Handbook of Constraint Programming
Edited by F. Rossi, P. van Beek and T. Walsh

Chapter 23

Vehicle Routing

Philip Kilby and Paul Shaw

This chapter looks a the use of Constraint Programming on an important industrial problem: that of constructing routes for vehicles to visit a set of customers at minimum cost, such as depicted in Figure 23.1. The methods are particularly aimed at the movement of people and goods by road.

This is a very important problem. In the USA in 2001, large trucking (6 or more tyres) logged more than two billion miles. Intercity trucking accounted for more than 1 trillion ton-miles of freight moved – 28% of the total freight ton-miles for the USA [15]. The costs of such movements are huge, so even small fractions of a percent in savings can have a substantial impact at a national level. Supply Chain Optimisation has become a key area for improvement by companies across the world, and vehicle operations will often play a key role in such a system.

The importance of the topic is reflected in the research interest. The literature on the

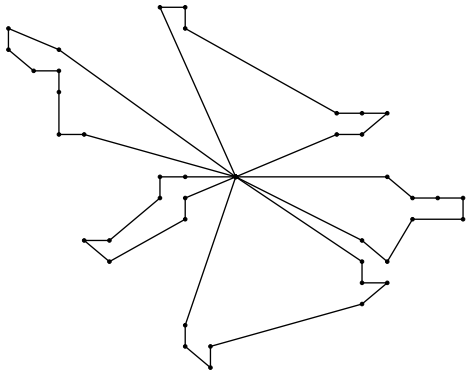


Figure 23.1: A Vehicle Routing Problem with 40 customers, 5 vehicles, and maximum 8 customers per vehicle.

topic is now very extensive – the variety of problems and solution methods explored is indicated in the surveys [111, 13, 14, 76, 35, 73].

An important factor in these problems is that no two companies will operate exactly alike – each has its own operational methods, and each brings their own constraints – like defined areas for drivers, vehicle/customer incompatibilities, and delivery time preferences. While traditional approaches capture the essence of the problem, a software system that is to deliver finished routes must be able to incorporate these myriad individual constraints.

Constraint Programming offers a valuable tool for specifying and solving these dirty, real-world problems. The ability to specify and add new constraints easily is very important.

We will begin by defining and formulating the VRP, and looking at some related problems. We then look at some of the methods employed to solve this problem – methods from the Operations Research literature, and those using a Constraint Programming formulation.

23.1 The Vehicle Routing Problem

The simplest Vehicle Routing Problem (VRP, also called the Capacitated Vehicle Routing Problem, CVRP) is defined formally as follows. The symbols defined here are summarised in Table 23.1 in Section 23.2.1.

A set of n customers is to be served by m vehicles. Each customer must be visited by exactly one vehicle. Customer i has demand r_i, and the sum of demands of customers assigned to vehicle k must be less than the vehicle capacity Q. All vehicles begin and end their route at a single depot.

The objective is to minimise the sum of travel costs, where the generalised cost of travelling from i to j is c_{ij}. Symmetric ($c_{ij} = c_{ji}$) and asymmetric variants exist. For brevity the models presented in this chapter assume symmetric costs, but are easily extended to the asymmetric case.

A number of extensions to the problem have been studied. In the VRP with heterogeneous fleet, vehicle capacities and cost of use differ. Multiple resource problems allow demands to be expressed in terms of a number of different resources l (r_i^l), and each vehicle has capacities specified in terms of each resource (Q_k^l). For example, both load and volume may be restricted in a vehicle, so must be accounted separately. A maximum dimension (for example length) may also be enforced.

The VRP with Time Windows (VRPTW) is often studied [13, 14]. A time window $[a_i, b_i]$ when delivery may commence is specified for each customer i, or multiple time windows may be specified. A vehicle arriving early is usually allowed to wait at the customer's location for the start of the time window. The addition of time constraints makes the problem much more difficult – even finding a feasible (let alone optimal) solution is $\mathcal{NP}$-hard [102].

An "open" version of the problem does not require the vehicle to return to the depot after the last customer. And finally, the multiple depot VRP allows vehicles to be housed at different depots. Usually the number of vehicles at each depot is given, although this may be a decision variable.

23.1.1 Side Constraints

A key feature of vehicle routing problems, as faced in industry, is the variety of constraints that can be added for individual companies. Examples include

- Particular vehicles may be physically constrained from visiting particular customers.
- Customers may have preferences for particular drivers.
- Drivers may have established but fuzzy delivery areas.
- A maximum time or length may be imposed on a route.
- A maximum time or distance between customers may be imposed.
- There may be a requirement to visit one customer before or after another.
- There may be incompatibilities between customers – for example their orders may not be safely carried on the same vehicle.
- Customer orders may be able to be "split", and carried on separate vehicles.
- A particular type of customer must appear on each route – for example each vehicle may be required to pick up a co-driver within the first hour, but the co-drivers are not assigned to specific vehicles *a-priori*.
- Customers may be able to be skipped at a cost.

23.1.2 Related Problems

A number of problems related to the vehicle routing problem have been studied.

The Travelling Salesman Problem (TSP) is the problem of finding the shortest path that visits a set of customers and returns to the first. It is a very well studied problem – see for example the recent book [56] or the reviews [78, 72, 64]. Given an assignment of customers to vehicles, the problem of routing the customers of a single vehicle is a TSP. The TSP with Time Windows ([39, 17]) is the analogous problem for the VRPTW – the VRPTW with a single vehicle. Many heuristics for the VRP and VRPTW work by allocating customers to vehicles, and then solving the resulting TSP or TSPTW problems.

Another related problem class is *Pickup and Delivery* (PDP) problems [83, 38]. Here, rather than goods being delivered from a depot, the goods are picked up en-route, and delivered later. *Dial-a-Ride* problems [29] are pickup and delivery problems where the cargo is people rather than goods. PDP problems have a rich set of constraints: time windows are usually present and precedence constraints ensure that goods are picked up before being delivered. In Dial-a-Ride problems, there may be constraints on maximum length of travel for a passenger, the maximum deviation, and/or the maximum number of intervening stops. The VRP with back-hauls [37] is another pickup-and-delivery type problem with the extra condition that all pickups occur after deliveries are complete. It models a situation where the vehicle is used to make a set of deliveries, and then picks up some items on the way back to the depot.

23.1.3 Reformulation

The VRP also has strong connections to classical scheduling. Several formulations for the VRP have been suggested which cast it as a Job Shop Scheduling Problem with transition times [8] or as an Open Shop Scheduling Problem [7]. Similarly, scheduling problems can be cast as Vehicle Routing Problems with precedence constraints.

However, the reformulation does not seem to offer any advantages on pure problems. Perhaps not surprisingly, scheduling software performs better on pure scheduling problems than VRP software on reformulated problems. Similarly VRP software is able to solve pure VRP instances better than scheduling software.

VRP sowtware ??

Beck *et al.*[7] offer some insights into why this is so. They identify five characteristics that distinguish the two problems: alternative resources, temporal constraints, ratio of operation duration to transition time, optimisation criterion, and temporal slack. They demonstrate that converted problems are atypical in each of these five areas. For example, under alternative resources, the authors note that in the VRP there are typically many vehicles capable of making a delivering; whereas in pure job-shop scheduling (JSP) there is exactly one feasible machine for each job. Other scheduling variants also limit the number of feasible machines for a job. When such a scheduling problem is converted to a routing problem, it has a structure not usually found in routing problems. Briefly, the other characteristics, in order, point to the following differences:

- In the VRP time windows on visits are usually independent, whereas time windows on the JSP are interdependent, making long chains.
- In the VRP a visit has small duration compared to the transit time. In the JSP, the reverse is true.
- In the VRP the usual optimisation criteria is first minimise vehicles, then to minimise travel. In the JSP the number of "vehicles" is fixed. The usual JSP criteria – minimise makespan, which equates to minimising the length of the longest route – is not usually studied in connection with the VRP.
- In the JSP the important temporal data is the operation's duration, while in the VRPTW it is the time window applied to the operation.

Constraint programming toolkits [61, 62] were applied to reformulated problems that fall between pure VRP and pure JSP. The study identified the problem impurities that affected the solving technologies, and in particular the impurities that could lead to technology failure. The study also demonstrated that under certain conditions pre-treating impure VRP with JSP technology (and vice versa, pre-treating impure JSP with VRP technology) could significantly improve performance.

23.2 Operations Research Approaches

The vehicle routing problem and its variants have been widely studied. Recent surveys are [111] on the VRP, and [13, 14] for the VRPTW. It is beyond the scope of this chapter to fully review these solution methods.

Constraint Programming approaches make extensive use of methods from the Operations Research literature. It is useful therefore to begin with a brief review of some of the

methods that have been proposed by the Operations Research community to solve these problems.

A variety of Constraint Programming approaches have also been developed – many of which use algorithms from the OR literature as building-blocks. These methods are reviewed in more detail in Section 23.3.

We begin by formulating the VRP and VRPTW more formally, using integer linear programming (ILP).

23.2.1 ILP Formulation

The standard VRP and VRPTW problems can be formulated as an ILP in a number of ways. A study of formulations is given in [80]. We use the following "3-index" formulation (from [67]).

Let the set of customers be $C = \{1 \dots n\}$, and vehicles $M = \{1 \dots m\}$. Use 0 to index the depot at route start, and $n+1$ to index the depot at route end. Define $N = C \cup \{0, n+1\}$.

The decision variables are x_{ijk} for $i, j \in N, k \in M$ where x_{ijk} is 1 if vehicle k travels directly from customer i to customer j, 0 otherwise. The constants used in the formulation are given in Table 23.1.

c_{ij}	The cost of travelling from i to j, $i, j \in N$
τ_{ij}	The time to travel from i to j, $i, j \in N$, incorporating the service time at customer i
δ_{ij}	The distance from i to j, $i, j \in N$
r_i	The demand at customer $i \in N$
Q_k	The capacity of vehicle $k \in M$
a_i	The earliest time service can start at customer $i \in N$
b_i	The latest time service can start at customer $i \in N$
K	A large integer

Table 23.1: Problem constants

For convenience, define $r_0 = 0, r_{n+1} = 0, a_0 = 0, a_{n+1} = 0, b_0 = K, b_{n+1} = K$. For time-window constrained problems, b_{n+1} can instead be set to a maximum route time if required.

Problem **VRP** =

$$\text{minimise} \quad z_{\text{VRP}} = \sum_{k \in M} \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ijk} \tag{23.1}$$

subject to

$$\sum_{k\in M}\sum_{j\in N} x_{ijk} = 1 \qquad \forall i \in C \tag{23.2}$$

$$\sum_{i\in C} r_i \sum_{j\in N} x_{ijk} \leq Q_k \qquad \forall k \in M \tag{23.3}$$

$$\sum_{j\in N} x_{0jk} = 1 \qquad \forall k \in M \tag{23.4}$$

$$\sum_{i\in N} x_{ihk} - \sum_{j\in N} x_{hjk} = 0 \qquad \forall h \in C,\ \forall k \in M \tag{23.5}$$

$$\sum_{i\in N} x_{i(n+1)k} = 1 \qquad \forall k \in M \tag{23.6}$$

$$\sum_{i,j\in S} x_{ijk} \leq \mid S \mid -1 \qquad \forall S \subseteq C \tag{23.7}$$

$$x_{ijk} \in \{0,1\} \qquad \forall i,j \in N,\ \forall k \in M \tag{23.8}$$

The objective 23.1 is to minimise the sum of the costs of used arcs. Constraints 23.2 ensures that each customer is visited exactly once. Constraints 23.3 enforce the capacity constraints (assuming a heterogeneous fleet). Constraints 23.4 to 23.6 together enforce the flow of vehicles from the start depot, through 0 or more customers to the end depot. Constraints 23.7 are required to ensure there are no subtours (cycles that do not include the depot). Unfortunately, this expands to an exponetial number of constraints, making formulation of this type impractical for real problems.

