Journal of the Operational Research Society (1995) 46, 1433–1446 0160-5682/95 $12.00

An Exchange Heuristic for Routeing Problems with Time Windows

JEAN-YVES POTVIN and JEAN-MARC ROUSSEAU
Université de Montréal, Canada

In this paper, we compare different exchange heuristics for vehicle routeing problems with time windows. We also introduce a new 2-opt* exchange heuristic, and show that a hybrid approach, based on Or-opt and 2-opt* exchanges, is particularly powerful for problems with time windows. Computational results are reported for randomly generated problems and for a standard test set.

Key words: heuristics, time windows, vehicle routeing

INTRODUCTION

Vehicle routeing and scheduling problems represent a very important part of any distribution system. Consequently, it is not surprising that the Vehicle Routeing Problem (VRP) has received a lot of attention in the past. This problem is concerned with the design of routes for a fleet of vehicles servicing a set of customers with known demands. The objective is to construct routes that service all customers at minimum cost and do not exceed the capacity of each vehicle[1].

In the Vehicle Routeing and Scheduling Problem with Time Windows (VRSPTW), these issues must be addressed with the additional complexity of allowable pick-up times or time windows at each customer location. Hence, no vehicle is allowed to arrive too late at a customer location. On the other hand, a waiting time is introduced in the route if the vehicle arrives too early. Quite often, a time window is also added to the depot, in order to define a 'scheduling horizon'. Each route must then start and end within the bounds of this window. The horizon acts similarly as a capacity constraint. For example, by enlarging either the capacity or the scheduling horizon, more customers can be serviced by the same vehicle. Two route construction heuristics for this problem are described by Solomon[2], and by Potvin and Rousseau[3].

In this paper, we first consider VRSPTWs with no capacity constraints (or vehicles of infinite capacity). In the literature, this class of problems is usually referred to as the Multiple Travelling Salesmen Problem with Time Windows (MTSPTW). Then, we consider Solomon's standard set of VRSPTWs[2].

The paper will describe and compare various iterative route improvement heuristics. Only a few improvement heuristics are useful when time windows are the main concern, and our objective is to evaluate them on problems with tight windows, large windows, and a mix of tight and large windows. To this end, the paper is organized along the following lines. First, the route improvement heuristics are introduced in general terms. Then we explain why the classical k-opt exchange heuristics are not well adapted to problems with time windows. We also explain how to modify a 2-opt to create a new 2-opt* exchange, which is particularly well suited to problems with time windows. Then, we briefly describe the Or-opt and Or-opt-1 exchanges[4], and propose a hybrid Or-opt and 2-opt* approach. Finally, computational results are reported on Solomon's standard set of problems[2], and on a set of randomly generated Euclidean problems.

Our intent is to write a paper that will be easily readable and understandable. Accordingly, technical details may be found in the appendix, where a pseudo-code description of each exchange heuristic is provided.

Correspondence: J. Y. Potvin, Centre de recherche sur les transports, Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal, Québec, Canada H3C 3J7

ROUTE IMPROVEMENT HEURISTICS

There are many different ways to modify a solution in order to obtain a new improved solution. Usually, the modification step is embedded within a generic iterative improvement procedure of the following type.

Step 1. Generate an initial feasible solution.

Step 2. Modify the current solution to get a new improved feasible solution.

Step 3. Repeat Step 2 until no more improvement is possible (i.e. a local optimum has been reached).

One of the best known approaches of this type is the k-opt exchange heuristic of Lin[5]. In Step 2 of the generic framework, k links in the current routes are exchanged for k new links. If the routes cover N links, there are $O(N^k)$ ways to select the k links to be replaced in order to generate a new solution. The set of new solutions is usually referred to as the 'neighbourhood' of the current solution. These solutions are generated and evaluated one by one, and the first solution that provides an improvement over the current solution is selected as the new current solution.

The complexity of Step 2 of the generic framework is typically polynomial, but the number of iterations needed to reach a local optimum is exponential in the worst case[6]. However, apart from some rare pathological cases, this approach quickly converges to a local optimum. In the following, we will refer to an $O(N^k)$ exchange heuristic when the size of the neighbourhood is $O(N^k)$.

