# Local Search in Constraint Programming: Application to the Vehicle Routing Problem

**Article** · October 1997

Source: CiteSeer

CITATIONS
17

READS
313

**5 authors**, including:

Philip Kilby
The Commonwealth Scientific and Industrial Research Organisation
**66** PUBLICATIONS **1,063** CITATIONS
SEE PROFILE

Patrick Prosser
University of Glasgow
**137** PUBLICATIONS **3,706** CITATIONS
SEE PROFILE

Paul Shaw
IBM
**43** PUBLICATIONS **2,536** CITATIONS
SEE PROFILE

# Local Search in Constraint Programming: Application to the Vehicle Routing Problem

Bruno De Backer[1*], Vincent Furnon[1],
Philip Kilby[2], Patrick Prosser[2], and Paul Shaw[2]

[1] ILOG S.A, 9 rue de Verdun, 94253 Gentilly, France.
[2] Department of Computer Science, University of Strathclyde, Glasgow G1 1XH, Scotland, UK.

**Abstract.** Constraint programming is an appealing technology to use for vehicle routing problems. Traditional linear programming models do not have the flexibility or generality required by businesses wishing to model complex side constraints. This paper describes how a constraint programming framework for vehicle routing problems was implemented using ILOG Solver. A method for incorporating local search into a monotonic constraint programming toolkit is then described. Techniques for accelerating the method via filtering processes are also presented.
Computational tests using the constraint programming framework and a greedy search method were performed. Results indicate that moderately sized VRPs (100–200 nodes) can be solved to within 10% of the best known solution in seconds.
**Keywords:** Constraint Programming, Combinatorial problems, Iterative Improvement Techniques, Vehicle Routing Problem.

## 1 Introduction

Problems such as the Traveling Salesman Problem (TSP) or the Vehicle Routing Problem (VRP) can be found in almost every economic area. The efficient routing of pickup and delivery routes is vital for many shipping services whose added value and competitivity are based on short delays. Any improvement of the cost can yield substantial savings. It is estimated that distribution costs account for almost half of the total logistics costs [12].

Although vehicle routing problems have been studied by the operations research (OR) community since the fifties [5], many real-life problems involve side constraints (for example, complex capacity constraints) that make them difficult to model using standard OR techniques. On the other hand, constraint programming (CP) has proved to be a valuable tool for modelling and solving combinatorial problems that involve many specialized constraints, such as cardinality constraints, all-different constraints, and so on. Modelling these side constraints is important, especially when they represent legislation (for instance when drivers must be given a work break), and therefore must be upheld.

* Authors listed in alphabetical order.

Constraint programming is therefore an appealing technology to use when designing a tool that can easily model real-life problems. However, optimization in constraint programming is usually performed using a complete search method, namely branch and bound. Although branch and bound methods have provided exact solutions to large TSPs, they have not yet proved efficient enough to solve large VRP instances in reasonable times. This is probably why little work has been carried out on routing problems using CP. Only recently have small instances of the TSP (up to 30 visits) been solved using CP and branch and bound [2]. By contrast, local search techniques, often mixed with meta-heuristics, have delivered good results on large routing problems. Since our goal is to solve problems with more than one thousand visits routinely, local search becomes an obvious choice.

The problem with using local search within a constraint programming framework is that these frameworks are generally monotonic. That is, once a decision has been made (the domain of a constrained variable has been reduced), that decision cannot be changed except by backtracking to the point where the decision was made, undoing all other decisions made in between. Unfortunately, local search methods continually make changes where decisions made long ago are retracted without retracting decisions made from then until now.

The technical challenge of solving large routing problems is to design a framework for local search techniques in constraint programming. This prerequisite will allow us to implement a constraint programming tool for routing problems. An early attempt has been made by Puget [15] with a variant of 3-opt. Pesant and Gendreau [13] presented an implementation of local search with results on TSPs with time windows.

This article is organized as follows: In section 2, the VRP is described along with common side constraints. A constraint-based model for the VRP is then introduced, as are *path* constraints for propagating quantities along vehicle routes. A framework for implementing local search techniques is given in section 3, and exemplified through the case of vehicle routing problems in section 4. Section 5 discusses a search engine constructed using the framework previously described. Experiments are performed on over 100 VRPs.

## 2 The Vehicle Routing Problem

The TSP is an NP-hard problem [7] where a Hamiltonian circuit of minimal cost has to be found. In the VRP, the goal is to visit each customer exactly once, using a set of minimal cost routes, each of which starts from and ends at the same node (the depot). [1] The difference between the TSP and the VRP lies in the fact that additional constraints are generally imposed (for instance vehicle capacity) that make it necessary to create several routes. The TSP can be seen as a particular case of the VRP with only one vehicle and and no additional constraints. The cost function of the TSP is most often the sum of the values

[1] For the sake of clarity, the multiple-depot VRP is not considered in this article.

of the arcs making up the Hamiltonian circuit. The cost function of the VRP is usually more complex; for instance, a fixed cost may be added for each new route that is created. For a survey of the VRP, see [10].

