0.2	181.4	0.0	3.0	0.0	4.7	0.1	11.8
bccm-IF-3L-8C	1000	0.2	179.6	0.0	3.1	0.0	4.9	0.0	6.8
bccm-IF-3L-9A	1000	0.4	368.2	0.0	4.0	0.1	14.3	0.2	36.7
bccm-IF-3L-9B	1000	0.4	413.4	0.1	12.7	0.0	6.9	0.2	30.2
bccm-IF-3L-9C	1000	0.5	540.4	0.1	7.8	0.1	10.7	0.2	35.1
bccm-IF-3L-9D	1000	0.4	394.8	0.0	6.2	0.1	13.1	0.1	18.5
bccm-IF-3L-10A	1000	0.4	426.9	0.1	7.6	0.1	14.4	0.2	37.6
bccm-IF-3L-10B	1000	0.4	413.9	0.1	8.3	0.1	9.3	0.2	35.4
bccm-IF-3L-10C	1000	0.4	428.9	0.1	7.1	0.1	8.4	0.2	27.1
bccm-IF-3L-10D	1000	0.4	437.3	0.0	4.7	0.0	8.1	0.1	14.2
Mean time		0.2	207.9	0.0	4.3	0.0	6.1	0.1	14.3

Table B10
Computational times, in seconds, of heuristics for *gdb-IF-3L*.

Instance	α	IM	RM	PS	PSRL	ERC	ERCRL	RC	RCRL
gdb-IF-3L-1	1000	0.0	19.5	0.0	1.2	0.0	2.4	0.0	2.2
gdb-IF-3L-2	1000	0.0	27.8	0.0	2.5	0.0	2.6	0.0	3.3
gdb-IF-3L-3	1000	0.0	20.4	0.0	1.7	0.0	2.3	0.0	2.4
gdb-IF-3L-4	1000	0.0	16.4	0.0	1.3	0.0	1.5	0.0	1.8
gdb-IF-3L-5	1000	0.0	28.4	0.0	3.6	0.0	3.4	0.0	3.9
gdb-IF-3L-6	1000	0.0	20.8	0.0	1.5	0.0	1.5	0.0	1.8
gdb-IF-3L-7	1000	0.0	23.7	0.0	1.2	0.0	1.8	0.0	2.2
gdb-IF-3L-8	1000	0.1	95.8	0.0	3.8	0.0	5.0	0.0	6.6
gdb-IF-3L-9	1000	0.1	117.6	0.0	3.2	0.0	6.2	0.0	6.1
gdb-IF-3L-10	1000	0.0	27.9	0.0	3.3	0.0	2.3	0.0	3.1
gdb-IF-3L-11	1000	0.1	94.8	0.0	3.1	0.0	4.1	0.0	5.3
gdb-IF-3L-12	1000	0.0	24.2	0.0	3.5	0.0	2.1	0.0	2.3
gdb-IF-3L-13	1000	0.0	29.2	0.0	1.9	0.0	2.4	0.0	2.4
gdb-IF-3L-14	1000	0.0	18.5	0.0	2.4	0.0	1.8	0.0	2.3
gdb-IF-3L-15	1000	0.0	19.2	0.0	2.1	0.0	1.5	0.0	1.9
gdb-IF-3L-16	1000	0.0	32.2	0.0	1.8	0.0	2.3	0.0	2.6
gdb-IF-3L-17	1000	0.0	34.4	0.0	1.7	0.0	1.9	0.0	2.4
gdb-IF-3L-18	1000	0.1	59.1	0.0	1.9	0.0	2.6	0.0	3.4
gdb-IF-3L-19	1000	0.0	5.4	0.0	1.7	0.0	1.0	0.0	1.0
gdb-IF-3L-20	1000	0.0	21.5	0.0	3.0	0.0	2.1	0.0	2.4
gdb-IF-3L-21	1000	0.0	43.4	0.0	2.2	0.0	2.7	0.0	3.2
gdb-IF-3L-22	1000	0.1	94.4	0.0	3.0	0.0	5.7	0.0	6.4
gdb-IF-3L-23	1000	0.1	136.7	0.1	10.8	0.0	6.9	0.1	9.3
Mean time		0.0	44.0	0.0	2.7	0.0	2.9	0.0	3.4

Table B11
Computational times, in seconds, of heuristics for *bccm-IF*.

