Technical report 7.11.2016

A Fresh Ruin & Recreate Implementation for the Capacitated Vehicle Routing Problem

Jan Christiaens, Greet Vanden Berghe
KU Leuven, Department of Computer Science, CODeS & iMinds-ITEC
Gebr. De Smetstraat 1, 9000 Gent, Belgium, jan.christiaens@cs.kuleuven.be, greet.vandenberghe@cs.kuleuven.be

Problems such as the Capacitated Vehicle Routing Problem (CVRP) attract both mathematical modellers and heuristic approaches. At present, exact mathematical approaches are capable of solving some CVRP instances with up to 360 customers. The excessive computation times incurred by such exact models does, however, severely limit their practical applicability within logistics and transportation sectors. By contrast, heuristic approaches are generally faster and easier to adapt to other problems, albeit at the expense of solution quality. Ruin & recreate represents one such heuristic approach. However, recent ruin & recreate heuristics, while being deployed for more and more problems, have been concomitant with an additive trend whereby the quantity of ruin methods and recreate methods has been systematically increasing. Essentially, improved ruin & recreate results regularly coincide with challenging to reproduce methods. This paper's approach (ASB-RR), by contrast, is formed of a single ruin method, adjacent string removal, and a single recreate method, greedy insertion with blinks. ASB-RR exhibits low computation times, robustness and yields a high number of improved benchmark solutions when compared against state of the art CVRP algorithms. Furthermore, the approach may be easily redeployed for similar problems within the field of vehicle routing.

Key words: capacitated vehicle routing; ruin & recreate heuristic

1. Introduction

The capacitated vehicle routing problem (CVRP) was introduced by Dantzig and Ramser (1959) as the truck dispatching problem, a generalization of the traveling salesman problem (TSP). Despite over 55 years of research (Laporte 2009), the CVRP remains a highly active field of research and it continues to represent a significant computational challenge. Research concerning the problem mostly falls into one of two categories: exact algorithms or heuristics.

Developing exact algorithms is an interesting activity in itself while also providing objective function bounds which enable accurate heuristic quality assessment. Toth and Vigo (2002) provide a comprehensive overview of exact algorithms to solve CVRPs with either symmetric or asymmetric cost matrices. They primarily focus on Branch and Bound-based algorithms and review a set of original powerful approaches. The Branch-Cut-and-Price (BCP) algorithm by Fukasawa et al. (2006), improved upon previous Branch-and-Cut algorithms. Several improvements such as by Baldacci, Christofides, and Mingozzi (2008) are combined with new elements (Pecin et al. 2014), and prove capable of solving CVRPs of up to 350 customers and even 650 in some cases. Computation times can, however, be as long as five days, which makes such solvers impractical for many real-world purposes.

Heuristics, by contrast, are characterized as fast and adaptable methods. Absolute optimality is, however, not ensured - a necessary trade-off in terms of accommodating many real-world situations which rely on the presence of timely solutions. Heuristics generally employ specific neighborhoods which are explored by a metaheuristic to gradually improve candidate solutions. Prior to the latest improvements, classical neighborhoods such as 2-opt* (Potvin and Rousseau 1995) and CROSS-exchange (Taillard et al. 1997) are explored in single-solution-based heuristics. Larger neighborhoods are explored by Shaw (1998) and Pisinger and Røpke (2007) in a ruin & recreate framework. Prins (2004) contributed a hybrid genetic algorithm, the first population-based framework for VRPs, which improved significantly upon the prior approaches. Recently this approach was refined by Vidal et al. (2015), thereby realizing the state of the art VRP heuristic. While the initial ambition of heuristics is noble, they have gradually become increasingly complicated. The present paper seeks to remedy this by introducing a low-level, yet simultaneously powerful and fast, approach which is sufficiently adaptable and, as such, may be easily incorporated into any current or future VRP approach. This ruin & recreate approach's improvements to the state of the art will subsequently be demonstrated and documented.

Regarding the structure of the paper, it begins by first offering a problem definition of the CVRP. Following this a general introduction to heuristics and metaheuristics is provided, with particular attention paid to Simulated Annealing. Next, a comprehensive review of classical neighborhoods is presented. Expert readers already familiar with classical neighborhoods are free to skip this section, although some original terminology is introduced which helps unify the field and aid in the subsequent presentation of this paper's original contribution. An introduction to ruin & recreate is provided during the following section. Section 6 introduces ABS-RR, adjacent string removal & greedy insertion with blinks, this paper's ruin & recreate contribution. Computational results are detailed in the following section and the paper ultimately ends with a conclusion section summarising the results and delineating the scope and possibilities for future research.

2. Problem description and methodology

The CVRP, considered by the current paper, is defined as follows. Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ be a complete undirected graph in which $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ the set of edges. The vertices $v_i \in \mathcal{V}$ for

$i \in \{0, ..., n\}$ represent locations in a 2-dimensional space where v_0 corresponds to the *depot* and the other n vertices with *customers* having a demand q_i. Each edge (i, j) in $\mathcal{E} = \{(i, j) : i, j \in \mathcal{V}, i \neq j\}$ is associated with a *cost* c_{ij}. An unlimited homogeneous fleet of vehicles with capacity $\mathcal{Q}$ is situated at the depot. The CVRP consist of designing vehicle *tours* at minimum cost such that each customer is served exactly once by a single vehicle while each tour starts at the depot v_0, serves customers without exceeding the vehicle's capacity Q and finally ends at the same depot v_0.

3. Heuristics and metaheuristics

While exact mathematical optimization methods attempt to find the optimal solution for a given problem, heuristics search for reasonably good solutions within a small amount of computation time. *Local search* (LS) heuristics begin with an initial solution s_0 for the problem provided by a constructive method (CM). LS seeks better solutions by iteratively replacing the current solution s with an improving or equal neighbor solution $s \leftarrow s'$, a substitution referred to as a *move*. The neighborhood function $\mathcal{N}$ is a mapping of solution s to the set of its neighbors $s \mapsto \mathcal{N}(s)$ or simply its *neighborhood*. LS terminates in a local optimum $\bar{s}$ when $\mathcal{N}(\bar{s})$ contains no improving neighbor. By contrast, *metaheuristics* are capable of 'escaping' from such local optima by their ability of moving to lower quality neighbors. The subset $\mathcal{C}(s) \subseteq \mathcal{N}(s)$ represents a set of candidate solutions, from which s' is selected (neighbor selection NS) by a local search method and possibly accepted by the neighbor acceptance (NA) criterion. The metaheuristic terminates when a certain stop criterion (SC) is satisfied, for example computation time limit or maximum number of iterations. Fig. 1 introduces a schematic representation of a metaheuristic.

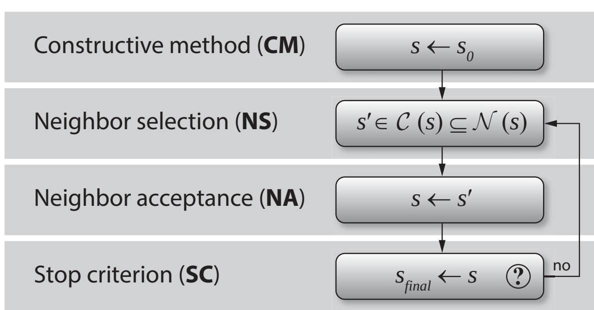

Figure 1 **Local search metaheuristic.**

A sequence of moves denotes a walk, or a search trajectory, through the solution space by iteratively moving from the current solution to one of its neighbors (Crainic and Toulouse 2003). The search trajectory in Fig. 2, for example, begins at the initial solution ($s \leftarrow s_0$). The neighborhood $\mathcal{N}(s_0)$ contains nine neighbors of which four are present in the set of candidates $\mathcal{C}(s_0)$. Solution

s_1 is accepted from $\mathcal{C}(s_0)$ by the first move m_1 $(s \leftarrow s_1)$ and neighborhood $\mathcal{N}(s_2)$ is reached after performing the second move m_2 $(s \leftarrow s_2)$.

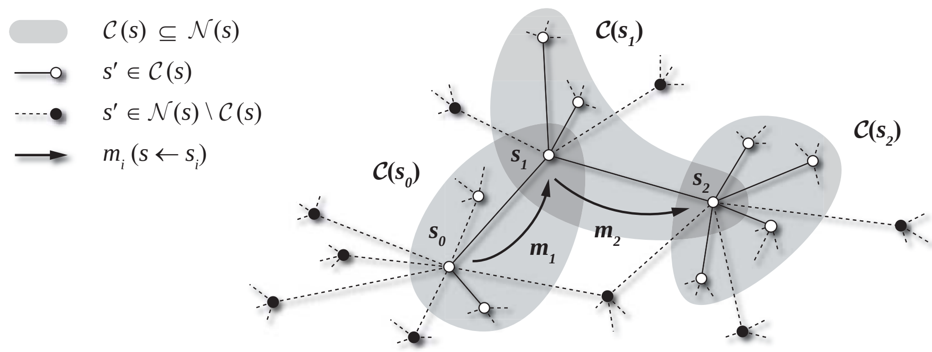

Figure 2 **Search trajectory through the solution space.**

Notice that neighborhood $\mathcal{N}(s_2)$ overlaps with initial neighborhood $\mathcal{N}(s_0)$, potentially resulting in metaheuristic 'cycling'. The *Tabu Search* metaheuristic (Glover 1986), for example, maintains characteristics of previously visited solutions in a set T to avoid such cycling behavior. *Simulated Annealing* is an often-employed meta-heuristic whose NA criterion accepts non-improving neighbors based upon the annealing process of metals. Given that the algorithm presented in Section 6 is guided by this metaheuristic, a more detailed explanation of Simulated Annealing is located within the following section.

3.1. Simulated Annealing

The field of metallurgy defines annealing as the process by which the physical properties of materials are modified through controlled heating and cooling. A statistical model concerning the energy changes in such annealing systems was developed by Metropolis et al. (1953). Based on this work, Kirkpatrick, Gelatt, and Vecchi (1983) introduced *Simulated Annealing* (SA) as a metaheuristic capable of solving combinatorial optimisation problems. The transition to a new state $s \leftarrow s'$ is dependent upon the energy change of the system $\Delta E = E(s') - E(s)$ and the current temperature T. While improving or equal quality neighbors $(\Delta E \leq 0)$ are always accepted, the transition to a non-improving neighbor occurs with an acceptance probability function $h(\Delta E, T) = exp(-\Delta E/T)$. Non-improving solutions are therefore more likely to be accepted at high values of T or low values of ΔE (Fig. 3a). This probability distribution is employed by SA as the probabilistic NA criterion.

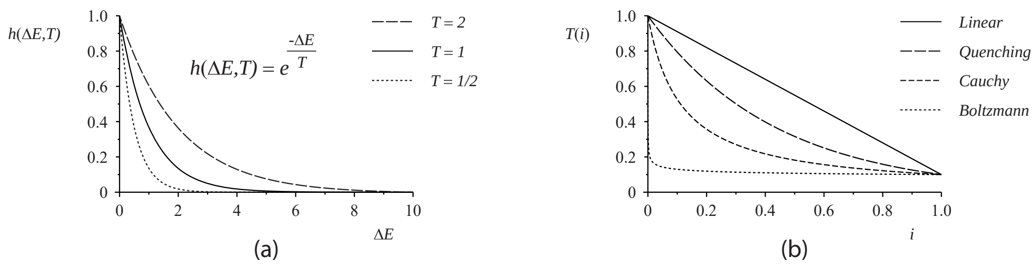

Figure 3 **Simulated Annealing: acceptance distribution (a) and cooling scheme (b).**

SA begins at an initial temperature, T_0, high enough such that all non-improving neighbors are likely accepted. Assuming the system is cooled to the theoretical final temperature $T_1 = 0$ over an infinite length of time, SA will converge to the problem's optimum solution. In practice, the system is cooled to a final temperature $T_1 > 0$ (the stop criterion). The cooling rate is specified by a *cooling scheme* $T(i)$ as a function of the normalized annealing time i $(0 \leq i \leq 1)$. Fig. 3b illustrates the envelopes of the *Linear*, *Exponential*, *Hyperbolic* and *Logarithmic* cooling schemes when the system is cooled from $T_0 = 1$ to $T_1 = 0.1$. The corresponding equations are provided by Table 1. The influence on the acceptance distribution is visualized in Fig. 4 for the Linear (a), Quenching (b) and Cauchy (c) cooling schemes.

