Chapter 5
The Vehicle Routing Problem with Time Windows

Guy Desaulniers
Oli B.G. Madsen
Stefan Ropke

5.1 ▪ Introduction

The *Vehicle Routing Problem with Time Windows* (VRPTW) is the extension of the *Capacitated Vehicle Routing Problem* (CVRP) where the service at each customer must start within an associated time interval, called a time window. Time windows may be hard or soft. In case of hard time windows, a vehicle that arrives too early at a customer must wait until the customer is ready to begin service. In general, waiting before the start of a time window incurs no cost. In the case of soft time windows, every time window can be violated barring a penalty cost. The time windows may be one-sided, e.g., stated as the latest time for delivery.

Time windows arise naturally in problems faced by business organizations which work on flexible time schedules. Specific problems with hard time windows include security patrol service, bank deliveries, postal deliveries, industrial refuse collection, grocery delivery, school bus routing, and urban newspaper distribution. Among the soft time window problems, dial-a-ride problems constitute an important example. In this chapter, we focus mainly on the hard time-window variant that has been the mostly studied one.

In the existing literature on the VRPTW, the number of vehicles available to serve the customers is usually considered unlimited and the objective function depends on the nature of the chosen solution method. For exact methods the objective is to minimize the total distance traveled. For heuristics the primary objective is to minimize the number of vehicles used and the secondary to minimize the total distance traveled. There may be exceptions to this general statement.

Since the CVRP is NP-hard, by restriction, the VRPTW is also NP-hard. In fact even finding a feasible solution to the VRPTW for a fixed number of vehicles is itself an NP-complete problem (Savelsbergh [109]). From a computational point of view one may distinguish between tight and loose time windows, and between narrow and wide time windows. A tight time window is a window which influences the solution; i.e., the window is an active constraint. A time window is considered as narrow if it is narrow relative to the planning horizon, e.g., 10 minutes compared to 12 hours. A narrow time window is not necessarily tight at the same time. If all the time windows are very wide, the VRPTW turns almost into a CVRP.

In the CVRP the geography is usually the important factor determining the shape of the routes. If the number of customers is small, say 20, a trained dispatcher can do very well in planning the routes only by looking at a map showing the location of the customers. However, if the capacity constraint is binding on some of the routes, it is much more difficult to overlook the planning situation. For the VRPTW when some of the routes are constrained by the capacity and others by the time windows, it is even more difficult to plan the routes manually. The interplay between the spatial and the temporal elements of the routes may result in optimal routes which are far from the classical picture of routes formed as clover leafs or petals. If the number of customers is increased to a realistic figure, say at least 100–200, it then becomes really difficult to make routes by hand. It is here that the computerized solution methods show their advantages.

The early works on the VRPTW were case study oriented (see Pullen and Webb [100], Knight and Hofer [69], and Madsen [79]). The solution methods were based on relatively simple heuristics. The first exact Branch-and-Bound algorithms appeared in the beginning of the 1980s (see Christofides, Mingozzi, and Toth [19], Baker and Rushinek [4], and Trienekens [118]). In 1987, Solomon [113] introduced benchmark instances involving 100 customers that were accepted as standard benchmark problems by most researchers working on the VRPTW and served as a catalyst to increase research on the VRPTW. In the following decade, many heuristics were developed, mostly local search ones, but also the first metaheuristics (tabu search and genetic algorithms). Several exact algorithms based on complex methodologies such as Lagrangian relaxation and column generation were also designed.

In the previous book on the VRP by Toth and Vigo [117], the most recent references on the VRPTW chapter by Cordeau et al. [21] dates back to 2000. The following sections will deal with the scientific progress made on the VRPTW since 2000. The reader can find additional information in the earlier surveys by Bräysy and Gendreau [16, 17] (heuristics and metaheuristics), Kallehauge [65] (exact algorithms), Potvin [96] (evolutionary algorithms), Gendreau and Tarantilis [47] (metaheuristics), Desaulniers, Desrosiers, and Spoorendonk [29] (exact algorithms), and Baldacci, Mingozzi, and Roberti [7] (exact algorithms).

This chapter is organized as follows. Section 5.2 presents mathematical formulations for the VRPTW. Sections 5.3 and 5.4 are devoted to the recent exact and the heuristic solution algorithms for the VRPTW, respectively. Each of these sections includes summary computational results. Section 5.5 discusses extensions of the VRPTW. Finally, conclusions are drawn in Section 5.6.

5.2 ▪ Mathematical Formulations

Starting from the notation given in Chapter 1, the VRPTW is defined on the directed graph $G = (V, A)$, where the depot is represented by the two vertices 0 and $n+1$, referred to as the *source* and *sink* vertices, respectively. Let $N = V \setminus \{0, n+1\}$ be the set of customer vertices. All feasible vehicle routes correspond to source-to-sink elementary paths in G. The converse is, however, not necessarily true; that is, some source-to-sink elementary paths in G may not represent feasible routes because they violate the time windows or the vehicle capacity. To simplify notation, zero demands and zero service times are defined for vertices 0 and $n+1$, i.e., $q_0 = q_{n+1} = s_0 = s_{n+1} = 0$. Furthermore, a time window is associated with them, i.e., $[a_0, b_0] = [a_{n+1}, b_{n+1}]$, where a_0 and b_0 are the earliest possible departure time from the depot and the latest possible arrival time at the depot, respectively. Assuming that the travel time matrix satisfies the triangle inequality, feasible solutions exist only if $a_0 \leq \min_{i \in V\setminus\{0\}}\{b_i - t_{0i}\}$ and

$$b_0 \geq \max_{i \in V\setminus\{0\}} \{\max\{a_0 + t_{0i}, a_i\} + s_i + t_{i,n+1}\}.$$

Note that an arc $(i,j) \in A$ can be omitted due to temporal considerations, if $a_i + s_i + t_{ij} > b_j$, or capacity limitations, if $q_i + q_j > Q$, or by other factors. Finally, let us mention that when vehicles are allowed to remain at the depot, especially in the case where the primary objective consists of minimizing the number of vehicles used, the arc $(0, n+1)$ with $c_{0,n+1} = t_{0,n+1} = 0$ must be added to the arc set A.

We first present a *Mixed-Integer Programming* (MIP) formulation for the VRPTW involving two types of variables: for each arc $(i,j) \in A$ and each vehicle $k \in K$, there is a binary arc-flow variable x_{ijk} that is equal to 1 if arc (i,j) is used by vehicle k, and 0 otherwise; and, for each vertex $i \in V$ and vehicle $k \in K$, there is a time variable T_{ik} specifying the start of service time at vertex i when serviced by vehicle k.

The VRPTW can be formulated as the following multi-commodity network flow model with time-window and capacity constraints:

$$\text{(5.1)} \qquad \text{(VRPTW1) minimize} \sum_{k\in K} \sum_{(i,j)\in A} c_{ij} x_{ijk}$$

$$\text{(5.2)} \qquad \text{s.t.} \sum_{k\in K} \sum_{j \in \delta^+(i)} x_{ijk} = 1 \qquad \forall\, i \in N,$$

$$\text{(5.3)} \qquad \sum_{j \in \delta^+(0)} x_{0jk} = 1 \qquad \forall\, k \in K,$$

$$\text{(5.4)} \qquad \sum_{i \in \delta^-(j)} x_{ijk} - \sum_{i \in \delta^+(j)} x_{jik} = 0 \qquad \forall\, k \in K,\ j \in N,$$

$$\text{(5.5)} \qquad \sum_{i \in \delta^-(n+1)} x_{i,n+1,k} = 1 \qquad \forall\, k \in K,$$

$$\text{(5.6)} \qquad x_{ijk}(T_{ik} + s_i + t_{ij} - T_{jk}) \leq 0 \qquad \forall\, k \in K,\ (i,j) \in A,$$

$$\text{(5.7)} \qquad a_i \leq T_{ik} \leq b_i \qquad \forall\, k \in K,\ i \in V,$$

$$\text{(5.8)} \qquad \sum_{i \in N} q_i \sum_{j \in \delta^+(i)} x_{ijk} \leq Q \qquad \forall\, k \in K,$$

$$\text{(5.9)} \qquad x_{ijk} \in \{0,1\} \qquad \forall\, k \in K,\ (i,j) \in A.$$

Objective function (5.1) aims at minimizing the total cost. Constraints (5.2) ensure that each customer is assigned to exactly one route. Next, constraints (5.3)–(5.5) define a source-to-sink path in G for each vehicle k. Additionally, constraints (5.6)–(5.7) and (5.8) guarantee schedule feasibility with respect to time windows and vehicle capacity, respectively. Note that, for a given k, the value of T_{ik} is meaningless whenever customer i is not visited by vehicle k. Finally, the arc-flow variables are subject to binary requirements (5.9).

Model (5.1)–(5.9) is nonlinear due to constraints (5.6) that can, however, be linearized as

$$\text{(5.6a)} \qquad T_{ik} + s_i + t_{ij} - T_{jk} \leq (1 - x_{ijk}) M_{ij} \qquad \forall\, k \in K,\ (i,j) \in A,$$

where M_{ij}, $(i,j) \in A$, are large constants that can be set to $\max\{b_i + s_i + t_{ij} - a_j, 0\}$.

The linear relaxation of model (5.1)–(5.5), (5.6a), (5.7)–(5.9) provides, in general, very weak lower bounds. This model has, however, a block-angular structure that can be exploited, where each block is composed of the constraints (5.3)–(5.5), (5.6a), (5.7)–(5.9) for a specific vehicle $k \in K$ and defines an *Elementary Shortest Path Problem with Resource Constraints* (ESPPRC). Applying the Dantzig–Wolfe decomposition principle [25] to this model yields the following set partitioning model once the (identical) vehicles and their corresponding variables are aggregated (see, e.g., Desrosiers et al. [33]). In this model, Ω denotes the set of all feasible routes, c_r the cost of route $r \in \Omega$, and a_{ir} the number of visits to customer $i \in N$ in route $r \in \Omega$ ($a_{ir} \in \{0,1\}$ when r is an elementary route. With each route $r \in \Omega$ is associated a binary path-flow variable y_r that takes value 1 if route r is selected in the solution and 0 otherwise. The set partitioning model is

$$\text{(5.10)} \qquad \text{(VRPTW2) minimize} \sum_{r\in\Omega} c_r y_r$$

$$\text{(5.11)} \qquad \text{s.t.} \sum_{r\in\Omega} a_{ir} y_r = 1 \qquad \forall\, i \in N,$$

$$\text{(5.12)} \qquad y_r \in \{0,1\} \qquad \forall\, r \in \Omega.$$

Objective function (5.10) seeks to minimize total cost. Set partitioning constraints (5.11) impose that each customer be visited exactly once by a vehicle. The binary requirements on the path-flow variables are expressed by (5.12). Notice that, as in the VRPTW literature, the above model assumes that the number of vehicles available to service the customers is unlimited, that is, $|K|$ is as large as needed. If this was not the case, the constraint

$$\text{(5.13)} \qquad \sum_{r\in\Omega} y_r \le |K|$$

enforcing the selection of at most one route per available vehicle would be added to the model.

The linear relaxation of model (5.10)–(5.12) produces better lower bounds than that of model (5.1)–(5.5), (5.6a), (5.7)–(5.9). On the other hand, the set partitioning model contains a huge number of variables, one per feasible route. In Section 5.3.1, we shall see how this difficulty can be overcome using column gene ration.

Several other models were proposed for the VRPTW. In particular, two-index formulations were used in conjunction with Branch-and-Cut algorithms (see Section 5.3.2). We present such a formulation below that involves one type of variables: for each arc $(i,j) \in A$, there is a binary variable x_{ij} that is equal to 1 if arc (i,j) is used in the solution and 0 otherwise. Denote by P the set of paths (not necessarily from the source-to-sink vertex) in G that do not respect the time-window constraints and by $A(p)$ the set of arcs in path $p \in P$. Let $r(S)$ be the minimum number of vehicles required to serve the customers in subset $S \subseteq N$ according to their demands. This number is, in general, replaced by $\lceil q(S)/Q \rceil$, where $q(S) = \sum_{i\in S} q_i$. The two-index formulation corresponds to

$$\text{(5.14)} \qquad \text{(VRPTW3) minimize} \sum_{(i,j)\in A} c_{ij} x_{ij}$$

$$\text{(5.15)} \qquad \text{s.t.} \sum_{j\in\delta^-(i)} x_{ji} = 1 \qquad \forall\, i \in N,$$

$$\text{(5.16)} \qquad \sum_{j\in\delta^+(i)} x_{ij} = 1 \qquad \forall\, i \in N,$$

$$\sum_{i \notin S} \sum_{j \in S} x_{ij} \geq r(S) \qquad \forall\, S \subseteq N,\ S \neq \emptyset, \tag{5.17}$$

$$\sum_{(i,j) \in A(p)} x_{ij} \leq |A(p)| - 1 \qquad \forall\, p \in P, \tag{5.18}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, (i,j) \in A. \tag{5.19}$$

The objective function (5.14) minimizes the total cost for serving the customers. Constraints (5.15)–(5.16) ensure that a vehicle arrives at and departs from each customer, respectively. Capacity inequalities (5.17) ensure that vehicle capacity is satisfied on all selected routes and also tighten the linear relaxation by imposing a minimum number of vehicles to serve every subset of customers S. Furthermore, they act as subtour elimination constraints. Infeasible path inequalities (5.18) forbid the selection of paths that do not respect the time windows. Finally, the flow variables x_{ij} are subject to binary requirements (5.19).

The two-index formulation (5.14)–(5.19) contains an exponential number of constraints (5.17) and (5.18). For practical-sized instances, they need to be generated dynamically as in a cutting plane algorithm. Additional valid inequalities can also be considered to tighten the linear relaxation of this model (see Section 5.3.2).

5.3 ▪ Exact Solution Methods

Numerous exact solution methods have been proposed for the VRPTW. The reader is referred to Cordeau et al. [21] for a survey on the methods that were developed before 2000. Since then, several algorithms were also designed. They can be classified into the following three families: Branch-and-Cut-and-Price, Branch-and-Cut, and reduced set partitioning.

5.3.1 ▪ Branch-and-Cut-and-Price

Branch-and-Price is a leading methodology for solving a wide variety of constrained vehicle routing and crew scheduling problems (see Barnhart et al. [9], Desaulniers, Desrosiers, and Solomon [28], and Lübbecke and Desrosiers [77]). For the VRPTW, it was introduced by Desrochers, Desrosiers, and Solomon [32]. It corresponds to a Branch-and-Bound in which the linear relaxations are solved by column generation. When cutting planes are generated to strengthen the linear relaxations, the method is called Branch-and-Cut-and-Price. The first Branch-and-Cut-and-Price for the VRPTW is due to Kohl et al. [70].

Column generation (Dantzig and Wolfe [25] and Gilmore and Gomory [49]) is an iterative process that can be applied for solving certain linear programs involving a huge number of variables such as the linear relaxation of the set partitioning model (5.10)–(5.12). In this context, the linear program is called the *Master Problem* (MP). At each iteration, a *Restricted Master Problem* (RMP) and an auxiliary problem, called the pricing problem or the subproblem, are solved sequentially. The RMP is the MP restricted to a subset of its variables. It is solved by a linear programming solver to provide a primal and a dual solution. Given this dual solution, the subproblem is problem-specific and consists of finding negative reduced cost columns (variables) for the RMP when some exist. When such columns are found, they are added to the RMP before starting a new iteration. If no negative reduced cost columns can be identified, the solution process stops and the current RMP primal solution is declared optimal for the MP.