### 2.1 Constraints

A real-life VRP often comes with many side constraints. Multiple capacity constraints can be expressed in several units. Some problems involve constraints where the total capacity of the vehicle cannot be used; instead, the loading of the vehicle must follow specific rules or legislation. This is for example the case in Europe with oil tanks.

Customers may also request to be served during specific time windows. There may be one time window or many disjoint time windows for each customer. Depots, parcels, goods, roads, drivers, and vehicles can also have time windows during which they are available. Finally, many kinds of compatibility constraints exist: some vehicles cannot access some customer sites, some parcels have to be delivered by a certain kind of vehicle, and so on.

### 2.2 A Constraint Based Model

Let $M$ be a set of indices corresponding to the vehicles, and $N$ be a set of indices corresponding to the customer visits. A finite-domain variable $next_i$ is associated with each visit $i$. $next_i = j$ if visit $j$ immediately follows visit $i$ in the route. Two visits per vehicle are introduced to represent visits to the depot. Let $S$ denote the set of indices of starting visits (*i.e.* the ones corresponding to departures from the depot), and $E$ denote the set of indices of ending visits (returns to the depot). So $next_i = j$ is possible iff $i \in N \cup S$ and $j \in N \cup E$. This ensures the conservation of the flow of vehicles through each customer and the depot.

### 2.3 Path Constraints

A path constraint deals with the propagation of accumulated quantities such as time and weight along a vehicle route. A path from $A$ to $B$ means that for all visits $i$ between $A$ and $B$ (including $A$ but not $B$), a constraint of the form

$$next_i = j \Rightarrow \sigma_i + f(i,j) \leq \sigma_j \tag{1}$$

is applied, where $\sigma_i$ represents the accumulated quantity. The function $f$ can take different forms depending on the quantity being accumulated.

The path constraint has been implemented in ILOG Solver. It is equivalent to a set of linear constraints in the form of (1). Path constraints on any number of quantities (such as weight, volume, time, distance *etc.*) can be applied simultaneously.

### 2.4 Core Constraints

For many problem classes with side constraints, some *core* constraints are always present or very common, regardless of the specializing side constraints. The core constraints in vehicle routing problems are related to quantities propagated along vehicle routes (by the path constraint), for instance time and capacity. Other types of core constraints can be observed in other problem classes. Time is restricted both by the working day and by time windows at customers. Capacity may be restricted in terms of weight, volume, number of pallet places, *etc.*

A method of modelling capacity constraints on vehicles is to propagate (via the path constraint) the load on the vehicle along each vehicle route. To model $\sigma_i$ as the load on arrival at visit $i$, $f(i,j)$ is the change in load associated with visit $i$. To indicate that all vehicles must not be overfilled at any time, constraints of the form $\forall i \in S\sigma_i = 0$ and $\forall i \in E\sigma_i \leq c_i$ are imposed, where vehicle $i$ has capacity $c_i$. Simultaneous pickups and deliveries can be modelled in a similar way, but this is not discussed here.

In the case of time constraints, $f(i,j)$ would be set to $s_i + t_{i,j}$, the service time at $i$ plus the travel time between $i$ and $j$. $\sigma_i$ would then represent the arrival time at $i$. Single or multiple service time windows could be set by further constraining the start of service variable for each visit. For visit $i$, a simple time window $[e_i, l_i]$ can then be expressed in ILOG Solver as $e_i \leq \sigma_i \leq l_i$. Defining disjoint time windows is also very easy. For example, ILOG Solver allows constraints such as $(e_i \leq \sigma_i \leq l_i) \vee (e'_i \leq \sigma_i \leq l'_i)$ to be written without introducing extra variables. The start and end of the working day for each vehicle can be represented by imposing time windows on visits in $S$ and $E$.

## 3 Local Search in Constraint Programming

Local search (iterative improvement) techniques are based on move operators (also called heuristics) that change some features of a given solution to obtain a new (better) solution. This allows the neighborhood of a solution to be explored. However, it is generally not possible to modify an instantiated solution using constraint programming because it implies non-monotonicity. A method for emulating this non-monotonic behaviour using constraint programming is given next.

### 3.1 Non-monotonic Approach

For a given problem, there usually are two types of variables: the decision variables, which record decisions made by the search algorithm, and the remainder, which are derived from the values of the decision variables (for instance, a variable representing the length of a TSP tour). When all the decision variables are instantiated, a solution is obtained; the other variables reach their final values by propagation. If there is a solution to the problem, saving the decision variables is enough to be able to restore this solution.

The implementation relies on two representations of the solution. There is a passive representation, which is the template against which new solutions are constructed, and which holds the "current solution" in terms of the states of decision variables. The second is an active representation that holds the constrained variables, and within which constraint propagation takes place. Decision variable states in the active representation are "checkpointed" (current domains are saved) before any changes are made, so that full domains can be restored later.