Table 1 **Cooling schemes.**

	Temperature	Cooling constant
Linear	$T(i) = T_0 + c.i$	$c = T_0 - T_1$
Exponential	$T(i) = T_0.c^i$	$c = T_1/T_0$
Hyperbolic	$T(i) = T_0/(1 + c.i)$	$c = (T_0 - T_1)/T_0$
Logarithmic	$T(i) = T_0/(1 + ln(c.i))$	$c = exp((T_0 - T_1)/T_1)$

When the linear scheme is applied (Fig. 4a), non-improving solutions are potentially accepted quite far into the search, thus implying a rather diversified search. The hyperbolic scheme (c) only accepts non-improving solutions at the very beginning of the search. An exponential scheme represents a good compromise for many researchers, wherein a diversified search is obtained during the first half of annealing-time, while the search becomes strongly intensified during the second half (b).

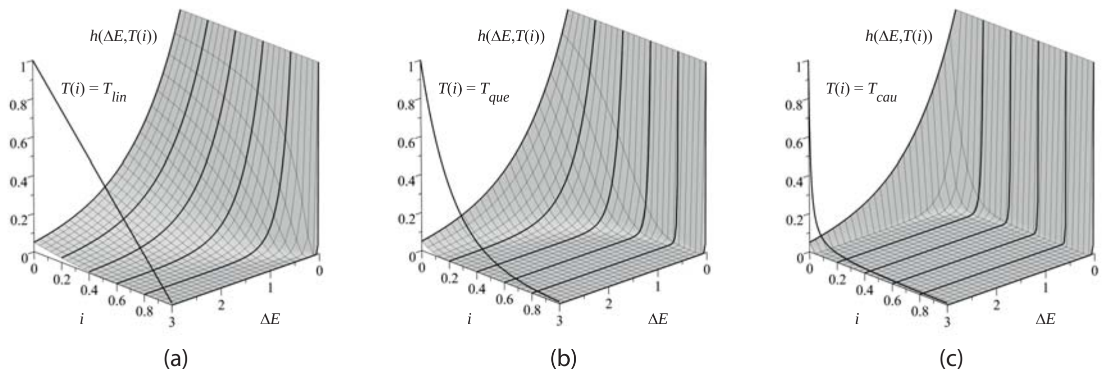

Figure 4 **Envelope of the acceptance probability h based on three cooling schemes: Linear (a), Exponential (b) and Hyperbolic (c).**

Most metaheuristics require *parameter tuning* when applied to a given problem as there are no parameter settings ensuring good performance on all problem types. For example, larger problems generally require more iterations to obtain gradual convergence. SA requires T_0 and T_1 to be ‘tuned’ which further impacts upon the cooling scheme shape, and therefore finding good values for these parameters proves a difficult task.

4. Classical neighborhoods

Before explaining the incentive behind the algorithm introduced by this paper, the present section first analyzes the local search improvement operators employed by TSP, CVRP and VRPTW (VRP with Time Windows) heuristics. Such an analysis will indicate the common mechanisms of the corresponding neighborhoods. While readers already familiar with such details are free to skip over this section, it does in fact contribute a comprehensive comparative overview of local search developments in a vehicle routing context.

The following subsections refer to a sequence of consecutive nodes in a tour as a *string*. Strings may contain zero customers to, at most, all customers served by the tour. The number of customers included in a string is referred to as the *cardinality* of the string and denoted by $|string|$. A string's origin tour T is referred to as T-*string*. The string's direction is defined relative to its origin tour A which may be preserved (A-$string^+$), reversed (A-$string^-$) or arbitrary (A-$string$) (see Fig. 5a). The term string may be substituted by the term *head* (B-*head*) or *tail* (B-*tail*) when the string includes, respectively, the first or last customer served by the tour B (Fig. 5b).

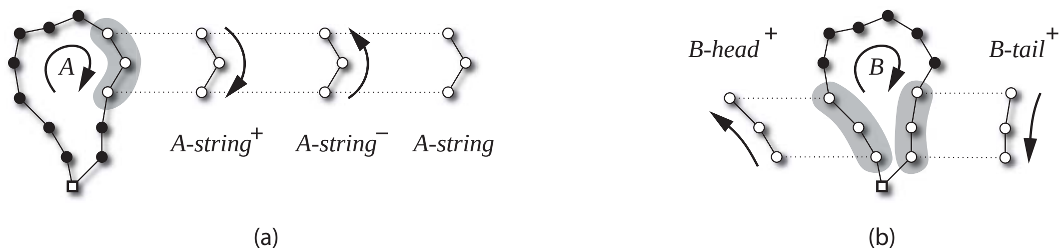

Figure 5 **String definitions.**

4.1. TSP neighborhoods

Optimising the TSP essentially implies reaching an optimum sequence for connecting an entire set of spatially-distributed nodes. Such a sequence cannot be optimum whenever an edge intersects another, as confirmed by Flood:

"There is one useful general theorem, which is quickly discovered by each one who considers the traveling-salesman problem. In the euclidean plane it states simply that the minimal tour does not intersect itself." (Flood 1956)

This theorem represents the foundation for the most powerful TSP heuristics, detailed in the following sections.

4.1.1. 2-opt (Croes 1958) One possible suggestion for remedying the optimality violation detailed by Flood is to locate such intersecting edges and reverse the direction of the intermediate nodes between these edges. The most basic implementation of such a suggestion corresponds to the well-known *2-opt* move (Croes 1958), which considers two non-consecutive edges. Fig. 6(a-b) illustrates a 2-opt move which removes two intersecting edges before reconnecting the nodes by two non-intersecting edges. Using the string terminology introduced earlier, we could interpret such a move as simply reversing the string's direction ($string^+ \rightarrow string^-$).

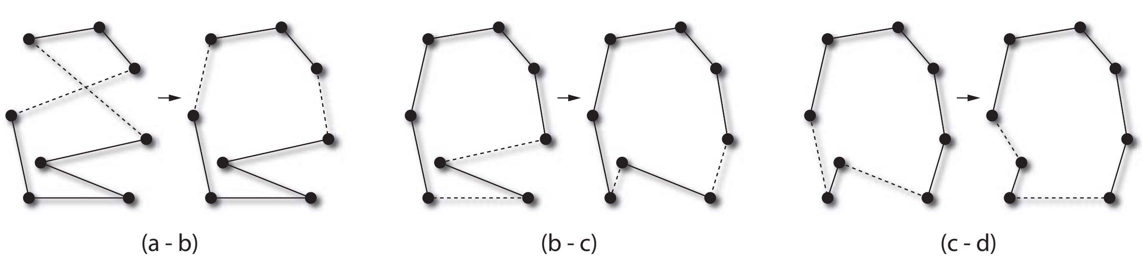

Figure 6 **Three 2-opt moves modifying a solution from (a) to (d).**

Although no pair of edges intersect in solution (b), solution (c) represents an improved neighbor in its 2-opt neighborhood. As with the $(b-c)$ move, (c) has no intersecting edges. The optimal solution (d) is obtained by a two-opt move applied to solution (c). Nevertheless, 2-opt does not guarantee generating the global optimum, as demonstrated by Fig. 7. Solution (a) has five neighbors in its 2-opt neighborhood while only three distinct solutions exist: (b), (c) and (d). Solution (a) has a tour length of 26 whereas neighbors (b), (c) and (d) have tour lengths of 26.06, 35.88 and 29.12 respectively. Therefore no improving solutions within the 2-opt neighborhood are available and solution (a) is said to be *2-optimal*. Metaheuristics may be employed to escape from this local optimum. Flood's theorem served as the foundation for the creation of the 2-opt move which is known to be reasonably effective with neighborhood sizes of $O(n^2)$, where n denotes the number of edges.

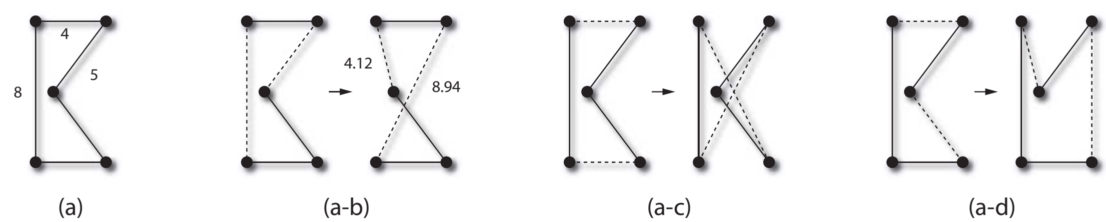

Figure 7 **A 2-optimal solution (a) without any improving neighbors (b, c and d).**

4.1.2. 3-opt (Lin 1965) The 2-opt move provides the basic principle for the *3-opt* move (Lin 1965), which considers three rather than two edges, thus enabling the exploration of a larger neighorhood. Fig. 8a reproduces the same solutions as Fig. 7a, proven 2-optimal by enumeration, and is consequently employed to illustrate 3-opt neighborhoods. Three edges are removed and replaced with others from the 3-opt neighborhood. If one of the removed edges were replaced by its original edge, it would result in a 2-opt neighbor. Due to this property, removing three consecutive edges in the current solution (a) always results in a 2-opt neighbor, given one of the three edges will always be replaced by its original edge. There are only five possibilities to remove three non-consecutive edges, of which only three - a_b, a_c and a_d - are symmetrically distinct. There exist three distinct possibilities for replacing the selected edges in (a_b). Only one option, (b), replaces all three edges with new ones, thus resulting in a neighbor not part of the 2-opt neighborhood. This situation also occurs in solutions (c) and (d).

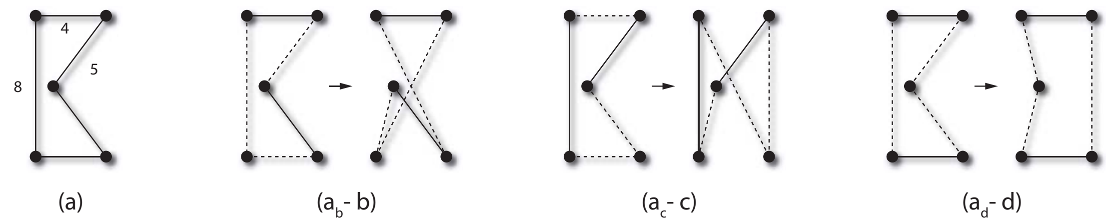

Figure 8 Three 3-opt neighbors (b, c and d) for the 2-optimal solution (a).

Evidently, it is possible to reach optimal solution d, which was not one of the 2-opt neighbors, using only improving moves. One should be aware, however, that exploring the 3-opt neighborhood increases the neighborhood size from $O(n^2)$ to $O(n^3)$.