For the VRPTW, the subproblem is an ESPPRC (see, e.g., Irnich and Desaulniers [61]) defined on network G. The resource constraints impose the time window and vehicle capacity constraints. Let α_i, $i \in N$, be the dual variables associated with constraints (5.11), and let $\alpha_0 = 0$. The subproblem can be stated as

$$\min_{r\in\Omega} \; c_r - \sum_{i\in N} a_{ir}\alpha_i \tag{5.20}$$

or, equivalently, as

$$\text{minimize} \sum_{(i,j)\in A} (c_{ij} - \alpha_i)x_{ij} \tag{5.21}$$

$$\text{s.t.} \quad \sum_{j\in\delta^+(0)} x_{0j} = 1, \tag{5.22}$$

$$\sum_{i\in\delta^-(j)} x_{ij} - \sum_{i\in\delta^+(j)} x_{ji} = 0 \qquad \forall\, j \in N, \tag{5.23}$$

$$\sum_{i\in\delta^-(n+1)} x_{i,n+1} = 1, \tag{5.24}$$

$$x_{ij}(T_i + s_i + t_{ij} - T_j) \leq 0 \qquad \forall\, (i,j)\in A, \tag{5.25}$$

$$a_i \leq T_i \leq b_i \qquad \forall\, i \in V, \tag{5.26}$$

$$\sum_{i\in N} q_i \sum_{j\in\delta^+(i)} x_{ij} \leq Q, \tag{5.27}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, (i,j)\in A, \tag{5.28}$$

where constraints (5.22)–(5.28) correspond to the constraints (5.3)–(5.9) of any specific vehicle k and index k is omitted. The objective function (5.21) aims at minimizing the path reduced cost, ensuring that a negative reduced cost column is found when one exists. Note that, in this formulation, path elementarity is imposed through the uniqueness of the time variable at each vertex and the positivity of the travel and service times.

In a Branch-and-Cut-and-Price algorithm, cuts and branching decisions are added to derive integer solutions when the computed solution of the MP is fractional. These cuts and branching decisions impact the MP or the subproblem definition. In the following paragraphs, we review the recent contributions on Branch-and-Cut-and-Price algorithms for the VRPTW.

Labeling Algorithms for the ESPPRC. The ESPPRC is known to be NP-hard (Dror [36]). It is usually solved by dynamic programming, namely, by a labeling algorithm (Irnich and Desaulniers [61]). In such an algorithm, a label E is a vector of $3+|N|$ components representing a partial path from the source vertex to any vertex i of G. Its components indicate the path reduced cost ($Z(E)$); its (earliest) start of service time at vertex i ($T(E)$); its cumulated load up to vertex i ($L(E)$); and whether vertex $\ell \in N$ can still be visited in an extension of this path ($W_\ell(E)$). A vertex can still be visited if it has not been visited yet and if it can be reached without violating its time window or the vehicle capacity. Starting from an initial label $E_0 = (0, a_0, \ldots, 0)$ at the source vertex, a labeling algorithm propagates the labels forwardly using extension functions towards the sink vertex. Given a label $E_i = (Z(E_i), T(E_i), L(E_i), (W_\ell(E_i))_{\ell\in N})$ representing a path ending at node i, its extension along an arc $(i,j)\in A$ yields a new label $E_j = (Z(E_j), T(E_j), L(E_j), (W_\ell(E_j))_{\ell\in N})$

at node j with

$$Z(E_j) = Z(E_i) + c_{ij} - \alpha_i, \tag{5.29}$$

$$T(E_j) = \max\{T(E_i) + s_i + t_{ij}, a_j\}, \tag{5.30}$$

$$L(E_j) = L(E_i) + q_j, \tag{5.31}$$

$$W_\ell(E_j) = \begin{cases} W_\ell(E_i) + 1 & \text{if } j = \ell, \\ \max\{W_\ell(E_i), U_\ell(T(E_j), L(E_j))\} & \text{otherwise,} \end{cases} \tag{5.32}$$

where $U_\ell(T(E_j), L(E_j))$ is equal to 1 if it is not possible to feasibly reach vertex $\ell \in N$ from vertex j with a start of service time at j equal to $T(E_j)$ and a load of $L(E_j)$. Label E_j corresponds to an infeasible partial path if $T(E_j) > b_j$, $L(E_j) > Q$, or $W_\ell(E_j) > 1$ for at least one customer $\ell \in N$. When one of these conditions is met, E_j is discarded.

To avoid enumerating all feasible partial paths, a labeling algorithm applies a dominance rule. This rule aims at identifying partial paths that cannot yield an optimal source-to-sink path. Given that the extension functions (5.29)–(5.32) are nondecreasing, the following proposition states the dominance rule used for the ESPPRC in the context of the VRPTW.

Proposition 1 (Desaulniers et al. [27]). *Let E and E' be two labels representing partial paths ending at the same vertex. Label E dominates label E' (which can be discarded) if*

$$Z(E) \le Z(E'), \tag{5.33}$$

$$T(E) \le T(E'), \tag{5.34}$$

$$L(E) \le L(E'), \tag{5.35}$$

$$W_\ell(E) \le W_\ell(E') \qquad \forall\, \ell \in N. \tag{5.36}$$

When equality holds for all components, one of the two labels must be kept.

For solving the linear relaxation of the VRPTW, Feillet et al. [40] were the first authors to propose a column generation algorithm that relies on an ESPPRC subproblem and the above labeling algorithm. They demonstrated empirically that, for the instances with relatively wide time windows, their algorithm produces much better lower bounds than a column generation algorithm based on the generation of paths allowing cycles (see the paragraphs on path relaxations below). It also finds more optimal integer solutions at the root node.

Different acceleration strategies for the labeling algorithm were developed. Noticing that the conditions of the dominance rule in Proposition 1 are only sufficient to identify dominated labels, Chabrier [18] proposed a modified dominance rule that permits the identification of dominated labels even if some of the conditions (5.36) do not hold: condition (5.33) must be replaced by

$$Z(E) \le Z(E') - \sum_{\ell \in N(E,E')} \alpha_\ell, \tag{5.37}$$

where $N(E,E')$ is the subset of customers $\ell \in N$ for which $W_\ell(E) > W_\ell(E')$. In their computational experiments, they considered this dominance rule only when $|N(E,E')| \le 2$. Otherwise, the rule of Proposition 1 is used.

To speed up the labeling algorithm, Righini and Salani [103] designed a bidirectional search that consists of extending labels forwardly from the source vertex and backwardly from the sink vertex before joining forward labels with backward labels to yield complete source-to-sink feasible paths. Forward and backward extensions are forbidden when a predetermined resource (time or load) reaches the midpoint of its possible values over the whole network G. In this way, the total number of labels generated is significantly reduced. Bounds on the reduced cost of any extension (computed through binary knapsack problems) can also be used to prune labels prematurely. Such bounds were also proposed by Feillet, Gendreau, and Rousseau [41]. These authors also introduced the concept of label loading and meta-extensions. Before starting the labeling algorithm, labels representing all partial paths (originating from the source vertex) of the paths associated with the basic variables in the current RMP are attached to the vertices of the network. Furthermore, meta-vertices corresponding to the ends of these paths are appended to the network. When (forward) labeling reaches one of these meta-vertices, a so called meta-extension is performed to extend the label directly to the sink vertex in the hope of finding rapidly negative reduced cost paths.

Another efficient acceleration strategy, called decremental state space relaxation, was developed independently by Boland, Dethridge, and Dumitrescu [14] and Righini and Salani [104]. This technique starts by solving the ESPRCC considering labels with no customer components $W_\ell()$, $\ell \in N$, that is, allowing all cycles. If no negative reduced cost paths (elementary or not) are found in this way, then column generation stops. Otherwise the paths with a negative reduced cost are tested for elementarity. If elementary ones are found, they are added to the RMP before starting a new column generation iteration. If none are identified, then a customer component $W_{\bar{\ell}}()$ is added to the labels, where $\bar{\ell}$ is chosen as a vertex visited more than once in the non-elementary path with the least reduced cost (other criteria for selecting $\bar{\ell}$ can also be considered). The labeling algorithm is then reapplied using enlarged labels. This process repeats until no negative reduced paths are found or at least one negative reduced cost elementary path is identified. With this strategy, the number of customer components to consider is relatively small (that is, not many iterations of the above search are required) unless the time windows and travel times allow many cycles.

For the difficult instances, solving the NP-hard ESPPRC subproblem is often the bottleneck of the Branch-and-Price algorithm: a large part of the total computational time is devoted to solving it. As is well known for various applications, heuristics can be used to solve the subproblem as long as they succeed to generate negative reduced cost columns. When this is not the case, an exact algorithm must be invoked to ensure the exactness of the overall column generation algorithm. Examples of heuristics that were used for the ESPPRC subproblem in the VRPTW context can be found in Chabrier [18] and Desaulniers, Lessard, and Hadjar [31]. Chabrier [18] proposes omitting conditions (5.36) in the dominance rule or to heuristically eliminate arcs from network G. On top of these ideas, Desaulniers, Lessard, and Hadjar [31] developed an efficient multi-start tabu search column generator that applies a tabu search procedure starting from the path associated with each variable in the current basis of the RMP. They present computational results which show that, with such heuristics, the exact labeling algorithm needs to be invoked only once or twice per linear relaxation most of the times. Another heuristic strategy, called limited search discrepancy, that comes from constraint programming was proposed by Feillet, Gendreau, and Rousseau [41]. For each vertex, a set of "good" outgoing arcs is determined based on their reduced cost at each column generation iteration. The other arcs are qualified as "bad" arcs. When solving the subproblem, a maximum number of bad arcs can be traversed in a path. When no negative reduced cost paths are found, this maximum is increased before executing heuristic labeling again.

Finally, Irnich et al. [62] proposed an exact procedure to eliminate arcs in the subproblem network that cannot be part of an optimal solution. Given an upper bound UB on the optimal value of an integer program and an optimal dual solution α to its linear relaxation providing a lower bound LB, it is known that any integer variable whose reduced cost with respect to α is greater than $UB - LB$ can be fixed to zero (see, e.g., Nemhauser and Wolsey [91, p. 389]). To eliminate arcs in the context of a Branch-and-Price algorithm for the VRPTW, this criterion can be applied as follows. After solving an MP in a Branch-and-Bound node, one obtains a dual solution α and a lower bound LB. If one can prove that the reduced costs (with respect to this dual solution) of all variables y_r associated with a path traversing arc $(i, j) \in A$ are greater than $UB - LB$, then arc (i, j) can be eliminated. Irnich et al. [62] showed how to compute a lower bound on the reduced cost of all paths containing an arc (i, j) simultaneously for all arcs in A. Their procedure consists of using forward and backward labeling to compute feasible shortest paths from the source vertex to all other vertices in V and from the sink vertex to all other vertices in V, respectively. Their approach ensures a maximum value for these lower bounds and, thus, the maximum number of eliminated arcs. Because the ESPPRC can be highly difficult to solve, a relaxed shortest path problem, such as the ones discussed below, can also be used in this arc elimination procedure, yielding, however, fewer eliminated arcs.

Labeling algorithms are widely used for solving the ESPPRC arising as a subproblem of the VRPTW. Alternative solution methods are rare. A constraint programming algorithm was designed by Rousseau, Gendreau, and Pesant [108]. Attempts at developing Branch-and-Cut algorithms were presented in recent conferences, but, to our knowledge, they were not successful enough to appear in refereed publications.

Path Relaxations. Given the difficulty of solving the ESPPRC subproblem, Branch-and-Price algorithms can rely on certain relaxed subproblems to generate columns. These subproblems allow the generation of paths containing cycles, but they are easier to solve. On the other hand, with such paths, the MP yields, in general, a weaker lower bound. Paths with cycles are eliminated through branching decisions.

In the first Branch-and-Price algorithm for the VRPTW, Desrochers, Desrosiers, and Solomon [32] used as a subproblem a *Shortest Path Problem with Resource Constraints* (SPPRC) and 2-cycle elimination; that is, all cycles are allowed except those of the form $i - j - i$ with $i, j \in V$. In this case, the labels contain no customer components $W_\ell()$, $\ell \in N$, and a maximum of two labels is kept at each vertex for each pair of admissible time and load values (see Irnich and Desaulniers [61]). More recently, Irnich and Villeneuve [63] considered the SPPRC with k-cycle elimination for any $k \geq 2$; that is, all cycles of length k or less are forbidden. Handling the case for $k \geq 3$ needs, however, sophisticated data structures. Test results showed that, compared to the 2-cycle elimination case, using 3-cycle and 4-cycle elimination can yield higher lower bounds for instances with wide time windows and much faster computational times. On the other hand, eliminating k-cycles for $k > 4$ does not bring a sufficient increase in the lower bound to compensate for the additional time spent solving the subproblem.

Desaulniers, Lessard, and Hadjar [31] proposed another path relaxation called partial elementarity. It consists of considering a subset of the customer components $W_\ell()$, $\ell \in N$, that is selected dynamically: after solving the MP, components that would forbid cycles present in the solution are added to this subset before solving the MP again. This iterative process continues until no cycles appear in the solution or the subset contains a prespecified maximum number of components. Note that this relaxation is akin to the decremental state space relaxation discussed above. With this approach, Desaulniers, Lessard, and Hadjar [31] showed that, for most tested 100-customer instances, a maximum of 40 customer components is sufficient to obtain only elementary paths in an MP solution.

Baldacci et al. [5] and Baldacci, Mingozzi, and Roberti [6] introduced the ng-path relaxation. With every customer $i \in N$ is associated a neighborhood $N_i \subset N$ that contains i and its "closest" neighbors. An ng-path can contain a cycle $i - \cdots - j - \cdots - i$ only if it contains a vertex j such that $i \notin N_j$. The rationale behind accepting such cycles is that because i is not a neighbor of j, then returning to i after visiting j is considered a long detour and, therefore, a route containing such a cycle should not be part of an optimal MP solution. The labeling algorithm described above can easily be modified to find ng-paths. Indeed, the sole modification concerns the extension functions of the customer components $W_\ell()$, $\ell \in N$, that must set to 0 all components such that $\ell \notin N_j$ when creating a label at vertex $j \in V$. Consequently, a maximum of $|N_j|$ customer components can take value 1, increasing the chance of dominating labels when $|N_j|$ is relatively small and accelerating the solution of the subproblem. In practice, neighborhoods of size 15 seem sufficient to ensure that the paths in the computed MP solution are elementary for instances with 100 customers. Note that the neighborhoods can also be built dynamically as for the partial elementarity relaxation that can be seen as a special case of the ng-path relaxation in which all neighborhoods N_i, $i \in N$, are identical.

Dual Variable Stabilization. Column generation is well known to be subject to dual value oscillations from one iteration to the next throughout the solution process (see Vanderbeck [119]). In the first half of the iterations, these oscillations are due to poor dual information that allows the generation of irrelevant columns (this phenomenon is called the *heading-in effect*). In the second half, primal degeneracy is often present, yielding multiple dual solutions and many generated columns that are not useful. Slow convergence (called the *tailing-off effect*) is thus observed; that is, the RMP objective value converges very slowly to the MP optimal value, while searching for a complementary dual optimal solution. Clearly, dual value oscillations increase the number of iterations required to achieve optimality.

To avoid such oscillations and reduce the number of iterations, a dual variable stabilization strategy can be applied. For the VRPTW, Rousseau, Gendreau, and Feillet [107] proposed computing at each iteration a dual solution corresponding to an interior point of the RMP optimal dual facet. To do so, they first solve modified RMPs (with various right-hand sides) to generate several optimal dual solutions. Then, the interior point is obtained as a convex combination of these dual solutions computed. This simple method allows a substantial reduction of the number of iterations required to solve an MP.

Another stabilization method was developed by Kallehauge, Larsen, and Madsen [67]. Instead of solving directly the RMP at each column generation iteration, they solve its dual. This approach corresponds to a Lagrangian relaxation method in which the Lagrangian dual problem is solved by a cutting-plane algorithm. To stabilize the Lagrangian multipliers (the dual variables), each of them is restricted to taking a value within a trust region centered around the value taken in the previous iteration. With this method, the number of iterations is not necessarily reduced (and may even increase substantially), but the duals of the RMP seem to be much easier to solve than their primal counterparts, yielding an overall reduction in computational time. This method is a sequel to the Lagrangian relaxation method of Kohl and Madsen [71] that used a bundle method to stabilize the dual variables.