The VRPTW can be formulated using an extra decision variable t_{ik}, the time vehicle $k \in M$ begins service at customer $i \in N$.

Problem **VRPTW** = minimise z_{VRP} subject to constraints 23.2 to 23.6 and 23.8, plus

$$t_{ik} + \tau_{ij} - K(1 - x_{ijk}) \leq t_{jk} \qquad \forall i,j \in N,\ \forall k \in M \tag{23.9}$$

$$a_i \leq t_{ik} \leq b_i \qquad \forall i \in N,\ \forall k \in M \tag{23.10}$$

Constraints 23.9 define the arrival times at each customer, and constraints 23.10 enforce the time windows. Constraints 23.7 are no longer required, as the time constraints will ensure there are no subtours.

23.2.2 Methods from the OR Literature

Since the VRP and its variants are $\mathcal{NP}$*-hard*, the focus has been very strongly on heuristic solution methods. We will describe the usual heuristic approach – initial route construction, improvement through local search, and meta-heuristic methods to escape local optima. We will also indicate the exact methods that have been described.

23.2.3 Construction Methods

Construction methods are used to create an initial solution. Many reported methods are based on an insertion procedure where a customer is selected, then inserted in a route in such a way as to minimise an incremental cost function. A number of constraints can be

enforced during construction. It is particularly easy to check within-route constraints (i.e. constraints that only deal with a single route), including time, precedence and capacity constraints. Methods vary in a number of ways.

Parallel versus sequential Sequential procedures focus on a single route, adding customers until no more will fit (e.g. [63]). Parallel methods build a set of routes simultaneously (e.g. [93]). Examples of both can be found in [107]

Seed customers The most successful parallel methods (such as Solomon's *I1* method [107]) use seed customers as the sole customer on a chosen number of routes. These customers act to guide the emerging routes.

Insertion order The order in which customers are inserted can be crucial. Some methods use a scoring system based proximity and other measures ([107, 63]. Some use *regret* (difference in cost between best and next-best insertion points) [36].

Insert position The objective function used to determine insert position may simply try to minimise the additional travel required to visit a customer. Others attempt to take time and other constraints into account, guiding the selection process to maximise the spare time or resources in resulting routes. (The same function may be used for both choosing the customer to insert and the insert position.)

Perhaps the most famous VRP construction method is the "Savings" method of Clarke and Wright [25]. The method starts with each customer on their own route. The "savings" obtained by combining two routes is $S_{ij} = c_{i0} + c_{0j} - c_{ij}$. The method selects the maximum S_{ij} for which the combined route is feasible, and iterates.

Other approaches to route construction include the "sweep" heuristic of Gillet and Miller [51]. Here, a ray centred at the depot is swept in a clockwise direction. Each customer it touches is added to the route until no more will fit. A new route is then begun.

23.2.4 Local Search Methods

Local search looks at the neighbourhood of the current solution to find improved solutions. The neighbourhood is defined in terms of one or more *move operators*: The neighbourhood of a solution is those solutions which can be generated by applying the move operator. The size of the neighbourhood is a key factor: the larger the neighbourhood, the more likely it is to contain good solutions. However, larger neighbourhoods are also more expensive to search.

Several of the operators used in solving routing problems are described below.

2-opt [81] Choose nodes i and j from the same tour, with $i+1 < j$. Delete arcs $i \rightarrow i+1$ and $j \rightarrow j+1$. Replace with $i \rightarrow j$ and $i+1 \rightarrow j+1$. This reverses the order of nodes from $i+1$ to j. Neighbourhood size is $O(n^2)$

3-opt [81] Analogous to *2-opt*, 3 arcs are deleted and re-attached in such as way that intermediate sections are not re-ordered. Neighbourhood size is $O(n^3)$. k-opts for $k > 3$ are also possible, but the cost is prohibitive.

Or-opt [86] Choose a parameter K, the maximum length considered. Remove each sequence of customers of length $k = K, \ldots$ down to 1 customers. Test re-inserting them in forwards and backwards orientation between each remaining pair of customers. Re-insert in the cheapest position. Neighbourhood size is $O(Kn^2)$.

Relocate [103] Subset of *Or-opt* moves – move a single customer from one route to the best position in another. Neighbourhood size is $O(n^2)$.

Exchange [103] Choose two customers in different routes, and swap their positions.

Cross [103] Extension of *Exchange*: Choose a sequence of customers $i_1 \ldots j_1$ of length at most K in one route, and $i_2 \ldots j_2$ of length at most K in another, and exchange the two sections. Neighbourhood size is $O(K^2n^2)$.

λ-exchange [87] As for cross except that the routes are re-optimised after the customers are exchanged. Neighbourhood size depends on re-optimisation method, but is at least $O(K^3n^3)$.

GENI exchange [50] An extension of the relocate that allows the receiving route to be minimally re-ordered to accommodate the new customer. It allows i to be inserted into the new route between non-consecutive nodes j and k by reinserting the segment $j + 1 \ldots k - 1$ to a different position.

Ejection chains [54] A sequence of customers is selected and inserted into another route. This may cause some sequence of customers to be "ejected" in order to accommodate the new ones. The procedure forms a chain of such ejections until no customers are left unassigned. This has proved to be a very powerful operator [96, 104, 21, 108]. Neighbourhood size depends on the restrictions imposed on the chain length, but is potentially very large.

Guided Local Search and Limited Discrepancy Search [106, 59] are methods introduced in the context of constraint programming and are discussed in Section 23.4.2.

Local search typically looks at a subset of these operators. The neighbourhood of the current solution will be examined to find cost-reducing moves. Either the first found, or the best found will then be executed. This leads the procedure to a local minimum within the specified neighbourhood.

23.2.5 Meta-Heuristics

Meta-heuristics are used to escape local minima. They do this in one of two ways – either the controlled acceptance of cost-increasing moves, or by neighbourhood expansion.

Many meta-heuristics have been applied to the VRP and related problems. Again, we can only give a brief indication of the variety.

Simulated Annealing [1] Acceptance of cost-increasing moves is controlled by a system parameter called *temperature* by analogy with a crystal annealing process. An increase in cost δ is accepted with probability proportional to $e^{-\delta/T}$. An updating procedure increases T as the process continues so that solutions converges to a (hopefully better) local minimum. Any combination of the local search operators just described can be used to form a search neighbourhood [10, 113, 23].

Tabu Search [52, 53] Tabu search allows a cost-increasing move, and then places the reversal on a "tabu list" so that the move cannot be "undone". This allows the neighbourhood of the new solution can be properly explored. A large number of methods have been suggested using Tabu search for vehicle routing problems – Bräysy and Gendreau [14] highlight fourteen notable applications. A few of the most successful are [26, 110, 24, 112].

Genetic [55] Genetic (also *Memetic* and *Evolutionary*) algorithms use an Darwinian evolution analogy. In the basic form, a population of solutions is created. Individuals are "crossed" to form a new solution with characteristics from both parents. Mutation operators (similar to the local search operators above) can also be applied. Implementations differ in initialisation, mutation and crossing methods. Again, many genetic methods have been suggested – Bräysy and Gendreau select seventeen for comparison. A few of the more successful are [60, 11, 79].

Variable Neighbourhood Search [58] This method considers a sequence of neighbourhoods of increasing size around the current solution. It exhausts one neighbourhood before moving on to the next largest, and will return to the smallest if a significant change has been made. This allows the solution to be improved as much as possible from low-cost, small neighbourhoods before investing the time to examine the larger neighbourhoods. Examples include [92, 12, 28]

23.2.6 Exact Methods

While much emphasis has been placed on heuristic solution, several optimal methods have been developed. Two such approaches are Column Generation and Lagrangian Relaxation. These methods are examined more closely in Section 23.5, where we look at using CP as a subproblem solver.

Other approaches are based on exploiting polyhedral properties of ILP formulations of the VRP. These have benefited from work done in the context of the TSP (e.g. [88]). Naddef and Rinaldi [85] review the branch and cut approach to solving the VRP. The Branch and Cut approach is based on an mixed-integer linear programming formulation of the problem. The linear relaxation of the problem is first solved. If the solution is not integer, *cuts* are then applied. These take the form of new constraints which remove fractional solutions without affecting the optimum. If an integer solution has not been found but no more effective cuts can be generated, a typical branch-and-bound search is employed. New cuts are applied at each branch. The process repeats until an integer solution is found and proved optimal in the usual way.

Cuts of various kinds have been explored [49, 84, 3, 2, 6, 27, 18, 75, 74].

23.3 Constraint Programming Approaches

The Constraint Programming approach to many problem types is to incrementally build a solution, backtracking when an infeasibility is detected, until a solution is found or the problem is proven to have no solution.

This method can be applied to the VRP, but it is usually too inefficient to even consider. In the vast majority of real-world cases, the existence of a solution is never in doubt. The real question is about the quality of the solution found.

In common with all applications of CP to optimisation problems, the search is over a large number of feasible solutions to find the best or near-best, rather than searching a large number of infeasible solutions to find the feasible one.

This change in emphasis is reflected in the way search is conducted. Local search and repair methods have been developed which are able to take advantage of a Constraint Programming framework.

These search methods are discussed in Section 23.5. We begin by presenting two effective constraint-based formulations of the VRP.

23.3.1 Formulating Routing Problems with Constraints

This section details a constraint programming formulation of the VRP with time and quantity of goods constraints maintained along each route. This model can be used as a basis for the capacitated vehicle routing problem, the vehicle routing problem with time windows, the pickup and delivery problem, and other vehicle routing problem variants. The main virtue of constraint programming is versatility, i.e. it allows the modeller to add different complications and extensions without adjustment to the basic model.

The constraint programming model described here is an extension of those described in [19, 90, 31], and also [94], one of the earliest references on the use of Constraints in routing problems. These works all use essentially the same method to model for the routing aspects: path constraints. The multi-vehicle aspects come from [31, 69]. The propagation rules that we describe in this section are implemented in the model described by [90]. They are also used by ILOG Solver [62] sand ILOG Dispatcher [61], which are C++ toolkits for general constraint programming and constraint-based vehicle routing respectively.

As previously, we have n customers orders and a fleet of m vehicles. The term *visit* will be used for each time a vehicle makes a stop. There is one visit per customer and two special visits per vehicle. These two additional visits per vehicle model the starting and stopping places for the vehicle. Let $C = \{1 \dots n\}$ form the customers, $M = \{1 \dots m\}$ form the vehicles and $V = \{1 \dots n + 2m\}$ form the visits. For the special first and last visits of each vehicle k we introduce the notations f_k and l_k. Visit $n + k$ is the first visit of vehicle k ($f_k = n + k$), while l_k is the last visit of vehicle k ($l_k = n + m + k$). The sets $F = \{n + 1 \dots n + m\}$ and $L = \{n + m + 1 \dots n + 2m\}$ indicate the set of first and last visits respectively.

1 ... n + 2M places! 2M are the depots!

To model routes, the integer variable p_i, $i \in V$ with domain $\{1 \dots n + 2m\}$ models the direct predecessor of each visit i. By convention, each "first" visit of a vehicle has as predecessor the vehicle's "last" visit ($\forall_{k \in M}\ p_{f_k} = l_k$). The predecessor variables form a permutation of V and are subject to the difference constraints:

$$p_i \neq p_j \quad \forall i, j \in V \wedge i < j \qquad (23.11)$$

~ two locations can't have the same predecessor

C := {1...n} items
M := {1...m} couriers
F := {n+1 ... n+m} starting depots
L := {n+m+1, ... n+2m} ending depots
V = C U F U L .