4.1.3. k-opt (Lin and Kernighan 1973) and or-opt (Or 1976) The *k-opt* heuristic developed by Lin and Kernighan (1973) represents the most general *-opt*, wherein the value of k is deduced by a clever mechanism embedded within the move itself. *Or-opt* exchanges (Or 1976) are subset of the 3-opt which also consider three edges. However, or-opt iteratively relocates a 3-$string^+$ to another location until no further improvements are possible. The procedure is subsequently executed with a 2-$string^+$ and, finally, a 1-*string* (single node).

4.2. CVRP neighborhoods

Given the CVRP represents a generalisation of the TSP, all TSP heuristics may be applied to a single tour in CVRP solutions as *intra-route* neighborhoods which are employed as either independent neighborhoods, parts of compound moves or post-processing procedures.

4.2.1. 2-opt as an inter-route operator In the 2-opt neighborhood, as previously detailed, two edges are selected and the direction of the enclosed string reversed ($string^+ \rightarrow string^-$). In a CVRP solution, two edges from different tours A and B are considered, enabling 2-opt to operate as an inter-route neighborhood. The edge selected in tour A splits the tour into A-$head^+$ and A-$tail^+$ strings, the edge selected in tour B splits it into B-$head^+$ and B-$tail^+$ strings. There are now two possibilities insofar as reconnecting the resulting strings. In Fig. 9a-b A-$tail^+$ and B-$tail^+$ are swapped by connecting them to B-$head^+$ and A-$head^+$, respectively. In this case, no strings are reversed. The second option is shown in Fig. 9a-c. B's head string is reversed (B-$head^-$) and connected to A's head-string (A-$head^+$). The same process occurs for the tail-strings, with a-tail being connected to B-$tail^-$.

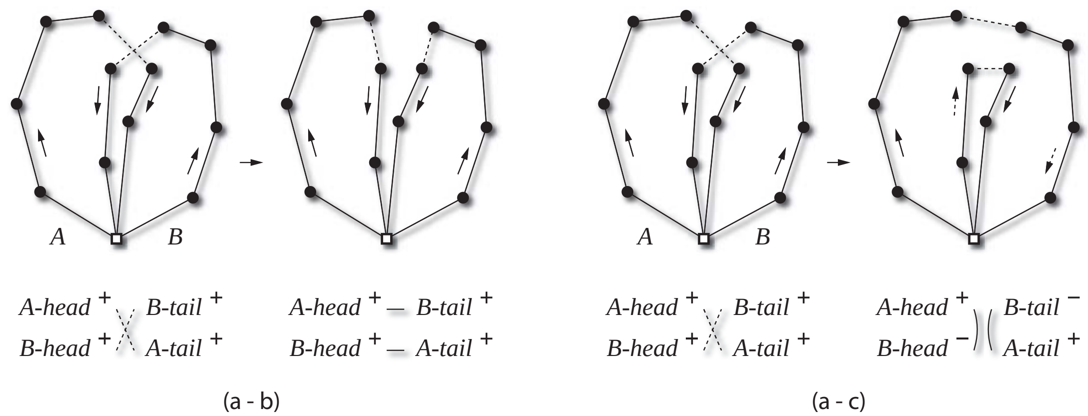

Figure 9 **Inter-route 2-opt move, string directions preserved (a-b) and reversed (a-c).**

4.2.2. Or-opt as an inter-route operator The or-opt neighborhood may be easily employed as a CVRP inter-route neighborhood by inserting the removed string into a different tour with sufficient capacity. However, when all vehicle capacities are completely utilized or all remaining capacities are less than the smallest demand, or-opt functions only as an inter-route neighborhood, provided customers are relocated to a new empty tour.

4.2.3. Single, double, pair+single and double pair procedures (Waters 1987) Greater capacity slack may be introduced into the solution by exploring other neighborhoods. Waters (1987) introduced single, double, pair+single and double pair procedures. These procedures remove one or two strings from the solution before reinserting them into optimal positions. The *single* procedure removes only one node (1-*string*) from the solution. The *double* procedure removes two 1-*string*s. The *pair+single* procedure removes both a 2-*string* (pair) and 1-*string* (single). Finally, two 2-*string*s are removed by the *double pair* procedure. The overall improvement method explores the two-opt neighborhood first, followed by the double pair, pair+single, double and finally the single neighborhood.

4.2.4. Chain-exchange (Fahrion and Wrede 1990) While Waters (1987) restricted the operation to string cardinalities of one and two, Fahrion and Wrede (1990) permitted two larger string cardinalities (M-*string* and P-*string*) through their *chain-exchange* procedure. String cardinalities M and P are bounded to be no more than one half of the average tour size as their research indicates that larger strings do not generally result in further improvements. Finally, as with Waters (1987), strings are reinserted into optimal positions.

4.3. VRPTW neighborhoods

The neighborhoods described in the previous section enable the reversal of the removed strings, a process which often results in time window infeasibilities for VRPTW solutions. Therefore, string order is usually preserved by the neighborhoods applied to VRPTWs.

4.3.1. Exchange, cross and relocate (Savelsbergh 1988) The *exchange*, *cross* and *relocate* neighborhoods are introduced in Savelsbergh's Ph.D. dissertation (Savelsbergh 1988) and related research paper (Savelsbergh 1992). These neighborhoods relocate strings between two tours A and B. *Exchange* simply swaps two strings: an A-*string*$^+$ is inserted at the B-*string*'s original location and vice-versa (Fig. 10a-b). *Cross* swaps the tails of two tours, equivalent to the inter-route extension of the two-opt neighborhood which preserves the string orders (Fig. 10c-d). Finally, *relocate* inserts an A-*string*$^+$ into another tour B (Fig. 10e-f). *Relocate* represents a generalization of the or-opt neighborhood given that the A-*string*'s cardinality is not limited to three.

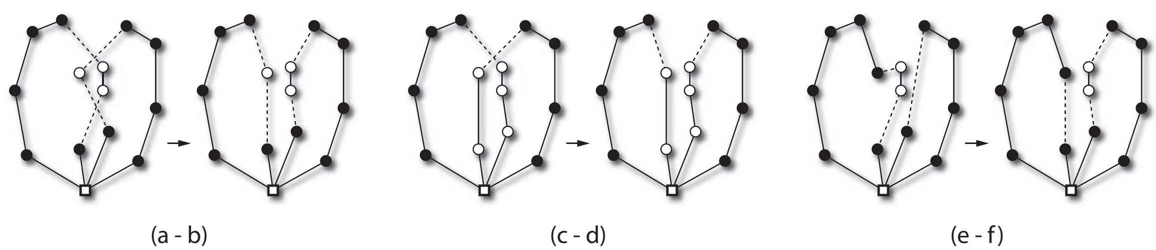

Figure 10 **Exchange (a-b), cross (c-d) and relocate (e-f).**

Neighborhoods are explored simultaneously, with the best neighbor in each iteration selected until no improvements are possible. A computational analysis of these neighborhoods is detailed by both Prosser and Shaw (1996) and Van Breedam (1994). These studies conclude that *relocate* improves solutions the most, and *exchange* the least.

4.3.2. 2-opt* (Potvin and Rousseau 1995) Equivalent to Savelsbergh's cross move, *2-opt** (Potvin and Rousseau 1995) generalized the 2-opt neighborhood into an inter-route neighborhood for the VRPTW, with the aim of preserving node order. Their first implementation applies 2-opt* until a 2-opt* optimal solution is found. Next, or-opt is applied to this solution until an or-opt optimal solution is found. The procedure is repeated until the optimum solution is achieved for both neighborhoods. The second implementation explores neighborhoods simultaneously. Results indicate their approaches perform well on VRPTW problems with tight time windows.

4.3.3. CROSS-exchange (Taillard et al. 1997) Taillard et al. (1997) combine Savelsbergh's inter-route *exchange*, *cross* and *relocate* neighborhoods in the *CROSS-exchange* neighborhood. They also enable *relocate* to operate within a single tour. NS is performed using a neighborhood reduction technique. An approximation matrix is employed which stores previous calculations, thus reducing computation time.

The following section will introduce ruin & recreate which enables one to interpret each of these classical neighborhoods in a more general manner.

5. Ruin & recreate strategies

The neighborhoods for VRP heuristics detailed throughout the previous section were defined via slight modifications of the incumbent solution. They therefore exhibit reasonable size complexity, implying a full neighborhood $\mathcal{N}(s)$ may be evaluated within a reasonable amount of computational time. Very large neighborhoods often occur when a significant part of the solution undergoes modification, making a full neighborhood exploration impractical. Therefore, neighbors may be selected by the NS from a reduced candidate set $\mathcal{C}(s) \subseteq \mathcal{N}(s)$.

Consider a simple NS which generates a single neighbor by first ruining the current solution before recreating it into a feasible one. The resulting neighbor is passed to the NA. Such an NS may be defined by a *ruin phase* $\mathcal{R}^-$ and *recreate phase* $\mathcal{R}^+$. This underlying concept is present in a variety of research papers. To our knowledge, it was established for the first time by Dees and Smith (1981) in their *Rip-Up and Reroute* strategies for wiring point-to-point connections in electronic design automation. Shaw (1998) introduced *Large Neighborhood Search* (LNS) wherein the ruin phase is implemented as the removal of related customers (related in terms of time, distance and being served by the same vehicle). A branch and bound technique optimally inserts the removed customers in the recreate phase. The term *Ruin & Recreate* (R&R) was introduced by Schrimpf et al. (2000) who applied the technique to a number of prominent problems including the TSP and VRPTW. They ruined solutions by removing either randomly selected customers, customers within a certain radius or consecutive customers in a single string. The solution is recreated by greedily reinserting the removed customers in a random order at minimum cost. Pisinger and Røpke (2007) defined several ruin & recreate methods which compete to modify the solution in their *Adaptive Large Neighborhood Search* (ALNS) framework. They applied their method to the Rich Pickup and Delivery Problem with Time Windows (RPDPTW) and implemented seven different strategies for selecting customers for removal: random, worst, related, cluster, time oriented, historical node-pair and historical request-pair. The solution is recreated using either greedy or regret insertion

(Potvin and Rousseau 1993) with or without noise function. An overview of these ruin & recreate methodological implementations is provided by Table 2.

Table 2 LNS, R&R and ALNS methodologies.

Shaw (1998) (LNS)			
$\mathcal{R}^- \rightarrow$	related removal	$\mathcal{R}^+ \rightarrow$	optimal insertion (branch and bound)
Schrimpf et al. (2000) (R&R)			
$\mathcal{R}^- \rightarrow$	random, radial or string removal	$\mathcal{R}^+ \rightarrow$	greedy insertion
Pisinger and Røpke (2007) (ALNS)			
$\mathcal{R}^- \rightarrow$	random, worst, related, cluster, time-oriented or historical removal	$\mathcal{R}^+ \rightarrow$	greedy or regret insertion with or without noise function

Recent heuristic development based on the ruin & recreate principle build upon the general ALNS framework by enlarging the set of ruin and insertion methods. Examples of such studies include, but are not limited to: Gilbert Laporte (2010), Ribeiro and Laporte (2012)

Given the generality of R&R, one may define the classical neighborhoods detailed in Section 4, as R&R strategies. Generally, all these neighborhoods are obtained by removing a number of strings ($\mathcal{R}^-$) and reinserting the, possibly reversed, strings into the solution at certain positions ($\mathcal{R}^+$). Evidently, recreating the solution via string insertions potentially yields fewer possibilities since it requires more free capacity in a single vehicle than inserting customers separately into multiple vehicles. Consequently, greedy insertion techniques are more likely to achieve feasible solutions by considering customers separately, rather than in the form of strings.