Valid Inequalities. As discussed by Desaulniers, Desrosiers, and Spoorendonk [30], valid inequalities in a column generation context can be defined on the variables of the compact formulation or solely on the variables of the extended formulation. In our case, the compact and extended formulations correspond to (5.1)–(5.9) and (5.10)–(5.12), respectively. Cutting planes of the first type can easily be rewritten in terms of the MP variables using the variable substitution of the Dantzig–Wolfe reformulation process (see Desaulniers et al. [27]). When involving only the arc-flow variables, they have the advantage of not modifying the structure of the subproblem. An example of such inequalities that can be applied to the VRPTW are the 2-path inequalities (Kohl et al. [70]). Let $S \subseteq N$ be a subset of customers and $\nu(S)$ the minimum number of vehicles required to service the customers in S while respecting vehicle capacity and time windows. Given a subset of customers S that cannot be serviced by a single vehicle (that is, with $\nu(S) \geq 2$), the corresponding 2-path cut forces the traversal of at least two arcs entering subset S in any feasible solution. The 2-path cuts are expressed by

$$\sum_{k\in K} \sum_{\substack{(i,j)\in A:\\ i\in V\setminus S,\, j\in S}} x_{ijk} \geq 2 \qquad \forall\, S \subseteq N \mid \nu(S) \geq 2 \tag{5.38}$$

or, equivalently, by

$$\sum_{r\in\Omega} \beta_r^S y_r \geq 2 \qquad \forall\, S \subseteq N \mid \nu(S) \geq 2, \tag{5.39}$$

where β_r^S is equal to the number of arcs in route r entering subset S. The separation of these cuts consists of finding a subset S such that $\nu(S) \geq 2$. Given a subset S (obtained by enumeration or by a heuristic), two sufficient conditions are typically checked to determine whether $\nu(S) \geq 2$. First, $r(S) = \lceil q(S)/Q \rceil \geq 2$ implies $\nu(S) \geq 2$. Second, if the traveling salesman problem with time windows defined over S is infeasible, then we deduce that $\nu(S) \geq 2$. To handle these inequalities in a column generation context, one simply needs to subtract the associated dual values from the reduced cost of the arcs involved in the cuts. The 2-path cuts can be generalized to ℓ-path cuts for $\ell \geq 2$ by replacing the right-hand side of (5.38) or (5.39) by ℓ whenever $\nu(S) = \ell$. If $\nu(S) = r(S)$, these inequalities are capacity inequalities. Otherwise, to prove that $\nu(S) > 2$ when $\nu(S) > r(S)$ is rather difficult, as it amounts to solving a VRPTW with the objective of minimizing the number of vehicles used to cover the customers in S.

Other families of valid inequalities of the first type (that is, defined on the arc-flow variables) can also be applied to the VRPTW, in particular, several that were developed for the CVRP or the traveling salesman problem, such as the comb inequalities and the odd-cat inequalities.

The valid inequalities of the second type cannot be expressed as linear combinations of arc-flow variables. They are more difficult to handle, as they require modifying the subproblem. For the VRPTW, Jepsen et al. [64] were the first to propose such inequalities, which are called the *Subset Row* (SR) inequalities and correspond to a subset of the Chvátal–Gomory rank 1 cuts for the set partitioning polytope. They are defined over subsets S of the constraints (5.11) as follows:

$$\sum_{r\in\Omega} \left\lfloor \frac{1}{k} \sum_{i\in S} a_{ir} \right\rfloor y_r \leq \left\lfloor \frac{|S|}{k} \right\rfloor \qquad \forall\, S \subseteq N,\ 0 < k < |S|. \tag{5.40}$$

In their computational experiments, Jepsen et al. considered only the SR inequalities for $|S| = 3$ and $k = 2$, which can be separated by enumeration. In this case, the SR inequalities correspond to a subset of the clique inequalities stipulating that, for any 3-customer subset S, at most one route covering at least two of these customers can be part of a feasible solution. Handling these 3-customer SR inequalities increases the complexity of the

ESPPRC subproblem, as each SR cut requires one additional resource, that is, one additional component in each label. More precisely, let $\mathcal{SR}$ be the subset of SR inequalities generated (with $|S| = 3$ and $k = 2$). Denote by S_g the subset S associated with inequality $g \in \mathcal{SR}$ and by ζ_g its nonpositive dual variable. This dual value must be subtracted from the reduced cost of a route that visits at least two customers in S_g, but it cannot be included in the arc reduced costs. In the labeling algorithm used for solving the subproblem, an additional component $B_g(\cdot)$ is added to the labels for each SR inequality $g \in \mathcal{SR}$ to count the number of customers visited in S_g. When extending a label E_i along an arc $(i,j) \in A$ to create a new label E_j, the dual values ζ_g are subtracted in the reduced cost extension function (5.29) for all cuts $g \in \mathcal{SR}$ such that $B_g(E_i) = 1$ and $B_g(E_j) = 2$. Furthermore, the dominance condition (5.33) is replaced by

$$Z(E) - \sum_{g \in \mathcal{SR}(E,E')} \zeta_g \leq Z(E'), \tag{5.41}$$

where $\mathcal{SR}(E,E')$ is the subset of SR cuts g such that $B_g(E) = 1$ and $B_g(E') = 0$. No direct dominance conditions are required on the $B_g(\cdot)$ components of a label. Finally, note that these modifications to the labeling algorithm need further adjustments when a path relaxation allowing cycles is used to generate columns.

The computational results obtained by Jepsen et al. [64] and Desaulniers, Lessard, and Hadjar [31] show that using the SR inequalities can significantly improve the lower bound computed at the root node of the search tree, completely closing the integrality gap for most tested instances. These cuts also help in reducing the computational time for most instances. For the most difficult ones, their impact on the subproblem complexity results in instances of the subproblem that are too hard to be solved in acceptable computational times.

Other valid inequalities defined directly on the y_r variables of the VRPTW extended formulation (5.10)–(5.12) and handled in a column generation framework were also studied recently: Chvátal–Gomory rank 1 cuts by Petersen, Pisinger, and Spoorendonk [93] and clique inequalities by Spoorendonk and Desaulniers [114]. In both cases, these inequalities can help raise the lower bounds achieved at the root node, but the difficulty in treating them does not necessarily allow a reduction of the computational time.

Branching Decisions. To derive integer solutions, the column and cut generation method is embedded into a Branch-and-Bound algorithm to yield a Branch-and-Cut-and-Price method. It is well known that it is difficult to take branching decisions directly on the MP variables y_r, $r \in \Omega$. Indeed, setting $y_r = 0$ implies adding a difficult constraint to the subproblem, namely, to forbid the (re)generation of route r. The branching decisions are rather defined on the arc-flow variables and can be imposed either in the subproblem by modifying the underlying network or in the MP by adding constraints.

The most popular strategy, used by Desrochers, Desrosiers, and Solomon [32], Irnich and Villeneuve [63], Kallehauge, Larsen, and Madsen [67], Kohl et al. [70], and by Desaulniers, Lessard, and Hadjar [31], among others, consists of branching on the aggregated flow on an arc $(i,j) \in A$, that is, on the value of $x_{ij} = \sum_{k \in K} x_{ijk}$. To impose $x_{ij} = 0$, the arc (i,j) is removed from the subproblem network. To impose $x_{ij} = 1$, all arcs $(i',j) \in A$ such that $i' \neq i$ and $i' \neq 0$, and all arcs $(i,j') \in A$ such that $j \neq j'$ and $j \neq n+1$, are removed from the network. In this case, all variables y_r in the current RMP associated with a route r that contains a removed arc must be deleted from the RMP.

Another popular strategy (see Desrochers, Desrosiers, and Solomon [32], Kohl et al. [70], and Irnich and Villeneuve [63]) that is not sufficient in itself to guarantee integrality is to branch on the total number of vehicles used $\sum_{r\in\Omega} y_r$. Such decisions are imposed by adding constraints in the MP.

Alternative branching strategies were also developed in specific works. In these strategies, the decisions were made on the time at which the service starts at a customer, on the possible successors of a customer along a route, or on the flow on the arcs entering or exiting a subset of customers.

5.3.2 ▪ Branch-and-Cut

Branch-and-Cut (see, e.g., Padberg and Rinaldi [92]) is another popular method for solving combinatorial optimization problems. It consists of a cutting plane method embedded into a Branch-and-Bound procedure. Cutting planes are added in each node of the search tree to tighten the linear relaxation as much as possible. When a linear relaxation is feasible, its computed optimal solution is fractional, and no more cutting planes can be identified by the implemented separation procedures; branching decisions are imposed to derive integer solutions.

Bard, Kontoravdis, and Yu [8] were the first to propose a Branch-and-Cut algorithm for the VRPTW in the special case where the objective function aims at minimizing the number of vehicles used (but it can easily be adapted for minimizing total traveling cost or distance). Their algorithm relies on the two-index model (5.14)–(5.19) to which the following relaxed time-window and capacity constraints are added dynamically:

$$T_i - T_j + (s_i + t_{ij})x_{ij} \le (1 - x_{ij})(b_i - a_j) \qquad \forall\, (i,j)\in A, \tag{5.42}$$

$$a_i \le T_i \le b_i \qquad \forall\, i \in V, \tag{5.43}$$

$$h_i - h_j + q_j \le (1 - x_{ij})(Q - q_j) \qquad \forall\, (i,j)\in A, \tag{5.44}$$

$$0 \le h_i \le Q \qquad \forall\, i \in V, \tag{5.45}$$

where T_i and h_i, $i \in V$, are resource variables indicating the start of service time and the accumulated load at vertex i, respectively. Furthermore, Bard, Kontoravdis, and Yu suggest lifting the infeasible path inequalities (5.18) by replacing their right-hand-side member with $|A(p)| - \max\{1, r(N(p))\}$, where $N(p)$ corresponds to the set of customer vertices in path $p \in P$. Their Branch-and-Cut algorithm also involves other valid inequalities, namely, comb, incompatible pair, D_k^-, and D_k^+ inequalities. To identify violated inequalities, they developed ad hoc heuristic separation algorithms.

Lysgaard [78] proposed another Branch-and-Cut algorithm for the VRPTW that relies on the two-index model (5.14)–(5.19), where the infeasible path inequalities (5.18) are replaced by the stronger tournament inequalities. Given an infeasible path $p = (v_1, v_2, \dots, v_\ell) \in P$, the corresponding tournament inequality is

$$\sum_{i=1}^{\ell-1} \sum_{j=i+1}^{\ell} x_{v_i v_j} \le |A(p)| - 1. \tag{5.46}$$

Lysgaard also introduces a new family of valid inequalities, called the reachability inequalities, that can be viewed as a strengthening of the ℓ-path inequalities. In fact, he develops different variants of these cuts. Let us present one. For a customer vertex $i \in N$, let $A_i^- \subset A$ be its reachable arc set, that is, the arc set containing all the arcs (j, j') that are part of a feasible path from the source vertex 0 to vertex i. For a subset of customers $U \subseteq N$, its reachable arc set is given by $A_U^- = \cup_{i \in U} A_i^-$. Furthermore, we say that a customer subset U (that may be a singleton) is conflicting if and only if the customers in U must be serviced on $|U|$ distinct routes in any feasible solution. A subset of the reachability inequalities is expressed as follows:

$$\sum_{\substack{(i,j)\in A\cap A_U^-:\\ i\in V\setminus S,\, j\in S}} x_{ij} \geq |U| \qquad \forall\, U \subseteq S \subseteq N \text{ such that } U \text{ is conflicting.} \tag{5.47}$$

The inequality for subsets U and S stipulates that at least $|U|$ vehicles must enter S through arcs allowing one to reach the customers in U. Symmetric inequalities involving the arcs exiting subset S (and reaching arc sets) were defined as well as inequalities involving semi-conflicting pairs of customers, that is, customers that can be visited on the same route but only consecutively. The separation of the reachability inequalities requires first computing the reachable and reaching arc sets for every customer vertex $i \in N$. Then, conflicting subsets and semi-conflicting customer pairs are enumerated. Finally, for each conflicting subset and each semi-conflicting pair, an exact polynomial-time procedure permits finding violated inequalities.

A third Branch-and-Cut algorithm was designed by Kallehauge, Boland, and Madsen [66]. Again, a modified version of the two-index formulation (5.14)–(5.19) serves as the basis of this algorithm. In the version used, the capacity constraints (5.17) are replaced by subtour elimination constraints; that is, $r(S)$ is replaced by 1. Furthermore, the infeasible path inequalities are defined not only for paths that violate time windows but also for those that do not respect vehicle capacity. As in Lysgaard [78], Kallehauge, Boland, and Madsen suggest using tournament inequalities (5.46) instead of infeasible path inequalities (5.18). Furthermore, based on a previous work on the asymmetric traveling salesman problem with replenishment arcs, they develop the following lifted path inequalities:

$$\sum_{i=1}^{\ell-1} \sum_{(v_i,j)\in\tilde{\delta}_{v_i}^+(p)} x_{v_i,j} \geq 1 \qquad \forall\, p=(v_1,v_2,\ldots,v_\ell)\in P, \tag{5.48}$$

where $\tilde{\delta}_{v_i}^+(p) = \{(v_i,j) \in A \mid j \neq v_1,\ldots,v_{i+1},\ T_{v_i} + s_{v_i} + t_{v_i,j} \leq b_j,\ \sum_{h=1}^{i} q_h + q_j \leq Q\} \cup \{(v_i, n+1) \in A\}$ for $i = 1,2,\ldots,\ell-1$ contains all the arcs leaving vertex v_i that allow escaping path p while respecting time windows and vehicle capacity. They show that, under certain assumptions, these lifted path inequalities are facet-defining. They also transfer to the VRPTW precedence inequalities that were introduced for the precedence constrained asymmetric traveling salesman problem. Indeed, the time windows allow one to define predecessor and successor relationships between the customers. Let $\pi(j) = \{i \in V \mid a_j + s_j + t_{ji} > b_i\}$ and $\sigma(j) = \{i \in V \mid a_i + s_i + t_{ij} > b_j\}$ be the predecessor and successor sets of customer $j \in N$. The precedence inequalities are given by

$$\sum_{\substack{(i,\ell)\in A:\\ i\in S\setminus\pi(j),\, j\in\overline{S}\setminus\pi(j)}} x_{ij} \geq 1 \qquad \forall\, S\subseteq N, j\in S, \tag{5.49}$$

$$\sum_{\substack{(i,\ell)\in A:\\ i\in\overline{S}\setminus\sigma(j),\, j\in S\setminus\sigma(j)}} x_{ij} \geq 1 \qquad \forall\, S\subseteq N, j\in S, \tag{5.50}$$

where $\overline{S} = V \setminus S$. The inequalities (5.49) and (5.50) are, more precisely, called the predecessor and successor inequalities (or π- and σ-inequalities), respectively. The Branch-and-Cut algorithm of Kallehauge, Boland, and Madsen [66] also involves odd-cat, D_k^-, and D_k^+ inequalities.

Letchford and Salazar González [74] compare the strength of the lower bounds provided by different formulations of the CVRP. In particular, they show that several types of valid inequalities that can be used to tighten a two-index formulation are implied by the set partitioning formulation when elementary routes are considered, whereas a subset of them are also implied when non-elementary routes are allowed. These results can be transferred to the VRPTW. These authors also introduce a two-commodity flow formulation for the VRPTW that allows them to derive a new family of valid inequalities. These inequalities are called projection inequalities because they are obtained by a projection onto the subspace of the arc-flow variables x_{ij}. They can thus be used with the two-index model (5.14)–(5.19). Intuitively, these inequalities state that, for any subset $S \subseteq N$ of customers, the sum of the upper time limits of the vehicles leaving S must be greater than or equal to the sum of the lower time limits of vehicles entering S plus the time spent by the vehicles inside S.