This is equivalent to stating (in the ILP model) that the in-degree and out-degree of each node must be equal to one. A difference constraint propagates according to the rule below. Following [90], we represent the current domain of variable x by $\{\!\{x\}\!\}$

$$x \neq y \ : \ \{\!\{x\}\!\} = \{a\} \rightarrow y \neq a \qquad (23.12)$$

this is for the propagation: if {{x}} contains only "a" then every other y ≠ x can't be "a"

where x and y are constrained integer variables, and a is an integer. Propagation rules here will be written as LHS $\rightarrow$ RHS where LHS is a logical combination of conditions

c_{ij}	The cost of travelling from i to j, $i, j \in N$
τ_{ij}	The time to travel from i to j, $i, j \in N$, incorporating the service time at customer i
δ_{ij}	The distance from i to j, $i, j \in N$
r_i	The demand at customer $i \in N$
Q_k	The capacity of vehicle $k \in M$
a_i	The earliest time service can start at customer $i \in N$
b_i	The latest time service can start at customer $i \in N$
K	A large integer

Table 23.1: Problem constants

on the variable domains whose truth can be easily checked. RHS is a unary constraint or conjunction thereof which can be enforced by simple domain filtering. The above rule only fires when the domain of x is reduced to the single value a. The RHS is enforced by removing the value a from the domain of y.

For each visit i, s_i models its direct successor. The successor variables are kept "coherent" (consistent) with the predecessor variables via the following constrained *element expressions*:

$$s_{p_i} = i \quad \forall i \in V - F \qquad p_{s_i} = i \quad \forall i \in V - L \tag{23.13}$$

The element constraint takes two integer variables y and z and a vector of integers or, in this case, integer variables x. The constraint specifies that z must be equivalent to the yth element of x. This type of constraint is nearly ubiquitous in constraint programming models, and, to the authors' knowledge, supported by all well-known constraint programming engines. It propagates as shown below.

$$\begin{array}{l} z = x[y] \\ \quad " \\ s[p_i] \end{array} \quad \begin{array}{l} \{\!\{y\}\!\} = \{a\} \wedge \exists b \notin \{\!\{z\}\!\} \rightarrow x[a] \neq b \\ \exists a \, \{\!\{x[a]\}\!\} \cap \{\!\{z\}\!\} = \emptyset \rightarrow y \neq a \\ \exists b (\forall a \in \{\!\{y\}\!\} \, b \notin \{\!\{x[a]\}\!\}) \rightarrow z \neq b \end{array} \tag{23.14}$$

The coherence constraints 23.13 can then be implemented as follows (we show only one of the two forms):

$$s_{p_i} = i \; : \; z = s[p_i] \wedge z = i \tag{23.15}$$

Note that the use of the predecessor and successor variables creates a symmetric model which will be reflected in the constraints that will now be introduced. Strictly speaking, the VRP solution space could be specified using only one of the variable sets (predecessor or successor) without changing the set of solutions of the problem. However, by adding both (redundant modelling), additional inferences can be made which can significantly prune the search space.

To model multiple vehicles, a "vehicle variable" v_i of domain $\{1 \ldots m\}$ is introduced for each visit i which represents the vehicle which performs visit i. Naturally, for the first and last visits, the constraints $\forall k \in M \; v_{f_k} = v_{l_k} = k$ are imposed. Along a route, all visits are performed by the same vehicle. This is maintained by constraints of the following form:

$$v_i = v_{p_i} \quad \forall i \in V - F \qquad v_i = v_{s_i} \quad \forall i \in V - L \tag{23.16}$$

These constraints form what is termed a *path constraint* as they maintain information along a path. The above is the simplest form of path constraint; more complex ones will be used to maintain the time and quantity of goods along a path. The above path constraint can be implemented using only element constraints.

Alternatively, less stringent form of consistency can be maintained, called "bounds consistency". This only requires the bounds – lowest and highest legal values – to be maintained. This is much cheaper to store and to use, but can be less powerful. In particular cases, however, it can be sufficient. We will write the bounds as as $\underline{x} := e$, meaning that

all values $i < e$ are removed from the domain of x; and $\overline{x} := e$, meaning that all values $i > e$ are removed from the domain of x.

The quantity of goods on the vehicle is one such variable for where the size of the domain and the mode of use means it is more efficient to maintain only bounds consistency. It is modelled by the introduction of a real or integer valued variable at each visit. Let $q_i \geq 0$ be a constrained variable representing the quantity of goods on the vehicle after performing visit i. Let $r_i \neq 0$ be the quantity of goods to be picked up at visit i, if this quantity is negative, it represents a drop off of goods. Then, the following path constraints maintain the load on the vehicles at each point in their route.

$$q_i = q_{p_i} + r_i \quad \forall i \in V - F \qquad q_i = q_{s_i} - r_{s_i} \quad \forall i \in V - L \tag{23.17}$$

As the q variables typically have large domains, only bounds consistency is maintained for efficiency. Propagation rules to maintain the above constraints are as follows:

$$\begin{array}{l} q_i = q_{p_i} + r_i \\ \text{Let } P \quad = \{k \mid k \in \{\!\{p_i\}\!\} \wedge \\ \qquad\qquad \min\{\!\{q_k\}\!\} + r_i \leq \max\{\!\{q_i\}\!\} \ \wedge \\ \qquad\qquad \max\{\!\{q_k\}\!\} + r_i \geq \min\{\!\{q_i\}\!\}\} \\ \text{Then} \\ \qquad \exists k \in \{\!\{p_i\}\!\}\, k \notin P \rightarrow p_i \neq k \\ \qquad \exists k \in P\, \forall l \in P \ \ \min\{\!\{q_k\}\!\} \leq \min\{\!\{q_l\}\!\} \rightarrow q_i \geq \min\{\!\{q_k\}\!\} + r_i \\ \qquad \exists k \in P\, \forall l \in P \ \ \max\{\!\{q_k\}\!\} \geq \max\{\!\{q_l\}\!\} \rightarrow q_i \leq \max\{\!\{q_k\}\!\} + r_i \end{array} \tag{23.18}$$

$$\begin{array}{l} q_i = q_{s_i} - r_{s_i} \\ \text{Let } S \quad = \{k \mid k \in \{\!\{s_i\}\!\} \wedge \\ \qquad\qquad \min\{\!\{q_k\}\!\} - r_k \leq \max\{\!\{q_i\}\!\} \ \wedge \\ \qquad\qquad \max\{\!\{q_k\}\!\} - r_k \geq \min\{\!\{q_i\}\!\}\} \\ \text{Then} \\ \qquad \exists k \in \{\!\{s_i\}\!\}\, k \notin S \rightarrow s_i \neq k \\ \qquad \exists k \in S\, \forall l \in S \ \ \min\{\!\{q_k\}\!\} \leq \min\{\!\{q_l\}\!\} \rightarrow q_i \geq \min\{\!\{q_k\}\!\} - r_k \\ \qquad \exists k \in S\, \forall l \in S \ \ \max\{\!\{q_k\}\!\} \geq \max\{\!\{q_l\}\!\} \rightarrow q_i \leq \max\{\!\{q_k\}\!\} - r_i \end{array} \tag{23.19}$$

Vehicles have limited capacity. To model this, element constraints are used to determine limits for the q variables: simple inequalities then restrict the a variables. Assume that the goods capacity of vehicle k is Q_k. The following constraints are then imposed:

$$q_i \leq Q_{v_i} \quad \forall i \in V \tag{23.20}$$

This formulation models heterogeneous fleets where each vehicle may have different capacity. Note that no constraints of the form $\forall i \in F\ q_i = 0$ are added, which would require that vehicles begin empty. By leaving these quantities otherwise unconstrained, different problems can be modelled. These include mixed pickup and drop off, where the vehicle can begin (partially) full and end (partially) full, and *en route* pickup and delivery problems (see next section).

Time is maintained roughly in the same manner as vehicle load except that waiting is normally allowed and so that path constraint maintains an inequality rather than an equality. Let $t_i \geq 0$ be a constrained variable which represents the time at which service for visit i begins.

The following constraints maintain time along vehicle routes:

$$t_i \geq t_{p_i} + \tau_{p_i,i} \quad \forall i \in V - F \qquad t_i \leq t_{s_i} - \tau_{i,s_i} \quad \forall i \in V - L \tag{23.21}$$

The propagation methods are omitted as the correspond closely to those already presented for propagating load (23.18, 23.19), the only difference being a cumulative inequality rather than a cumulative equality is maintained to allow waiting time.

Time windows on customers are specified by adding constraints on the t variables. For example, the constraint $a \leq t_i \leq b$ states that customer i must be visited between times a and b. Multiple time windows [90] can also be modelled; for example, the two constraints $a \leq t_i \leq d$ and $t_i \leq b \vee t_i \geq c$ indicates that customer i must be visited either between times a and b, or between times c and d ($a \leq b \leq c \leq d$).[1]

Different vehicles can have different availability windows; Suppose that O_k is the earliest starting time, or origin, of vehicle k and H_k is its latest finishing time, or horizon. Limits on the availability of vehicles are then expressed (again using element constraints and inequalities) as:

$$O_{v_i} \leq t_i \leq H_{v_i} \quad \forall i \in V \tag{23.22}$$

To avoid cycles of visits which do not involve a first and last visit (subtour elimination), a specialised constraint is described in section 23.3.3. However, one other simple way which propagates less, but has the advantage of not requiring custom constraints, is to make sure that each vehicle has a finite horizon, and that time for service for each visit is strictly positive.

Finally the cost function, which is normally the total distance travelled d, but can more generally involve other components, such as the number of vehicles used, is constrained as follows:

$$d = \sum_{i \in V-F} \delta_{p_i,i} \qquad d = \sum_{i \in V-L} \delta_{i,s_i} \tag{23.23}$$

In our case we should sum over each courier's route

where $\delta_{i,j}$ is the travel distance from visit i to j. Note the use of *both* the predecessor and successor variables to constrain the cost, which is nearly always more effective during search than using one single set. Each term in the sums is maintained by an element constraint, and the total sum by a summation constraint, whose propagation details will be skipped here, except to say that again, only bounds on variables are maintained in the sum.

When the cost-per-kilometer varies according to the vehicle used, the cost function can be generalised. Let C_k be the cost per unit distance of vehicle k. The cost function is then specified as:

$$d = \sum_{i \in V-F} C_{v_i} \delta_{p_i,i} \qquad d = \sum_{i \in V-L} C_{v_i} \delta_{i,s_i} \tag{23.24}$$

These costs are maintained as those above, except that there is an additional multiplication constraint per term. This second cost function will not be considered further here.

[1] There are various ways of handling constraint disjunction in constraint programming systems. ILOG Solver takes a simple approach to propagating the constraint $c_1 \vee c_2$—when one of the disjuncts becomes violated, the other disjunct is added as a hard constraint in the system. Implication can also be implemented this way by rewriting $c_1 \Rightarrow c_2$ as $\neg c_1 \vee c_2$).

23.3.2 Extensions of the Model

The constraint programming model of the vehicle routing problem described encompasses various standard OR models, such as the vehicle routing problem with capacity constraints (CVRP), the vehicle routing problem with (multiple) time windows (VRPTW), the multi-depot vehicle routing problem (MDVRP), the open vehicle routing problem (OVRP), and the site-dependent vehicle routing problem (SDVRP). (See [98] for a full description of these classifications.) Moreover, any a problem with a mix of features of these basic problems types can be easily accommodated.

By adding additional constraints the model can also be used on pickup-and-delivery problems (PDP), including problems with back-hauls. PDP problems are modelled with negative values of r_i for a delivery, and positive r_i for a pickup.