In distinct contrast to the recent trend of introducing more and more $\mathcal{R}^-$ and $\mathcal{R}^+$ methods, the present paper introduces a simplified yet powerful R&R implementation using a single $\mathcal{R}^-$ and $\mathcal{R}^+$ method: *adjacent string removal* and *greedy insertion with blinks* (ASB-RR), respectively. The $\mathcal{R}^-$ and $\mathcal{R}^+$ combines elements of classical neighborhoods and a modified version of greedy insertion methods employed in recent R&R heuristics (Table 3), with the significant additional benefit of being highly reproducible.

Table 3 Classical neighborhoods and ASB-RR methodologies.

Classical neighborhoods			
$\mathcal{R}^- \rightarrow$	string removal	$\mathcal{R}^+ \rightarrow$	string insertion
ASB-RR			
$\mathcal{R}^- \rightarrow$	adjacent string removal	$\mathcal{R}^+ \rightarrow$	greedy insertion with blinks

6. Adjacent String Removal & Greedy Insertion with Blinks

The following principle guides the implementation of the ASB-RR ruin & recreate phases: *"Customers should be removed in the ruin phase with the ambition of improving inter-route relocations in the recreate phase."* Furthermore, one should not only consider which customers are to be relocated but simultaneously also consider the consequent *capacity slack* and *spatial slack* engendered by their removal.

Spatial slack implies vehicles are free to serve all customers without incurring large detours or significantly greater travel distances. There exists vehicle flexibility with regard to which customers to serve. In essence, vehicles are not bound to a specific geographic region. *Spatial bond*, by contrast, implies travelling to certain customers would introduce significantly longer detours and, consequently, greater costs. There exists less freedom when choosing which customers to serve and their geographic location is a deciding criteria.

In Fig. 11 below, for example, (f) represents the ruined state with the most spatial slack. By contrast, ruined states (b) and (d) emerging from (a) and (c), respectively, continue to be bonded to the same regions.

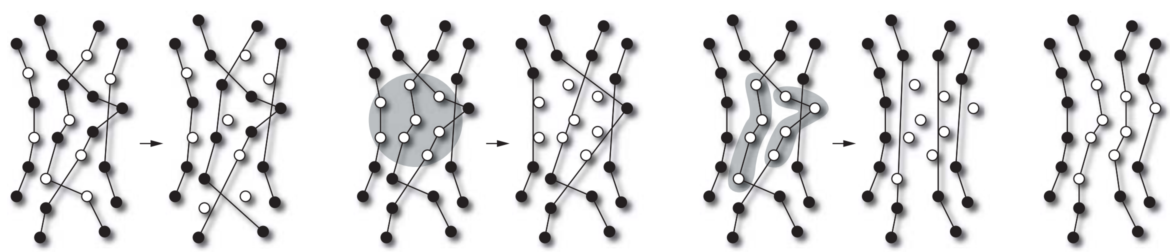

Figure 11 **Random (a-b), radial (c-d), adjacent string (e-f) removal and recreated state (g) which may emerge only from ruined state (f).**

This results in the following three propositions:

PROPOSITION 1. ***Remove a 'sufficient' number of customers***
A small number of removed customers (such as 1 to 5) fails to introduce sufficient capacity slack and subsequently ease customer relocations during the recreate phase.

PROPOSITION 2. ***Remove 'adjacent' customers***
Removing non-adjacent customers (randomly-selected customers such as in Fig. 11a-b), generally introduces capacity slack scattered across multiple tours. Removed customers are unlikely to

benefit from the capacity slack introduced by other removed customers which often results in single-customer relocations.

PROPOSITION 3. ***Remove adjacent 'strings'***
Customers which are only distance-adjacent (as in the case of radial removal, Fig. 11c-d) often introduce scattered capacity slack and minimal spatial slack if the removed customers are non-consecutive (d). An example of removing adjacent strings is depicted in Fig. 11e-f, the only ruined state from which (g) may emerge.

6.1. Ruin phase - Adjacent string removal

Proposition 1 states the ruin phase should remove a sufficient number of customers by removing adjacent strings. The number of strings k and the cardinality of these strings l may be selected randomly from the ranges $[K_{min}, K_{max}]$ and $[L_{min}, L_{max}]$, respectively. By fixing the average computational effort to an average number of removed customers $\bar{c}$, it is possible to investigate the influence of removing many strings of small cardinality or few strings of high cardinality. Parameter $\bar{c}$ may be expressed as a function of the average number of strings $\bar{k}$ and the average string length $\bar{l}$, as demonstrated in Eq. 1.

$$\bar{c} = \bar{k}.\bar{l} = \frac{K_{min} + K_{max}}{2}.\frac{L_{min} + L_{max}}{2} \qquad K, L \in \mathbb{N}_{>0} \qquad (1)$$

Solutions may consist of many short tours serving few customers or only a few long tours serving many customers, depending upon the specific problem instance. Therefore, it may not be possible to remove K_{max} strings if the solution contains fewer tours, or to remove strings of cardinality L_{max} in cases where all tours are short. Given that it is more likely that the number of served customers in a tour is smaller than the number of tours in a solution, especially for large instances. K_{max} is dynamically calculated while the values of $\bar{c}$, L_{min}, L_{max} and K_{min} are fixed input parameters, as per Eq. 2.

$$K_{max} = \frac{4.\bar{c}}{L_{min} + L_{max}} - K_{min} \qquad K_{max} \in \mathbb{R}_{>0} \qquad (2)$$

Given that the input parameter L_{max} may be too large in the current solution, an adjusted value l_{max}, limited by the average tour length $\overline{|t|}$ (Eq. 3), is used to calculate the adjusted value k_{max} (Eq. 4).

$$l_{max} = min(\overline{|t|}, L_{max}) \qquad l_{max} \in \mathbb{R}_{>0} \qquad (3)$$

$$k_{max} = \frac{4.\bar{c}}{L_{min} + l_{max}} - K_{min} \qquad k_{max} \in \mathbb{R}_{>0} \qquad (4)$$

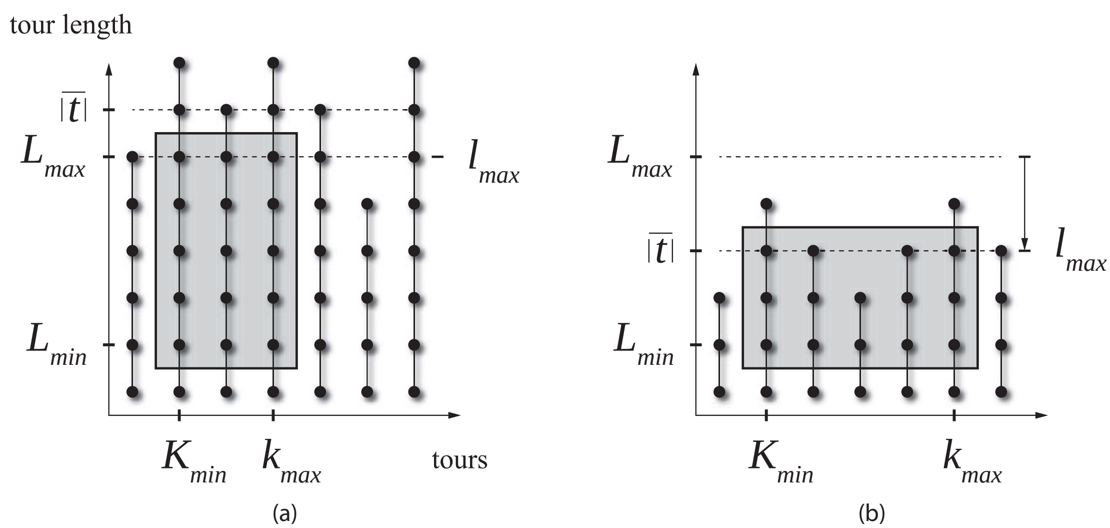

Figure 12 **Ranges for k and l when $|\bar{t}| > L_{max}$ (a) and $|\bar{t}| < L_{max}$ (b).**

Fig. 12 illustrates the adjustment of k_{max} according to the current solution. Assume the following parameter settings: $\bar{c} = 12$, $L_{min} = 2$, $L_{max} = 6$ and $K_{min} = 2$. The current solution's average tour cardinality $\overline{|t|}$ is equal to 7 (Fig. 12a), which is greater than L_{max}. Therefore l_{max} is set to 6 (Eq. 3) and k_{max} becomes 4 (Eq. 4). When the average tour cardinality $\overline{|t|}$ is smaller than L_{max} (4 and 6 respectively in Fig. 12b), l_{max} is set to 4 and k_{max} becomes 6. This enables the selection of a random integer value for k, the number of strings to be removed in the range $[K_{min}, k_{max}]$ by way of Eq. 5. Similarly, for each string, an integer value for its cardinality is randomly selected in the range $[L_{min}, l_{max,t}]$ by way of Eq. 7. Although l_{max} is adjusted for the current solution, its value may be too large according to a certain tour, as illustrated in Fig. 12. Therefore, the adjusted value $l_{max,t}$ (Eq. 6) is required by Eq. 7 to be no greater than the current tour cardinality $|t|$.

$$k = \lfloor\, U(\,[K_{min},\, k_{max} + 1[\,\,)\,\rfloor \qquad k \in \mathbb{N}_{>0} \tag{5}$$

$$l_{max,t} = min(\,|t|,\, l_{max}) \qquad l_{max,t} \in \mathbb{R}_{>0} \tag{6}$$

$$l_t = \lfloor\, U(\,[L_{min},\, l_{max,t} + 1[\,\,)\,\rfloor \qquad l_t \in \mathbb{N}_{>0} \tag{7}$$

The procedural steps of the ruin phase are given by Algorithm 1. First, values for l_{max}, k_{max} and k are calculated by employing the settings found in Table 4.

Table 4 **ASB-RR parameter settings for the Ruin method.**

$\bar{c} = 10$
An average of 10 removed customers has been proven 'sufficient' after exhaustive analysis. A figure of 10 implies the removal of at least 1 and at most 19 customers.
$L_{min} = 1,\ L_{max} = 10$
Removed string lengths should be between 1 and 10 since removing longer strings failed to improve final results. This confirms the observations of Fahrion and Wrede (1990) whose approach, while similar, limits string cardinality to half the average tour length.
$K_{min} = 1$
The minimum number of removed strings is set to 1. By employing Eq. 5 and the previous settings, the number of removed strings k_{max} will be in the range $[1, 3]$ for solutions with large tours ($\overline{\lvert t \rvert} \geq L_{max} = 10$) and $[1, 19]$ for solutions' tours exclusively of cardinality one ($\overline{\lvert t \rvert} = 1$). This situation almost always only occurs during the first iteration of the search (see Section 6.3)

Algorithm 1 ASB-RR - Ruin method

```
procedure REMOVECUSTOMERS(s)
    l_max, k_max, k ← calculate(Eq. 3,4,5)
    C ← ∅                                        ▷ Set of removed customers
    T ← ∅                                        ▷ Set of ruined tours
    c_seed ← randomCustomer(s)
    for c ∈ adj(c_seed) and |T| < k do           ▷ Increasing distance to seed
        if tour(c) ∉ T then
            l_t ← calculate(Eq. 7)
            C ← C ∪ removeSelected(c, l)
            T ← T ∪ tour(c)
        end if
    end for
    return C
end procedure
```

Removed customers are stored in set C and their original tour in set T. Adjacent strings are selected near a randomly chosen seed customer c_{seed}. For each customer $c \in P$ an adjacency list $adj(c)$ of all customers ordered by increasing distance from c is assumed to be available. The list

$adj(c)$ includes customer c as its first element. List $adj(c_{seed})$ is iterated over to select strings close to c_{seed}. If a customer's serving tour $t(c)$ has not yet been ruined ($t(c) \notin T$), customers are removed from $t(c)$ via the 'string' or 'split string' procedure. Thereafter the ruined tour is added to set T. This procedure is repeated until k strings are removed, implying $|T| = k$.