5.3.3 ▪ Reduced Set Partitioning

Baldacci, Mingozzi, and Roberti [6] (see also Baldacci et al. [5]) developed an efficient procedure to reduce the size of the VRPTW set partitioning model (5.10)–(5.12) by fixing to 0 the value of a very large number of its variables. The reduced set partitioning model is then solved using a MIP commercial solver such as CPLEX. The following criterion, which is valid for any integer linear program, is used to fix variables to 0: Given an integer linear program and an upper bound U on its optimal value, a non-negative integer variable takes value zero in every optimal integer solution if its reduced cost with respect to a feasible dual solution of a linear programming relaxation exceeds $U - L$, where L is the cost of the dual solution that provides a lower bound on the optimal value. The variable can therefore be fixed to 0 or, equivalently, eliminated from the problem formulation.

The upper bound U can be set to the cost of a good heuristic solution. To compute a lower bound L and a feasible dual solution $(\pi_i)_{i \in N \cup \{0\}}$ of the linear relaxation of (5.10)–(5.12), Baldacci, Mingozzi, and Roberti [6] proposed a dual ascent method that combines Lagrangian relaxation, subgradient optimization, and column generation. In this method, up to four heuristics can be used sequentially. In the first heuristic, routes can contain cycles and may not respect either vehicle capacity or time windows. In the second heuristic, ng-routes are generated to yield a better dual solution. In the third heuristic, elementary paths are considered. Finally, the fourth heuristic integrates subset row inequalities defined on subsets of three customers. The dual solution obtained with a heuristic serves as the starting point of the next heuristic.

Given the final computed dual solution $(\pi_i)_{i \in N \cup \{0\}}$ and its cost L, a dynamic algorithm is applied to find all feasible routes in network G whose reduced cost with respect to $(\pi_i)_{i \in N \cup \{0\}}$ is less than or equal to $U - L$. Only the variables associated with these routes are considered in the reduced set partitioning model that also includes the subset row inequalities identified by the fourth heuristic. When there are too many variables, only a subset of the variables (those with the least reduced costs) is enumerated by dynamic programming, yielding an overall heuristic solution method.

5.3.4 ▪ Summary Computational Results for Exact Methods

For more than two decades, the exact solution methods have been tested on the VRPTW benchmark instances proposed by Solomon [113]. These 100-customer instances are

Table 5.1. *Results of the most recent exact methods.*

Instances		JPSP08		DLH08		BMR11	
Series	No.	No. Solved	Avg. Time	No. Solved	Avg. Time	No. Solved	Avg. Time
C1	9	9	468	9	18	9	25
RC1	8	8	11,005	8	2,150	8	276
R1	12	12	27,412	12	2,327	12	251
C2	8	7	2,795	8	2,093	8	40
RC2	8	5	3,204	6	15,394	8	3,767
R2	11	4	35,292	8	63,068	10	28,680
Total	56	45		51		55	

divided into two classes of 29 (class 1) and 27 (class 2) instances, each containing three series (C, RC, and R). In general, the time windows are relatively narrow in class 1 and wide in class 2. The locations of the customers are clustered in the C instances, random in the R instances, and mixed in the RC series.

For each series of 100-customer instances, Table 5.1 reports computational results obtained by the three most recent exact algorithms, namely, those of Jepsen et al. [64], Desaulniers, Lessard, and Hadjar [31], and Baldacci, Mingozzi, and Roberti [6], abbreviated by JSPS08, DLH08, and BMR11, respectively. In this table, the first two columns indicate the instance series and the number of instances it contains. For each series and each algorithm, we report the number of instances that were solved to optimality (no time limit imposed) and the average computational time in seconds to solve them. These times are those reported by the authors and were obtained on computers with different characteristics: P4 at 3.0 GHz for JSPS08, AMD Opteron at 2.6 GHz for DLH08, and IBM Intel Xeon X7350 at 2.93 GHz for BMR11. The last line of Table 5.1 provides the total number of instances solved by each algorithm.

These results clearly show that the instances in class 2 are much more difficult to solve than those in class 1 because wide time windows increase the number of feasible routes and the number of customers per feasible route, yielding harder to solve instances. Furthermore, we observe that, on average, the clustered instances are the easiest to solve and the totally random instances the hardest. The results also show the rapid evolution of the recent algorithms. Before the paper of Jepsen et al. [64], only 35 of the 56 instances were solved to optimality. With the introduction of the SR inequalities in a Branch-and-Cut-and-Price algorithm, Jepsen et al. raised this number to 45. Using an efficient tabu search column generator, the Branch-and-Cut-and-Price algorithm of Desaulniers, Lessard, and Hadjar [31] solved six additional instances. With their approach based on a reduced set partitioning model and relying on ng-paths, Baldacci, Mingozzi, and Roberti [6] solved all instances except one due to a lack of memory. Their computational times are much smaller than the previous ones, but it should be noted that their method requires an upper bound and, for their experiments, they used the best known upper bounds from the literature. Note that, in a recent conference presentation, Ropke [105] reported solving all Solomon instances using a Branch-and-Cut-and-Price algorithm relying on a strong branching strategy. Because no technical paper describing the algorithm is available yet, this approach was not covered above.

In the future, one can expect to see more computational results for the Gehring and Homberger benchmark instances [45], which extend the Solomon data set to instances involving 200, 400, 600, 800, and 1000 customers. To the best of our knowledge, only

Larsen [73], Cook and Rich [20], and Kallehauge, Larsen, and Madsen [67] report solving some of these instances.

5.4 ▪ Heuristics

Heuristics are solution methods that can often find good-quality feasible solutions relatively quickly. However, there is no guarantee regarding solution quality. Heuristics are, thus, tested empirically, and their performance is judged by their computational results. Currently, most VRPs encountered in the industry are solved using heuristics because of their speed and their ability to handle large instances. Tradition dictates that a hierarchical objective function is used when heuristics are applied: the first priority is to minimize the number of vehicles used and the second to minimize the cost of the traversed arcs. This differs from the exact algorithms that do not consider the number of vehicles in the objective function.

In the past decade research on heuristic methods for the VRPTW has focused on metaheuristics and little research has been conducted on construction heuristics. An exception is the work of Ioannou, Kritikos, and Prastacos [59], which proposes a relatively fast construction heuristic which can produce solutions of good quality considering the devoted computational time. Construction heuristics for the VRPTW were surveyed by Bräysy and Gendreau [16].

5.4.1 ▪ Basics

A central concept in most successful heuristics for the VRPTW is that of *local search*. Local search algorithms are based on neighborhoods. Let $\mathscr{S}$ be the set of feasible solutions to a given VRPTW instance, and let $c : \mathscr{S} \to \mathbb{R}$ be a function that maps from a solution to the *cost* to this solution. The set $\mathscr{S}$ is finite, but it is often extremely large. Since the VRPTW is a minimization problem, our goal is to find a solution s^* for which $c(s^*) \leq c(s)$ for all $s \in \mathscr{S}$. However, with heuristics, we are willing to settle for a solution that might be slightly inferior to s^*.

Let $\mathscr{P}(\mathscr{S})$ be the set of subsets of solutions in $\mathscr{S}$. We define a *neighborhood function* as a function $N : \mathscr{S} \to \mathscr{P}(\mathscr{S})$ that maps from a solution s to a subset of solutions $N(s)$. This subset is called the *neighborhood* of s. A solution s is said to be *locally optimal* or a *local optimum* with respect to a neighborhood $N(s)$ if $c(s) \leq c(s')$ for all $s' \in N(s)$. With these definitions, we can describe a steepest descent algorithm (see Algorithm 5.1). The algorithm takes an initial solution s as input. At each iteration, it finds the best solution s' in the neighborhood $N(s)$ of the current solution s (line 4). If s' is better than s (line

Algorithm 5.1 Steepest descent

```
1: input: Initial solution $s \in \mathscr{S}$
2: done = false
3: **while** done $\neq$ true **do**
4:     $s' \in \arg\min_{s'' \in N(s)} \{c(s'')\}$
5:     **if** $c(s') < c(s)$ **then**
6:         $s = s'$
7:     **else**
8:         done = true
9: return s
```

5), then s' replaces s as the current solution (line 6). Lines 3–8 are repeated as long as s' is an improved solution. When the loop stops, the algorithm returns s as the best solution found. The algorithm is called a steepest descent algorithm because it always chooses the best solution in the current neighborhood. As we will see in the following, there are other ways to use neighborhoods to explore the solution space.

In general, larger neighborhoods lead to solutions of better quality when the neighborhoods are used, for example, in a steepest descent algorithm. The drawback of larger neighborhoods is that the evaluation of all solutions is more time consuming in a large neighborhood compared to a smaller one unless clever algorithmic ideas can be used to speed up the search. In the following subsections, we review a number of neighborhoods that have been used in the most successful VRPTW heuristics. We classify these neighborhoods into two categories: *traditional* and *large* neighborhoods. The first category contains the neighborhoods whose size is growing polynomially with n in a controlled manner such that all solutions in the neighborhood can be evaluated by explicit enumeration. Polynomial-sized neighborhoods whose growth is so rapid that they cannot be searched explicitly and neighborhoods whose size is growing exponentially with n fall in the second category (as in Ahuja et al. [1]).

5.4.1.1 ▪ Traditional Neighborhoods

Traditional neighborhoods can be divided into two categories: *intra-route* and *inter-route*. The neighborhoods in the former category contain solutions in which a single route is changed with respect to the reference solution, whereas, in the second category, they contain solutions obtained by moving customers between two or more routes. A thorough survey of VRPTW neighborhoods can be found in Bräysy and Gendreau [16]. Below we review the traditional neighborhoods that have been used in the most important metaheuristics of the past decade.

2-opt Neighborhood (intra-route). A 2-opt neighborhood (see Figure 5.1) contains solutions obtained by removing two arcs from a route and replacing them by two other arcs to reconnect the route, while changing the orientation of the subpath that does not contain the depot. In the figure the square represents the depot and the circles are customers. The dashed arcs correspond to subpaths in network G (see Section 5.2), involving one or several arcs, while each solid arc corresponds to a single arc in G. Notice that changing the orientation of the subpath $(i+1,\dots,j+1)$ can be problematic because of the time windows.

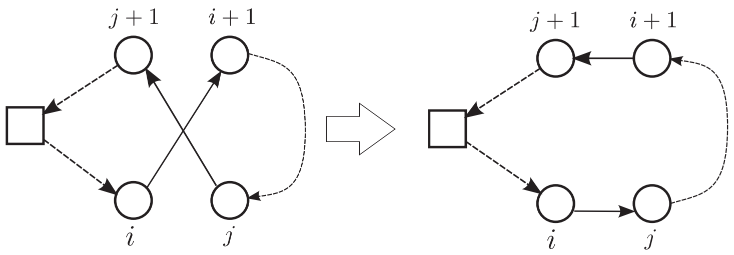

Figure 5.1. *Illustration of the 2-opt neighborhood.*

Or-opt Neighborhood (intra-route). Each solution in the Or-opt neighborhood (see Figure 5.2) is defined by relocating a subpath $(i+1,\dots,j)$ to a different position in a route. The orientation of the relocated subpath is preserved. The move is carried out by

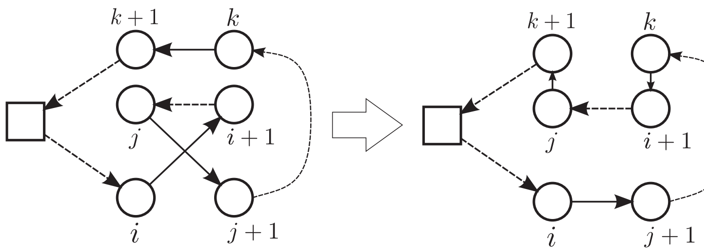

Figure 5.2. *Illustration of the Or-opt neighborhood.*

removing arcs $(i,i+1)$, $(j,j+1)$, and $(k,k+1)$ and replacing them by arcs $(i,j+1)$, $(k,i+1)$, and $(j,k+1)$. Typically the neighborhood is reduced by considering only subpaths containing a limited number of customers or by trying only insertions that are close to the original position. Bräysy [15] introduced a slight variant of the Or-opt neighborhood, denoted IOPT, by allowing the relocated subpath to be reversed when it is reinserted.

2-opt* Neighborhood (inter-route). The 2-opt* neighborhood (see Figure 5.3) is defined similarly to the 2-opt neighborhood, but its solutions are derived by modifying two routes instead of one. Two arcs, $(i,i+1)$ and $(j,j+1)$, from two distinct routes are removed, and the routes are reconnected by inserting the arcs $(i,j+1)$ and $(j,i+1)$. The effect is that the tails of the two routes are exchanged. The 2-opt* neighborhood does not change the orientation of the subpaths, as opposed to the 2-opt.

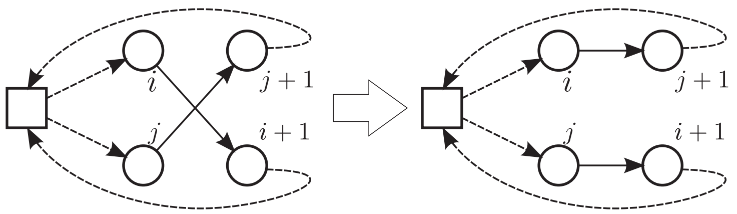

Figure 5.3. *Illustration of the 2-opt* neighborhood.*

Cross Exchange Neighborhood (inter-route). In the cross exchange neighborhood (see Figure 5.4), two subpaths are selected and their positions are exchanged. This is done by removing four arcs $(i,i+1)$, $(j,j+1)$, $(k,k+1)$, and $(l,l+1)$ and replacing them by the arcs $(i,k+1)$, $(l,j+1)$, $(k,i+1)$, and $(j,l+1)$. The size of the cross exchange neighborhood is typically reduced by considering only subpaths containing a limited number of customers. We note that the cross exchange neighborhood can be used in an intra-route fashion where the subpaths to be exchanged belong to the same route.

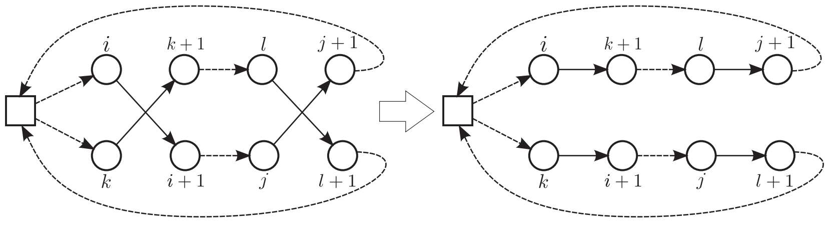

Figure 5.4. *Illustration of the cross exchange neighborhood.*

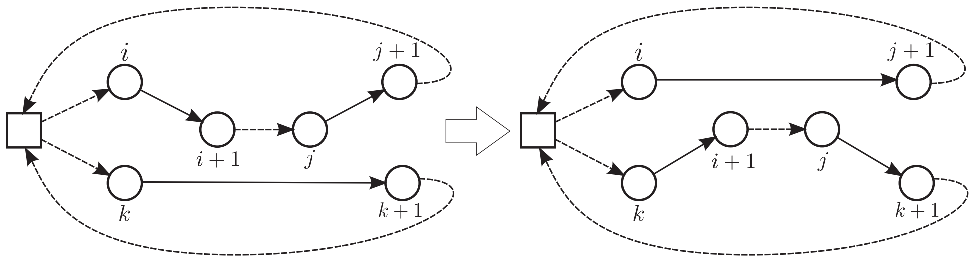

Figure 5.5. *Illustration of the path relocation neighborhood.*

Path Relocation Neighborhood (inter-route). In the path relocation neighborhood (see Figure 5.5), the solutions are obtained by relocating a subpath from one route to another one. This is done by removing three arcs $(i, i+1)$, $(j, j+1)$, and $(k, k+1)$ and replacing them by the arcs $(i, j+1)$, $(j, k+1)$, and $(k, i+1)$. This neighborhood can be seen as a special case of the cross exchange neighborhood if we allow one empty subpath in the latter neighborhood. The length of the relocated subpath is typically limited, and some implementations allow the subpath to be reversed when it is reinserted.