The CP system is only used to check the validity of solutions and determine the domains of constrained variables, not to *search* for solutions. The search is performed by an iterative improvement procedure that makes changes to the passive representation. When the procedure needs to check the validity of a potential solution, the passive representation is handed to the CP system for checking. If the CP system says the solution is legal, then the new solution is kept in the passive representation as the "current" one. Otherwise, the decision made in the passive representation is reversed.

When the CP system performs validity checks it instantiates the set of decision variables in the active representation using the passive representation. As a part of checking, constraint propagation using *all* constraints (core constraints and side constraints) takes place. If the constraint check succeeds, then the domains of all constrained variables are saved for later use in fast legality checks on core constraints (see section 3.2).

The approach presented in [13] differs from the one above in that it makes more extensive use of backtracking. Such an approach has been tested by the authors but it proved less efficient. For example, in makes an efficient implementation of the filters presented in section 3.2 more difficult.

### 3.2 Improving the Speed of Constraint Checking

In local search techniques, improvement heuristics generally only modify a very small part of a solution. Therefore, testing the complete solution using the CP system can be inefficient. We have tried to avoid this inefficiency in two ways: by reducing the amount of work carried out by the CP system to perform each check, and performing fast special-purpose checks for particular constraint types.

The first method applies to the order in which the decision variables in the active representation should be instantiated. Roughly speaking, instantiating the decision variables that are more likely to lead to failure is an efficient mechanism. Usually, this involves the decision variables that have just been changed by the most recent move operator. If the solution is illegal, this is more likely to lead to failure earlier.

The second method of increasing efficiency is to perform special-purpose "core constraint" checks directly (using the domains of constrained variables preserved from the last propagation, as described earlier). A detailed description of how this procedure can be carried out for routing problems is given in section 4.2.

Another improvement can be applied which is also generally applicable to local search within any application. Before testing if the move is feasible, the

variation of cost can be computed. In the case of a descent approach, if the cost value increases, the move will not be accepted and the checking phase is not necessary.

# 4 The Case of the VRP

In this section, it is shown how the above approach to non-monotonicity can be efficiently used for VRPs.

## 4.1 Decision Variables and Heuristics

For VRPs, the move operators modify the position of visits in a route or move them from one route to another. In the VRP model, the decision variables are the variables $next_i$ associated to each visit $i$. Once the constraints are posted, a standard heuristic can be used to obtain a first solution, after which all the $next$ variables are saved in the passive representation. To modify the solution, the $next$ variables are set to their saved values except for those that the move operator modifies, which are set to their new values. If the new solution is feasible and its cost is lower, then the decision variables are saved in the passive representation.

## 4.2 Improving the Speed of Constraint Checking for the VRP

To reduce the number of constraint checks, for any move, the variation in cost implied by the move is first computed. For the VRP, in the case where the cost of a solution is its total length, the variation of cost between two solutions is the difference between the sum of the lengths of the added arcs and the sum of the lengths of the arcs removed. We use a descent approach in this paper, and so if the variation in cost is non-negative, the solution will be rejected and the move is not checked for validity.

We can reduce the work required to perform checks by instantiating the decision variables in the routing plan in a particular order. The $next$ variables for the routes affected by the move operator just applied should be instantiated first. Furthermore, if the move is a relocate operator, the route that the visit is moved *to* should be instantiated first, as it will now be more constrained.

Knowledge of time or capacity windows can also be used to filter out infeasible solutions using special core constraint checks. For vehicle routing problems, we consider core constraints as any specified by a path constraint. We use filters similar to those described in [17]. The main difference here is in the fact that *all* the constraints (including side constraints) are taken into account to tighten the bounds of the constrained variables, so making the check more effective. Once a solution is set (all $next$ variables are bound), the lower and upper bounds of the $\sigma$ variables are known. In the case of the capacitated VRP with time windows, $\sigma_i$ can be either the arrival time or the load of the vehicle at visit $i$. These bounds are saved after each successful move to be used in testing the validity of the core constraints during subsequent moves.

As an example, such a filter for the heuristic that inserts visit $k$ between visit $i$ and visit $j$ works as follows:

- $\sigma_i$ is the arrival time at visit $i$
  Let $tmin_i$ be the saved propagated lower bound of $\sigma_i$ and $tmax_j$ denote the saved propagated upper bound of $\sigma_j$. Let $Tmin_k$ and $Tmax_k$ be the initial bounds of $\sigma_k$. The new solution is not feasible if $tmin_i + s_k + t_{ik} > Tmax_k$ or $Max(tmin_i + s_k + t_{ik}, Tmin_k) + s_j + t_{kj} > tmax_j$.
- $\sigma_i$ is the load of the vehicle at visit $i$
  Let $wmin_i$ and $wmax_i$ be the saved propagated bounds of $\sigma_i$. The new solution is infeasible if $q_k > wmax_i - wmin_i$.

Similar filters have also been implemented in the case of an exchange of visits or parts of routes.