Assume that each pickup and deliver order o has a pickup visit o_p and a delivery visit o_d. For each order o, the following constraints are imposed, which state the the pickup and delivery visits must be carried out by the same vehicle and that the pickup must be performed before the delivery.

$$v_{o_p} = v_{o_d} \qquad t_{o_p} < t_{o_d} \tag{23.25}$$

For a back-haul problem, interleaving of pickup and delivery visits is excluded via the following unary constraints on successor and predecessor variables:

$$s_i \neq j \quad \forall i \in P \forall j \in D \quad \text{where } P = \{i \,|\, r_i > 0\} \text{ and } D = \{i \,|\, r_i < 0\} \tag{23.26}$$

which state that no direct link from pickup to delivery is allowed.

23.3.3 Increased Propagation

The model already detailed is a valid and general model for various flavours of vehicle routing problem. The basic model can also be further enriched to solve more complex real-world problems (see section 23.6). However, the constraint propagation of this model can be significantly improved by considering the problem structure as a whole rather than individual constraints. This section deals with such additional propagation algorithms.

Eliminating cycles

This constraint was introduced in [19, 90] and is a very efficient way of avoiding cycles in constraint programming vehicle routing models. The essence is simple: for any chain of customers, it is forbidden to go from the end of a chain to its start. This idea is illustrated in figure 23.2 adapted very slightly from the one shown in [90].

To maintain consistency of this constraint, first assume that a notification is sent to any interested parties whenever a variable is changed (as proposed in [114]). In particular, the NoCycle constraint need only be informed when a p variable is bound to a particular value.

Two values b_i, e_i are associated with each visit $i \in V - F$ which represent respectively, the visits which begin and end the chain involving i. The value of b_i is only valid if i is at the end of a chain. Likewise, the value of e_i if only valid if i is at the beginning of a chain. With these variables, the NoCycle constraint can be propagated in $O(1)$ time each time a p variable is bound to a value. The method is as follows:

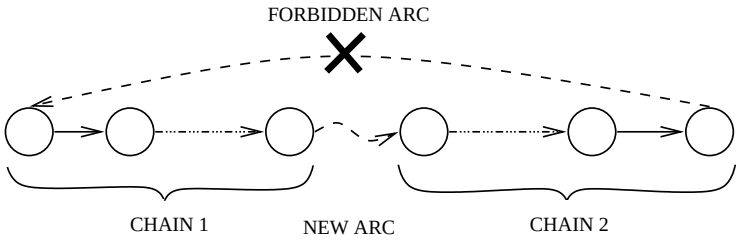
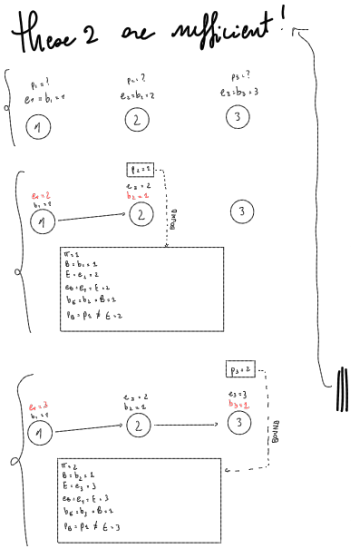


Figure 23.2: Operation of the NOCYCLE constraint



INITIALISE
forall i **in** $V - F$, $b_i = e_i = i$
forall i **in** $V - F$
if HASVALUE(p_i)
SENDEVENT(BOUND(p_i)) { see below }

when BOUND(p_i)
π = value(p_i) { $\pi \rightarrow i$ is new arc }
$B = b_\pi$ { B is first visit of new chain }
$E = e_i$ { E is the last visit of new chain }
$e_B := E$ { := is reversible assignment }
$b_E := B$ { := is reversible assignment }
$p_B \neq E$ { disallow the chain looping }

Where "reversible assignment" is noted, this means that the assignment is undone on backtracking, so that the chains maintain their integrity after returning from a dead end.

useless

Connectivity

Caseau and Laburthe [19] propose to go further than subtour eliminations for the TSP by performing a strong connection check to see if all nodes can be reached from the start node. This can be generalised to the model described here by performing a connection check to find the set of visits that can be reached by each vehicle. Let R_k^F be the set of visits reachable from visit f_k (including f_k) in a *forwards* direction—that is, following directed arcs $i \rightarrow j$ where $i \in V - L \wedge j \in \{\!\{s_i\}\!\} \wedge k \in v_i \wedge k \in v_j$. This set can be computed by a marking algorithm of complexity linear in the number of arcs considered. Likewise, let R_k^B be the set of visits reachable from visit l_k (including l_k) in a *backwards* direction. Let the set of reachable visits of vehicle k be $R_k = R_k^F \cap R_k^B$. There are two situations where search has reached a dead end and can backtrack. First, if $f_k \notin R_k$, then it means that their is no route from the first visit of vehicle k to its last visit. Second, if $\cup_{k \in M} R_k \neq V$, then there are some visits which cannot be visited by any vehicle. In fact, the second condition subsumes the first, but we highlight the first condition as it can be more efficient to test.

Caseau and Laburthe [19] also propose that this pruning rule can be transformed into a propagation rule, fixing any arcs which are *necessary* to a connection (that it, without the

arc, the already outlined algorithm would detect a dead end). Such an arc $i \rightarrow j$ would then be fixed by setting $s_i = j$.

Unfortunately the authors reported that the reduction in search space realized was very small, and did not compensate the large computational cost – detecting such arcs with their algorithm has complexity $O(|V|a)$, where a is the number of arcs considered. However, it is interesting to note here as an indication of the type of search space reduction that can be investigated.

Permutation of predecessor variables

Constraints 23.11 state that all p variables must be different. The well known global *all-diff* constraint of Régin [95] can be used to efficiently maintain generalised arc consistency for a set of variables that must all take different values. See Chapter 7 "Global Constraints" for more information.

23.3.4 An Alternative Formulation

In the formulation presented here, the solution requires a single value for successor and predecessor of a visit to be identified. Pesant *et al.*[90] present an alternative formulation in the context of the TSP that requires the set of *all* successors and predecessors of the visit to be identified.

Two set variables, B_i and A_i, are maintained for each visit that determine *all* visits which must come before (respectively after) visit i. In a TSP, any visit j other than i must either come before or after i. In a VRP, this is not the case, and we identify three possibilities (a) i before j on the same vehicle route ($i \in B_j \wedge j \in A_i$) (b) j before i on the same vehicle route ($j \in B_i \wedge i \in A_j$) (c) i and j served by different vehicles ($i \notin B_j \cup A_j \wedge j \notin B_i \cup A_i$). The A and B sets are maintained by examination of the time constraints between pairs of visits. This leads to the following constraints for all visits $i \in V$:

$$\begin{array}{ll} \text{(a) } B_i \cap A_i = \emptyset & \text{(b) } j \in A_i \Leftrightarrow i \in B_j \\ \text{(c) } s_i = j \Leftrightarrow A_i = A_j \cup \{j\} & \text{(d) } j \in A_i \wedge l \in A_j \Rightarrow l \in A_i \\ \text{(e) } v_i \neq v_j \vee j \in B_i \vee j \in A_i & \text{(f) } A_i \cap B_j \neq 0 \Rightarrow s_i \neq j \\ \text{(g) } j \in A_i \Rightarrow t_j \geq t_i + \tau^\star_{i,j} & \text{(h) } j \in B_i \Rightarrow t_j \leq t_i - \tau^\star_{j,i} \end{array} \tag{23.27}$$

Here $\tau^\star_{i,j}$ is the shortest path (in terms of time) between the start of service of i and the start of service of j. This can be found using a shortest path algorithm, but any lower bound is legal to use, including $\tau_{i,j}$ (making the reasonable assumption that the triangle inequality holds on travel times). In fact in [90] $\tau^\star_{i,j} = \tau_{i,j}$.

In the above, constraint (a) says that no visit can be both before and after another visit; constraint (b) states that if i is before j, then j is after i; constraint (c) links the direct successors with the complete successors; constraint (d) enforces the transitive closure which states that if i is before j and j is before l, then i is before l; constraint (e) says that if i and j are served by the same vehicle, they must be ordered; constraint (f) says that when at least one visit occurs between two other visits, those two visits cannot be directly linked; finally constraint (h) requires that when visits i and j are ordered in a certain sense, there must be a minimal time gap between them.

These constraints result in more powerful propagations eliminating certain arcs from consideration, or enforcing that two visits must be performed by different vehicles, for example. The before and after sets can also be used to tighten time bounds on visits to strengthen the method proposed in [90]. The following constraints are valid for any customer visit $i \in C$.

$$\begin{array}{ll} d_B = \sum_{j \in B_i} t_{j,s_j} & \{\text{Total duration before } i\} \\ d_A = \sum_{j \in A_i} t_{p_j,j} & \{\text{Total duration after } i\} \\ t_i \geq t_{f_{v_i}} + d_B & t_i \leq t_{l_{v_i}} - d_A \end{array} \tag{23.28}$$

That is, the earliest time that visit i can be started is the start time of the vehicle on which i will be serviced, plus the travel for all visits up to i. A symmetrical constraint limits the latest time that visit i can be completed.

23.3.5 Lower Bounds and Cost-based Propagation

NOT GOOD!

The cost can be used to limit search through a global constraint. The basic constraint programming model described in Section 23.3.1 will perform propagation through constraints 23.23. Consider in particular, the first of the constraints: CST: $d = \sum_{i \in V-F} \delta_{p_i,i}$. Suppose further that a goal cost G has been identified, so that we require $d \leq G$. (Such a cost is usually provided by finding a solution to the VRP with some cost $G + \epsilon$, resulting in the new tighter cost bound G.) Constraint CST will maintain lower and upper bounds on d computed from a sum of terms, each term of which is the distance from each node's predecessor to the node. During search, not all p_i variables will be bound to a single value, and *bounds* on the contribution of each term will be computed by propagation of the element constraints which constitute each term. Assume that T_i is the term in CST corresponding to visit i. Then:

$$\underline{T_i} := \min_{j \in p_i} \delta_{j,i} \qquad \overline{T_i} := \max_{j \in p_i} \delta_{j,i} \tag{23.29}$$

The summation will then maintain bounds on d, the total distance travelled, notably the lower bound:

$$d_{LB} = \sum_{i \in V-F} \underline{T_i} \tag{23.30}$$

The propagation rule $\underline{d} := LB$, ensures that this lower bound is enforced.

Of course, whenever $\sum_{i \in V-F} \underline{T_i} > G$, the domain of d will become empty and force the search to backtrack. However, what happens more often is that propagation will occur, removing arcs which would, if they appeared in the final solution, exceed the cost bound G. Consider the lower bounds on the cost when one visit i has as predecessor a visit h other than its closest predecessor. The new lower bound $d^{i,h}_{LB}$ would then be:

$$d^{i,h}_{LB} = \sum_{j \in V-F \setminus i} \underline{T_i} + \delta_{h,i} \tag{23.31}$$

The following propagation rule then applies:

$$\exists h \in p_i \;\; d^{i,h}_{LB} > G \rightarrow p_i \neq h \quad \forall i \in V - F \tag{23.32}$$

This logic can also be symmetrically applied to the successor variables.

Although the above method performs cost-based propagation, the lower bound d_{LB} is poor as it does not consider any routing aspects, such as the fact that all predecessor (or successor) variables must take different values. This has the effect of significantly weakening the propagation. Accordingly, better bounds have been proposed in the context of constraint programming, as well as a cost-based propagation methods based on reduced costs.

Better lower bounds

Several better lower bounds have been introduced in the literature which have different complexity/strength trade offs. A simple, strong bound was introduced by Caseau and Laburthe [19] for the TSP, based on a *regret* concept. In general terms, the regret R_x of a variable x is the difference in the cost to assign the variable its best value compared to the cost to assign it to its *second* best. Maximum regret is often used as a variable ordering heuristic.