Instance	α	IM	RM	PS	PSRL	ERC	ERCRL	RC	RCRL
bccm-IF-1A	1000	0.1	70.2	0.0	4.3	0.0	5.2	0.0	5.6
bccm-IF-1B	1000	0.1	69.5	0.0	5.8	0.0	4.9	0.0	5.5
bccm-IF-1C	1000	0.1	68.7	0.0	5.3	0.0	5.9	0.0	6.3
bccm-IF-2A	1000	0.1	63.5	0.0	3.3	0.0	4.1	0.0	4.6
bccm-IF-2B	1000	0.1	51.6	0.0	4.1	0.0	5.5	0.0	6.1
bccm-IF-2C	1000	0.1	56.2	0.1	7.6	0.0	4.7	0.0	5.0
bccm-IF-3A	1000	0.1	58.6	0.0	3.8	0.0	3.4	0.0	4.0
bccm-IF-3B	1000	0.1	56.2	0.0	2.4	0.0	4.5	0.0	4.9
bccm-IF-3C	1000	0.1	55.0	0.0	3.3	0.0	3.8	0.0	4.2
bccm-IF-4A	1000	0.2	223.3	0.0	4.7	0.1	10.8	0.1	12.5
bccm-IF-4B	1000	0.2	234.3	0.1	8.3	0.1	10.5	0.1	12.6
bccm-IF-4C	1000	0.2	222.3	0.1	11.5	0.1	10.2	0.1	11.8
bccm-IF-4D	1000	0.2	227.3	0.1	6.5	0.0	8.1	0.1	9.7
bccm-IF-5A	1000	0.2	203.1	0.2	22.1	0.1	8.7	0.1	9.5
bccm-IF-5B	1000	0.2	199.9	0.1	10.3	0.0	8.2	0.1	9.3
bccm-IF-5C	1000	0.2	209.6	0.1	14.0	0.1	8.8	0.1	9.6
bccm-IF-5D	1000	0.2	198.4	0.1	14.1	0.0	7.9	0.1	8.8
bccm-IF-6A	1000	0.1	116.6	0.0	4.9	0.0	7.6	0.1	9.0

Table B11 (*continued*)

Instance	α	IM	RM	PS	PSRL	ERC	ERCRL	RC	RCRL
bccm-IF-6B	1000	0.1	127.6	0.0	3.8	0.0	7.5	0.1	8.5
bccm-IF-6C	1000	0.1	118.0	0.1	11.9	0.1	8.8	0.1	9.6
bccm-IF-7A	1000	0.2	203.7	0.1	10.9	0.1	10.3	0.1	11.6
bccm-IF-7B	1000	0.2	219.9	0.1	10.8	0.1	10.5	0.1	11.3
bccm-IF-7C	1000	0.2	211.4	0.1	18.2	0.1	10.4	0.1	11.3
bccm-IF-8A	1000	0.2	189.5	0.1	10.6	0.0	6.8	0.0	8.1
bccm-IF-8B	1000	0.2	194.9	0.1	12.0	0.0	7.0	0.0	8.2
bccm-IF-8C	1000	0.2	180.8	0.1	14.4	0.0	7.8	0.1	8.8
bccm-IF-9A	1000	0.5	492.4	0.2	20.9	0.1	11.9	0.1	13.5
bccm-IF-9B	1000	0.6	636.7	0.1	18.3	0.1	11.4	0.1	12.8
bccm-IF-9C	1000	0.4	392.1	0.2	30.3	0.1	11.6	0.1	13.2
bccm-IF-9D	1000	0.4	396.7	0.2	26.3	0.1	11.9	0.1	13.9
bccm-IF-10A	1000	0.7	745.7	0.2	20.2	0.1	9.6	0.1	11.7
bccm-IF-10B	1000	0.7	685.7	0.2	22.3	0.1	9.9	0.1	11.7
bccm-IF-10C	1000	0.4	435.4	0.1	11.4	0.1	9.4	0.1	11.4
bccm-IF-10D	1000	0.4	431.2	0.1	18.7	0.1	10.3	0.1	11.7
Mean time		0.2	236.6	0.1	11.7	0.0	8.2	0.1	9.3