One should recall that all customers are iterated over by increasing distance from c_{seed} using $adj(c_{seed})$. This iteration continues until a customer c is encountered who is served by a not-yet-ruined tour $t(c)$. When such a customer is found, no customer served by $t(c)$ preceded c in the list $adj(c_{seed})$. All other customers succeed c in list $adj(c_{seed})$, indicating they are further from c_{seed} than c. Therefore, c represents the customer closest to c_{seed} out of all customers served by $t(c)$, and is denoted as $\breve{c}$.

The 'string' procedure removes a random string of length l which includes customer $\breve{c}$. Including $\breve{c}$ implies all removed strings are adjacent to c_{seed} and, consequently, adjacent to all other removed strings. In essence, only adjacent strings are removed. An example where $l = 3$ is illustrated in Fig. 13. One of the three possible strings is randomly selected for removal.

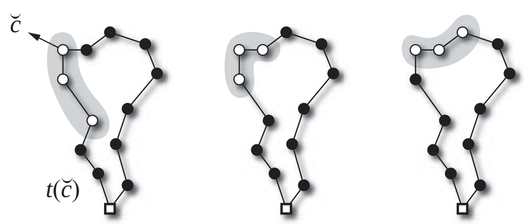

Figure 13 **Possible 'string' removals which include $\breve{c}$ from tour $t(\breve{c})$ when $l = 3$.**

The 'split string' procedure begins much like 'string', by randomly selecting a string of cardinality $l + m$ which includes customer $\breve{c}$ (Fig. 14a). However, the ruin phase bypasses and preserves a random substring of m intervening customers as shown in Fig. 14b. The number of preserved customers m is determined as follows. Initially $m = 1$ and the current value of m is maintained if a random number is smaller than α ($U([0,1[) < \alpha = 0.01$) or when the maximum value for m is reached ($m = |t(c)| - l$). If neither of these conditions is satisfied, m is incremented ($m = m + 1$) and the incrementation process repeats. This results in values for $m = 1, 2, 3, ..., m_{max}$ having respective probabilities $p = 1.00\%, 0.99\%, 0.98\%, ..., (100 - p_{m=1} - p_{m=2} - p_{m=3} - ... - p_{m=m_{max}-1})\%$.

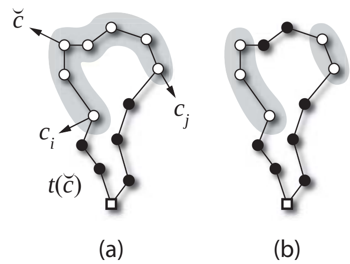

Figure 14 **'Split string' removal when $l = 5$ and $m = 2$.**

Both 'string' and 'split string' are executed with equal probabilities. Fig. 15 represents three distinct situations concerning how tours are ruined. In half of cases the 'string' procedure removes a string, introducing spatial slack (Fig. 15a-b), whereas in the other half tours are ruined by the 'split string' procedure. Furthermore, there exists a very small value for m, occurring with a very low probability, where the original tour is preserved (c-d). Otherwise $m = m_{max}$, where the removed customers are close to the depot (e-f).

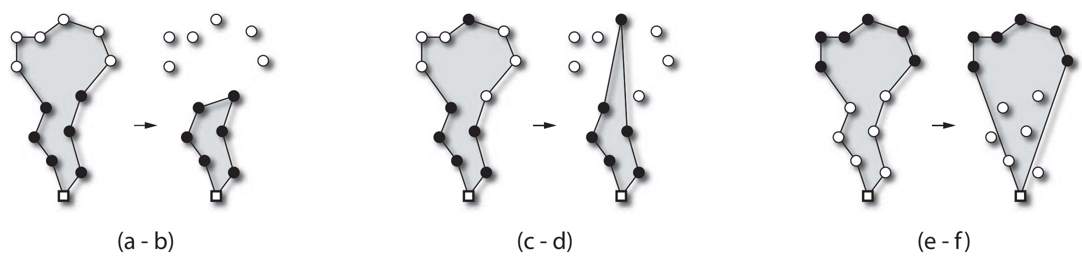

Figure 15 **'String' removal (a-b) and 'split sting' removal (c-d), (e-f).**

Fig. 16 presents an example of the ruin phase where two strings are removed. First, the seed customer c_{seed} is randomly selected (a). Following this, the list $adj(c_{seed})$, is iterated over. Since c_{seed} is always the first element in the list and no tours are ruined, $\check{c} = c_{seed}$. A string of length $l = 4$ is removed by the 'string' procedure which includes $\check{c}$ (b-c). In (d) $adj(c_{seed})$ is iterated over until the next customer is found who is served by a not-yet-ruined tour. The second string is also removed by the 'string' method and l is, coincidentally, 4 again (e). The final ruined state is illustrated in (f).

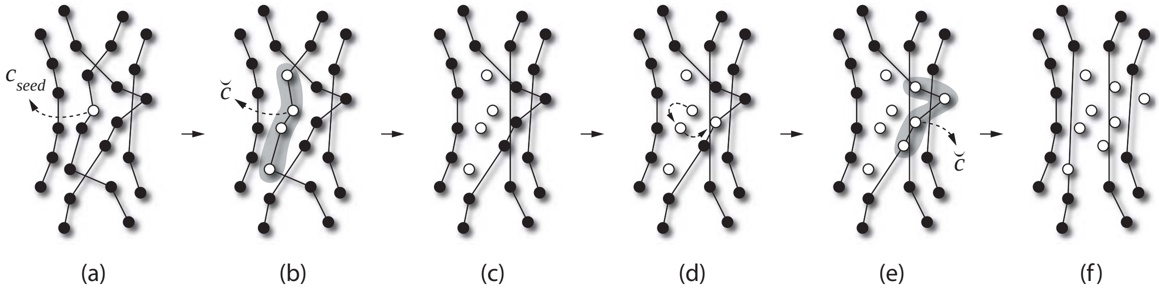

Figure 16 **An example of the ASB-RR ruin method..**

6.2. Recreate phase - Greedy insertion with blinks

The recreate phase is based on greedy insertion which sequentially inserts a set of customers C into a solution s (Algorithm 2). First, set C is sorted by one of the following orders: *Random*, *Demand*, *Far* or *Close*. 'Random' enables the set's insertion without any ordering. 'Demand' sorts customers by demand, placing those with the largest demand first. 'Far' inserts the most-distant customers from the depot first, thereby introducing spatial bond in the existing tours. Finally, 'Close' inserts customers closest to the depot first. Set C is sorted by Random, Demand, Far and Close by weights 4, 4, 2 and 1, respectively.

While basic greedy places each customer at the best position, one may deviate slightly from the best position. Pisinger and Røpke (2007) apply *greedy insertion with noise function* by adding a randomized noise term to the insertion cost. This paper, by contrast, introduces *greedy insertion with blinks*. Each customer $c \in C$ is inserted into solution s at the 'best' position P as follows. All current tours part of the solution are iterated over in a random order. When a tour t has enough capacity slack to serve c, all positions inside this tour, $P_t \in r$, are iterated over. Each position is evaluated with a probability of $1-\beta$, otherwise skipping the position as if the algorithm 'blinks'. If a position P_t is found for which the cost of inserting c is lower than the current best position P, P_t becomes the new best position ($P \leftarrow P_t$). If no position was found in existing tours, a new empty tour is created to serve c, otherwise c is inserted at P. $\beta = 0$ denotes that customers are always inserted at the best position, a strategy proven to be sub-optimal after experimentation. During the experiments reported in Section 7, it was observed that a blink rate of 1% ($\beta = 0.01$) was the most effective in terms of improving final solution quality. A result that may, at first, appear somewhat counter-intuitive.

Algorithm 2 ASB-RR - Recreate

procedure INSERTCUSTOMERS(C, s)
  $C \leftarrow sort(C)$
  for $c \in C$ **do**
    $P \leftarrow null$ ▷ Best insert position
    for $t \in s$ (which can serve c) **do**
      for P_t **in** t **do**
        if $U[0,1[< 1-\beta$ **then** ▷ 'Blink' sometimes
          if $costAt(P) < costAt(P_{best})$ **then**
            $P \leftarrow P_t$
          end if
        end if
      end for
    end for
    if $P = null$ **then**
      $t \leftarrow createEmptyTour()$
      $P \leftarrow positionIn(t)$
    end if
    insertCustomerAt(P)
  end for
end procedure

There exist multiple positions or options for inserting a customer when they are about to be inserted. Out of these options, one may make a random choice or utilize a heuristic to pick the best option (greedy insertion). The effect of blinks is reflected by the probability p of selecting a specific option based on its rank r. A blink rate of β implies the best option, which is ranked first $r = 1$, is selected with probability $p(1) = (1-\beta)$. If the best ranked option is blinked over, the probability of selecting the second best option is $p(2) = (1-\beta).\beta$. Selection probability of the third ranked option equals $p(3) = (1-\beta).\beta^2$. The selection probability for each rank is expressed by exponential function:

$$p(r) = (1-\beta).\beta^{(r-1)} \qquad r \in \{1, ..., \infty\} \tag{8}$$

Notice how the blinking algorithm itself is unaware of each option's rank, it only blinks with probability β while iterating over all options. Thus, blinking results in rank-based selection probabilities without requiring one to rank the options first, in contrast to *heuristic-biased stochastic* *sampling* (HBSS) (Bresina 1996), a closely related selection algorithm which is compared against the blinking algorithm in Appendix A.

6.3. Constructive method

The heuristic begins with the simplest one-to-one relationship: each vehicle serves only one customer (OTO-CM). Clearly, this assignment process represents the worst solution possible. No additional effort was dedicated to improving the CM since it was observed that the computational effort required for creating a higher-quality initial solution is better spent by the improvement method. Moreover, the OTO-CM solution is not biased towards any specific solution structure or customer clustering.

6.4. Simulated annealing

The ASB-RR neighborhood is guided by Simulated Annealing (see Section 3.1). The temperature of the system is lowered by the exponential cooling schedule: $T(i) = T_0.c^i$ where $c = T_1/T_0$. The search begins at an initial temperature $T_0 = 100$ and ends at the final temperature $T_0 = 1$. The number of iterations it is determined as a function of problem size v by linear interpolation as in Eq. 9. The minimum problem size is $v_{min} = 100$ and maximum $v_{max} = 1000$, as per Uchoa et al. (2014). The present study set $it(v_{min}) = 30.10^6$ and $it(v_{max}) = 300.10^6$, thus enabling direct comparison of the present paper's calculation times with those from the aforementioned paper.

$$it(v) = it(v_{min}) + \frac{it(v_{max}) - it(v_{min})}{v_{max} - v_{min}}(v - v_{min}) \tag{9}$$

7. Computational results

Uchoa et al. (2014) introduced a benchmark set since *"the existing sets became too easy, are too artificial or do not cover the wide range of characteristics found in real applications"*. The new benchmark set contains 100 instances where problem size ranges from 100 to 1000 customers, covering a wide variety of characteristics. The accompanying technical report includes the results of three state of the art methods (Table 5): BCP (Pecin et al. 2014), ILS-SP (Subramanian, Uchoa, and Ochi 2013) and UHGS (Vidal et al. 2014).

Table 5 State of the art methods.

BCP	Branch-Cut-and-Price (2014)	Exact
ILS-SP	Iterated Local Search with Set Partitioning (2013)	Heuristic
UHGS	Unified Hybrid Genetic Search (2014)	Heuristic

This study's experiments were performed on Xeon(R) CPU E5-2650 v2 @ 2.60GHz, and compared against the results of ILS-SP and UHGS, which were conducted on Xeon CPU @ 3.07 GHz. The full benchmark set is divided into three sets based on the problem size: Small, Medium and Large (Table 6).