Speedup Techniques. In local search, it is important to implement neighborhood evaluation efficiently in order to search as many solutions as possible within the time allocated for the search. Each move in the neighborhoods mentioned above can be evaluated in constant time using the techniques described in Kindervater and Savelsbergh [68], but for large instances this might not be enough. It is therefore common to truncate the neighborhood search in a heuristic way such that some moves that do not seem promising are not evaluated. This can be done by considering only moves that connect nearby customers; time-window considerations can also be taken into account. Two examples of such *filtering* can be found in Hashimoto and Yagiura [53] and Nagata, Bräysy, and Dullaert [88].

Irnich [60] introduced *sequential search* for the VRPTW. Sequential search is able to speed up local search by using exact filtering where large subsets of the possible moves can be skipped based on cost considerations. Computational results show impressive speedups for large-scale VRPTW instances. Sequential search has, to the best of our knowledge, not yet been used in the most successful VRPTW heuristics.

5.4.1.2 ▪ Large Neighborhoods

The use of large neighborhoods in heuristics has been surveyed by Ahuja et al. [1] and Pisinger and Ropke [95]. Methods for searching large neighborhoods can be divided in two categories: exact and heuristic evaluation methods. An exact evaluation should always identify the best solution within the neighborhood, while a heuristic evaluation may well miss it. Some exponential-sized neighborhoods can be searched exactly in polynomial time. An example is the assignment neighborhood for the *Traveling Salesman Problem* (TSP), which is described by Ahuja et al. [1]. Other exponential-sized neighborhoods are searched exactly by solving NP-hard optimization problems. For example, the cyclic exchange neighborhood of Thompson and Psaraftis [116] can be searched exactly using the algorithms presented in Chapter 4 of Dumitrescu [38].

Exact evaluation of large neighborhoods have not been used in the most successful heuristics for the VRPTW, but large neighborhood heuristic evaluation is an important component in several recent heuristics (for example, see Lim and Zhang [75]). In the

VRPTW literature, the most common large neighborhoods are encountered in the *Large Neighborhood Search* (LNS) framework put forward by Shaw [111] (and the closely related ruin and recreate algorithm of Schrimpf et al. [110]). The key elements in LNS are the *destroy* and *repair* operators. A destroy operator partially disintegrates a solution, while a repair operator reconstructs a complete solution starting from a partial solution. The two operators are used repeatedly, as illustrated in the pseudo-code of Algorithm 5.2. In this pseudo-code x is the current solution, x^* is the best solution observed during the search, and x' is a temporary solution. The function $d(\cdot)$ destroys a solution, $r(\cdot)$ rebuilds a complete solution, and $c(\cdot)$ calculates the cost of a solution. The algorithm repeats lines 4 to 8 until a stopping criterion is met (e.g., a maximum number of iterations). In line 4 the destroy and repair operators are applied. In line 5 a test determines whether the new solution x' should be accepted as the current solution. In the most simple case only improving solutions are accepted.

Algorithm 5.2 Large neighborhood search

```
1: Input: initial solution $x$
2: $x^* = x$
3: **while** stop criterion not met **do**
4:     $x' = r(d(x))$
5:     **if** accept$(x, x')$ **then**
6:         $x = x'$
7:     **if** $c(x') < c(x^*)$ **then**
8:         $x^* = x'$
9: return $x^*$
```

An illustration of how the destroy/repair operators can be used to improve a VRPTW solution is shown in Figure 5.6. In this example the destroy operator removes a subset of customers from their routes and the repair operator reinserts unassigned customers into the routes.

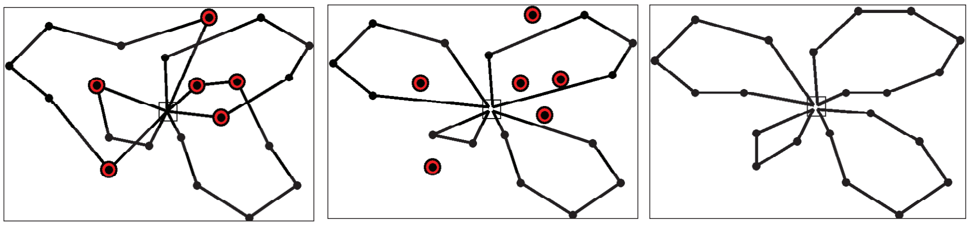

Figure 5.6. *Example of destroy/repair operation. Left figure shows the initial solution with customers to be removed marked. The middle figure illustrates the partial solution after the destroy operation. The right figure shows the solution after the repair operation where the unassigned customers are reinserted.*

5.4.1.3 ▪ Allowing Infeasible Solutions

When designing metaheuristics for the VRPTW, one must decide whether infeasible solutions should be allowed during the search. When allowed, they are typically penalized in the objective function by one or more terms that aim at reducing infeasibility. Allowing infeasible solutions makes it easier to maneuver in the solution space, providing shortcuts between areas of high-quality solutions. On the other hand, it adds complexity to the algorithm: evaluation of the objective may be more difficult, penalty parameters must be adjusted (potentially in a adaptive way), and one must ensure that feasible solutions are visited at least occasionally. Despite these drawbacks, allowing infeasible solutions appears to be an important tool for finding high-quality solutions.

In the current metaheuristics three types of infeasibilities are allowed: time window, vehicle capacity, and customer service violations. The last type occurs when some customers are not visited. It is the most common type in the algorithms that minimize the number of vehicles used (see the next subsection). Time window and capacity violations are typically allowed together and lead to a penalized objective function $\bar{c}(s)$:

$$\bar{c}(s) = c(s) + \alpha q(s) + \beta w(s),$$

where $c(s)$ is the standard objective function and $q(s)$ (resp., $w(s)$) measures the total capacity (resp., time-window) violation over all routes. α and β are parameters that are usually adjusted during the search, reacting to the performance of the algorithm. A good example of how the α and β parameters can managed is given in Cordeau, Laporte, and Mercier [23].

Time-window violation along a route $r = (v_0 = 0, v_1, v_2, \ldots, v_k, v_{k+1} = n+1)$ has traditionally been calculated straightforwardly: the start of service time T_{v_i} at vertex v_i is computed as

$$T_{v_i} = \begin{cases} a_0 & \text{if } i = 0, \\ \max\{a_{v_i}, T_{v_{i-1}} + s_{v_{i-1}} + t_{v_{i-1}v_i}\} & \text{otherwise,} \end{cases}$$

and the total violation is $\sum_{i=1}^{k+1} \max\{0, T_{v_i} - b_{v_i}\}$. The drawback of this procedure is that the evaluation of a move in the traditional neighborhoods can no longer be performed in constant time. Instead, Nagata [85] proposed an artificial start of service time evaluation

$$T'_{v_i} = \begin{cases} a_0 & \text{if } i = 0, \\ \max\{a_{v_i}, \min\{b_{v_i}, T'_{v_{i-1}}\} + s_{v_{i-1}} + t_{v_{i-1}v_i}\} & \text{otherwise} \end{cases}$$

that moves back the start of service time to the end of the time window whenever it is violated. Time-window violation is given by $\sum_{i=1}^{k+1} \max\{0, T'_{v_i} - b_{v_i}\}$, a formula that still captures time-window violations and makes it possible to evaluate in constant time a single move in the traditional neighborhoods (see Vidal et al. [120]).

5.4.1.4 ▪ Minimizing the Number of Vehicles Used

The vehicle minimization/traveling cost minimization is typically carried out in a two-phase manner where one first seeks to find the least number of vehicles necessary and then keeps this number fixed while minimizing the total traveling cost. If the heuristic allows visiting infeasible solutions, then one can select a number of vehicles m and create a (possibly infeasible) solution with that number of vehicles. Then a search is conducted with the aim of finding a feasible solution. If the search is successful, the number of vehicles is reduced by one by deleting a complete route before repeating the search. Otherwise the number of vehicles is increased by one and the search is repeated if no feasible solutions have been found yet. Using a computed lower bound on the number of vehicles, it is possible to stop the search for a fewer number of vehicles when the lower bound is met. The search for the minimum number of vehicles can also be aborted when a sufficient effort has been invested. Nagata and Bräysy [87] have devised one of the most effective route minimization procedure.

5.4.2 ▪ Single Trajectory Search

Starting from a single solution, single trajectory search algorithms generate a sequence of solutions that can be seen as a trajectory through the solution space. At every iteration, only the current solution is used to determine the next one. In the following paragraphs, we review two families of single trajectory search algorithms that were applied to solve the VRPTW, namely, iterated local search and LNS algorithms. Note that these algorithms can rely on tabu search and simulated annealing schemes.

5.4.2.1 ▪ Iterated Local Search

Iterated local search (Lourenço, Martin, and Stützle [76]) is a simple metaheuristic that is based on a series of steepest descent local searches. A pseudo-code of this algorithm is given in Algorithm 5.3. Starting from an initial solution x, the algorithm iterates (lines 4 to 9) between perturbing the current solution x to obtain a temporary solution x' and applying local search to improve this solution until a stopping criterion is met (e.g., a maximum number of iterations). In line 4, x is perturbed, for example, by applying one or more random moves defining a chosen neighborhood function. In line 5, the local search algorithm is typically a descent algorithm, but it could also be a more advanced metaheuristic like tabu search (see, e.g., Cordeau and Maischberger [24]). In lines 6 and 7, the current solution x is updated if the solution x' obtained from the local search is accepted. In lines 8 and 9, the best known solution x^* is updated, if necessary.

Algorithm 5.3 Iterated local search

```
1: Input: initial solution $x$
2: $x^* = x$
3: while stop criterion not met do
4:     $x' = \text{perturb}(x)$
5:     $x' = \text{localsearch}(x')$
6:     if $\text{accept}(x, x')$ then
7:         $x = x'$
8:     if $c(x') < c(x^*)$ then
9:         $x^* = x'$
10: return $x^*$
```

Ibaraki et al. [57] developed an iterated local search algorithm for a variant of the VRPTW where the time windows are represented by convex, piecewise linear penalty functions. The VRPTW is a special case of this problem when the penalty function is set appropriately (infinite penalty outside the feasible region). Because of the penalty functions, it is non-trivial to determine the best possible starting time for a given route. This problem is called the *Optimal Start Time Problem* (OSTP). The authors present a dynamic programming algorithm for the OSTP and describe how the dynamic programming algorithm can be sped up in a local search algorithm. Note that the OSTP was previously studied by Dumas, Soumis, and Desrosiers [37], who considered arbitrary convex penalty functions.

Their iterated local search algorithm uses a number of neighborhoods: Or-opt (or IOPT), 2-opt, 2-opt*, path relocation, and cross exchange. Visiting infeasible solutions with respect to time windows or capacity is allowed, but penalized with the penalization parameters set adaptively. When a local minimum is encountered, a perturbation that consists of performing one to three random cross exchange moves is applied.

Hashimoto, Yagiura, and Ibaraki [54] proposed an iterated local search algorithm for a time-dependent VRPTW with soft time windows by iterated local search. As before, it is a non-trivial task to determine the optimal starting time of a route, and a dynamic programming algorithm is presented. The iterated local search algorithm uses three neighborhoods: the 2-opt* and cross exchange inter-route neighborhoods and the Or-opt intra-route neighborhood. When a local optimum is encountered, the current solution is perturbed by applying a single, random cross exchange move to the solution. The algorithm is used to solve VRPTW instances by allowing time window violations and adjusting penalties appropriately.

Cordeau and Maischberger [24] introduced an iterated local search algorithm that relies on an updated version of the tabu search algorithm of Cordeau, Laporte, and Mercier [22, 23] as the local search algorithm. The tabu search algorithm allows solutions that are infeasible with respect to both the time windows and vehicle capacity. It uses a simple neighborhood based on customer relocation. The perturbation step is inspired by LNS and consists of removing a cluster of customers and reinserting them in a random order. Each customer is reinserted in a route using a cheapest-insertion policy.

Another iterated local search algorithm was proposed by Lim and Zhang [75]. Unlike the above iterated local search algorithms, their algorithm does not allow infeasible solutions but yields, nevertheless, good results. In our opinion, the Lim and Zhang algorithm is, however, more complex than the others.

5.4.2.2 ▪ Large Neighborhood Search

The original LNS heuristic by Shaw [111] already solved the VRPTW with good results on a reduced set of instances, but it was only with the work of Bent and Van Hentenryck [11] that the method's ability for obtaining good solutions to the VRPTW was fully revealed. Bent and Van Hentenryck used a two-phase approach. In the first phase a simulated annealing heuristic based on traditional neighborhoods is applied to minimize the number of vehicles, while, in the second phase, LNS is invoked to minimize travel costs. In the LNS algorithm, solutions are destroyed by removing groups of related customers, while the repair mechanism uses a Branch-and-Bound algorithm to reinsert the customers. The Branch-and-Bound algorithm is truncated using *limited discrepancy search* (Harvey and Ginsberg [52]) in order to keep the computational times low. Only improving solutions are accepted.

Pisinger and Ropke [94] solved the VRPTW (and other VRP variants) by transforming each instance into a pickup-and-delivery problem with time windows that is then solved using the LNS heuristic developed in Ropke and Pisinger [106]. Several destroy and repair operators are implemented, and an adaptive mechanism is used to favor the application of the most successful destroy/repair operators in the previous iterations. Because of this feature the heuristic is termed *Adaptive Large Neighborhood Search* (ALNS). The repair operators are based on quick, greedy heuristics. Simulated annealing is used as an outer metaheuristic framework to decide whether deteriorating solutions should be accepted.

Another paper that proposes an LNS heuristic relying on quick (greedy) repair methods is due to Mester and Bräysy [80]. They embedded the LNS moves within an evolution strategy metaheuristic (see Beyer and Schwefel [12]) and also exploited traditional neighborhoods during the search.

Prescott-Gagnon, Desaulniers, and Rousseau [98] developed an ALNS heuristic based on a much stronger repair operator. Solution destruction is performed using four removal heuristics, while the repair step uses heuristic Branch-and-Price, adapted from the

Branch-and-Cut-and-Price algorithm of Desaulniers, Lessard, and Hadjar [31]. In order to make each repair step sufficiently fast the original algorithm is modified in several ways: (1) the column generation subproblem is only solved heuristically using a tabu search algorithm; (2) for large instances, the column generation process is stopped prematurely if the master problem objective value has not improved for a given number of iterations; (3) no cuts are generated; and (4) when branching is required, the value of the largest fractional variable in the master problem is permanently fixed to one without any backtracking possibility. To diversify the search, all feasible solutions generated by the repair step are accepted, even if the new solution is worse than the current solution.

5.4.3 ▪ Population-Based Search

A number of metaheuristics are based on the idea of maintaining a pool of solutions, called a *population*, that evolves at each iteration of the solution process. Unlike in single trajectory search, new solutions are derived from a population of solutions that offers diversity in itself. Below, we survey the recent works on the VRPTW based on evolutionary and path relinking algorithms, two families of population-based search heuristics.

5.4.3.1 ▪ Evolutionary Algorithms

Evolutionary algorithms combine solutions from the current population to produce offsprings. The most fit of them are then retained to update the population. Examples of this type of algorithms are *genetic algorithms* (Holland [56] and Reeves [101]) and *memetic algorithms* (Moscato [81] and Moscato and Cotta [82]).