Although the bound was introduced for the TSP, it can be used for the VRP without any significant change. First, let each visit i have associated its closest allowable visit $\kappa_i = argmin_{j \in p_i} \delta_{j,i}$. Also for each visit $i \in V - F$ where $|\{\!\{p_i\}\!\}| > 1$ the second closest visit can be defined as: $\kappa_i^2 = argmin_{j \in p_i \setminus \kappa_i} \delta_{j,i}$. Then the regret R_i of a visit i as $R_i = \delta_{\kappa_i^2,i} - \delta_{\kappa_i,i}$.

The regret-based bound recognises that if i is the closest direct predecessor to both visit j and visit l, then in any solution at least one of j and l cannot be directly followed by i. In this case, an extra distance can be added to d_{LB} equal to the minimum *regret* of j and l.

In the general case, suppose that for any visit $i \in V - L$, i is the closest allowable direct predecessor of a set of visits K_i. Then the regret based lower bound d_{LBR} is given by:

$$d_{LBR} = d_{LB} + \sum_{i \in V-L} R(i, K_i) \tag{23.33}$$

where $R(\emptyset) = 0$ and $R(X) = \sum_{j \in X} R_j - max_{j \in X} R_j$ otherwise.

This bound adds the smallest $|K_i|$ regrets to the basic bound d_{LB} for each contested visit (the visit with the largest regret in the set is assume to link to its closest visit). The bound works very well in practice, as it significantly strengthens the basic bound and is efficient to maintain. A symmetric bound can be computed using the successor variables in place of the predecessor variables, and both can be used for pruning.

Other bounds (for the TSP) have also been introduced based either on minimum spanning trees (MST) and minimum spanning aboresences (MSA). For instance Pesant *et al.*[90] maintain a minimum spanning tree via an incremental version of Kruskal's algorithm [4] which adjusts the MST when required. Again for the TSP, [19] proposes computing the MSA rooted at a single node. The latter goes further, proposing that a bound based on Lagrangian relaxation of the MSA problem can provide very good bound – very often within 1% of the optimum solution. However, the authors emphasise that because of the cost of computing the lower bounds, these methods should be used when appropriate and with care. For instance, such bounds become much weaker when side constraints such as time windows (especially tight ones) are added. In this case, the high investment does not pay off.

Cost-based propagation

Focacci, Lodi and Milano [46, 47, 45] propose a more effective global constraint allowing cost-based propagation for the TSP and TSP with time windows which can also be applied to the VRP. The idea is that the assignment problem relaxation of the TSP is used (where the subtour constraint is relaxed) to provide a lower bound on the objective. The Hungarian algorithm [71], or an incremental version of it, provides this lower bound d_H (which is a minimum cost matching of visits to visits according to the domains of the predecessor or successor variables).

However, the Hungarian algorithm also produces a *reduced cost* $\bar{c}_{i,j}$ for each possible arc i, j. This reduced cost is a lower bound on the increase in d_H if j was used as the successor of visit for i, instead of the one proposed in the solution to the assignment problem.

A new propagation rule based on the cost goal G and reduced costs can then be used:

$$\exists i \in V - L \, \exists j \in \{\!\{s_i\}\!\} \; d_H + \bar{c}_{i,j} > G \rightarrow s_i \neq j \tag{23.34}$$

This method is covered in detail in Chapter 7 "Global Constraints", or see [46] for a description in the context of the TSPTW.

23.4 Constraint Programming in Search

We have seen that Constraint Programming can offer many advantages when solving routing problems, due to the increased pruning achieved through propagation. We have also seen, in Section 23.2.4, that local search methods have been effective in solving the problems.

However, a difficulty arises when one attempts to put these two methods together in a naïve way. In local search, we may move a customer from one route to another, assigning a new successor $s_i = j$. Later, we may decide to move node j, so that s_i receives a new value. However, this contradicts a basic operating principle in classic Constraint Programming: so called "chronological backtracking". Under chronological backtracking, decisions must be undone in the reverse of the order they were made. So in order to undo the decision $s_i = j$, we would have to undo all operations performed since that time. This would undo all the progress made by local search, and hence is unacceptable.

Two ways around this problem seem to have been identified so far.

The first is to allow a heuristic or meta-heuristic to control search. In this case, the constraint system is used simply as a rule checker. The second way is to insulate the Constraint Programming system from the changes being made at the lower level, by wrapping up local search changes within an *operator*, which is then used within the constraint system. The use of these sorts of operators also allows the user to identify parts of the search that can be handled using traditional backtrack search. These methods are discussed in more detail below.

A third way of using CP is as a subproblem solver. Here, an independent search process generates a sequence of subproblems – usually closely related to the original routing problem – which are solved by CP. This sort of use has more of the flavour of the second class identified above, as the CP system is being used as more than just a rule checker; a full CP framework is being used repeatedly. We will look at some of these methods in Section 23.5.

See also Chapter 8 "Local Search" for methods for doing local search. Focacci *et al.*[48] also look at the use of local search within a constraint framework.

23.4.1 Constraint Programming as a Rule-Checker

Perhaps the easiest way to use a constraint programming framework in solving the Vehicle Routing Problem is to simply use CP-based methods as a rule checker to be applied to individual routes, or collections of routes. The main advantage over other possible methods is that CP is very expressive, and a wide variety of side constraints can be specified and checked efficiently. In addition, a CP-based solver that can handle the core VRP constraints can often be used without algorithmic modification to handle a problem with additional side constraints ([31, 106]).

Using CP as a rule-checker means that the search procedure is handled outside the constraint system. The constraint system is simply called each time a new route (or partial route) is to be tested. However, the CP system is still able to perform the usual propagations, limiting the scope of decision variable and potentially identifying infeasible partial solutions.

De Backer *et al.*[31] discuss in detail the use of CP within a non-chronological framework. They use two representations of the solution – an *active* representation within a constraint programming system that is only instantiated when a constraint check is required, and an *passive* representation used by the local search routines. Within this framework, a full test of all constraints through the whole (partial) solution can be very expensive. They therefore describe a number of methods for improving the efficiency of the CP system. We site three examples – details can be found in the reference.

First, many local search operators operate using a particular criterion – for example identifying the node that can be moved at least cost or maximum regret. A large gain in efficiency can be made by simply testing that criterion first, before performing any constraint checks. So rather than seeing whether it is legal to perform a particular move, the system should first see whether the cost of the move is less that the best found so far. If not, then the constraint checks can be skipped altogether.

Another possibility for improving efficiency is to have specialised propagators for the core constraints. For example, each "move operator" within a local search method may have its own propagator which examines the values produced during the path variable propagation. Illegal moves can then be identified very quickly. Even though they concentrate on the core constraints, since any side constraints will also be potentially affecting the path constraints, these propagators make good use of information from *all* constraints.

Finally it is important to observe that many move operators affect only a subset of the routes in a solution. Most constraint checking can therefore be limited to just those routes that have changed.

De Backer *et al.*[31] describe the use a CP framework for implementing several of the local search and meta-heuristic methods described in Section 23.2.4. The methods proved to be effective – several new "best known" solutions where produced.

23.4.2 Local Search within a Constraint framework

As noted above, *operators* can be defined within a Constraint Programming framework to insulate the Constraint System from non-chronological backtrack. This allows more of

the search to take place within a Constraint framework, and hence allows more scope for propagations and other techniques to prune the search space.

Many of the operators used within this type of framework are based on serial insertion and block deletion. These types of modification are well suited to use within a Constraint Programming framework, as they allow the propagations described in Section 23.3.1 to be used in full – both narrowing the potential sites for insertion, and quickly identifying partial solutions that cannot lead to a feasible solution. The formulation is also able to cope with arbitrary side constraints easily. Within-route constraints (like capacity) – which usually form the bulk of constraints – can be checked during insertion.

One such insertion-based technique, developed specifically for use in constraint programming environment, is Large Neighbourhood Search [106]. In this method, a group of "related" customers is removed, and then re-inserted into the existing runs with optimal cost. The customers are related geographically, temporally, or by use of a particular resource. For example, a simple relatedness function is $\mathcal{R}(i,j) = \frac{1}{c_{ij}+V_{ij}}$ where c_{ij} is the cost of travel from i to j, and V_{ij} is $K > 0$ if i and j are on the same vehicle, 0 otherwise. K is chosen to be comparable to the c_{ij}s.

The number of visits removed starts at one, and increases if no improvements have been found, up to some maximum (30 is used in the cited reference). Re-insertion uses exact branch-and-bound procedure with constraint propagation, which can find the minimum insertion cost rapidly.

The method has proved particularly successful on Vehicle Routing Problems, and has since been used in Constraint-based methods for other problems, including crew scheduling [105] and maximal satisfaction problems [82].

The strength of propagations from insertion-style local search is further exploited in a technique proposed by Caseau *et al.*[21]. Three meta-heuristics that involve repeated insertion – Large Neighbourhood Search, Limited Discrepancy Search, and Ejection Chains – are combined. The paper uses these three methods as building-blocks in a system designed to discover new heuristics based on automated "learning". Depending on the data presented, the methods produces heuristics, made up of calls to these building blocks, that are able to produce good solutions. The CP system is acting as a rule checker here, but in addition, traditional CP techniques are used to solve small TSPs with side constraints as described in [19] that appear as subproblems.

Another general-purpose search procedure that has been developed within the constraint literature is Limited Discrepancy Search (LDS) [59, 20, 21]. Many problem-solving methods involve a sequence of steps, a set of actions that can be taken at each step, and a heuristic for ordering the preference for those actions. Following the first preference at each step gives a solution to the problem. LDS systematically looks at the solutions that differ from the heuristic solution by making a different choice at a number of points. So for instance a 1-discrepancy would follow the heuristic at all but one step. At that step it would take the *second-best* choice according to the heuristic. Going against the heuristic at stage one (then following it for the rest of the procedure) gives a 1-discrepancy solution. Going against the heuristic at stage two gives another such solution, etc. So if there were n steps in the solution, n different 1-discrepancy solutions can be generated. A 2-discrepancy solution either twice uses the heuristic's second-best choice during construction, or once uses the third-best choice. In the context of the VRP, LDS can be used during construction. For example, in choosing the next customer to insert, or the insert position.

The Constraint framework is exploited in a method described by Pesant and Gendreau [89] to implicitly search large neighbourhoods. As mentioned previously, large neighbourhoods are more likely to contain good solutions, or allow good solutions to be discovered in fewer steps. However, such neighbourhoods can be expensive to search. Pesant and Gendreau describe a method that allows these larger neighbourhoods to be searched efficiently using branch and bound.

They characterise a neighbourhood $\mathcal{N}$ of a particular solution by a set of finite-domain variables $V = \{v_1, \ldots, v_n\}$, so that each feasible combination of values $\bar{v} = v1 \times \ldots \times v_n$ maps to exactly one neighbour of the current solution, and vice-versa. A systematic, branch and bound exploration of feasible values of $\bar{v}$ then implicitly explores the neighbourhood efficiently. Using lower-bounding functions allows non-productive areas of the neighbourhood to be identified and eliminated.

For example, in the TSP with time windows, an effective neighbourhood is the orientation-preserving 3-opt neighbourhood described in Section 23.2.4. This can be characterised using 3 indices – I, J and K – defining the break points, with the constraint $I \prec J \prec K$ (where $I \prec J$ means "I precedes J in the current tour"). Each feasible I, J, K combination represents one possible 3-opt exchange (and all 3-opt exchanges are represented by an I, J, K combination). Peasant and Gendreau describe how this I, J, K space can be explored using a branch and bound tree of depth three, using two bounding procedures to prune the search.

Bounding, propagation and pruning are being used to implicitly eliminate subsets of neighbours. This allows the more complex and larger neighbourhood structures (e.g. [109]) to be explored effectively.

Rousseau *et al.*[99] describe an approach, based strongly on the ideas presented in [89]. In this hybrid CP/OR approach, methods such as (Large Neighbourhood Search, GENI exchange, Ejection Chains) are embedded as operators within a constraint programming framework. Again, the building blocks chosen are based on serial insertion and deletion – precisely the operators able to benefit most strongly from propagation.