The non-monotonic approach can therefore be summed up in the following way:

**if** the cost is reduced **then**
  **if** the solution is not infeasible according to the filters **then**
    copy in values of decision variables
    **if** the solution is feasible **then**
      save all the variables that have changed
    **endif**
    backtrack the CP solver
  **endif**
**endif**

Table 1 shows results that were obtained when applying the set of techniques described above to two capacitated VRPs with time windows, one with 100 visits, the other with 500 visits. There are three cases to consider. The basic case performs full constraint testing after each attempted neighbourhood move. The case with a cost filter does not perform a full constraint test if the cost is not reduced. The final case performs a cost check and a constant time insertion test which both must succeed before performing a full constraint test.

**Table 1.** Application of performance enhancement techniques to search

| Visits | No. of Full Constraint Tests | | | Time (sec) | | |
|---|---|---|---|---|---|---|
| | basic | + cost filter | + insertion filter | basic | + cost filter | + insertion filter |
| 100 | 451442 | 8875 | 6926 | 112.7 | 5.0 | 4.2 |
| 500 | 3634385 | 74978 | 35007 | N/A | 98.5 | 56.0 |

Checking the cost improvement before performing any constraint check provides a very important speedup. The constant-time insertion test becomes more important for larger problem sizes.

## 5 Experimental results

A simple search engine for the VRP was built on top of the constraint programming framework already discussed. A "best accept" strategy was used; at each point in search the neighbourhood move selected is the one giving the largest decrease in cost. The search stops when there are no cost decreasing moves (at a local minimum). The objective is to minimize the total length of the vehicle routes.

### 5.1 Improvement Heuristics

Four improvement heuristics were used, and are shown in figure 1. The two-opt heuristic works by reversing part of a single route. The relocate heuristic moves a visit from one route to another. The exchange heuristic swaps two visits in different routes. The cross heuristic swaps the end portions of two routes, and can join two routes or split one in two. The two-opt heuristic is due to [11]. The other heuristics are described in [16].

### 5.2 Objective Function

A *virtual* vehicle exists that has associated with it the visits that have not been included in the routing plan. The virtual vehicle is a convenience introduced for the search engine; the constraint toolkit does not interact with it.

The objective function is the total distance travelled by all vehicles (including the virtual vehicle). The distance travelled by the virtual vehicle when performing a visit is equal to $\alpha_1(d_{0i} + d_{i0}) + \alpha_2$ where $d_{ij}$ is the distance between customers $i$ and $j$ (customer 0 is the depot). When $\alpha_1 = 1$ and $\alpha_2 = 0$, the cost of the virtual vehicle making the visit is equal to that of a single vehicle being dispatched from the depot to make only that single visit. Higher values of $\alpha_1$ and $\alpha_2$ make it more costly not to perform the visits by real vehicles.

The use of a virtual vehicle makes modelling dynamic systems simpler. When a new visit arrives, it is included in the schedule by assigning it to the virtual vehicle. The iterative improvement then automatically includes it at an appropriate position.

### 5.3 Search Method

Search begins by assuming all customer visits will be carried out by the virtual vehicle, and then uses the four improvement heuristics to reduce the cost of the solution. (One could also use construction heuristics, *e.g.* [4], to build a solution before applying the best accept search.) New states generated by the heuristics that violate time windows or capacity constraints are rejected by the methods described earlier. There is an additional restriction that the only heuristic that can be used in association with the virtual vehicle is the relocation of a visit from the virtual vehicle to a real vehicle.