Table 6 **Subsets of the complete benchmark.**

	n	Instances	# Instances
Small	100 – 250	X-n101-k25 – X-n247-k47	32
Medium	250 – 500	X-n251-k28 – X-n491-k59	36
Large	500 – 1000	X-n502-k39 – X-n1001-k43	32
Complete	100 – 1000	X-n101-k25 – X-n1001-k43	100

The ILS-SP, UHGS and ASB-RR heuristics were run 50 times. The average (Avg*), best (Best*) and the average computation time in minutes (T) from these obtained results are aggregated in Appendix B. The minimum (Min), maximum (Max), average (Avg), median (Median) and number of best (# Best) values from these aggregated results are presented in Table 7. Heuristics are ranked first (the best), second and third place by use of the colors dark gray, gray, and light gray respectively.

Table 7 **Summarized computational results.**

	ILS-SP			UHGS			ASB-RR		
	Avg*	Best*	T	Avg*	Best*	T	Avg*	Best*	T
Small (100-250)									
Min	0.00	0.00	0.1	0.00	0.00	1.4	0.00	0.00	0.2
Max	2.50	1.19	17.8	0.30	0.06	20.4	0.50	0.16	18.4
Avg	0.31	0.12	2.4	0.07	0.00	6.0	0.11	0.01	5.2
Median	0.19	0.00	1.6	0.03	0.00	5.4	0.05	0.00	5.0
# Best	9	18	25	22	30	0	8	24	7
Medium (250-500)									
Min	0.00	0.00	2.0	0.00	0.00	6.5	0.05	0.00	9.8
Max	1.69	0.89	60.6	0.58	0.21	86.7	1.35	0.28	58.4
Avg	0.56	0.20	23.1	0.27	0.04	30.3	0.22	0.04	27.0
Median	0.47	0.15	16.8	0.28	0.02	22.4	0.20	0.02	22.4
# Best	8	9	24	8	18	3	20	20	9
Large (500-1000)									
Min	0.00	0.00	27.3	0.11	0.00	33.1	0.06	0.00	60.9
Max	2.18	1.89	792.8	0.92	0.47	560.8	0.38	0.15	412.7
Avg	0.93	0.66	195.7	0.46	0.22	268.6	0.17	0.04	152.0
Median	0.94	0.56	144.7	0.43	0.18	258.6	0.16	0.03	144.8
# Best	1	2	15	1	2	2	30	29	15
Complete (100-1000)									
Min	0.00	0.00	0.1	0.00	0.00	1.4	0.00	0.00	0.2
Max	2.50	1.89	792.8	0.92	0.47	560.8	1.35	0.28	412.7
Avg	0.60	0.32	71.7	0.27	0.09	98.8	0.17	0.03	60.0
Median	0.47	0.15	17.7	0.26	0.03	22.4	0.15	0.01	22.4
# Best	18	29	64	31	50	5	58	73	31

UHGS outperforms the other methods for Small instances. For the Medium set, UHGS and ASB-RR perform equally well insofar as obtaining the best solutions (Best*). Additionally, it was observed that ASB-RR became the most robust method. Meanwhile, for the Large set of instances, ASB-RR outperforms both heuristics on all aspects - it is the most robust method which finds the best solutions in the least amount of computation time. These observations are supported by the results of the Wilcoxon rank-sum test. The average results obtained by ASB-RR (Avg*, Appendix B) are compared against the results of ILS-SP and UHGS in Table 8. When statistical significance exceeds 95%, cells are colored gray. The null-hypothesis (X=Y) is always rejected indicating ASB-RR significantly differs from both other heuristics. ASB-RR obtains better results (X>Y) except for the small benchmark instances, where UHGS represents the best approach.

Table 8 **P-values of the Wilcoxon rank-sum test.**

	X=Y	X>Y	X<Y
X=ILS-SP, Y=ASB-RR			
Small	0,026626	0,987409	0,013313
Medium	0,000052	0,999976	0,000026
Large	0,000001	1,000000	0,000000
Complete	0,000000	1,000000	0,000000
X=UHGS, Y=ASB-RR			
Small	0,024641	0,012320	0,988427
Medium	0,002555	0,998787	0,001278
Large	0,000001	1,000000	0,000000
Complete	0,000000	1,000000	0,000000

Detailed results are presented in Appendix B. The best known solution (BKS) is provided for each instance if proven optimal by the BCP method. Furthermore, ASB-RR method improved 36 BKSs and obtained 31 results equal to the BKS during the experiment of 50 runs. Throughout the entire experimental campaign, 40 BKSs were improved and 40 equal solutions were found which are marked dark and light gray respectively in the BKS column of Appendix B.

8. Conclusions

This paper introduced a single ruin and single recreate method - ASB-RR - which exhibited low computation times, robustness and high performance for large CVRP problems containing 500-1000 customers. Across all benchmark instances (Uchoa et al. 2014) ASB-RR was on average the most robust, high-quality solution method, improving 40 Best Known Solutions, while simultaneously averaging the shortest computation times.

Laporte (2009) provided much of the motivation behind this paper's original contribution when writing: *"There is, however, a sense that several of the most successful metaheuristics are over-engineered and one should now attempt to produce simple and flexible algorithms capable of handling a larger variety of constraints, even if this were to translate into a small loss in accuracy"*

Reapproaching ruin & recreate by shifting away from a perspective increasingly weighed down by a degree of entrenched complexity and its constituent highly-detailed approaches and instead returning to a more fundamental, low-level perspective can result in the adaptation of older technology to meet a more computationally demanding present and future. Indeed, despite Laporte's comment that such a shift is worthwhile even if a small loss in accuracy were incurred, this approach's results promisingly indicate that no such loss takes place.

The contemporaneous trend of adding more and more ruin methods and recreate methods, while often resulting in good quality or improving results, represents a more intellectual than practical endeavour. By examining the functionality of such approaches and distilling their essences into a single ruin & recreate method - ASB-RR - a previously challenging reproducibility is achieved while simultaneously producing, on average, better results. To reiterate, ASB-RR is not necessarily restricted to CVRP optimisation and may be easily adapted to a variety of other vehicle routing problems. Problems concerning, for example, time windows or pick-up and delivery, may also be successfully optimized via minimal algorithmic modification. ASB-RR demonstrates an impressive convergence speed which engenders an array of future real-world applications, thus providing a multitude of exciting avenues for future research.

Acknowledgments

Work funded by IWT 130855 grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen) in cooperation with Conundra (www.conundra.eu), and supported by the Belgian Science Policy Office (BELSPO) in the Interuniversity Attraction Pole COMEX (http://comex.ulb.ac.be). Statistical advice provided by Wim Vancroonenburg (KU Leuven), editorial consultation provided by Luke Connolly (KU Leuven).

Appendix A: Blinks and HBSS

Iterative sampling (Langley 1992) builds an empty solution into a complete solution, which represents a 'sample' of the solution space, by incrementally making random decisions at each decision point. For example when applied to VRP, an unserved customer is randomly selected and inserted at a random position in the partial solution until all customers have been inserted. By contrast, decisions may be made based solely on a specific heuristic. As such, an unserved customer is inserted at the best position by *greedy insertion*. Meanwhile, in *Heuristic-biased Stochastic Sampling* (HBSS) (Bresina 1996), the heuristic's preferred options are applied with a certain probability as follows. The heuristic ranks all R options at a decision point. A rank-based weight $bias(r)$ is assigned to each option by way of a specific *bias function*. The probability $p(r)$ of selecting an option based on its rank r is obtained by dividing its weight by the sum of all option weights:

$$p(r) = \Big(\sum_{k=1}^{R} bias(k)\Big)^{-1} bias(r) \qquad\qquad r \in \{1, ..., R\} \qquad (10)$$

Table 9 presents the rank-based probabilities obtained by the following bias functions: constant, logarithmic, linear, cubic and exponential. For each bias function, the left column represents $p(r)$ assuming there exist only five options ($R = 5$) at the considered decision point. Thirty options are assumed ($R = 30$) within the right hand column and the remaining probability of selecting one of the options $r = 6..30$ is detailed in the last row.

Table 9 HBSS rank selection probabilities.

	Constant		Logarithmic		Linear		Cubic		Exponential	
r	1		$log^{-1}(r+1)$		r^{-1}		r^{-3}		e^{-r}	
1	0.200	0.033	0.339	0.109	0.438	0.250	0.843	0.832	0.636	0.632
2	0.200	0.033	0.214	0.069	0.219	0.125	0.105	0.104	0.234	0.233
3	0.200	0.033	0.170	0.055	0.146	0.083	0.031	0.031	0.086	0.086
4	0.200	0.033	0.146	0.047	0.109	0.063	0.013	0.013	0.032	0.031
5	0.200	0.033	0.131	0.042	0.088	0.050	0.007	0.007	0.012	0.012
6-30	-	0.833	-	0.678	-	0.429	-	0.013	-	0.006

Clearly, probabilities are dependent upon the number of options. With regard to the logarithmic bias function, the probability $p(r)$ of selecting the best option $r = 1$ is three times smaller when thirty options are available (presented in the right column) compared to when only five options are available (presented left). This influence is negligible when the bias function quickly converges to zero and the partial sum $\sum_{k=1}^{R} bias(k)$ converges to the infinite sum ($R = \infty$). This occurs for strong bias functions, such as the cubic and exponential bias function.

In contrast to HBSS, *greedy insertion with blinks*, introduced by the present paper in Section 6.2, does not require one to explicitly rank the options in advance to obtain rank-based selection probabilities. Nevertheless, the exponential rank-based probability function of the blinking algorithm (Eq. 8 and 11) may be expressed as HBSS with exponential bias function as follows:

$$p(r) = (1-\beta)\ \beta^{(r-1)} \qquad r \in \{1,...,\infty\} \tag{11}$$

$$= \left(\frac{1-\beta}{\beta}\right)\beta^r \tag{12}$$

$$= \left(\frac{\beta}{1-\beta}\right)^{-1}\beta^r \tag{13}$$

Based on the sum of an infinite geometric sequence $\sum_{k=0}^{\infty} ab^k = \frac{a}{1-b}$ for $|b|<1$ and $a=b$:

$$= \left(\sum_{k=0}^{\infty}\beta^{(k+1)}\right)^{-1}\beta^r \tag{14}$$

$$= \left(\sum_{k=1}^{\infty}\beta^{k}\right)^{-1}\beta^r \tag{15}$$

which allows the blinking algorithm's probability function (Eq. 11) to be expressed as HBSS (Eq. 10) with exponential bias function ($bias(r) = \beta^r$) when there exists an inifinite number of options ($R=\infty$):

$$p(r) = \left(\sum_{k=1}^{R} bias(k)\right)^{-1} bias(r) \qquad \begin{cases} r \in \{1,...,R\} \\ R=\infty \\ bias(r) = \beta^r = \left(\frac{1}{\beta}\right)^{-r} \end{cases} \tag{16}$$

The condition $R=\infty$ implies HBSS bias weights should be divided by the infinite sum $\sum_{k=1}^{\infty} bias(k)$, a value approximated by the partial sum for strong bias functions. For example, HBSS probabilities with $bias(r) = e^{-r}$ are equal to those obtained by the blinking algorithm when $\beta = \frac{1}{e}$, while the blink rate $\beta = 0.01$ (applied in Section 6.2) is equal to HBSS with $bias(r) = 100^{-r}$. These results are presented below in Table 10.