Memetic algorithms hybridize genetic algorithms with local search and potentially problem-specific algorithms. The pseudo-code of a simple memetic algorithm is sketched in Algorithm 5.4. The size of the initial population created in line 1 is obviously an important parameter. Lines 2 to 6 constitute the main loop of the algorithm which can be stopped according to various criteria based on the number of iterations performed, the elapsed time, or a measure of convergence. In line 3, new solutions forming set S are constructed by combining existing solutions from P. This operation is called a *crossover*. In line 4, mutation is applied to a subset of the solutions in S to create a new subset of offsprings S', perturbing in this way the offspring set. The mutation step is not included in all memetic algorithms. In line 5, local search is executed on each offspring solution in S' to generate a set S'' of improved solutions. Sometimes this local search step is seen as a mutation operation. The solutions in S'' are then used to update the population P (line 6). A simple way to perform this update consists of keeping the $|P|$ best solutions from the set $P \cup S''$, but a more advanced procedure is typically used to favor diversity in population P. A classical genetic algorithm is obtained by removing the local search step (line 5) in Algorithm 5.4.

Algorithm 5.4 Memetic algorithm

```
1: Create an initial pool $P$ of solutions
2: **while** stopping criterion is not met **do**
3:     $S = \text{crossover}(P)$
4:     $S' = \text{mutation}(S)$
5:     $S'' = \text{localsearch}(S')$
6:     $P = \text{updatepool}(P, S'')$
7: return best solution observed during the search
```

A vital component in memetic algorithms is the crossover algorithm. Below we give an overview of the crossover methods used in the most successful evolutionary algorithms for the VRPTW.

EAX Crossover. The EAX crossover algorithm is based on a strong crossover algorithm for the TSP that was first introduced by Nagata and Kobayashi [89] and has been at the foundation of the best evolutionary algorithms for the TSP since then (see, e.g., Nagata and Kobayashi [90]). The EAX algorithm was later adapted to the CVRP (Nagata [84] and Nagata and Bräysy [86]) and, finally, to the VRPTW by Nagata, Bräysy, and Dullaert [88]. In five steps, the EAX algorithm combines two solutions α and β defined by the arc sets A_α and A_β, respectively. Step 1 constructs a graph $G_{\alpha\beta}$ containing the arcs $A_{\alpha\beta} = (A_\alpha \cup A_\beta) \setminus (A_\alpha \cap A_\beta)$. Step 2 partitions $A_{\alpha\beta}$ into a number of *$\alpha\beta$-cycles*. An $\alpha\beta$-cycle is a cycle in $G_{\alpha\beta}$ alternating between arcs in A_α and arcs in A_β and such that all arcs in A_α have an opposite orientation to that of the arcs in A_β. Step 3 builds an *E-set* by combining *$\alpha\beta$-cycles*, the simplest strategy being to select a single *$\alpha\beta$-cycle* as an E-set. Step 4 creates an offspring solution defined by the arcs in $(A_\alpha \setminus (E \cap A_\alpha)) \cup (E \cap A_\beta)$, where E are the arcs in the selected E-set. Because this offspring solution may contain subtours, step 5 eliminates subtours one by one by deleting some arcs in the subtours and adding others to connect the resulting pieces. Notice that the built offspring is dependent on the choices made in steps 2, 3, and 5. Furthermore, it does not necessarily satisfy the time-window and capacity constraints. To obtain a feasible solution, a repair procedure performing 2-opt*, Or-exchange, node relocation, and node exchange moves is applied. Solutions that remain infeasible after the repair step are discarded, while feasible solutions are improved using local search. The EAX crossover was later used by Błocho and Czech [13], who embedded it in a parallel algorithm.

OX Crossover and Giant Tour Representation. The OX crossover is a classic crossover for permutation-based solution representations (see, e.g., Falkenauer and Bouffouix [39]). Figure 5.7 illustrates this crossover for permutations of the numbers from 1 to 9. One chooses two random numbers i and j and creates an offspring permutation by copying the piece of parent 1 from position i to j into position i to j of the offspring. The remaining part of the offspring is found by copying elements of parent 2, not already in the offspring, into position $j+1$ and forward (in a circular fashion), respecting the order of parent 2.

position	1	2	3	4	5	6	7	8	9
Parent 1	4	3	8	2	9	6	5	7	1
Parent 2	5	7	3	2	8	4	9	1	6
offspring	3	4	8	2	9	6	1	5	7

Figure 5.7. *OX crossover example with $i = 3$ and $j = 6$.*

In order to apply the OX crossover for the VRPTW, one needs to define a mapping from a solution to a permutation and back again. To do so for the CVRP, Prins [99] proposed a mapping from a solution to a permutation based on a *giant tour representation* of a solution. This representation selects an ordering of the routes in the solution and then lists the customers of the first route (in order of appearance), the customers of the second route, and so on. To transform a permutation back into a CVRP solution, Prins suggested a *split* procedure (introduced by Beasley [10]) that cuts the permutation into a number of segments, each constituting a route. The cutting points are determined optimally (that is, they must minimize the total cost of the resulting solution) in polynomial time using a shortest path calculation in an auxiliary network. Recently, Vidal et al. [120] applied the same crossover operator to the VRPTW with great success. Their algorithm allows the split procedure to generate infeasible solutions by adding penalized arcs in the auxiliary network corresponding to infeasible routes. A local search heuristic based on traditional neighborhoods can be called with a certain probability to try to repair the infeasible solutions. Both feasible and infeasible solutions are kept separately in the solution pool.

Multiparent Recombination Repoussis, Tarantilis, and Ioannou [102] took a different approach to obtaining offspring solutions. Their operator produces an offspring from a single parent by taking the entire population into account. In each generation the algorithm counts the number of occurrences m_{ij} of each arc (i,j) in the entire population. An arc is considered promising if it occurs in more than θ solutions in the population. The threshold θ is calculated in each generation based on the current diversity of the population. If the population is diverse, θ is lower than if the population contains similar solutions. Offsprings are generated from a parent solution using an LNS algorithm. The destroy operator removes customers adjacent to arcs (i,j) for which $m_{ij} < \theta$. The customers are reinserted by a greedy randomized heuristic. The generated offsprings are then improved using traditional neighborhood searches in two phases: tabu search is applied in the vehicle number minimization phase, whereas guided local search aims at distance minimization.

5.4.3.2 ▪ Path Relinking

Hashimoto and Yagiura [53] developed a path relinking algorithm for the VRPTW that works with a pool of solutions and allows infeasible solutions which are penalized. The penalty weights are controlled adaptively depending on whether feasibility is easy or hard to attain. In each outer iteration of the algorithm, two solutions σ_A and σ_B from the pool are selected randomly. The algorithm transforms σ_A into σ_B by a series of 2-opt* and Or-opt moves. This generates a number of solutions in between the σ_A and σ_B solutions. In the inner loop, some of these solutions are improved using 2-opt*, cross exchange, and Or-opt neighborhoods. Certain generated solutions are added to the solution pool if they improve the quality of the pool, taking into account total infeasibility, time infeasibility, and capacity infeasibility.

5.4.4 ▪ Metaheuristic Overview

This section provides an overview of the features of the heuristics cited above. The following abbreviations identify them: BC12: Błocho and Czech [13], BVH04: Bent and Van Hentenryck [11], CM12: Cordeau and Maischberger [24], HY08: Hashimoto and Yagiura [53], HYI08: Hashimoto, Yagiura, and Ibaraki [54], IINSUY08: Ibaraki et al. [57], LZ07: Lim and Zhang [75], MB05: Mester and Bräysy [80], NB09: Nagata and Bräysy [86], NBD10: Nagata, Bräysy, and Dullaert [88], PDR09: Prescott-Gagnon, Desaulniers, and Rousseau [98], PR07: Pisinger and Ropke [94], RTI09: Repoussis, Tarantilis, and Ioannou [102], and VCGP13: Vidal et al. [120].

Table 5.2 compares the features of these heuristics. The columns in this table indicate the neighborhoods used as presented in Sections 5.4.1.1 and 5.4.1.2, including the maximum string size considered in the cross-exchange and path relocation neighborhoods

Table 5.2. *Metaheuristic comparison.*

	Intra			Inter					Metaheuristic							Recomb.			Infeas.			
	Or-Opt	2-opt	2-opt*	Cross-exchange	Path relocation	LNS	Neighborhood filtering	Parallel metaheuristic	ILS	SA	TS	GLS	PR	EA	EAX	OX+split	Multiparent	TW	Capacity	Customers	Richer than VRPTW	Vehicle minimization
BVH04	X	X	X	1	1	X				X												X
MB05				1	1	X						X		X								X
PR07						X				X										X	X	X
LZ07	X	X	X	X		CE			X		X									X	X	X
HYI08	X		X	3					X									A			X	
HY08	X		X	3			X						X					A	X			
IINSUY08	X	X	X	3	3		X		X									A	X		X	X
RTI09		X	X	4	1	(X)					X	X		X			X					X
PDR09						X					X									X		X
NB09			X	1	1		X					X						B	X	X		X
NBD10	X		X	1	1		X							X	X			B	X			X
BC12								X						X	X			A	X			X
CM12					1			X	X		X							A	X		X	
VCGP13		X	X	2	2		X							X		X		B	X		X	X

(columns 2 to 7); if a heuristic filtering of the neighborhoods was used (column 8); if parallel processing can be exploited (column 9); which metaheuristic frameworks are used among iterated local search (ILS), simulated annealing (SA), tabu search (TS), guided local search (GLS), path relinking (PR), evolutionary algorithms or strategies (EA) (columns 10 to 15); the crossover operator used in evolutionary algorithms (columns 16 to 18); if time windows can be violated during the search (column 19), where an "A" means that the traditional way of measuring time-window violation is used, while a "B" rather refers to the method of Nagata [85]; if the capacity and the all-customers-served constraints can be violated (columns 20 and 21); if the algorithm is able to solve VRP variants other than the VRPTW (column 22); and if the algorithm can minimize the number of vehicles used (column 23). Algorithms without this last feature are dependent on being initialized with the desired number of vehicles. Notice that BC12 uses local search, but the authors have not specified which neighborhoods are used. LZ07 does not specify the maximum string size for cross exchanges: this is indicated by an "X" in the corresponding column. All heuristics with "X" in column 7 apply the destroy/repair idea from Shaw's LNS [111]. The parentheses around the "X" for RTI09 highlight a quite different usage of this destroy/repair idea. Finally, LZ07 use large neighborhoods, but of the cyclic exchange type, which explains the "CE" in column 7.

Some immediate observations are that local search is playing a part in all the chosen heuristics and that most of the algorithms allow visiting infeasible solutions.

5.4.5 ▪ Summary Computational Results for Heuristics

In this section we compare the most important heuristics from the past decade based on the results they achieved on the data sets of Solomon [113] and Gehring and Homberger [45]. Tables 5.3 and 5.4 provide computational results for the Solomon and Gehring–Homberger instances, respectively. For each pair of heuristic and instance sets, we report three numbers: the *Cumulative Number of Vehicles* (CNV) summed over all instances in the set, the *Cumulative Total Distance* (CTD), and the average time spent per instance in minutes. Computational times have been normalized to match the performance of an Intel Xeon 2.93 GHz computer. Normalization factors were found using the CINT2006 and CINT2000 benchmarks (see the Standard Performance Evaluation Corporation [115]). We acknowledge that such normalization is far from being exact. Note that some papers do not contain enough information to report average computational times, which explains the blank entries in the two tables. To report the computational time for the BC12 parallel heuristic we multiplied the number of cores used by the computational time. This is of course not completely fair because the heuristic will not be able to take advantage of all processing units at the same time, but it is the best we can do.

Table 5.3. *Results on the Solomon instances (sorted according to CNV first and CTD second).*

	NBD10	VCGP13	RTI09	PDR09	BVH04	HYI08
CNV	405	405	405	405	405	405
CTD	57187	57196	57216	57240	57273	57282
Time (min)	16.9	13.4	33.7	97.0		29.7
	PR07	LZ07	HY08	CM12	IINSUY08	
CNV	405	405	405	406	407	
CTD	57332	57368	57484	57199	57545	
Time (min)	15.3	15.6	16.2	392.8	9.9	

Table 5.4. *Gehring–Homberger results (sorted according to total CNV first and total CTD second). Columns* 200, 400, ..., 1000 *show results for each size class. The last column shows total CNV and CTD and average time.*

		200	400	600	800	1000	Overall
	CNV	694	1380	2065	2734	3417	10290
NB09	CTD	-	-	-	-	-	-
	Time (min)	40.5	80.9	121.4	161.9	202.4	121.4
	CNV	694	1381	2066	2736	3419	10296
BC12	CTD	168088	388939	792278	1354477	2056153	4759935
	Time (min)	933.1	3740.7	6175.2	7473.0	8244.9	5313.4
	CNV	694	1381	2068	2739	3420	10302
VCGP13	CTD	168092	388013	786373	1334963	2036700	4714141
	Time (min)	42.0	170.5	497.0	1075.0	1745.0	705.9
	CNV	694	1383	2068	2737	3420	10302
HY08	CTD	169070	392507	800982	1367971	2085125	4815655
	Time (min)	31.5	64.0	95.6	127.1	159.6	95.6
	CNV	694	1381	2067	2738	3424	10304
NBD10	CTD	168067	388466	789592	1357695	2045720	4749540
	Time (min)	13.8	54.6	85.3	93.1	119.1	73.2
	CNV	694	1381	2066	2739	3428	10308
RTI09	CTD	169163	395936	816326	1424321	2144830	4950576
	Time (min)	56.5	113.1	169.6	226.2	282.7	169.6
	CNV	694	1382	2068	2742	3429	10315
LZ07	CTD	169296	393695	802681	1372427	2071643	4809742
	Time (min)	55.3	175.4	383.5	752.6	1105.9	494.5
	CNV	694	1383	2069	2747	3430	10323
HYI08	CTD	171018	406109	847470	1442957	2204728	5072282
	Time (min)	59.2	118.5	177.9	237.1	296.5	177.8
	CNV	694	1385	2071	2745	3432	10327
PDR09	CTD	168556	389011	800561	1391344	2096823	4846295
	Time (min)	171.3	287.7	339.4	417.0	523.6	347.8
	CNV	694	1387	2070	2750	3431	10332
IINSUY08	CTD	170484	398938	825172	1421225	2155374	4971193
	Time (min)	19.7	39.5	59.3	79.0	98.8	59.3
	CNV	694	1385	2071	2758	3438	10346
PR07	CTD	169042	393210	807470	1358291	2110925	4838938
	Time (min)	48.4	49.6	57.5	71.3	83.6	62.1
	CNV	694	1389	2082	2765	3446	10376
MB05	CTD	168573	390386	796172	1361586	2078110	4794827
	Time (min)	2.9	6.1	14.4	52.0	215.3	58.1
	CNV	697	1394	2091	2778	3468	10428
BVH04	CTD	168502.6	410111.6	858040	1469790	2266959	5173403
	Time (min)	-	-		-	-	-

Heuristics were often tested using several parameter configurations. In Tables 5.3 and 5.4 we report the results from the best configuration (and, typically, the most time-consuming configuration) in order to make the comparison consistent. When the heuristic was executed multiple times, e.g., 10 times, on each instance, we report the best solution found in the 10 tests and the time for performing all 10 runs. Note that for BVH04 we have obtained the Gehring–Homberger results from the technical report version of Bent and Van Hentenryck [11]. For IINSUY08, we chose the ILS-1 configuration and, for PR07, we chose the results produced by letting the algorithm run for 50,000 iterations.

The results on the Solomon instances show that the best heuristics proposed are all very close to one another in terms of solution quality. Most of them produce a CNV of 405. Many of the heuristics show similar computational times, but a few methods have significantly higher running times.