Basic operators are defined which remove and insert customers in a route. New operators can then be defined – for instance the following code implements a simple ejection chain:

NEC(c) :- Insert(c, R) $\vee$ *(Remove(C,R)* $\wedge$ *Insert (c,R)* $\wedge$ *NEC(C))*

where R and C are variables representing any route or customer. The code attempts to insert c in a feasible route or, if no such route can be found, removes another customer from a route, inserts c in that route, and recurses to find a new home for C. Obviously this method must be modified to prevent cycling etc, but the flavour of CP-based programming is evident.

In the method described, the construction phase, a local search phase and a post-optimisation improvement phase are all defined in terms of the operators. They follow the ideas of [89], exploring the neighbourhoods defined by each operator using branch-and-bound search within a CP framework. As with the previous method, propagation and pruning are working to reduce the number of solutions actually visited.

23.5 Using Constraint Programming as a Subproblem Solver

We focus here on methods that involve solving a sequence of constrained subproblems. These subproblems often share many of the constraints of the original problem, and hence Constraint Programming may be a useful technique to employ.

An example is the method described by Caseau and Laburthe [20] for the VRP. This is an insertion-based technique that interleaves insertion with local search. It uses CP to check the feasibility of insertions and of the neighbourhood moves used in local search. However, it also uses a constraint-based framework to solve the TSPTW sub-problems exactly, using the method described in [19]. Because the whole system is set up within a constraint-based framework, it is easy to implement Limited Discrepancy Search [59] as a solution improvement technique.

Shaw [106] uses a CP-based solver to find exact solutions to the TSPTW subproblems arising in the context of the Large Neighbourhood Search procedure (see Section 23.2.4).

Easton *et al.*[40] use a branch-and-price algorithm to solve the Tavelling Tournament Problem – which is related to the routing problems we examine here. In this approach, integer programming is used to solve the master problem, while constraint programming is used to solve the pricing problem. This approach is discussed in more detail in [66]. The branch-and-price approach is also used in [30] to solve the Vehicle Routing Problem with Time Windows.

Other examples of CP subproblem solvers are given in the next two sections.

23.5.1 Set Covering or Set Partitioning with Column Generation

Set Covering/Partitioning with Column Generation is a linear programming technique used commonly in the Operations Research literature to solve a variety of routing, scheduling and related problems [34, 97, 22, 5, 16, 32, 116]. Column Generation is also the subject of a recent book [33].

In the VRP context, indicator variables are used to specify potential routes: $a_{ik} = 1$ if customer i is visited by route k, 0 otherwise. Columns therefore represent a potential route, indicating which customers are covered, but not the order. The order and the cost c_k of route k can be calculated when the column is constructed, and stored separately.

The approach is to generate a large set K of potential routes (columns), then solve a set partitioning or set covering problem to choose the set of routes which covers all customers at minimum cost. The decision variable is y_k, $y_k = 1$ if route k is used in the solution, 0 otherwise.

$$\textbf{SP}\text{: minimise } \sum_{k \in K} c_k y_k \tag{23.35}$$

subject to

$$\sum_{k \in K} y_k a_{ik} = 1 \quad \forall i \in N \tag{23.36}$$

$$y_k \in \{0, 1\} \ \ \forall k \in K \tag{23.37}$$

The formulation above is the Set Partitioning formulation, requiring each customer to be covered exactly once. **SP** has the advantage that special structure within the constraint matrix can allow the problem to have integer properties [101]. However, the Set Covering formulation, which allows customers to be visited more than once, is generally somewhat easier to solve. The formulation (**SC**) is obtained by replacing "=" by "$\geq$" in equation 23.36. Repeat visits can usually be handled by simply deleting one. In the following all comments regarding formulation **SP** apply equally to **SC**

Due to the number of potential columns, it is not practical to include all possible routes in K. However, the method of Column Generation allows us to converge to the optimum solution. After solving **SP**, reduced costs for each customer can be obtained using the dual variable associated with each equation 23.36. The Column Generation subproblem can then be solved, where columns are generated by finding paths with negative reduced cost – i.e. sets of customers where the cost of travel is exceeded by the sum of customer reduced costs. This is a type of prize-collecting TSP [42]. An optimal solution to the relaxed form of **SP** is obtained when no more negative reduced-cost paths can be generated.

Constraint programming can be used both in initial column construction phase, and again in the column generation phase. One of the main advantages of the method is that any within-route constraints can be enforced by the column construction procedure, independent of the Set-Partitioning solution mechanism. The method can therefore handle classical VRP, and VRPTW problems, as well as a variety of side constraints. Previously, Dynamic Programming was used [34], but this does not have the flexibility of CP which allows, for example, multiple time windows to be added without fuss [91].

Solving the prize-collecting TSP subproblem in the column generation phase has to obey the same constraints as the original route-construction method (including any side constraints) – it simply has a modified objective. The same constraint-based formulation can therefore be used to solve the prize collection sub-problem.

This approach has been described by Rouseau *et al.*[100] in the context of the TSPTW, but most of the discussion applies to the VRPTW as well. They describe arc elimination and search strategies which speed up solving the prize-collection subproblem.

Junker *et al.*[65] discuss a framework for embedding efficient algorithms into a constraint framework for solving problems that arise in column generation methods. These methods are developed further in [41].

23.5.2 Lagrangian Relaxation

Lagrangian Relaxation is also a standard Operations Research technique, and descriptions can be found in many Operations Research texts (e.g. [70, 115]). The technique is applied to linear programming formulations of a problem where an "easy" problem is being complicated by some additional constraints. The idea is to move the complicating constraints into the objective with a penalty (Lagrangian) multiplier, leaving a much simpler subproblem. Using duality theory, optimal multipliers can then be found by solving a series of subproblems, yielding optimal solutions to the original problem.

There are various ways to apply this technique to the VRP, or VRPTW, depending on the relaxed constraints. Fisher [43, 44] used Lagrangian Relaxation to solve the VRP and VRPTW problems. The relaxation used there leaves a "minimum K-tree" subproblem. More recently, Lagrangian relaxation has been applied to the VRPTW [67]. In the approach described there, the constraints which ensure that each customer is visited exactly

once are relaxed – these are constraints 23.2 in the formulation in Section 23.2.1. This gives the Lagrangian objective

$$\text{minimise } z_{\text{LR}}(\lambda) = \sum_{i \in N} \sum_{j \in N} \sum_{k \in M} c_{ij} x_{ijk} - \sum_{i \in N} \lambda_i \left(\sum_{j \in N} \sum_{k \in M} x_{ijk} - 1 \right) \quad (23.38)$$

We wish to find the optimal set of multipliers λ_i, and must therefore solve Problem **LR** = maximize$_{\lambda \in \mathbb{R}^n}$ $z_{\text{LR}}(\lambda)$ subject to constraints 23.3 to 23.10 (without the redundant subtour constraints 23.7).

LR divides into m independent (difficult) problems – one for each vehicle. As with the Column Generation method described above, the subproblem is a sort of prize-collecting TSP, with the Lagrange multiplier λ_i associated with each customer quantifying the "prize" for a vehicle to visit. (At termination, λ_i therefore gives an indication of the "cost" of visiting each customer.) For the strict definition of VRPTW, [67] develop a dynamic programming-based method to solve the elementary shortest path with time windows and capacity constraints subproblem. However, a constraint-based approach could be used which would allow arbitrary side constraints to be applied.

The Lagrangian Relaxation method is an iterative procedure. At iteration t, the current solution is x_{ijk}^t The Lagrangian problem is then solved to find the optimal multiplier λ_i^t for $i \in N$. Unfortunately, in the case of the VRPTW, this optimization is also a difficult problem. [67] describe a cutting plane algorithm using trust regions to solve the Lagrangian dual problem for the pure VRPTW. Given these multipliers, a new x_{ijk}^{t+1} is calculated using the subproblem solver – CP or Dynamic Programming.

Traditionally Lagrangian Relaxation methods have suffered from being tied to a specific formulation of the problem. Constraint Programming allows the possibility to solve subproblems in the presence of a wide variety of side constraints, making the method more widely applicable. However, an exact solution to the subproblem is usually required in order to guarantee convergence of the method.

Benoist *et al.*[9] used Lagrange Relaxation within a constraint programming framework to solve a related routing problem - the "Travelling Tournament Problem" faced in constructing round-robin sports schedules. The method allows for a variety of constraints in composing a draw while minimising the travel time for players. This work is also interesting for the collaborative architecture described that allows CP and Lagrangian Relaxation to work effectively together.

23.6 CP-VRP in the Real World

23.6.1 Real-World Constraints

Up to this point, we have discussed the Vehicle Routing Problem and its variants as they are studied in academia. Starting from the very pure statement of the VRP originally proposed, new constraints and variants reflecting real-world practice have been studied over time.

However, the main difference between VRP in academe and routing practice in the real world remains the almost incredible variety of constraints and objective functions that are seen in day-to-day use – many of which have never been examined in the academic literature.

To bring this into focus, we present a number of examples of operating practice that have been seen by staff at a leading supplier of software for Vehicle Routing.

- Minimising vehicles is seldom an optimisation criteria in day-to-day scheduling. The fleet size is determined periodically, but in between times, drivers are typically on contract, and so are paid for some minimum time whether they drive or not. As a result of this, there is often a constraint that route time exceeds the paid minimum, while being less than the contracted maximum time.
- Meal and rest breaks, within a certain time of starting, and importantly, within a certain time relative to one another, must be inserted into the route automatically
- Subcontracting is common, whereby a shipment is sent using another carrier. The constraint that all visits are completed is therefore often dropped in favour of a cost term in the objective that will automatically drop uneconomic visits.
- A situation has been seen whereby the company only pays for travel to and from the first and last visits of the route, not for the distance on the route itself. There is a constraint which forces the route to be linear, rather than petal-shaped, which in effect forces the last visit to be one of the most distant from the first. In fact the constraint is that the total length of the route is no more than 30% greater than the distance from the first to the last stop.
- In some workplaces, some drivers have secured a strong negotiation position which allows them to choose which stops they wish to perform. Others are able to choose some of their route.
- Cross-docking is another very common practice. For example, a major supermarket has a regional distribution centre. This centre receives goods from suppliers, and distributes goods to stores. To avoid double handling and storage costs, where-ever possible the deliveries are “cross-docked”: the goods are taken straight across the dock from a suppliers vehicle to the distribution vehicle. If the supermarket is using its own vehicles to pick up from the supplier, then this constraint is a movable time-window linking two or more routes.
- A similar inter-tour constraint occurs when one vehicle delivers to a scheduled service, such as a ferry or train, and another picks up from the other end. There may be a choice of many scheduled services to use, but two routes must coordinate on the same service. Note that the receiving vehicle is not necessarily identified, but one of a number of vehicles must be tasked to meet the scheduled service.
- In (telecommunications) technician dispatching there can be a requirement to have two technicians at different locations at the same time, so that they can perform end-to-end tests of communications hardware. A constraint is required to force these two visits into different vehicles and for the two visits to occur in the same time window.
- Some long-haul companies use trailer change-over. Here, one vehicle (perhaps while making other deliveries) carries a trailer. At some location and time (decision variables) it meets another vehicle and the trailer is moved to the second vehicle for onward travel. This may be repeated a number of times. Each such change-over represents an inter-tour constraint. Again the vehicles involved are variable.

- The resources at a site may be limited. For instance there may be a limited number of docks or forklifts. The number of vehicles visiting at any one time must therefore be limited. This will be a consideration for example in a supermarket where multiple vehicles are sent to replenish stocks.