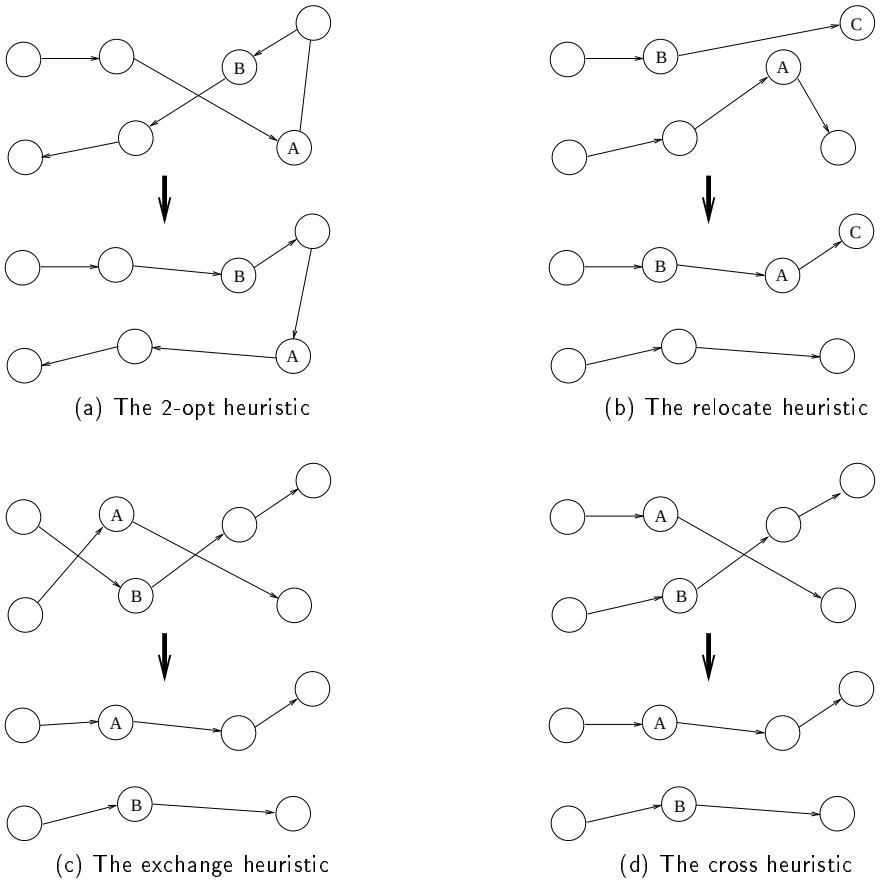


**Fig. 1.** Some Improvement Heuristics for the VRP

To make a move that reduces cost maximally, the cost of moving to all neighbourhood states must be computed so that the best may be chosen. This set of moves can be large, and so the speed of a best accept system compared to a random descent one can be low, especially when many cost reducing moves exist. However, application of any heuristic only deletes and adds a few arcs. Since the cost function is based on distance, the costs of moves involving changed arcs are all that need to be re-evaluated after each successful move.

### 5.4 Computational Results

The algorithm was tested on capacitated problems with and without time windows. The problems with time windows used were those of Solomon [18], which have 100 customers. Those without time windows were from [3,6,20,21] and have various numbers of customers. The $\alpha$ parameters were set to $\alpha_1 = 1.025$

and $\alpha_2 = 5 \times 10^{-4}\beta$ where $\beta$ is the distance between the two most remote sites in the problem. (Experiments for deciding on good values for the $\alpha$ parameters were conducted in [14].) Throughout the search procedure, double precision floating point numbers are used to represent time, distance, and capacity.

Solomon's problems are split into three categories—*r* problems with random customer distribution, *c* problems with clustered distribution, and *rc* with a mixture. Each of these categories is subdivided into series 1 and series 2 problems. Series 1 problems have a short scheduling horizon, and typically require 10–20 vehicles to serve the 100 customers. Series 2 have a long scheduling horizon and need only 2–4 vehicles to serve the customers. The depot is central in all problems.

The problems without time windows have been split into a number of classes according to the source of the problems, represented by the first 1–3 letters of the problem name (see table 2 for names). These problems are mostly regular with a central depot. Exceptions are the *f*, *fr*, and *tai* classes. The *f* class has heavy clustering. The *fr* class is regular, but with very non-central depots away from the customers. The *tai* class is variable and features clustering, non-central depots and depots within and outside clusters. Over 90% of the problems without time windows had an average route length of under 12.

The results of running the greedy algorithm are shown in table 2. The problem name, cost attained and the excess over the best known cost are tabulated. (For the problems without time windows the name indicates the size of the problem—when two numbers are present in the name, the larger one indicates the number of locations.) The algorithm is deterministic, so only one run was needed for each problem. The results can therefore be considered as representing *average* performance.

The results presented in [19] were used as the best for Solomon's problems. The best solutions to the other problems (as well as the problems themselves) were taken from `http://borneo.gmd.de/~andy/Benchmarks.html`.

The mean excess over the best solutions was 10.01% for problems without time windows, and 8.23% for problems with time windows. We consider these results to be of good quality and would not expect them to improve to any great degree (for instance, to within 5% of the best solution on average) without the use of a meta-heuristic. For the problems without time windows, 52% of the problems had a cost excess of under 10% and 17% had a cost excess of under 5%. For the problems with time windows, 66% had a cost excess of under 10% and 36% had a cost excess of under 5%.

For problems with time windows, it is clear that the poorest performance is obtained on the *r2* class; random problems with long vehicle routes. The best is obtained on what may be termed the "opposite" class *c1* with clustered customers and shorter vehicle routes. The results for problems without time windows do not appear to indicate any significant variation in performance with problem class.