Table 10 **Comparison of rank selection probabilities for HBSS and Blinks.**

	HBSS: $bias(r)$			Blinks: β	
r	e^{-r}	100^{-r}	r	$\frac{1}{e}$	0.010
1	0.632	0.990	1	0.632	0.990
2	0.233	0.010	2	0.233	0.010
3	0.086	0.000	3	0.086	0.000
4	0.031	0.000	4	0.031	0.000
5	0.012	0.000	5	0.012	0.000
6-30	0.006	0.000	6-∞	0.006	0.000

Appendix B: Detailed results

Table 11 Aggregated computational results (Small).

		BKS			BCP		ILS-SP				UHGS				ASB-RR			
#	Name	BKS	nv	Opt	RLB	T	Gap	Avg*	Best*	T	Gap	Avg*	Best*	T	Gap	Avg*	Best*	T
1	X-n101-k25	27591	26	✓	27591	0.1	0.00	27591.0	27591	0.1	0.00	27591.0	27591	1.4	0.00	27591.0	27591	0.8
2	X-n106-k14	26362	14	✓	26362	3.5	0.05	26375.9	26362	2.0	0.08	26381.8	26378	4.0	0.07	26381.5	26362	1.3
3	X-n110-k13	14971	13	✓	14971	0.3	0.00	14971.0	14971	0.2	0.00	14971.0	14971	1.6	0.00	14971.1	14971	1.0
4	X-n115-k10	12747	10	✓	12747	2.1	0.00	12747.0	12747	0.2	0.00	12747.0	12747	1.8	0.00	12747.0	12747	0.2
5	X-n120-k6	13332	6	✓	13234	88.1	0.04	13337.6	13332	1.7	0.00	13332.0	13332	2.3	0.00	13332.0	13332	1.6
6	X-n125-k30	55539	30	✓	55539	2.5	0.24	55673.8	55539	1.4	0.01	55542.1	55539	2.7	0.03	55556.3	55542	3.1
7	X-n129-k18	28940	18	✓	28897	2.5	0.20	28998.0	28948	1.9	0.03	28948.5	28940	2.7	0.03	28948.8	28940	1.5
8	X-n134-k13	10916	13	✓	10840	399.1	0.29	10947.4	10916	2.1	0.17	10934.9	10916	3.3	0.22	10940.1	10916	2.8
9	X-n139-k10	13590	10	✓	13590	17.0	0.10	13603.1	13590	1.6	0.00	13590.0	13590	2.3	0.04	13595.4	13590	2.0
10	X-n143-k7	15700	7	✓	15634	1553.0	0.29	15745.2	15726	1.6	0.00	15700.2	15700	3.1	0.04	15705.8	15700	2.1
11	X-n148-k46	43448	47	✓	43448	0.3	0.01	43452.1	43448	0.8	0.00	43448.0	43448	3.2	0.05	43469.2	43448	2.8
12	X-n153-k22	21220	23	✓	21140	37.7	0.85	21400.0	21340	0.5	0.03	21226.3	21220	5.5	0.04	21229.4	21220	5.6
13	X-n157-k13	16876	13	✓	16876	1.0	0.00	16876.0	16876	0.8	0.00	16876.0	16876	3.2	0.02	16878.6	16876	3.7
14	X-n162-k11	14138	11	✓	14053	187.0	0.16	14160.1	14138	0.5	0.02	14141.3	14138	3.3	0.13	14157.1	14138	3.4
15	X-n167-k10	20557	10	✓	20476	1024.0	0.25	20608.7	20562	0.9	0.03	20563.2	20557	3.7	0.02	20560.8	20557	3.2
16	X-n172-k51	45607	53	✓	45549	3.8	0.02	45616.1	45607	0.6	0.00	45607.0	45607	3.8	0.03	45619.2	45607	5.3
17	X-n176-k26	47812	26	✓	47721	9.2	0.92	48249.8	48140	1.1	0.30	47957.2	47812	7.6	0.08	47849.6	47812	5.2
18	X-n181-k23	25569	23	✓	25511	18.2	0.01	25571.5	25569	1.6	0.09	25591.1	25569	6.3	0.04	25579.8	25569	5.5
19	X-n186-k15	24145	15	✓	23980	7305.0	0.17	24186.0	24145	1.7	0.01	24147.2	24145	5.9	0.14	24178.4	24149	4.0
20	X-n190-k8	16980	8	✓	16939		0.96	17143.1	17085	2.1	0.05	16987.9	16980	12.1	0.03	16984.9	16980	9.1
21	X-n195-k51	44225	53	✓	44225	2.4	0.02	44234.3	44225	0.9	0.04	44244.1	44225	6.1	0.17	44298.5	44241	6.1
22	X-n200-k36	58578	36	✓	58455	901.7	0.20	58697.2	58626	7.5	0.08	58626.4	58578	8.0	0.10	58636.1	58578	6.7
23	X-n204-k19	19565	19	✓	19484	501.6	0.31	19625.2	19570	1.1	0.03	19571.5	19565	5.3	0.50	19662.3	19565	4.9
24	X-n209-k16	30656	16	✓	30480	1303.0	0.36	30765.4	30667	3.8	0.08	30680.4	30656	8.6	0.04	30669.4	30656	6.0
25	X-n214-k11	10856	11		10809		2.50	11126.9	10985	2.3	0.20	10877.4	10856	10.2	0.48	10908.6	10873	8.7
26	X-n219-k73	117595	73	✓	117595	0.5	0.00	117595.0	117595	0.8	0.01	117604.9	117595	7.7	0.05	117650.4	117595	8.0
27	X-n223-k34	40437	34	✓	40311	303.5	0.24	40533.5	40471	8.5	0.15	40499.0	40437	8.3	0.23	40529.9	40448	7.6
28	X-n228-k23	25742	23	✓	25657	252.3	0.21	25795.8	25743	2.4	0.14	25779.3	25742	9.8	0.19	25790.9	25744	10.5
29	X-n233-k16	19230	17		19070		0.55	19336.7	19266	3.0	0.30	19288.4	19230	6.8	0.21	19269.7	19232	8.1
30	X-n237-k14	27042	14	✓	26930	1398.0	0.14	27078.8	27042	3.5	0.09	27067.3	27042	8.9	0.18	27089.7	27042	7.2
31	X-n242-k48	82751	48	✓	82589	819.0	0.15	82874.2	82774	17.8	0.24	82948.7	82804	12.4	0.16	82884.4	82775	9.9
32	X-n247-k47	37274	51	✓	37256	9.5	0.63	37507.2	37289	2.1	0.03	37284.4	37274	20.4	0.13	37323.2	37274	18.4

Table 12 Aggregated computational results (Medium).

		BKS			BCP		ILS-SP				UHGS				ASB-RR			
#	Name	BKS	nv	Opt	RLB	T	Gap	Avg*	Best*	T	Gap	Avg*	Best*	T	Gap	Avg*	Best*	T
33	X-n251-k28	38684	28	✓	38473	4767.0	0.40	38840.0	38727	10.8	0.29	38796.4	38699	11.7	0.28	38791.0	38687	9.8
34	X-n256-k16	18880	17	✓	18826	1255.0	0.02	18883.9	18880	2.0	0.00	18880.0	18880	6.5	0.05	18888.9	18880	11.5
35	X-n261-k13	26558	13		26407		1.17	26869.0	26706	6.7	0.27	26629.6	26558	12.7	0.32	26642.3	26558	11.8
36	X-n266-k58	75478	58	✓	75350	150.1	0.11	75563.3	75478	10.0	0.37	75759.3	75517	21.4	0.19	75617.8	75478	10.8
37	X-n270-k35	35291	36	✓	35156	3422.0	0.21	35363.4	35324	9.1	0.22	35367.2	35303	11.2	0.20	35362.2	35323	11.4
38	X-n275-k28	21245	28	✓	21245	3.7	0.05	21256.0	21245	3.6	0.17	21280.6	21245	12.0	0.11	21268.6	21245	13.3
39	X-n280-k17	33503	17		33286		0.80	33769.4	33624	9.6	0.31	33605.8	33505	19.1	0.37	33628.1	33529	17.7
40	X-n284-k15	20226	15		20139		1.10	20448.5	20295	8.6	0.30	20286.4	20227	19.9	0.30	20286.6	20240	15.3
41	X-n289-k60	95151	61	✓	94928		0.31	95450.6	95315	16.1	0.33	95469.5	95244	21.3	0.21	95352.2	95233	14.3
42	X-n294-k50	47167	51		46911		0.19	47254.7	47190	12.4	0.20	47259.0	47171	14.7	0.23	47274.5	47210	14.7
43	X-n298-k31	34231	31	✓	34105	531.1	0.37	34356.0	34239	6.9	0.18	34292.1	34231	10.9	0.13	34276.0	34234	14.5
44	X-n303-k21	21744	21		21546		0.70	21895.8	21812	14.2	0.49	21850.9	21748	17.3	0.15	21776.5	21751	17.3
45	X-n308-k13	25859	13		25587		0.94	26101.1	25901	9.5	0.14	25895.4	25859	15.3	1.35	26207.7	25931	25.7
46	X-n313-k71	94044	72		93851		0.27	94297.3	94192	17.5	0.24	94265.2	94093	22.4	0.15	94182.4	94063	18.9
47	X-n317-k53	78355	53	✓	78334	4.6	0.00	78356.0	78355	8.6	0.04	78387.8	78355	22.4	0.05	78392.4	78355	22.0
48	X-n322-k28	29848	28		29722		0.48	29991.3	29877	14.7	0.36	29956.1	29870	15.2	0.27	29927.6	29849	16.9
49	X-n327-k20	27546	20		27378		0.97	27812.4	27599	19.1	0.30	27628.2	27564	18.2	0.31	27631.4	27608	21.6
50	X-n331-k15	31102	15	✓	31027		0.43	31235.5	31105	15.7	0.19	31159.6	31103	24.4	0.08	31128.2	31122	20.4
51	X-n336-k84	139135	86		138706		0.23	139461.0	139197	21.4	0.29	139534.9	139210	38.0	0.17	139373.4	139209	22.8
52	X-n344-k43	42068	43		41881		0.51	42284.0	42146	22.6	0.33	42208.8	42099	21.7	0.22	42158.5	42079	21.5
53	X-n351-k40	25928	41		25809		0.86	26150.3	26021	25.2	0.33	26014.0	25946	33.7	0.21	25982.1	25938	26.5
54	X-n359-k29	51505	29		51381		1.11	52076.5	51706	48.9	0.42	51721.7	51509	34.9	0.14	51577.8	51505	23.1
55	X-n367-k17	22814	17		22747		0.83	23003.2	22902	13.1	0.11	22838.4	22814	22.0	0.09	22833.4	22814	36.1
56	X-n376-k94	147713	94	✓	147713	3.3	0.00	147713.0	147713	7.1	0.03	147750.2	147717	28.3	0.05	147783.6	147721	32.0
57	X-n384-k52	65943	53		65681		0.65	66372.5	66116	34.5	0.50	66270.2	66081	40.2	0.25	66107.4	65963	25.9
58	X-n393-k38	38269	38		38167		0.49	38457.4	38298	20.8	0.28	38374.9	38269	28.6	0.33	38394.1	38331	30.4
59	X-n401-k29	66187	29		65971		0.80	66715.1	66453	60.4	0.27	66365.4	66243	49.5	0.09	66248.5	66189	38.0
60	X-n411-k19	19718	19		19640		1.20	19954.9	19792	23.8	0.13	19743.8	19718	34.7	0.26	19768.5	19731	58.4
61	X-n420-k130	107798	130	✓	107704	115.0	0.04	107838.0	107798	22.2	0.12	107924.1	107798	53.2	0.08	107879.2	107817	47.9
62	X-n429-k61	65483	62		64930		0.40	65746.6	65563	38.2	0.25	65648.5	65501	41.5	0.17	65593.6	65485	35.0
63	X-n439-k37	36391	37	✓	36289		0.14	36441.6	36395	39.6	0.17	36451.1	36395	34.5	0.23	36473.8	36426	42.1
64	X-n449-k29	55269	29		54928		1.69	56204.9	55761	59.9	0.51	55553.1	55378	64.9	0.26	55411.2	55272	38.0
65	X-n459-k26	24173	26		23931		1.20	24462.4	24209	60.6	0.41	24272.6	24181	42.8	0.29	24242.2	24175	56.5
66	X-n469-k138	221909	140		221429		0.12	222182.0	221909	36.3	0.32	222617.1	222070	86.7	0.14	222227.1	221984	48.0
67	X-n480-k70	89458	70		89235		0.46	89871.2	89694	50.4	0.34	89760.1	89535	67.0	0.11	89559.2	89458	50.5
68	X-n491-k59	66510	59		66263		1.08	67226.7	66965	52.2	0.58	66898.0	66633	71.9	0.20	66645.5	66517	51.4