When a problem is constrained, each modification to the current solution must also be checked for feasibility. The feasibility test is straightforward for problems with capacity constraints, because the total demand on a route does not depend on the ordering of the customers. Accordingly, the demand shifted from one route to another during an exchange is simply added or subtracted to the current load and compared with the capacity. On the other hand, the ordering of the customers affects solution feasibility for problems with time windows. In the worst case, the service time at each customer must be checked to assess the feasibility of the new solution. Accordingly, the feasibility test is of the order of $O(N)$. However, different techniques have been proposed to speed-up the computation[7,8]. In particular, Savelsbergh proposes a method that reduces the checking effort to constant time[8]. Other techniques have also been proposed by Psaraftis in the context of precedence-constrained routeing problems[9].

In the following, we introduce our 2-opt* exchange heuristic and the well-known Or-opt[4]. Both approaches can be used to generate high quality solutions in a reasonable amount of computation time, because the size of their neighbourhood is of the order of $O(N^2)$. In this paper, the 3-opt exchange heuristic is used for comparison purposes. The size of the 3-opt neighbourhood is of the order of $O(N^3)$, and this method is much more computationally expensive.

A 2-OPT* EXCHANGE HEURISTIC

Classical k-opt exchange heuristics are not well adapted to problems with time windows, because most exchanges do not preserve the orientation of the routes. Figure 1 provides an example of a 2-opt exchange, where links $(1, 2)$ and $(3, 4)$ are replaced by links $(1, 3)$ and $(2, 4)$. Since customers are typically sequenced according to their time window's upper bound (those with the lowest upper bounds being the first to be serviced), reversing some portion of a route is likely to produce an infeasible solution.

For problems with multiple routes, it is possible to look at another way of exchanging two pairs of links. First, we use the fact that any MTSP problem can be transformed into an equivalent Travelling Salesman Problem (TSP)[10,11]. We get this equivalence by creating M copies of the depot, each connected to the other nodes in the same way as the original depot. In such a network, any MTSP solution looks as a TSP solution (see Figure 2(a)). Then, a

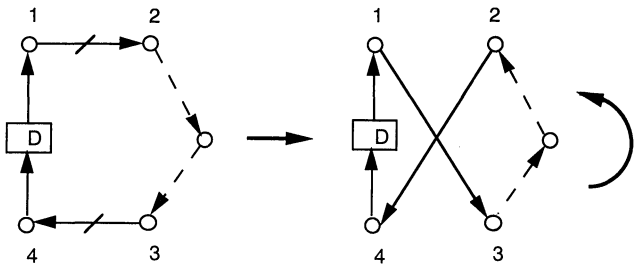

FIG. 1. *Exchange of links* (1, 2), (3, 4) *for links* (1, 3), (2, 4).