**Table 2.** Costs Values Using Best Accept Search

| No Time Windows | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Problem | Cost | Excess | Problem | Cost | Excess | Problem | Cost | Excess |
| c100 | 941.01 | 13.90% | c100b | 839.50 | 2.43% | c120 | 1134.01 | 8.82% |
| c150 | 1218.50 | 18.48% | c199 | 1476.90 | 14.36% | c50 | 570.98 | 8.84% |
| c75 | 970.49 | 16.19% | en101k14 | 1155.45 | 7.28% | en101k8 | 941.01 | 15.18% |
| en22k4 | 394.76 | 5.27% | en23k3 | 577.44 | 1.48% | en30k3 | 509.11 | -4.66% |
| en33k4 | 847.37 | 1.48% | en51k5 | 570.98 | 9.59% | en76k10 | 914.37 | 9.90% |
| en76k14 | 1126.19 | 9.13% | en76k7 | 798.11 | 16.85% | en76k8 | 830.09 | 12.94% |
| f134 | 1236.36 | 6.31% | f71 | 263.61 | 8.94% | fran51k5 | 2834.60 | 11.07% |
| frbn51k5 | 4645.91 | 10.70% | frcn51k5 | 4109.45 | 8.89% | pn101k4 | 810.62 | 19.03% |
| pn16k8 | 451.95 | 3.90% | pn19k2 | 233.58 | 10.18% | pn20k2 | 234.00 | 6.36% |
| pn21k2 | 236.19 | 11.94% | pn22k2 | 239.50 | 10.88% | pn22k8 | 619.07 | 2.67% |
| pn23k8 | 548.76 | -0.95% | pn40k5 | 506.10 | 10.50% | pn45k5 | 564.56 | 10.70% |
| pn50k10 | 729.07 | 4.75% | pn50k7 | 604.23 | 9.07% | pn50k8 | 678.14 | 4.49% |
| pn51k10 | 785.80 | 5.48% | pn55k10 | 726.37 | 8.58% | pn55k15 | 996.95 | 16.47% |
| pn55k7 | 658.78 | 25.72% | pn55k8 | 630.58 | 9.48% | pn60k10 | 815.07 | 15.45% |
| pn60k15 | 1005.93 | 11.15% | pn65k10 | 852.05 | 7.58% | pn70k10 | 951.17 | 14.05% |
| pn76k4 | 687.52 | 16.73% | pn76k5 | 697.83 | 10.59% | tai100a | 2232.59 | 9.02% |
| tai100b | 2219.33 | 14.36% | tai100c | 1648.99 | 17.16% | tai100d | 1885.78 | 19.26% |
| tai150a | 3351.54 | 9.70% | tai150b | 2843.15 | 4.22% | tai150c | 2484.37 | 5.15% |
| tai150d | 2800.49 | 5.45% | tai385 | 27053.50 | 10.71% | tai75a | 1835.83 | 13.44% |
| tai75b | 1416.86 | 5.37% | tai75c | 1474.39 | 14.20% | tai75d | 1562.49 | 14.43% |
| Time Windows | | | | | | | | |
| Problem | Cost | Excess | Problem | Cost | Excess | Problem | Cost | Excess |
| c101 | 828.94 | 0.00% | c102 | 852.06 | 2.79% | c103 | 883.37 | 6.68% |
| c104 | 931.28 | 12.91% | c105 | 828.94 | 0.00% | c106 | 828.94 | 0.00% |
| c107 | 828.94 | 0.00% | c108 | 861.24 | 3.90% | c109 | 849.25 | 2.45% |
| c201 | 591.56 | 0.00% | c202 | 639.96 | 8.18% | c203 | 643.62 | 8.87% |
| c204 | 766.13 | 29.72% | c205 | 609.36 | 3.48% | c206 | 590.25 | 0.30% |
| c207 | 594.31 | 1.02% | c208 | 632.81 | 7.56% | r101 | 1650.22 | 2.65% |
| r102 | 1509.70 | 5.28% | r103 | 1302.97 | 7.95% | r104 | 1075.31 | 9.50% |
| r105 | 1430.61 | 3.88% | r106 | 1290.02 | 3.03% | r107 | 1146.87 | 5.63% |
| r108 | 1058.72 | 9.68% | r109 | 1282.63 | 8.11% | r110 | 1220.54 | 12.98% |
| r111 | 1173.43 | 9.57% | r112 | 1084.09 | 13.68% | r201 | 1337.75 | 6.61% |
| r202 | 1213.55 | 11.53% | r203 | 1074.53 | 17.70% | r204 | 828.08 | 0.42% |
| r205 | 1169.95 | 10.04% | r206 | 996.10 | 19.58% | r207 | 994.61 | 22.07% |
| r208 | 782.54 | 10.41% | r209 | 975.41 | 14.08% | r210 | 1025.64 | 6.01% |
| r211 | 937.45 | 18.00% | rc101 | 1676.80 | 3.28% | rc102 | 1589.34 | 7.57% |
| rc103 | 1361.17 | 22.63% | rc104 | 1217.81 | 7.22% | rc105 | 1667.12 | 8.03% |
| rc106 | 1465.41 | 5.81% | rc107 | 1359.16 | 10.45% | rc108 | 1189.46 | 4.36% |
| rc201 | 1410.34 | 12.92% | rc202 | 1317.68 | 13.18% | rc203 | 1026.12 | 1.20% |
| rc204 | 908.11 | 12.56% | rc205 | 1331.69 | 3.50% | rc206 | 1197.84 | 3.37% |
| rc207 | 1211.15 | 12.24% | rc208 | 971.88 | 16.54% | | | |

For reasons of space, all CPU times cannot be shown, but the CPU time required was quite low. Using a P120 laptop PC, solutions to problems with 100–200 customers were obtained in under 20 seconds.

### 5.5 Benefits of the Heuristics

Since the experiments conducted so far use four improvement heuristics (which is rather unusual), studies to determine the benefit of each move operator were performed. For our experimental analysis, each problem is solved using every combination of the four move operators being enabled/disabled. To ensure that all visits were completed, disabling the relocation heuristic only disabled relocations between real vehicles; visits can always be moved from the virtual vehicle to a real one.