Table B12
Computational times, in seconds, of heuristics for *gdb-IF*.

Instance	α	IM	RM	PS	PSRL	ERC	ERCRL	RC	RCRL
gdb-IF-1	1000	0.0	24.1	0.0	2.4	0.0	3.4	0.0	3.5
gdb-IF-2	1000	0.0	34.7	0.0	4.1	0.0	4.2	0.0	4.8
gdb-IF-3	1000	0.0	24.4	0.0	4.9	0.0	3.6	0.0	3.9
gdb-IF-4	1000	0.0	19.6	0.0	3.5	0.0	4.2	0.0	4.3
gdb-IF-5	1000	0.0	37.3	0.1	7.9	0.0	4.5	0.0	4.5
gdb-IF-6	1000	0.0	21.6	0.0	2.8	0.0	2.5	0.0	2.7
gdb-IF-7	1000	0.0	23.5	0.1	7.8	0.0	5.2	0.0	5.5
gdb-IF-8	1000	0.1	110.5	0.1	11.6	0.0	5.0	0.0	6.1
gdb-IF-9	1000	0.2	164.6	0.0	5.5	0.0	5.6	0.1	8.4
gdb-IF-10	1000	0.0	28.5	0.0	4.9	0.0	4.6	0.0	4.6
gdb-IF-11	1000	0.1	108.3	0.0	5.6	0.0	6.5	0.0	7.1
gdb-IF-12	1000	0.0	30.3	0.0	6.1	0.0	4.6	0.0	4.7
gdb-IF-13	1000	0.0	34.2	0.0	6.1	0.0	4.9	0.0	4.3
gdb-IF-14	1000	0.0	26.9	0.0	4.8	0.0	5.1	0.0	4.9
gdb-IF-15	1000	0.0	17.9	0.0	1.5	0.0	1.4	0.0	1.6
gdb-IF-16	1000	0.0	46.2	0.1	11.9	0.0	7.3	0.0	7.2
gdb-IF-17	1000	0.0	37.9	0.0	5.3	0.0	3.5	0.0	3.7
gdb-IF-18	1000	0.1	70.4	0.1	11.2	0.0	5.5	0.0	5.6
gdb-IF-19	1000	0.0	5.7	0.0	1.2	0.0	1.6	0.0	1.7
gdb-IF-20	1000	0.0	24.2	0.0	5.0	0.0	4.4	0.0	4.5
gdb-IF-21	1000	0.1	55.4	0.1	9.5	0.0	6.3	0.0	6.5
gdb-IF-22	1000	0.1	114.5	0.1	11.3	0.0	6.0	0.0	6.3
gdb-IF-23	1000	0.2	158.0	0.2	30.6	0.1	8.8	0.1	9.7
Mean time		0.1	53.0	0.1	7.2	0.0	4.7	0.0	5.0

Appendix C. Supplementary data

Supplementary data associated with this paper can be found in the online version at http://dx.doi.org/10.1016/j.cor.2015.10.010.