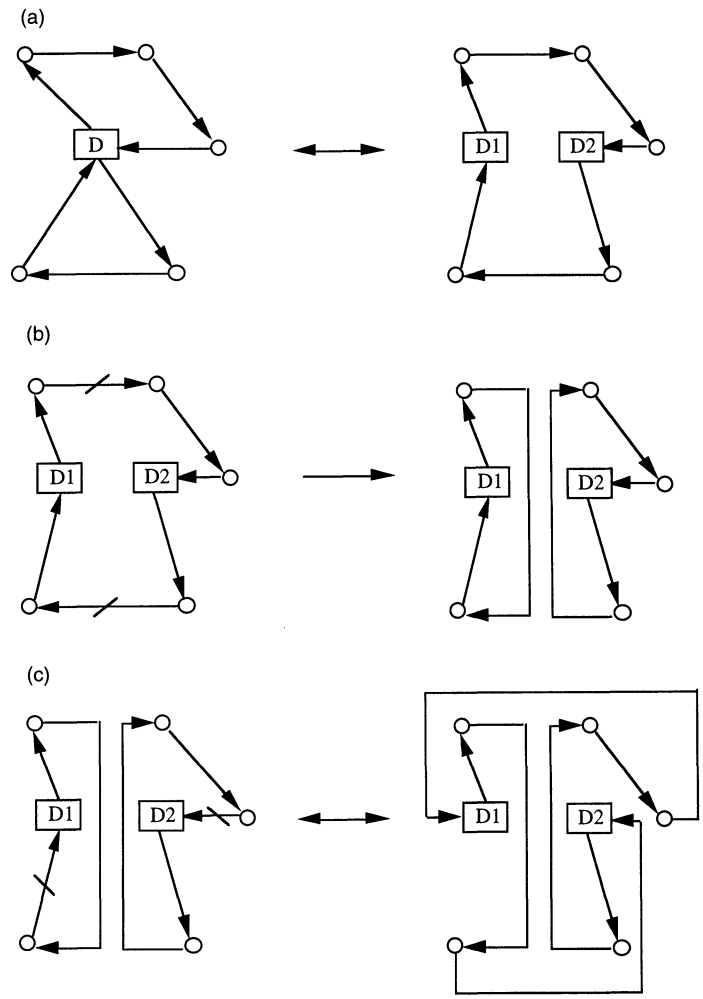

FIG. 2. *A 2-opt* exchange.*

special 2-opt* exchange is applied: two links are replaced by two new links, so as to break the tour into two different subtours (see Figure 2(b)). Of course, the new solution is valid if and only if there is at least one copy of the depot in each subtour.

The 2-opt* exchange is particularly powerful for problems with time windows, because it preserves the orientation of the routes, and introduces the last customers of a given route (i.e. those with late time windows) at the end of the first customers of another route (i.e. those with early time windows). Hence, the new solution is likely to be feasible. As depicted in

Figure 2(c), the subtours are then easily merged back into an equivalent single tour by linking the last customer in each subtour to the other copy of the depot (note that the links connecting a given customer to each copy of the depot are all equivalent).

If we look at the solution before and after the 2-opt* exchange, four new links have been introduced into the solution, including the two links for merging the two subtours into an equivalent single tour. However, given that the last two links are determined by the first two (i.e. there is no choice), the size of the 2-opt* neighbourhood is of the order of $O(N^2)$.

It is worth noting that the 2-opt* heuristic is useful when the two links to be replaced are in two different routes. It is not useful otherwise, because a subtour with no copy of the depot is created. We can easily conclude that the 2-opt* heuristic is not useful if the initial route construction heuristic produces an optimal clustering of the customers (i.e. if only sequencing adjustments within the routes can improve the solution).

In order to get inter-route as well as intra-route improvements, a classical 2-opt exchange is applied when the links to be replaced are in the same route, and a 2-opt* exchange is applied otherwise. When the time windows are not very tight, reversing a small portion of a route may be feasible. In this context, the intra-route 2-opt exchanges can provide some improvement. In the following, we refer to this approach as the 2-opt/2-opt* exchange heuristic.

The next section now introduces two other powerful exchange heuristics for problems with time windows, namely the Or-opt-1 and Or-opt.

THE OR-OPT-1 AND OR-OPT EXCHANGE HEURISTICS[4]

These two approaches are well-known 'node exchange heuristics'. An Or-opt-1 exchange considers each customer in turn in the current solution and tries to improve the solution by inserting the customer at another location. The Or-opt heuristic extends Or-opt-1 by considering sequences of one, two and three adjacent customers in the solution (see Figure 3).

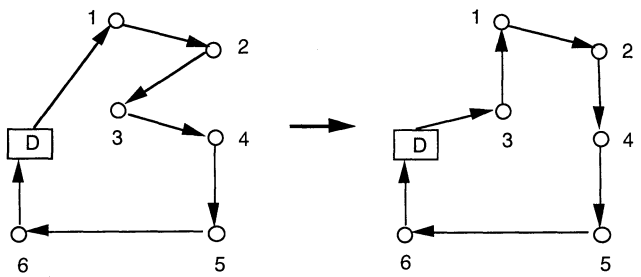

FIG. 3. *Moving the sequence of customers* (1, 2) *between customers* 3 *and* 4.