Table 3 shows the percentage increase in cost over those reported in table 2. In the column headings, $r$ stands for relocate, $e$ for exchange, $t$ for two-opt, and $c$ for cross. A letter in the column heading indicates that a move is enabled. There is insufficient space to produce results of this type for all problems, so mean excesses for each problem class are shown.

From examination of table 3 one can see that in both the problems with and without time windows, disabling the exchange heuristic resulted in the smallest increase in cost (0.29% and 0.98%), whereas disabling the relocate heuristic resulted in the largest increase (4.07% and 1.90%). Disabling the exchange heuristic would appear to have a very small effect. This can quite readily be explained. First of all, the exchange operation can be performed by two relocate operations in many cases. (This is not true only when the intermediate state is either illegal or of higher cost.) Moreover, the exchange heuristic will only be of benefit if the two nodes to be interchanged are quite close together. When vehicle routes do not overlap considerably (which is normal), nodes that are close together will most likely be in the same route, and so not subject to exchange. The only place where vehicle routes are likely to overlap is near the depot, when routes come together. In this situation (where the beginnings or ends of routes are being interchanged) the cross heuristic can do the job required.

The only case in which disabling the exchange heuristic had an quite obvious detrimental effect was in class *fr*. These problems have the depot positioned very non-centrally (outside the bounding box of the customers). Here, routes will tend to overlap more, since each vehicle is dispatched in a similar direction to the customers. In this situation, the exchange operator can be of considerable benefit.

When using combinations of two operators, there is again agreement on the most and least useful combinations which are relocate/cross and exchange/two-opt respectively. For the single operator trials, all classes favour relocate most and cross next, but with disagreement on the least useful. The problems with time windows appear to find exchange on its own as the worst, while for the remaining problems this order is reversed. A closer examination of the data reveals that time windows are not the likely cause. Recall that most of the problems examined result in solutions with average route lengths of under 12 stops, except problem

**Table 3.** Cost Excess Over Using All Heuristics

| No Time Windows | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Class | ret | rec | rtc | etc | re | rt | rc | et | ec | tc |
| c | 1.21% | 2.25% | 1.58% | 1.52% | 2.97% | 2.30% | 3.00% | 6.58% | 4.55% | 3.52% |
| e | 1.67% | 2.28% | 0.68% | 1.18% | 3.14% | 3.73% | 3.03% | 9.42% | 4.30% | 2.72% |
| f | 5.85% | 5.44% | 1.54% | 7.37% | 6.57% | 5.79% | 5.28% | 20.89% | 9.38% | 8.40% |
| fr | 0.12% | 0.85% | 2.46% | -0.80% | 0.91% | 2.46% | 2.82% | 1.22% | 1.57% | 3.23% |
| p | 0.57% | 0.89% | 0.85% | 1.36% | 1.49% | 2.11% | 1.79% | 3.03% | 2.16% | 2.40% |
| tai | 3.52% | 1.51% | 0.69% | 3.07% | 4.15% | 4.03% | 2.11% | 13.42% | 5.74% | 4.59% |
| Means | 1.71% | 1.65% | 0.98% | 1.90% | 2.75% | 3.03% | 2.44% | 7.59% | 3.91% | 3.39% |

| Class | r | e | t | c | none |
|---|---|---|---|---|---|
| c | 3.11% | 7.99% | 8.29% | 6.51% | 9.29% |
| e | 4.84% | 11.03% | 12.01% | 6.38% | 12.54% |
| f | 6.76% | 21.14% | 22.44% | 10.19% | 23.41% |
| fr | 2.82% | 2.51% | 4.16% | 3.57% | 4.50% |
| p | 2.34% | 3.94% | 4.57% | 3.24% | 5.09% |
| tai | 4.95% | 14.75% | 15.11% | 8.01% | 16.13% |
| Means | 3.68% | 8.76% | 9.44% | 5.56% | 10.14% |

| Time Windows | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Class | ret | rec | rtc | etc | re | rt | rc | et | ec | tc |
| c1 | 1.15% | 3.07% | 0.30% | 2.34% | 4.17% | 1.40% | 3.33% | 9.88% | 5.03% | 2.93% |
| c2 | 1.19% | 6.44% | 0.03% | 3.95% | 5.72% | 0.72% | 6.57% | 11.46% | 11.33% | 6.32% |
| r1 | 2.70% | 1.27% | 1.11% | 2.20% | 3.11% | 3.84% | 1.16% | 7.99% | 4.43% | 3.13% |
| r2 | 6.84% | 3.83% | 0.34% | 6.74% | 10.55% | 7.10% | 3.68% | 19.64% | 11.76% | 7.60% |
| rc1 | 2.81% | 2.49% | 1.26% | 2.70% | 3.78% | 4.43% | 4.05% | 9.81% | 4.48% | 4.18% |
| rc2 | 7.69% | 5.05% | -1.72% | 6.63% | 8.95% | 7.47% | 5.35% | 21.04% | 13.51% | 7.54% |
| Means | 3.78% | 3.52% | 0.29% | 4.07% | 6.04% | 4.25% | 3.79% | 13.20% | 8.26% | 5.21% |