Results on the Gehring and Homberger instances show more variations in CNV and average computational time. We can conclude that Nagata and Bräysy [87] obtain the best results in terms of the primary objective and that the race is very close among the best algorithms. Matching the results in Table 5.4 with the algorithm features from Table 5.2, one can observe that allowing infeasible solutions, especially time and capacity violations, are key factors to a successful metaheuristic. We also note that the population-based search methods yield high-quality solutions. Furthermore, we conclude that the Solomon instances are no longer sufficient for comparing state-of-the-art heuristics and that large-sized instances are necessary to highlight differences.

5.4.6 ▪ Other Measures of Quality

Running time and solution quality achieved are obvious performance indicators for a metaheuristic. However, they are not the only ones. As mentioned in Chapter 4 it also makes sense to look at simplicity, number of parameters and parameter sensitivity, flexibility, and robustness. Several of the heuristics that were highlighted in Chapter 4 as especially promising regarding one or more of these indicators are also able to solve the VRPTW. Among the heuristics only mentioned in this chapter we would like to highlight the iterated local search algorithms of Hashimoto, Yagiura, and Ibaraki [54] and Ibaraki et al. [57]. The core metaheuristics in these papers are very simple, but good results are obtained nevertheless. Note also that the heuristic of Prescott-Gagnon, Desaulniers, and Rousseau [98] appears to be flexible in the sense that it potentially can handle many VRP variants by making small adjustments to the subproblem that generates routes. A drawback of this heuristic is that it is rather complex to implement since it embeds a well-tuned Branch-and-Price framework.

5.5 ▪ Extensions

This section surveys the scientific literature on extensions of the VRPTW. It contains three parts. The first two are dedicated to the split-delivery VRPTW and the time-dependent VRPTW that have been studied for more than a decade now. The last part briefly comments on three extensions that were recently addressed.

5.5.1 ▪ Split Deliveries

The *Split-Delivery Vehicle Routing Problem with Time Windows* (SDVRPTW) consists of a VRPTW where the demand of each customer can be fulfilled by several vehicles and where demands greater than the vehicle capacity is allowed. The literature on the SDVRPTW is quite limited. Frizzell and Giffin [44] and Mullaseril, Dror, and Trudeau [83] developed simple construction and improvement heuristics. In the first paper the proposed heuristics are especially tailored for the problem with a special network structure. In the second paper the proposed heuristic is applied to a real-life problem of managing the trucks for distributing feed in a cattle ranch. In Ho and Haugland [55], a solution method based on tabu search is proposed. The well-known Solomon test instances are applied with modified customer demands.

Gendreau et al. [46] introduced an exact Branch-and-Cut-and-Price method. As in many VRPTW algorithms the lower bounds in the search tree are computed by a column generation algorithm. However, for the SDVRPTW, an additional difficulty is that the quantities delivered are also decision variables. These decisions on the delivered quantities are taken in the MP leading to an exponential number of constraints and a complexified ESPPRC subproblem. The algorithm succeeded at solving modified Solomon instances with up to 50 customers. Desaulniers [26] designed an exact Branch-and-Cut-and-Price method for the SDVRPTW. To avoid an exponential number of constraints in the MP due to the quantities delivered, the delivered quantities are handled in the column generation subproblem, which is a combined ESPPRC and an LP-relaxed bounded knapsack problem. Enhancements to this algorithm were proposed in Archetti, Bouchard, and Desaulniers [2]. Within one hour of computational time, the resulting algorithm succeeded at solving to optimality 262 of the 504 modified Solomon instances (namely, 168, 86, and 8 instances, with 25, 50, and 100 customers, respectively).

5.5.2 ▪ Time-Dependent Costs or Travel Times

The *Time-Dependent Vehicle Routing Problem with Time Windows* (TDVRPTW) consists of a VRPTW where the travel time or travel cost between two locations depends on the time of the day. This can occur, for example, when congestion during rush hours result in longer travel times. An important, desirable, and reasonable property for this type of problems is the *First-In-First-Out* (FIFO) property, which stipulates that a vehicle leaving a location i at any time t to go to a location j cannot arrive earlier at j than another identical vehicle that left location i before t and went directly to j.

The TDVRPTW has received very little attention in the scientific literature compared to the CVRP or the VRPTW. The first authors to deal with the TDVRP after 2000 were Ichoua, Gendreau, and Potvin [58], who developed a parallel tabu search heuristic for the case with soft time windows (that is, barring a penalty, service can start later than a time-window upper bound). Fleischmann, Gietz, and Gnutzmann [43] present a general heuristic-based framework for the implementation of time-varying travel times in various vehicle routing algorithms, including TDVRPTW. Haghani and Jung [51] propose a genetic algorithm. Hashimoto, Yagiura, and Ibaraki [54] design an iterated local search heuristic that takes into account time-dependent service times, traveling times, and traveling costs in the case with soft time windows. Donati et al. [35] propose a solution based on an ant colony heuristic. Soler, Albiach, and Martínez [112] transform the TDVRPTW through several steps into an asymmetric CVRP. They solve very small instances to optimality, but none of realistic size. Figliozzi [42] develops an iterative route construction and improvement heuristic and introduces new replicable test problems. Ghiani and Guerriero [48] exploit some properties of the model of Ichoua, Gendreau, and Potvin [58] and show that this model is quite general.

To conclude, a few heuristic methods have been designed for variants of the TDVRPTW. Only one exact method suitable for very small-sized instances was also proposed.

5.5.3 ▪ Driver Considerations, Multiple Use of Vehicles, and Multiple Time Windows

The VRPTW with driver regulations takes into account various regulations concerning the working hours, breaks, and rests of the drivers. These regulations are appearing more and more often all around the world and may differ from area to area. For this problem, Goel [50] introduced an LNS heuristic based on local search operators, Prescott-Gagnon et al. [97] developed an LNS method based on a column generation heuristic, and Kok et al. [72] proposed a dynamic programming heuristic. According to the reported computational results obtained on instances involving 100 customers over a one-week horizon, the algorithm of Prescott-Gagnon et al. outperforms the other two algorithms.

The CVRP with multiple use of vehicles is a variant of the classical CVRP that allows one to assign several routes to the same vehicle over a finite planning horizon (e.g., one work day) in a context where vehicle availability, vehicle fixed costs, or maximum working time per vehicle must be considered. Azi, Gendreau, and Potvin [3] addressed the case with time windows and designed an exact Branch-and-Price solution method. They solved instances with up to 40 customers.

Doerner et al. [34] studied the VRP with multiple interdependent time windows in which each customer may be required to be visited several times and the elapsed time between two consecutive visits must not exceed a maximum time. This VRPTW variant is inspired by a blood transportation application. The authors developed an exact algorithm and heuristic algorithms. Their results show that the heuristic algorithms find solutions reasonably close to optimal solutions in a fraction of a second.

5.6 ▪ Conclusions and Future Research Directions

The previous sections have reported a considerable progress in VRPTW methodologies in the last decade.

Concerning the exact methods the research has concentrated on stabilization of the dual variables, effective solutions of the resource constrained shortest path subproblems, introduction of new valid inequalities, and smarter Branch-and-Bound techniques. The result is that it is now possible to solve to optimality all the Solomon instances. The computational time needed has also been reduced considerably during the last decade but may, however, be still quite long.

Concerning the heuristic methods the research has focused almost solely on metaheuristics. It appears that population-based search methods, large neighborhoods, and allowing infeasible solutions during the search are important ingredients in current state-of-the-art heuristics.

We envision that research for the VRPTW will proceed. On the exact methods, research will focus on the development of better solution algorithms for solving the subproblem and on the introduction of new cuts and smarter Branch-and-Bound procedures. An important challenge is the design of an efficient separation algorithm for the 2-path cuts which are currently separated by enumerating a large number of customer subsets and solving a TSP with time windows for each subset. Even more challenging is the development of a separation algorithm for the 3-path cuts that may greatly help to close the integrality gap. With such research endeavors, we can foresee that more of the Gehring and Homberger instances will be solved to optimality.

We expect that research on heuristics will continue but at a slightly slower pace compared to what was witnessed in the last decade. This prediction is based on the observation that most research currently aims at solving CVRP variants that are richer than the classical VRPTW. In terms of future research directions we believe that we still have not seen the full potential of methods combining mathematical programming and metaheuristics. We also think that the development of heuristics that can match the solution quality of the best methods of today in a fraction of the time would be of interest for practical applications where interactive planning seems more widespread than batch planning.

As in the past, the future exact and heuristic methods for the VRPTW will also serve as a basis for solving more complex VRPs. Finally, we think that the VRPTW will remain a privileged platform for validating the usefulness of new research ideas for exact and heuristic methods due to the high-level maturity of the existing VRPTW solution methods.

Bibliography

[1] R. K. Ahuja, Ö. Ergun, J. Orlin, and A. Punnen, *A survey of very large-scale neighborhood search techniques*, Discrete Applied Mathematics, 123 (2002), pp. 75–102.

[2] C. Archetti, M. Bouchard, and G. Desaulniers, *Enhanced branch-and-price-and-cut for vehicle routing with split deliveries and time windows*, Transportation Science, 45 (2011), pp. 285–298.

[3] N. Azi, M. Gendreau, and J.-Y. Potvin, *An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles*, European Journal of Operational Research, 202 (2010), pp. 756–763.

[4] E. K. Baker and S. Rushinek, *Large scale implementation of a time oriented vehicle scheduling model*, Technical Report, US Department of Transportation, Urban Mass Transit Administration, Washington, D.C., 1982.

[5] R. Baldacci, E. Bartolini, A. Mingozzi, and R. Roberti, *An exact solution framework for a broad class of vehicle routing problems*, Computational Management Science, 7 (2010), pp. 229–268.

[6] R. Baldacci, A. Mingozzi, and R. Roberti, *New route relaxation and pricing strategies for the vehicle routing problem*, Operations Research, 59 (2011), pp. 1269–1283.

[7] ———, *Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints*, European Journal of Operational Research, 218 (2012), pp. 1–6.

[8] J. Bard, G. Kontoravdis, and G. Yu, *A branch-and-cut procedure for the vehicle routing problem with time windows*, Transportation Science, 36 (2002), pp. 250–269.

[9] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance, *Branch-and-price: Column generation for solving huge integer programs*, Operations Research, 46 (1998), pp. 316–329.

[10] J. E. Beasley, *Route first — cluster second methods for vehicle routing*, Omega, 11 (1983), pp. 403–408.

[11] R. Bent and P. Van Hentenryck, *A two-stage hybrid local search for the vehicle routing problem with time windows*, Transportation Science, 38 (2004), pp. 515–530.

[12] H.-G. Beyer and H.-P. Schwefel, *Evolution strategies—A comprehensive introduction*, Natural Computing, 1 (2002), pp. 3–52.

[13] M. Błocho and Z. J. Czech, *A parallel memetic algorithm for the vehicle routing problem with time windows*, Parallel Computing, submitted, 2012.

[14] N. Boland, J. Dethridge, and I. Dumitrescu, *Accelerated label setting algorithms for the elementary resource constrained shortest path problem*, Operations Research Letters, 34 (2006), pp. 58–68.

[15] O. Bräysy, *A reactive variable neighborhood search for the vehicle-routing problem with time windows*, INFORMS Journal on Computing, 15 (2003), pp. 347–368.

[16] O. BRÄYSY AND M. GENDREAU, *Vehicle routing problem with time windows, part* I*: Route construction and local search algorithms*, Transportation Science, 39 (2005), pp. 104–118.

[17] ——, *Vehicle routing problem with time windows, part* II*: Metaheuristics*, Transportation Science, 39 (2005), pp. 119–139.

[18] A. CHABRIER, *Vehicle routing problem with elementary shortest path based column generation*, Computers & Operations Research, 33 (2006), pp. 2972–2990.

[19] N. CHRISTOFIDES, A. MINGOZZI, AND P. TOTH, *State-space relaxation procedures for the computation of bounds to routing problems*, Networks, 11 (1981), pp. 145–164.

[20] W. COOK AND J. L. RICH, *A parallel cutting plane algorithm for the vehicle routing problem with time windows*, Technical Report, Computational and Applied Mathematics, Rice University, Houston, TX, 1999.

[21] J.-F. CORDEAU, G. DESAULNIERS, J. DESROSIERS, M. M. SOLOMON, AND F. SOUMIS, *VRP with time windows*, in The Vehicle Routing Problem, P. Toth and D. Vigo, eds., SIAM, Philadelphia, 2002, ch. 7, pp. 157–193.

[22] J.-F. CORDEAU, G. LAPORTE, AND A. MERCIER, *A unified tabu search heuristic for vehicle routing problems with time windows*, Journal of the Operational Research Society, 52 (2001), pp. 928–936.

[23] ——, *Improved tabu search algorithm for the handling of route duration constraints in vehicle routing problems with time windows*, Journal of the Operational Research Society, 55 (2004), pp. 542–546.

[24] J.-F. CORDEAU AND M. MAISCHBERGER, *A parallel iterated tabu search heuristic for vehicle routing problems*, Computers & Operations Research, 39 (2012), pp. 2033–2050.

[25] G. B. DANTZIG AND P. WOLFE, *The decomposition algorithm for linear programming*, Operations Research, 8 (1960), pp. 101–111.

[26] G. DESAULNIERS, *Branch-and-price-and-cut for the split delivery vehicle routing problem with time windows*, Operations Research, 58 (2010), pp. 179–192.

[27] G. DESAULNIERS, J. DESROSIERS, I. IOACHIM, M. M. SOLOMON, F. SOUMIS, AND D. VILLENEUVE, *A unified framework for deterministic time constrained vehicle routing and crew scheduling problems*, in Fleet Management and Logistics, T. Crainic and G. Laporte, eds., Kluwer, Norwell, MA, 1998, pp. 57–93.

[28] G. DESAULNIERS, J. DESROSIERS, AND M. M. SOLOMON, *Column generation*, Springer, New York, 2005.

[29] G. DESAULNIERS, J. DESROSIERS, AND S. SPOORENDONK, *The vehicle routing problem with time windows: State-of-the-art exact solution methods*, in Wiley Encyclopedia of Operations Research and Management Science, Vol. 8, J. Cochrane, ed., Wiley, New York, 2010, pp. 5742–5749.

[30] ——, *Cutting planes for branch-and-price algorithms*, Networks, 58 (2011), pp. 301–310.

[31] G. DESAULNIERS, F. LESSARD, AND A. HADJAR, *Tabu search, generalized k-path inequalities, and partial elementarity for the vehicle routing problem with time windows*, Transportation Science, 42 (2008), pp. 387–404.

[32] M. DESROCHERS, J. DESROSIERS, AND M. M. SOLOMON, *A new optimization algorithm for the vehicle routing problem with time windows*, Operations Research, 40 (1992), pp. 342–354.

[33] J. DESROSIERS, Y. DUMAS, F. SOUMIS, AND M. M. SOLOMON, *Time constrained routing and scheduling*, in Network Routing, vol. 8 of Handbooks in Operations Research and Management Science, M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, ed., Elsevier Science, Amsterdam, 1995, pp. 35–139.

[34] K. DOERNER, M. GRONALT, R. HARTL, G. KIECHLE, AND M. REIMANN, *Exact and heuristic algorithms for the vehicle routing problem with multiple interdependent time windows*, Computers & Operations Research, 35 (2008), pp. 3034–3048.

[35] A. V. DONATI, R. MONTEMANNI, N. CASAGRANDE, A. E. RIZZOLI, AND L. M. GAMBARDELLA, *Time dependent vehicle routing problem with a multi ant colony system*, European Journal of Operational Research, 185 (2008), pp. 1174–1191.

[36] M. DROR, *Note on the complexity of the shortest path models for column generation in VRPTW*, Operations Research, 42 (1994), pp. 977–979.

[37] Y. DUMAS, F. SOUMIS, AND J. DESROSIERS, *Optimizing the schedule for a fixed vehicle path with convex inconvenience costs*, Transportation Science, 24 (1990), pp. 145–152.

[38] I. DUMITRESCU, *Constrained Path and Cycle Problems*, PhD thesis, Department of Mathematics and Statistics, The University of Melbourne, Australia, 2002.