References

[1] Bartolini E, Cordeau J-F, Laporte G. Improved lower bounds and exact algorithm for the capacitated arc routing problem. Math Program 2013;137(1-2):409–52.
[2] Bautista J, Fernández E, Pereira J. Solving an urban waste collection problem using ant heuristics. Comput Oper Res 2008;35(9):3020–33.
[3] Belenguer JM, Benavent E. A cutting plane algorithm for the capacitated arc routing problem. Comput Oper Res 2003;30(5):705–28.
[4] Belenguer J-M, Benavent E, Lacomme P, Prins C. Lower and upper bounds for the mixed capacitated arc routing problem. Comput Oper Res 2006;33(12):3363–83.
[5] Benavent E, Campos V, Corberán A, Mota E. The capacitated arc routing problem: lower bounds. Networks 1992;22(7):669–90.
[6] Bode C, Irnich S. Cut-first branch-and-price-second for the capacitated arc-routing problem. Oper Res 2012;60(5):1167–82.
[7] Brandão J, Eglese R. A deterministic tabu search algorithm for the capacitated arc routing problem. Comput Oper Res 2008;35(4):1112–26.
[8] Chu F, Labadi N, Prins C. A scatter search for the periodic capacitated arc routing problem. Eur J Oper Res 2006;169(2):586–605.
[9] Constantino M, Gouveia L, Mourão MC, Nunes AC. The mixed capacitated arc routing problem with non-overlapping routes. Eur J Oper Res; 2015;244(2):445–456.
[10] Corberán, Á, Laporte G. Arc routing: problems, methods, and applications, vol. 20. Philadelphia:SIAM; 2015.
[11] Corberán A, Prins C. Recent results on arc routing problems: an annotated bibliography. Networks 2010;56(1):50–69.
[12] Coutinho-Rodrigues J, Rodrigues N, Clímaco J. Solving an urban routing problem using heuristics: a successful case study. Belgian J Oper Res Stat Comput Sci 1993;33 (1):2.
[13] Del Pia A, Filippi C. A variable neighborhood descent algorithm for a real waste collection problem with mobile depots. Int Trans Oper Res 2006;13(2):125–41.
[14] Doulabi SHH, Seifi A. Lower and upper bounds for location-arc routing problems with vehicle capacity constraints. Eur J Oper Res 2013;224(1):189–208.