| Class | r | e | t | c | none |
|---|---|---|---|---|---|
| c1 | 3.55% | 11.39% | 10.70% | 5.45% | 13.49% |
| c2 | 6.50% | 15.15% | 12.44% | 11.96% | 16.05% |
| r1 | 4.41% | 9.34% | 12.50% | 5.34% | 13.35% |
| r2 | 10.72% | 23.20% | 21.65% | 13.60% | 25.03% |
| rc1 | 5.18% | 11.30% | 13.65% | 6.08% | 14.78% |
| rc2 | 9.90% | 25.30% | 23.26% | 13.84% | 27.92% |
| Means | 6.70% | 15.78% | 15.70% | 9.25% | 18.34% |

classes *c2*, *r2*, and *rc2* which have longer routes. For these problem classes (and more marginally for class *c1*, which has the longest mean route length amongst the other classes) two-opt outperforms exchange. This is intuitive; the two-opt heuristic will work better when the vehicle routes are longer since there is a larger number of potential two-opts that can be performed. What is more surprising is that even though two-opt reverses parts of routes, it can still be of significant benefit to problems with time windows.

## 6 Conclusion

A constraint programming model of the vehicle routing problem has been introduced. This model uses a general "path constraint" to constrain quantities that accumulate over vehicle routes, such as time and capacity. This makes modelling time windows and different types of capacity constraints simple, but does not remove the ability to impose other types of side constraint.

To allow large problems to be solved routinely, a method by which local search can be used within constraint programming was developed and presented. This technique is general, and can be applied to many problem classes. Application of various filtering techniques can result in a large increase in the number of heuristics applied per second.

A greedy search engine for the VRP using the "best accept" method has been constructed. Problems can usually be solved to within 10% (average case) of the best solution published with this simple method. Additional experiments indicated that when the depot is central disabling the exchange operator resulted in almost no cost increase. The 2-opt heuristic was also found to be useful for problems with time windows.

The work described here is being extended by the authors towards real applications. This work will address different routing scenarios such as pickup and delivery, multiple depots, and operational dynamics. In addition, initial experiments using meta-heuristics have already produced results [1,8] that are very close to, and in many cases better than, the best known solutions.

## Acknowledgment

The production of this paper was supported by the GreenTrip project, a research and development undertaking partially funded by the ESPRIT Programme of the Commission of the European Union as project number 20603. The partners in this project are Pirelli (I), ILOG (F), SINTEF (N), Tollpost-Globe (N), and University of Strathclyde (UK). WWW—`http://www.si.sintef.no/GreenTrip`

## References

1. B. De Backer and V. Furnon. Meta-heuristics in constraint programming: Experiments with tabu search on the vehicle routing problem. In *Proceedings of the 2nd International Conference on Meta-heuristics*, 1997.

2. Y. Caseau and F. Laburthe. Solving small TSPs with constraints. In *Proceedings the 14th International Conference on Logic Programming*, 1997.
3. N. Christofides, A. Mingozzi, and P. Toth. The vehicle routing problem. *Combinatorial Optimization*, pages 315–338, 1979.
4. G. Clarke and G. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568–581, 1964.
5. G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6:80, 1959.
6. M. Fisher. Optimal solution of vehicle routing problems using minimum K-trees. *Operations Research*, 42:626–642, 1994.
7. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W H Freeman, 1979.
8. P. Kilby, P. Prosser, and P. Shaw. Guided local search for the vehicle routing problem. In *Proceedings of the 2nd International Conference on Meta-heuristics*, 1997.
9. B. J. LaLonde and P. H. Zinzser. Customer service: Meaning and measurement. National Council of Physical Distribution Management, 1976.
10. G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *EJOR*, 59:345–358, 1992.
11. S. Lin. Computer solutions of the traveling salesman problem. *Bell Systems Technology Journal*, 44:2245–2269, 1965.
12. Institute of Logistics and Distribution Management. The 1985 survey of distribution costs. Queens Square, Corby, Northants, UK, 1985.
13. G. Pesant and M. Gendreau. A view of local search in constraint programming. In *CP-96*. Springer-Verlag, 1996.
14. P. Prosser and P. Shaw. Study of greedy search with multiple improvement heuristics for vehicle routing problems. Technical Report RR/96/201, Department of Computer Science, University of Strathclyde, Glasgow, January 1997.
15. J.-F. Puget. Object-oriented constraint programming tranportation problems. In *Advanced Software Technology in Air Transport*, 1992.
16. M. W. P. Savelsbergh. Computer aided routing. Centrum voor Wiskunde en Informatica, Amsterdam, 1988.
17. M. W. P. Savelsbergh. An efficient implementation of local search algorithm for constrained routing problems. *EJOR*, 47:75–85, 1990.
18. M. M. Solomon. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, 35:254–265, 1987.
19. 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.
20. E. D. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23:661–676, 1993.
21. Problems at `http://borneo.gmd.de/~andy/Benchmarks.html`.