[39] E. FALKENAUER AND S. BOUFFOUIX, *A genetic algorithm for job shop*, in Proceedings of the 1991 IEEE International Conference on Robotics and Automation, 1991, pp. 824–829.

[40] D. FEILLET, P. DEJAX, M. GENDREAU, AND C. GUEGUEN, *An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems*, Networks, 44 (2004), pp. 216–229.

[41] D. FEILLET, M. GENDREAU, AND L.-M. ROUSSEAU, *New refinements for the solution of vehicle routing problems with branch and price*, INFOR, 45 (2007), pp. 239–256.

[42] M. A. FIGLIOZZI, *The time dependent vehicle routing problem with time windows: Benchmark problems, an efficient solution algorithm, and solution characteristics*, Transportation Research Part E: Logistics and Transportation Review, 48 (2012), pp. 616–636.

[43] B. FLEISCHMANN, M. GIETZ, AND S. GNUTZMANN, *Time-varying travel times in vehicle routing*, Transportation Science, 38 (2004), pp. 160–173.

[44] P. W. FRIZZELL AND J. W. GIFFIN, *The split delivery vehicle scheduling problem with time windows and grid network distances*, Computers & Operations Research, 22 (1995), pp. 655–667.

[45] H. GEHRING AND J. HOMBERGER, *A parallel two-phase metaheuristic for routing problems with time windows*, Asia-Pacific Journal of Operational Research, 18 (2001), pp. 35–47.

[46] M. GENDREAU, P. DEJAX, D. FEILLET, AND C. GUEGUEN, *Vehicle routing with time windows and split deliveries*, Technical Report 2006-851, Laboratoire Informatique d'Avignon, Avignon, France, 2006.

[47] M. GENDREAU AND C. D. TARANTILIS, *Solving large-scale vehicle routing problems with time windows: The state of the art*, Technical Report 2010-04, CIRRELT, Montréal, Canada, 2010.

[48] G. GHIANI AND E. GUERRIERO, *A note on the Ichoua et al.* (2003) *travel time model*, Technical Report, Optimization Online, January 2012.

[49] P. C. GILMORE AND R. E. GOMORY, *A linear programming approach to the cutting stock problem*, Operations Research, 9 (1961), pp. 849–859.

[50] A. GOEL, *Vehicle routing and scheduling with drivers' working hours*, Transportation Science, 43 (2009), pp. 17–26.

[51] A. HAGHANI AND S. JUNG, *A dynamic vehicle routing problem with time-dependent travel times*, Computers & Operations Research, 32 (2005), pp. 2959–2986.

[52] W. D. HARVEY AND M. L. GINSBERG, *Limited discrepancy search*, in Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, Montréal, Canada, 1995.

[53] H. HASHIMOTO AND M. YAGIURA, *A path relinking approach with an adaptive mechanism to control parameters for the vehicle routing problem with time windows*, Lecture Notes in Computer Science, 4972 (2008), pp. 254–265.

[54] H. HASHIMOTO, M. YAGIURA, AND T. IBARAKI, *An iterated local search algorithm for the time-dependent vehicle routing problem with time windows*, Discrete Optimization, 5 (2008), pp. 434–456.

[55] S. C. HO AND D. HAUGLAND, *A tabu search heuristic for the vehicle routing problem with time windows and split deliveries*, Computers & Operations Research, 31 (2004), pp. 1947–1964.

[56] J. H. HOLLAND, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*, MIT Press, Cambridge, MA, 1992.

[57] T. IBARAKI, S. IMAHORI, K. NONOBE, K. SOBUE, T. UNO, AND M. YAGIURA, *An iterated local search algorithm for the vehicle routing problem with convex time penalty functions*, Discrete Applied Mathematics, 156 (2008), pp. 2050–2069.

[58] S. ICHOUA, M. GENDREAU, AND J.-Y. POTVIN, *Vehicle dispatching with time-dependent travel times*, European Journal of Operational Research, 144 (2003), pp. 379–396.

[59] G. IOANNOU, M. KRITIKOS, AND G. PRASTACOS, *A greedy look-ahead heuristic for the vehicle routing problem with time windows*, Journal of the Operational Research Society, 52 (2001), pp. 523–537.

[60] S. IRNICH, *A unified modeling and solution framework for vehicle routing and local search-based metaheuristics*, INFORMS Journal on Computing, 20 (2008), pp. 270–287.

[61] S. IRNICH AND G. DESAULNIERS, *Shortest path problems with resource constraints*, in Column Generation, G. Desaulniers, J. Desrosiers, and M. Solomon, eds., Springer, New York, 2005, ch. 2, pp. 33–65.

[62] S. IRNICH, G. DESAULNIERS, J. DESROSIERS, AND A. HADJAR, *Path reduced costs for eliminating arcs*, INFORMS Journal on Computing, 22 (2010), pp. 297–313.

[63] S. IRNICH AND D. VILLENEUVE, *The shortest path problem with resource constraints and k-cycle elimination for $k \geq 3$*, INFORMS Journal on Computing, 18 (2006), pp. 391–406.

[64] M. JEPSEN, B. PETERSEN, S. SPOORENDONK, AND D. PISINGER, *Subset-row inequalities applied to the vehicle-routing problem with time windows*, Operations Research, 56 (2008), pp. 497–511.

[65] B. KALLEHAUGE, *Formulations and exact algorithms for the vehicle routing problem with time windows*, Computers & Operations Research, 35 (2008), pp. 2307–2330.

[66] B. KALLEHAUGE, N. BOLAND, AND O. B. G. MADSEN, *Path inequalities for the vehicle routing problem with time windows*, Networks, 49 (2007), pp. 273–293.

[67] B. KALLEHAUGE, J. LARSEN, AND O. B. G. MADSEN, *Lagrangian duality applied to the vehicle routing problem with time windows*, Computers & Operations Research, 33 (2006), pp. 1464–1487.

[68] G. A. P. KINDERVATER AND M. W. P. SAVELSBERGH, *Vehicle routing: Handling edge exchanges*, in Local Search in Combinatorial Optimization, E. Aarts and J. Lenstra, eds., Wiley, Chichester, UK, 1997, pp. 337–360.

[69] K. KNIGHT AND J. HOFER, *Vehicle scheduling with timed and connected calls: A case study*, Operational Research Quarterly, 19 (1968), pp. 299–310.

[70] N. KOHL, J. DESROSIERS, O. B. G. MADSEN, M. M. SOLOMON, AND F. SOUMIS, *2-Path cuts for the vehicle routing problem with time windows*, Transportation Science, 33 (1999), pp. 101–116.

[71] N. KOHL AND O. B. G. MADSEN, *An optimization algorithm for the vehicle routing problem with time windows based on Lagrangean relaxation*, Operations Research, 45 (1997), pp. 395–406.

[72] A. L. KOK, C. M. MEYER, H. KOPFER, AND J. M. J. SCHUTTEN, *A dynamic programming heuristic for the vehicle routing problem with time windows and European community social legislation*, Transportation Science, 44 (2010), pp. 442–454.

[73] J. LARSEN, *Parallellization of the vehicle routing problem with time windows*, PhD thesis IMM-PHD-1999-62, Department of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark, 1999.

[74] A. N. LETCHFORD AND J. J. SALAZAR GONZÁLEZ, *Projection results for vehicle routing*, Mathematical Programming, Ser. B, 105 (2006), pp. 251–274.

[75] A. LIM AND X. ZHANG, *A two-stage heuristic with ejection pools and generalized ejection chains for the vehicle routing problem with time windows*, INFORMS Journal on Computing, 19 (2007), pp. 443–457.

[76] H. R. LOURENÇO, O. C. MARTIN, AND T. STÜTZLE, *Iterated local search: Framework and applications*, in Handbook of Metaheuristics, M. Gendreau and J.-Y. Potvin, eds., Springer, New York, 2nd ed., 2010, pp. 363–397.

[77] M. E. LÜBBECKE AND J. DESROSIERS, *Selected topics in column generation*, Operations Research, 53 (2005), pp. 1007–1023.

[78] J. LYSGAARD, *Reachability cuts for the vehicle routing problem with time windows*, European Journal of Operational Research, 175 (2006), pp. 210–223.

[79] O. B. G. MADSEN, *Optimal scheduling of trucks - A routing problem with tight due times for delivery*, in Optimization Applied to Transportation Systems, H. Strobel, R. Genser and M. Etschmaier, eds., IIASA, International Institute for Applied System Analysis, Laxenburgh, 1976, pp. 126–136.

[80] D. MESTER AND O. BRÄYSY, *Active guided evolution strategies for large-scale vehicle routing problems with time windows*, Computers & Operations Research, 32 (2005), pp. 1593–1614.

[81] P. MOSCATO, *On evolution, search, optimization, genetic algorithms and martial arts: Toward memetic algorithms*, Caltech Concurrent Computation Program 826, California Institute of Technology, 1989.

[82] P. MOSCATO AND C. COTTA, *A modern introduction to memetic algorithms*, in Handbook of Metaheuristics, M. Gendreau and J.-Y. Potvin, eds., Springer, New York, 2nd ed., 2010, pp. 141–183.

[83] P. A. MULLASERIL, M. DROR, AND P. TRUDEAU, *Split-delivery routing heuristics in livestock feed distribution*, Journal of the Operational Research Society, 48 (1997), pp. 107–116.

[84] Y. NAGATA, *Edge assembly crossover for the capacitated vehicle routing problem*, Lecture Notes in Computer Science, 4446 (2007), pp. 142–152.

[85] ——, *Efficient evolutionary algorithm for the vehicle routing problem with time windows: edge assembly crossover for the VRPTW*, in Proceedings of the 2007 IEEE Congress on Evolutionary Computation, 2007. CD-ROM.

[86] Y. NAGATA AND O. BRÄYSY, *Edge assembly based memetic algorithm for the capacitated vehicle routing problem*, Networks, 54 (2009), pp. 205–215.

[87] ——, *A powerful route minimization heuristic for the vehicle routing problem with time windows*, Operations Research Letters, 37 (2009), pp. 333–338.

[88] Y. NAGATA, O. BRÄYSY, AND W. DULLAERT, *A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows*, Computers & Operations Research, 37 (2010), pp. 724–737.

[89] Y. NAGATA AND S. KOBAYASHI, *Edge assembly crossover: A high-power genetic algorithm for the travelling salesman problem*, in Proceedings of the 7th International Conference on Genetic Algorithms, 1997, pp. 450–457.

[90] ——, *A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem*, INFORMS Journal on Computing, 25 (2013), pp. 346–363.

[91] G. L. NEMHAUSER AND L. A. WOLSEY, *Integer and Combinatorial Optimization*, John Wiley & Sons, New York, 1988.

[92] M. W. PADBERG AND G. RINALDI, *Optimization of a* 532*-city symmetric traveling salesman problem by branch-and-cut*, Operations Research Letters, 6 (1987), pp. 1–7.

[93] B. PETERSEN, D. PISINGER, AND S. SPOORENDONK, *Chvátal-Gomory rank-*1 *cuts used in a Dantzig-Wolfe decomposition of the vehicle routing problem with time windows*, in The Vehicle Routing Problem: Latest Advances and New Challenges, B. L. Golden, S. Raghavan, and E. A. Wasil, eds., vol. 43 of Operations Research/Computer Science Interfaces Series, Springer, New York, 2008, pp. 397–419.

[94] D. PISINGER AND S. ROPKE, *A general heuristic for vehicle routing problems*, Computers & Operations Research, 34 (2007), pp. 2403–2435.

[95] ——, *Large neighborhood search*, in Handbook of Metaheuristics, M. Gendreau and J.-Y. Potvin, eds., Springer-Verlag, Berlin, 2nd ed., 2010, ch. 13, pp. 399–419.

[96] J.-Y. POTVIN, *State-of-the art review: Evolutionary algorithms for vehicle routing*, INFORMS Journal on Computing, 21 (2009), pp. 518–548.

[97] E. PRESCOTT-GAGNON, G. DESAULNIERS, M. DREXL, AND L.-M. ROUSSEAU, *European driver rules in vehicle routing with time windows*, Transportation Science, 44 (2010), pp. 455–473.

[98] E. PRESCOTT-GAGNON, G. DESAULNIERS, AND L.-M. ROUSSEAU, *A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows*, Networks, 54 (2009), pp. 190–204.

[99] C. PRINS, *A simple and effective evolutionary algorithm for the vehicle routing problem*, Computers & Operations Research, 31 (2004), pp. 1985–2002.

[100] H. PULLEN AND M. WEBB, *A computer application to a transport scheduling problem*, Computer Journal, 10 (1967), pp. 10–13.

[101] C. R. REEVES, *Genetic algorithms*, in Handbook of Metaheuristics, M. Gendreau and J.-Y. Potvin, eds., Springer, New York, 2nd ed., 2010, pp. 109–139.

[102] P. P. REPOUSSIS, C. D. TARANTILIS, AND G. IOANNOU, *Arc-guided evolutionary algorithm for the vehicle routing problem with time windows*, IEEE Transactions on Evolutionary Computation, 13 (2009), pp. 624–647.

[103] G. RIGHINI AND M. SALANI, *Symmetry helps: bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints*, Discrete Optimization, 3 (2006), pp. 255–273.

[104] ——, *New dynamic programming algorithms for the resource constrained elementary shortest path problem*, Networks, 51 (2008), pp. 155–209.

[105] S. ROPKE, *Branching decisions in branch-and-cut-and-price algorithms for vehicle routing problems*, International Workshop on Column Generation, June 10–13, 2012, Bromont, Canada.

[106] S. Ropke and D. Pisinger, *An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows*, Transportation Science, 40 (2006), pp. 455–472.

[107] L.-M. Rousseau, M. Gendreau, and D. Feillet, *Interior point stabilization for column generation*, Operations Research Letters, 35 (2007), pp. 660–668.

[108] L.-M. Rousseau, M. Gendreau, and G. Pesant, *Solving VRPTWs with constraint programming based column generation*, Annals of Operations Research, 130 (2004), pp. 119–216.

[109] M. W. P. Savelsbergh, *Local search in routing problems with time windows*, Annals of Operations Research, 4 (1985), pp. 285–305.

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

[111] P. Shaw, *Using constraint programming and local search methods to solve vehicle routing problems*, in CP-98 (Fourth International Conference on Principles and Practice of Constraint Programming), vol. 1520 of Lecture Notes in Computer Science, Springer, Berlin, 1998, pp. 417–431.

[112] D. Soler, J. Albiach, and E. Martínez, *A way to optimally solve a time-dependent vehicle routing problem with time windows*, Operations Research Letters, 37 (2009), pp. 37–42.

[113] M. M. Solomon, *Algorithms for the vehicle routing and scheduling problem with time window constraints*, Operations Research, 35 (1987), pp. 254–265.

[114] S. Spoorendonk and G. Desaulniers, *Clique inequalities applied to the vehicle routing problem with time windows*, INFOR, 48 (2010), pp. 53–67.

[115] Standard Performance Evaluation Corporation. *SPEC CPU2006 benchmark suite*. http://www.spec.org, 2006.

[116] P. M. Thompson and H. N. Psaraftis, *Cyclic transfer algorithms for multivehicle routing and scheduling problems*, Operations Research, 41 (1993), pp. 935–946.

[117] P. Toth and D. Vigo, eds., *The Vehicle Routing Problem*, SIAM, Philadelphia, 2002.

[118] H. W. J. M. Trienekens, *The time constrained vehicle routing problem*, Technical Report, Erasmus University, Rotterdam, The Netherlands, 1982.

[119] F. Vanderbeck, *Implementing mixed integer column generation*, in Column Generation, G. Desaulniers, J. Desrosiers, and M. M. Solomon, eds., Springer, New York, 2005, ch. 12, pp. 331–358.

[120] T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins, *A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time windows*, Computers & Operations Research, 40 (2013), pp. 475–489.