Table 13 **Aggregated computational results (Large).**

		BKS			BCP		ILS-SP				UHGS				ASB-RR			
#	Name	BKS	nv	Opt	RLB	T	Gap	Avg*	Best*	T	Gap	Avg*	Best*	T	Gap	Avg*	Best*	T
69	X-n502-k39	69230	39		69120		0.17	69346.8	69284	80.8	0.14	69328.8	69253	63.6	0.06	69274.7	69243	60.9
70	X-n513-k21	24201	21		24053		0.96	24434.0	24332	35.0	0.40	24296.6	24201	33.1	0.38	24292.1	24238	77.1
71	X-n524-k137	154594	155	✓	154533	212.1	0.27	155005.0	154709	27.3	0.25	154979.5	154774	80.7	0.14	154807.2	154651	151.4
72	X-n536-k96	94988	96		94409		0.75	95700.7	95524	62.1	0.36	95330.6	95122	107.5	0.20	95173.2	95006	74.7
73	X-n548-k50	86701	50		86604		0.20	86874.1	86710	64.0	0.34	86998.5	86822	84.2	0.11	86798.0	86710	64.5
74	X-n561-k42	42722	42		42495		0.96	43131.3	42952	68.9	0.34	42866.4	42756	60.6	0.34	42868.1	42774	73.8
75	X-n573-k30	50719	30		50575		0.90	51173.0	51092	112.0	0.39	50915.1	50780	188.2	0.17	50804.6	50737	113.0
76	X-n586-k159	190423	159		189950		0.26	190919.0	190612	78.5	0.22	190838.0	190543	175.3	0.09	190600.7	190484	86.3
77	X-n599-k92	108490	93		108000		0.82	109384.0	109056	73.0	0.53	109064.2	108813	125.9	0.18	108688.6	108548	75.4
78	X-n613-k62	59556	62		59323		1.49	60444.2	60229	74.8	0.68	59960.0	59778	117.3	0.29	59731.3	59585	88.1
79	X-n627-k43	62210	43		62018		1.12	62905.6	62783	162.7	0.50	62524.1	62366	239.7	0.17	62317.1	62219	89.3
80	X-n641-k35	63737	35		63228		1.36	64606.1	64462	140.4	0.71	64192.0	63839	158.8	0.18	63850.3	63750	92.5
81	X-n655-k131	106780	131	✓	106766	41.5	0.00	106782.0	106780	47.2	0.11	106899.1	106829	150.5	0.06	106844.6	106813	109.6
82	X-n670-k126	146451	133		146211		0.84	147676.0	147045	61.2	0.53	147222.7	146705	264.1	0.18	146720.4	146451	198.9
83	X-n685-k75	68261	75		67925		1.07	68988.2	68646	73.8	0.58	68654.1	68425	156.7	0.16	68369.0	68271	135.1
84	X-n701-k44	81934	44		81694		1.35	83042.2	82888	210.1	0.68	82487.4	82293	253.2	0.16	82065.4	81974	122.5
85	X-n716-k35	43414	35		43113		1.75	44171.6	44021	225.8	0.52	43641.4	43525	264.3	0.16	43483.8	43426	158.3
86	X-n733-k159	136250	160		135748		0.58	137045.0	136832	111.6	0.25	136587.6	136366	244.5	0.10	136389.3	136255	143.2
87	X-n749-k98	77365	98		76924		1.18	78275.9	77952	127.2	0.65	77864.9	77715	313.9	0.19	77509.2	77380	146.3
88	X-n766-k71	114525	71		114108		1.06	115738.0	115443	242.1	0.54	115147.9	114683	383.0	0.21	114761.1	114590	174.4
89	X-n783-k48	72445	48		71728		1.76	73722.9	73447	235.5	0.78	73009.6	72781	269.7	0.30	72660.7	72492	170.2
90	X-n801-k40	73331	40		73124		0.92	74005.7	73830	432.6	0.55	73731.0	73587	289.2	0.14	73436.7	73347	137.1
91	X-n819-k171	158267	172		157558		0.73	159425.0	159164	148.9	0.40	158899.3	158611	374.3	0.10	158423.0	158305	172.5
92	X-n837-k142	193813	142		193245		0.63	195027.0	194804	173.2	0.34	194476.5	194266	463.4	0.08	193976.9	193824	166.8
93	X-n856-k95	89007	95		88839		0.30	89277.6	89060	153.7	0.26	89238.7	89118	288.4	0.14	89131.3	89050	160.0
94	X-n876-k59	99331	59		98880		1.09	100417.0	100177	409.3	0.56	99884.1	99715	495.4	0.15	99483.2	99388	217.4
95	X-n895-k37	53946	37		53147		1.88	54958.5	54713	410.2	0.92	54439.8	54172	321.9	0.26	54085.8	53993	212.5
96	X-n916-k207	329247	207		328588		0.52	330948.0	330639	226.1	0.29	330198.3	329836	560.8	0.08	329509.5	329299	215.3
97	X-n936-k151	132926	158		132496		1.21	134530.0	133592	202.5	0.44	133512.9	133140	531.5	0.14	133117.3	133014	412.7
98	X-n957-k87	85482	87		85328		0.53	85936.6	85697	311.2	0.40	85822.6	85672	432.9	0.16	85620.0	85546	202.4
99	X-n979-k58	119008	58		118399		1.05	120253.0	119994	687.2	0.42	119502.1	119194	554.0	0.09	119120.4	119065	276.6
100	X-n1001-k43	72404	43		71812		2.18	73985.4	73776	792.8	0.76	72956.0	72742	549.0	0.17	72528.1	72415	284.3

References

Baldacci R, Christofides N, Mingozzi A, 2008 *An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. Mathematical Programming* 115(2):351–385.

Bresina JL, 1996 *Heuristic-biased stochastic sampling. AAAI/IAAI, Vol. 1*, 271–278.

Crainic TG, Toulouse M, 2003 *Handbook of Metaheuristics*, chapter Parallel Strategies for Meta-Heuristics, 475–513 (Boston, MA: Springer US).

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

Dantzig GB, Ramser JH, 1959 *The truck dispatching problem. Management Science* 6(1):80–91.

Dees W, Smith I RJ, 1981 *Performance of interconnection rip-up and reroute strategies. Design Automation, 1981. 18th Conference on*, 382–390.

Fahrion R, Wrede M, 1990 *On a principle of chain-exchange for vehicle-routeing problems (1-vrp). The Journal of the Operational Research Society* 41(9):821–827.

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

Fukasawa R, Longo H, Lysgaard J, Aragão MPd, Reis M, Uchoa E, Werneck RF, 2006 *Robust branch-and-cut-and-price for the capacitated vehicle routing problem. Mathematical Programming* 106(3):491–511.

Gilbert Laporte FV Roberto Musmanno, 2010 *An adaptive large neighbourhood search heuristic for the capacitated arc-routing problem with stochastic demands. Transportation Science* 44(1):125–135.

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

Kirkpatrick S, Gelatt CD, Vecchi MP, 1983 *Optimization by simulated annealing. SCIENCE* 220(4598):671–680.

Langley P, 1992 *Systematic and nonsystematic search strategies. Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, 145–152 (San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.).

Laporte G, 2009 *Fifty years of vehicle routing. Transportation Science* 43(4):408–416.

Lin S, 1965 *Computer solutions of the traveling salesman problem. Bell System Technical Journal* 44(10):2245–2269.

Lin S, Kernighan BW, 1973 *An effective heuristic algorithm for the traveling-salesman problem. Operations Research* 21(2):498–516.

Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E, 1953 *Equation of state calculations by fast computing machines. The Journal of Chemical Physics* 21(6):1087–1092.

Or I, 1976 *Traveling Salesman-Type Combinatorial Problems and their relation to the Logistics of Regional Blood Banking* (Xerox University Microfilms).

Pecin D, Pessoa A, Poggi M, Uchoa E, 2014 *Improved Branch-Cut-and-Price for Capacitated Vehicle Routing*, 393–403 (Cham: Springer International Publishing).

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

Potvin JY, Rousseau JM, 1993 *A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. European Journal of Operational Research* 66(3):331–340.

Potvin JY, Rousseau JM, 1995 *An exchange heuristic for routeing problems with time windows. The Journal of the Operational Research Society* 46(12):1433–1446.

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

Prosser P, Shaw P, 1996 *Study of greedy search with multiple improvement heuristics for vehicle routing problems.*

Ribeiro GM, Laporte G, 2012 *An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. Computers & Operations Research* 39(3):728–735.

Savelsbergh MW, 1992 *The vehicle routing problem with time windows: Minimizing route duration. ORSA journal on computing* 4(2):146–154.

Savelsbergh MWP, 1988 *Computer Aided Routing.* Ph.d. dissertation, Centrum voor Wiskunde en Informatica, Amsterdam, Amsterdam.

Schrimpf G, Schneider J, Stamm-Wilbrandt H, Dueck G, 2000 *Record breaking optimization results using the ruin and recreate principle. Journal of Computational Physics* 159(2):139–171.

Shaw P, 1998 *Using constraint programming and local search methods to solve vehicle routing problems. Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming*, 417–431, CP '98 (London, UK, UK: Springer-Verlag).

Subramanian A, Uchoa E, Ochi LS, 2013 *A hybrid algorithm for a class of vehicle routing problems. Computers & Operations Research* 40(10"):2519–2531.

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

Toth P, Vigo D, 2002 *Models, relaxations and exact approaches for the capacitated vehicle routing problem. Discrete Applied Mathematics* 123:487–512.

Uchoa E, Pecin D, Pessoa A, Poggi M, Subramanian A, Vidal T, 2014 *New benchmark instances for the capacitated vehicle routing problem.* Technical report, Research Report Engenharia de Produção, Universidade Federal Fluminense.

Van Breedam A, 1994 *An Analysis of the Behavior of Heuristics for the Vehicle Routing Problem for a selection of problems with Vehicle-related, Customer-related, and Time-related Constraints.* Ph.d. dissertation, University of Antwerp, Faculty of Applied Economics, Antwerp.

Vidal T, Crainic TG, Gendreau M, Prins C, 2014 *A unified solution framework for multi-attribute vehicle routing problems. European Journal of Operational Research* 234(3):658–673.

Vidal T, Crainic TG, Gendreau M, Prins C, 2015 *Time-window relaxations in vehicle routing heuristics. Journal of Heuristics* 21(3):329–358.

Waters CDJ, 1987 *A solution procedure for the vehicle-scheduling problem based on iterative route improvement. The Journal of the Operational Research Society* 38(9):833–839.