- Overtime rates must be taken into account in the objective, but are often expressed in a complicated fashion, including discontinuities, and penalty payments. E.g. $\$x$ up to the first half hour, $\$y$ for each subsequent half hour, plus $\$z$ meal allowance (plus 30 minutes break) if working more than 2 hours.

- In technician dispatching, the truck inventory must be maintained. The inventory of up to 300 types of parts may have to be tracked, and only technicians with sufficient supplies assigned. The technician must return to base to restock when appropriate, or may be able to be resupplied en-route by either meeting another technician and swapping parts, or by a special delivery from base.

- Lots of little, but still important, preferences have been seen, for example shipments to a particular customer in a week must be done by the same driver; or a husband and wife must/may not work together as a team.

Note that while many of the situations listed can be addressed using the models discussed here, some require more elaborate models, and some can only be approximately modelled. In addition, some would require bespoke techniques to find good solutions.

23.6.2 Dynamic Operation

Another important aspect of solving vehicle routing problems in the real world is the dynamic nature of the problems: the problem is shifting even as it is being solved.

The dynamism may be small - where some visits may be added or deleted without changing the general structure of the route, or the entire routing process may be driven by the stops being added, such as in taxi or parcel delivery operations.

Other dynamic aspects of operations include

- A company may have a list of clients it visits, but not all clients will require visiting each day. The stops to be performed may only be finalised as close to dispatch time – or even after.

- Traffic incidents and road works can alter the time taken to drive between two stops.

- It may be difficult to calculate *a-priori* how long a visit will require. If visits take longer than expected, parts of a planned route may have to be re-scheduled.

- Vehicles may break down, requiring visits to be re-scheduled

Algorithms to address these dynamic features of the problem are just beginning to be addressed in the literature [68, 77].

23.6.3 Vehicle Routing Software

The importance of vehicle routing to companies' operations is reflected in the use of vehicle routing software packages. The journal *OR/MS Today* conducts a regular survey of routing software. In their most recent survey [57], twenty routing software systems were analysed. The companies involved reported a total of more than 8,000 systems sold. We refer the reader to a survey such as this for details of available systems.

It should be remembered that the routing aspects are only small part of a routing system. Other features required of software include

- The ability to *geo-code* addresses - that is turn an address into a map location.
- The ability to calculate travel distances and times from one map point to another.
- A graphical user interface for displaying routes.
- The ability to change routes manually, preferably using a graphical interface.
- A method of easily specifying and entering constraints.
- Interfacing with other systems, such as billing and invoice systems.

23.7 Conclusions

We have seen how Constraint Programming can be applied to an important industrial problem. The Constraint Programming approaches have used, and advanced, the substantial body of research on the problem from the Operations Research community. Methods from the OR literature form the basis for many successful CP approaches.

Before the advent of Constraint Programming, the usual approach to solving these problems was to formulate the problem at hand in such a way that the main objectives and core constraints could be handled efficiently. If any side constraints were present, these were often handled in an "ad-hoc" manner. Unfortunately this leads to a wide variety of formulations, each of which is often specific to a fairly narrow class of problems.

The advantage Constraint Programming brings is a much more general method of handling the core and side constraints of routing problems. The formulation presented here is able to model a very wide variety of problems classes seen in the literature and in the real world with the *same set* of variables and core constraints. This formulation can handle capacity, time and incompatibility constraints, and various types of pickup and delivery problem.

In addition, a very wide variety of side constraints can be modelled without affecting the core model. We have already mentioned many different types of side constraints, but there are a large number of others in used in companies around the world. Many of these can be incorporated into the model with very low cost. We have shown how the expressive nature of the model allows constraints to be specified and checked efficiently.

Routing problems of the sort examined here are all $\mathcal{NP}$-hard. Efficiency in exploring a chosen search space is therefore paramount. We have shown how a combination of propagators for generic constraints, along with bespoke propagations, can substantially reduce the number of search nodes actually visited, without affecting the solution quality.

We have shown how performing local search within a Constraint Programming framework can benefit from the pruning of the search space.

Constraint Programming systems allow flexibility in how a new type of constraint, seen for the first time in a particular company, is handled. First, the new constraint can usually be incorporated relatively simply into the model described. As soon as this is done, automatic methods within the Constraint system are usually able to immediately use the constraint to prune search trees. In addition, if the constraint is seen as being core to solving the problem, bespoke propagators can be fashioned which increase the degree to which the search tree is pruned.

We have also seen how Constraint Programming systems offer advantages when used as subproblem solvers. In methods such as Column Generation and Lagrangian Optimisation, subproblems are often generated with an eclectic mix of constraints. Some of these are common to other routing systems, and others are particular to the subproblem environment. Constraint Programming allows the methods developed to solve the routing aspects to be leveraged to solve the related subproblem.

In the future, it would appear that there is still much to be done in terms of hybrid OR and CP methods. This is a very active area of research, and some very interesting methods using advanced techniques from both disciplines are already being seen. It is clear that techniques from Constraint Programming will continue influence developments in this area.

Acknowledgements

The authors wish to acknowledge the immense contribution of Patrick Prosser to their understanding of constraint techniques, particularly as applied to the Vehicle Routing Problem. Thanks Pat.

Philip Kilby is supported by the Australian Research Council and National ICT Australia (NICTA). NICTA is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

Bibliography

[1] E. Aarts, J. H. M. Korst, and P. J. M. Van Laarhoven. Simulated annealing. In E. Aarts and J. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 91–120. John Wiley & Sons, Chichester, 1997.

[2] N. R. Achuthan, L. Caccetta, and S. P. Hill. A new subtour elimination constraint for the vehicle routing problem. *European Journal of Operational Research*, 91(3): 573–586, 1996.

[3] N. R. Achuthan, L. Caccetta, and S. P. Hill. An improved branch-and-cut algorithm for the capacitated vehicle routing problem. *Transportation Science*, 37(2):153–169, 2003.

[4] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.

[5] R. Anbil, J. J. Forrest, and W. R. Pulleyblank. Column generation and the airline crew pairing problem. *Documenta Mathematica Journal*, Extra Volume ICM III: 677–686, 1998.

[6] J. Araque, G. Kudva, T. Morin, and J. Pekny. A branch-and-cut algorithm for vehicle routing problems. *Annals of Operations Research*, 50:37, 1994.

[7] J. Beck, P. Prosser, and E. Selensky. Vehicle routing and job shop scheduling: What's the difference? In M. Kaufmann, editor, *13th International Conference on Automated Planning and Scheduling ICAPS'03*, 2003.

[8] J. C. Beck, P. Prosser, and E. Selensky. On the reformulation of vehicle routing problems and scheduling problems. In *Proceedings of SARA 2002, Symposium on Abstraction, Reformulation and Approximation*, volume 2371 of *Lecture Notes in Computer Science*, pages 282–289, Berlin, 2002. Springer-Verlag.

[9] T. Benoist, F. C. Laburthe, and B. Rottembourg. Lagrange relaxation and constraint programming collaborative schemes for traveling tournament problems. In *Proceedings CP-AI-OR'01, Ashford 2001*, 2001.

[10] R. Bent and P. Van Hentenryck. A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, 38(4):515, 2004.

[11] J. Berger, M. Barkaoui, and O. Bräysy. A route-directed hybrid genetic approach for the vehicle routing problem with time windows. *INFOR*, 41:179–194, 2003.

[12] O. Bräysy. A reactive variable neighborhood search for the vehicle-routing problem with time windows. *INFORMS Journal on Computing*, 15(4):347 – 368, 2003.

[13] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part I: Route construction and local search algorithms. *Transportation Science*, 39(1): 104–118, 2005.

[14] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part II: Metaheuristics. *Transportation Science*, 39(1):119, 2005.

[15] Bureau of Transportation Statistics. *National Transportation Statistics (NTS) 2004*. Bureau of Transportation Statistics, 2005.

[16] S. Butt and D. Ryan. An optimal solution procedure for the multiple tour maximum collection problem using column generation. *Computers and Operation Research*, 26(4):427–441, 1999.

[17] R. W. Calvo. A new heuristic for the traveling salesman problem with time windows. *Transportation Science*, 34(1):113–124, 2000.

[18] V. Campos, A. Corberán, and E. Mota. Polyhedral results for a vehicle routing problem. *European Journal of Operations Research*, 52:75, 1991.

[19] Y. Caseau and F. Laburthe. Solving small TSPs with constraints. In *Proceedings of the 14th International Conference on Logic Programming*, pages 316–330. The MIT Press, 1997.

[20] Y. Caseau and F. C. Laburthe. Heuristics for large constrained vehicle routing problems. *Journal of Heuristics*, 5(3):281–303, 1999.

[21] Y. Caseau, F. C. Laburthe, and G. Silverstein. A meta-heuristic factory for vehicle routing problems. In J. Jaffar, editor, *Proceedings, Principles and Practice of Constraint Programming – CP'99. Alexandria, VA, USA, October 11-14, 1999.*, volume 1713 of *Lecture Notes in Computer Science*, pages 144–159, Heidelberg, 2004. Springer.

[22] D. G. Cattrysse, M. Salomon, and L. N. Van Wassenhove. A set partitioning heuristic for the generalised assignment problem. *Europenal Journal of Operational Research*, 72:167–174, 1994.

[23] W. Chiang and R. Russell. Simulated annealing metaheuristics to vehicle routing problems with time windows. *Annals of Operations Research*, 63:3–27, 1996.

[24] W.-C. Chiang and R. A. Russel. A reactive tabu search metaheuristic for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 9(4):417–430, 1997.

[25] G. Clarke and J. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568–581, 1964.

[26] 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(8):928–936, 2001.

[27] G. Cornuéjols and F. Harche. Polyhedral study of the capacitated vehicle routing problem. *Mathematical Programming*, 60:21, 1993.

[28] P. I. Cowling and R. Keuthen. Embedded local search approaches for routing optimization. *Computers & Operations Research*, 32(3):465–490, 2005.

[29] T. G. Crainic, F. Malucelli, M. Nonato, and F. C. Guertin. Meta-heuristics for a class of demand-responsive transit systems. *INFORMS Journal on Computing*, 17 (1):10–24, 2005.

[30] E. Danna and C. Le Pape. Accelerating branch-and-price with local search: A case study on the vehicle routing problem with time windows. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, pages 99–130. Kluwer Academic Publishers, 2005.

[31] B. De Backer, V. Furnon, P. Prosser, P. Kilby, and P. Shaw. Solving vehicle routing problems using constraint programming and metaheuristics. *Journal of Heuristics*, 6(4):501–523, 2000.

[32] G. Desaulniers, J. Lavigne, and F. Soumis. Multi-depot vehicle scheduling problems with time windows and waiting costs. *European Journal of Operational Research*, 111:479–494, 1998.

[33] G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors. *Column Generation*. Springer, 2005.

[34] M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354, March 1992.

[35] J. Desrosiers, Y. Dumas, M. M. Solomon, and F. C. Soumis. Time constrained routing and scheduling. In M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, chapter 2, pages 35–139. North-Holland, Amsterdam, 1995.

[36] M. Diana and M. M. Dessouky. A new regret insertion heuristic for solving large-scale dial-a-ride problems with time windows. *Transportation Research Part B: Methodological*, 38(6):539–557, 2004.

[37] C. Duhamel, J.-Y. Potvin, and J.-M. Rousseau. A tabu search heuristic for the vehicle routing problem with backhauls and time windows. *Transportation Science*, 31 (1):49–59, Feb 1997.

[38] Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(1):7–22, 1991.

[39] Y. Dumas, J. Desrosiers, E. Gelinas, and M. M. Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations Research*, 43(2): 367–371, 1995.

[40] K. Easton, G. Nemhauser, and M. Trick. Solving the travelling tournament problem:

A combined integer programming and constraint programming approach. In *Practice and Theory of AutomatedTimetabling IV*, pages 100 – 109. Springer-Verlag, Heidelberg, 2003.