[15] Dror M, editor. Arc routing: theory, solutions, and applications. Boston: Kluwer Academic Publishers; 2000.
[16] Ghiani G, Guerriero F, Improta G, Musmanno R. Waste collection in Southern Italy: solution of a real-life arc routing problem. Int Trans Oper Res 2005;12(2):135–44.
[17] Ghiani G, Guerriero F, Laporte G, Musmanno R. Tabu search heuristics for the arc routing problem with intermediate facilities under capacity and length restrictions. J Math Model. Algorithms 2004;3(3):209–23.
[18] Ghiani G, Improta G, Laporte G. The capacitated arc routing problem with intermediate facilities. Networks 2001;37(3):134–43.
[19] Ghiani G, Laganá D, Laporte G, Mari F. Ant colony optimization for the arc routing problem with intermediate facilities under capacity and length restrictions. J Heuristics 2010;16(2):211–33.
[20] Golden BL, DeArmon JS, Baker EK. Computational experiments with algorithms for a class of routing problems. Comput Ind Eng 1983;10(1):47–59.
[21] Golden BL, Wong RT. Capacitated arc routing problems. Networks 1981;11(3):305–15.
[22] Gouveia L, Mourão MC, Pinto LS. Lower bounds for the mixed capacitated arc routing problem. Comput Oper Res 2010;37(4):692–9.
[23] Grandinetti L, Guerriero F, Laganà D, Pisacane O. An optimization-based heuristic for the multi-objective undirected capacitated arc routing problem. Comput Oper Res 2012;39(10):2300–9.
[24] Groves G, Le Roux J, Van Vuuren J. Network service scheduling and routing. Int Trans Oper Res 2004;11(6):613–43.
[25] Lacomme P, Prins C, Ramdane-Chérif W. Competitive memetic algorithms for arc routing problems. Ann Oper Res 2004;131(4):159–85.
[26] Liu T, Jiang Z, Geng N. A memetic algorithm with iterated local search for the capacitated arc routing problem. Int J Prod Res 2013;51(10):3075–84.
[27] Martinelli R, Poggi M, Subramanian A. Improved bounds for large scale capacitated arc routing problem. Comput Oper Res 2013;40(8):2145–60.
[28] Martinez C, Loiseau I, Resende M, Rodriguez S. BRKGA algorithm for the capacitated arc routing problem. Electron Notes Theoret Comput Sci 2011;281:69–83.
[29] Mei Y, Li X, Yao X. Cooperative co-evolution with route distance grouping for large-scale capacitated arc routing problems. IEEE Trans Evol Comput 2014;18(3):435–49.
[30] Mourão MC, Almeida MT. Lower-bounding and heuristic methods for a refuse collection vehicle routing problem. Eur J Oper Res 2000;121(2):420–34.
[31] Mourão MC, Amado L. Heuristic method for a mixed capacitated arc routing problem: a refuse collection application. Eur J Oper Res 2005;160(1):139–53.
[32] Mourão MC, Nunes AC, Prins C. Heuristic methods for the sectoring arc routing problem. Eur J Oper Res 2009;196(3):856–68.
[33] Muyldermans L, Pang G. A guided local search procedure for the multi-compartment capacitated arc routing problem. Comput Oper Res 2010;37(9):1662–73.
[34] Pearn WL. Approximate solutions for the capacitated arc routing problem. Comput Oper Res 1989;16(6):589–600.
[35] Polacek M, Doerner Karl F, Hartl RF, Maniezzo V. A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. J Heuristics 2008;14(5):405–23.
[36] Rardin RL, Uzsoy R. Experimental evaluation of heuristic optimization algorithms: a tutorial. J Heuristics 2001;7(3):261–304.
[37] Rosa BD, Improta G, Ghiani G, Musmanno R. The arc routing and scheduling problem with transshipment. Transport Sci 2002;36(3):301–13.
[38] Santos L, Coutinho-Rodrigues J, Antunes CH. A web spatial decision support system for vehicle routing using google maps. Decis Support Syst 2011;51(1):1–9.
[39] Santos L, Coutinho-Rodrigues J, Current JR. Implementing a multi-vehicle multi-route spatial decision support system for efficient trash collection in Portugal. Transp Res Part A: Policy Pract 2008;42(6):922–34.
[40] Santos L, Coutinho-Rodrigues J, Current JR. An improved heuristic for the capacitated arc routing problem. Comput Oper Res 2009;36(9):2632–7.
[41] Talbi E. Metaheuristics: from design to implementation. New Jersey: Wiley; 2009.
[42] Ulusoy G. The fleet size and mix problem for capacitated arc routing. Eur J Oper Res 1985;22(3):329–37.
[43] Usberti FL, França PM, França ALM. Grasp with evolutionary path-relinking for the capacitated arc routing problem. Comput Oper Res 2013;40(12):3206–17.
[44] Willemse EJ, Joubert JW. Applying min-max *k* postmen problems to the routing of security guards. J Oper Res Soc 2012;63(2):245–60.
[45] Willemse EJ, Joubert JW. Constructive heuristics for the residential waste collection problem. In: Willemse EJ, Ittmann H, Yadavalli V, editors. Proceedings of the 2011 ORSSA annual conference, 18–21 September 2011, Elephant Hills Hotel, Victoria Falls, Zimbabwe; 2011. p. 19–28.
[46] Willemse EJ, Joubert JW. Splitting procedures and benchmark problems for the mixed arc routing problem with intermediate facilities under capacity and length restrictions; 2015. Available online from ⟨https://www.sites.google.com/site/wasteoptimisation/capacitated-arc-routing-problems/working-papers⟩[accessed 10-09-15].