[41] T. Fahle, U. Junker, S. E. Karisch, N. Kohl, M. Sellmann, and B. Vaaben. Constraint programming based column generation for crew assignment. *Journal of Heuristics*, 8(1):59–81, 2002.

[42] D. Feillet, P. Dejax, and M. Gendreau. Traveling salesman problems with profits. *Transportation Science*, 39(2):188, 2005.

[43] M. L. Fisher. Optimal solution of vehicle routing problems using minimum K-trees. *Operations Research*, 42(4):626–642, 1994.

[44] M. L. Fisher and K. O. Jörnsten. Vehicle routing with time windows: Two optimization algorithms. *Operations Research*, 45(3):488–492, 1997.

[45] F. Focacci, A. Lodi, and M. Milano. Solving TSP with time windows with constraints. In D. De Schreye, editor, *Logic Programming – Proceedings of the 1999 International Conference on Logic Programming*, pages 515–529, Cambridge, MA., 1999. MIT Press.

[46] F. Focacci, A. Lodi, and M. Milano. Embedding relaxations in global constraints for solving TSP and TSPTW. *Annals of Mathematics and Artificial Intelligence*, 34: 291–311, 2002.

[47] F. Focacci, A. Lodi, and M. Milano. A hybrid exact algorithm for the TSPTW. *INFORMS Journal on Computing*, 14(4):403–417, 2002.

[48] F. Focacci, F. Laburthe, and A. Lodi. Local search and constraint programming. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 369–403. Kluwer Academic Publishers, 2003.

[49] R. Fukasawa, J. Lysgaard, M. P. de Aragao, M. Reis, E. Uchoa, and R. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. In D. Bienstock and G. Nemhauser, editors, *Integer Programming and Combinatorial Optimization - IPCO 2004*, volume 3064, pages 1–15, Berlin, 2004. Springer-Verlag.

[50] M. Gendreau, A. Hertz, and G. Laporte. New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40(6):1086–1094, 1992.

[51] B. Gillet and L. R. Miller. A heuristic algorithm for the vehicle dispatch problem. *Operations Research*, 22:340–349, 1974.

[52] F. Glover. Tabu search, part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[53] F. Glover. Tabu search, part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[54] F. Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1-3):223–253, 1996.

[55] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, Reading, 1989.

[56] G. Gutin and A. Punnen, editors. *The Traveling Salesman Problems and its Variations*. Kluwer Academic Publishers, Dordrecht, 2002.

[57] R. Hall. Vehicle routing software survey. *OR/MS Today*, 31(3), 2004.

[58] P. Hansen and N. Mladenovic. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001.

[59] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In C. S. Mellish,

editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, volume 1, pages 607–615, Montréal, Québec, Canada, 1995. Morgan Kaufmann.

[60] J. Homberger and H. Gehring. A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *European Journal of Operational Research*, 162(1):220–238, 2005.

[61] ILOG S.A. *ILOG Dispatcher 4.0 User's Manual*. ILOG S.A., 9 Rue de Verdun, 94253 Gentilly Cedex, France, .

[62] ILOG S.A. *ILOG Solver 6.0 User's Manual*. ILOG S.A., 9 Rue de Verdun, 94253 Gentilly Cedex, France, .

[63] 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(5):523–537, 2001.

[64] M. Junger, G. Reinelt, and G. Rinaldi. The traveling salesman problem. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, Wiley Interscience Series in Discrete Mathematics. John Wiley & Sons, 1997.

[65] U. Junker, S. E. Karisch, N. Kohl, B. Vaaben, T. Fahle, and M. Sellmann. A framework for constraint programming based column generation. In J. Jaffar, editor, *5th International Conference of Principles and Practice of Constraint Programming – CP'99*, volume 1713 of *Lecture Notes in Computer Science*, pages 261–275. Springer, 2004.

[66] G. N. K. Easton and M. Trick. CP based branch and price. In M. Milano, editor, *Constraint and Integer Programming: Toward a Unified Methodology*, chapter 7. Springer, 2004.

[67] B. Kallehauge, J. Larsen, and O. B. Madsen. Lagrangean duality applied on vehicle routing with time windows. Technical Report IMM-TR-2001-9, IMM, Technical University of Denmark, DK-2800 Kgs. Lyngby - Denmark, 2001.

[68] P. Kilby, P. Prosser, and P. Shaw. Dynamic VRPs: A study of scenarios. APES Technical Report APES-06-1998, Department of Computer Science, Strathclyde University, Glasgow, Scotland, September 1998.

[69] P. Kilby, P. Prosser, and P. Shaw. A comparison of traditional and constraint-based heuristic methods on vehicle routing problems with side constraints. *Constraints*, 5 (4):389–414, 2000.

[70] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*, volume 21 of *Algorithms and Combinatorics*. Springer, Berlin, 2nd edition, 2002.

[71] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[72] G. Laporte. The traveling salesman problem: an overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:231–247, 1992.

[73] G. Laporte. The routing problem: An overview of exact and approximate algorithms. *European Journal of Operations Research*, 59:345–358, 1992.

[74] G. Laporte and Y. Nobert. Comb inequalities for the vehicle routing problem. *Methods of Operations Research*, 51:271, 1984.

[75] G. Laporte, Y. Nobert, and M. Desrouchers. Optimal routing with capacity and distance restrictions. *Operations Research*, 33:1050, 1985.

[76] G. Laporte, M. Gendreau, J.-Y. Potvin, and F. Semet. Classical and modern heuris-

tics for the vehicle routing problem. *International Transactions in Operational Research*, 7(4-5):285–300, 2000.

[77] A. Larsen, O. B. G. Madsen, and M. M. Solomon. The a priori dynamic traveling salesman problem with time windows. *Transportation Science*, 38(4):459, 2004.

[78] E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley and Sons, Chichester, 1985.

[79] A. Le Bouthillier and T. G. Crainic. A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. *Computers & Operations Research*, 32 (7):1685–1708, 2005.

[80] A. N. Letchford and J.-J. Salazar-González. Projection results for vehicle routing. *Mathematical Programming*, 105(2):251–274, 2006.

[81] S. Lin. Computer solutions of the traveling salesman problem. *Bell Systems Technical Journal*, 44:2245–2269, 1965.

[82] L. Lobjois, M. Lemaître, and G. Verfaillie. Large neighbourhood search using constraint propagation and greedy reconstruction for valued CSP resolution. In *ECAI Workshop on "Modelling and Solving Problems with Constraints" (14th European Conference on Artificial Intelligence, ECAI 2000), Berlin, Germany, 20 - 25 August 2000*. ECAI, 2000.

[83] Q. Lu and M. Dessouky. An exact algorithm for the multiple vehicle pickup and delivery problem. *Transportation Science*, 38(4):503, 2004.

[84] J. Lysgaard, A. N. Letchford, and R. W. Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2): 423–445, 2004.

[85] D. Naddef and G. Rinaldi. Branch and cut. In P. Toth and D. Vigo, editors, *Vehicle Routing*, volume 9 of *SIAM Monographs on Discrete Mathematics and Applications*. SIAM, 2000.

[86] I. Or. *Travelling Salesman-Type Combinatorial Problems and Their Relation to the Logistics of Blood-Banking*. PhD thesis, Department of Industrial Engineering and Management Sciences, Northwest University, Evanston, IL., 1976.

[87] I. H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41:421–451, 1993.

[88] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33:60, 1991.

[89] G. Pesant and M. Gendreau. A view of local search in constraint programming. In E. C. Freuder, editor, *Principles and Practice of Constraint Programming - CP96*, volume 1118 of *Lecture Notes in Computer Science*, pages 353–366. Springer, 1996.

[90] G. Pesant, M. Gendreau, J. Potvin, and J. Rousseau. An exact constraint logic programming algorithm for the travelling salesman with time windows. *Transportation Science*, 32(1), 1998.

[91] G. Pesant, M. Gendreau, J.-Y. Potvin, and J.-M. Rousseau. On the flexibility of constraint programming models: From single to multiple time windows for the traveling salesman problem. *European Journal of Operational Research*, 117(2):253–263, 1999.

[92] M. Polacek, R. F. Hartl, K. Doerner, and M. Reimann. A variable neighborhood search for the multi depot vehicle routing problem with time windows. *Journal of Heuristics*, 10(6):613–627, 2004.

[93] J.-Y. Potvin and J.-M. Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operations Research*, 66:331–340, 1993.

[94] J.-F. Puget. Object oriented constraint programming for transportation problems. In *Proceedings of Advanced Software Technology in Air Transport ASTAIR'92*, London, 1992. Royal Aeronautical Society.

[95] J.-C. Regin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), Volume 1*, pages 362–367. AAAI, 1994.

[96] C. Rego and C. Roucairol. Parallel tabu search heuristic based on ejection chains for the vehicle routing problem. In I. Osman and J. Kelly, editors, *Metaheuristics: Theory and Applications*. Kluwer Academic Publishers, 1996.

[97] C. C. Ribeiro and F. Soumis. A column generation approach to the multiple-depot vehicle scheduling problem. *Operations research*, 42(1):41–52, 1994.

[98] S. Ropke and D. Pisinger. A unified heuristic for vehicle routing problems with backhauls. *European Journal of Operational Research*. To appear.

[99] L.-M. Rousseau, M. Gendreau, and G. Pesant. Using constraint-based operators to solve the vehicle routing problem with time windows. *Journal of Heuristics*, 8(1): 43–58, 2002.

[100] L.-M. Rousseau, M. Gendreau, G. Pesant, and F. Focacci. Solving VRPTWs with constraint programming based column generation. *Annals of Operations Research*, 130:199–216, 2004.

[101] D. M. Ryan and J. C. Falkner. On the integer properties of scheduling set partitioning models. *European Journal of Operational Research*, 35(3):442–456, 1988.

[102] M. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4(285-305), 1985.

[103] M. W. P. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, 4(2):146–154, 1992.

[104] J. Schulze and T. Fahle. A parallel algorithm for the vehicle routing problem with time window constraints. *Annals of Operations Research*, 86:585–607, 1999.

[105] M. Sellmann, K. Zervoudakis, P. Stamatopoulos, and T. Fahle. Crew assignment via constraint programming: Integrating column generation and heuristic tree search. *Annals of Operations Research*, 115(1):207–225, 2002.

[106] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher and J.-F. Puget, editors, *Fourth International Conference on Principles and Practice of Constraint Programming (CP '98)*. Springer-Verlag, 1998.

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

[108] H. Sontrop, P. van der Horn, and M. Uetz. Fast ejection chain algorithms for vehicle routing with time windows. In M. J. Blesa, C. Blum, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics*, volume 3636 of *Lecture Notes in Computer Science*, pages 78–89. Springer-Verlag, Berlin, 2005.

[109] E. Taillard, P. Badeau, M. Gendreau, F. Guertain, and J.-Y. Potvin. A new neighbourhood structure for the vehicle routing problem with time windows. Technical Report CRT-95-66, Centre de Recherche sur les Transports, University of Montreal, 1995.

[110] É. D. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31:170–186, 1997.

[111] P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*, volume 9 of *SIAM Monographs on Discrete Mathematics and Applications*. SIAM, 2002.

[112] P. Toth and D. Vigo. The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, 15(4):333–346, 2003.

[113] A. Van Breedam. Improvement heuristics for the vehicle routing problem based on simulated annealing. *European Journal of Operational Research*, 86(3):480–490, 1995.

[114] P. Van Hentenryck, Y. Deville, and C. Teng. A generic arc consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.

[115] L. A. Wolsey and G. L. Nemhauser. *Integer and Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley-Interscience, New York, 1999.

[116] H. Xu, Z. Chen, S. Rajagopal, and S. Arunapuram. Solving a practical pickup and delivery problem. *Transportation Science*, 37(3):347–364, 2003.