Although both heuristics are focusing on nodes rather than links, it is easy to see that the Or-opt exchanges are a subset of 3-opt exchanges. As mentioned before, most 3-opt exchanges do not preserve the orientation of the routes, and are likely to generate infeasible solutions. Accordingly, the Or-opt heuristic focuses on a small subset of 3-opt exchanges that are more likely to be feasible. It moves only small sequences of adjacent customers, which are typically 'close' together from a spatial and temporal point of view, and inserts them at a new location, while preserving the orientation of the routes. In this way, the Or-opt exchange heuristic can generate solutions that are close in quality to those of a 3-opt for problems with time windows, but in much less computation time.

As opposed to the 2-opt* exchanges, the Or-opt-1 and Or-opt exchanges are suitable for inter-route and intra-route modifications. The next section will now explain how to use the strengths of the Or-opt and 2-opt* methods to build a powerful hybrid heuristic.

THE HYBRID 2-OPT*/OR-OPT HEURISTIC

Some interesting points emerged from the previous description of the Or-opt and 2-opt* heuristics. First, these methods explore different neighbourhoods of the current solution (although, in a few cases, a 2-opt* exchange can be equivalent to an Or-opt exchange). Secondly, the 2-opt* introduces important inter-route modifications, while the Or-opt only slightly modifies the solution. In other words, the 2-opt* heuristic moves in the solution space by considering new solutions that are very different from the current one, while the Or-opt focuses on finer refinements. These considerations underline what could be gained by merging these two 'complementary' approaches into a hybrid one, and by exploring both neighbourhoods.

We tested two different implementations of this hybrid heuristic.

(1) In the first implementation, the 2-opt* and Or-opt neighbourhoods are explored in turn. First, the 2-opt* heuristic is applied until a 2-opt* optimal solution is found (that is, a local optimum with respect to the 2-opt* exchange heuristic). Then, the Or-opt exchange heuristic is applied to the 2-opt* optimal solution until an Or-opt optimal solution is found. This procedure is repeated until the solution stabilizes (i.e. when it is 2-opt* *and* Or-opt optimal).

    In a second attempt, we start again from the initial solution and repeat the whole procedure, with the Or-opt now applied before the 2-opt*.

    At the end, we simply select the best of the two solutions.

(2) In the second implementation, the Or-opt and 2-opt* neighbourhoods are merged together. That is, we look both at Or-opt and 2-opt* exchanges for an improvement to the current solution.

    First, the two neighbourhoods are merged in such a way that all 2-opt* exchanges are considered before the Or-opt exchanges.

    In a second attempt, and starting again from the initial solution, the two neighbourhoods are merged in such a way that all Or-opt exchanges are considered before the 2-opt* exchanges.

    At the end, we select the best of the two solutions.

It should be noted that a detailed description of the two implementations is given in the appendix.

COMPUTATIONAL RESULTS

In order to evaluate the new hybrid heuristic introduced in the previous section, a first experiment was performed on a set of randomly generated problems with tight windows, large windows, and a mix of tight and large windows (but no capacity constraints). Then, the experiment was repeated with Solomon's standard set of VRSPTWs[2]. The heuristics were coded in C, and the tests were run on a Sparc workstation. It is worth noting that these implementations do not exploit Savelbergh's method for checking solution feasibility in constant time[8]. However, since all heuristics tested in this paper would have equally benefited from this technique, the comparison between the heuristics in regard to the computation time holds whether the feasibility test is of the order of $O(N)$ or $O(1)$ (clearly, Savelsbergh's method does not impact solution quality).

In each experiment, the overall objective was to minimize the number of routes first, and then, to minimize the total route time (i.e. the sum of travel time, waiting time and service time). In other words, a solution with fewer routes was always preferred over a solution with more routes, independently of the total route time. The service time at each customer was set to 0 for the random problems. For Solomon's problems, a total service time value of 9000 must be added to the sum of travel time and waiting time for each problem in sets C1 and C2, while a value of 1000 is added to each problem in the remaining sets.

Note finally that the travel times are equivalent to the corresponding Euclidean distances.

Random problems

For these experiments, we created Euclidean problems with $N = 100$ customers, by randomly generating their coordinates in a $[0, 100] \times [0, 100]$ square, using a uniform distribution.

The centre of each time window was randomly generated within the scheduling horizon. We created problems of type T with tight time windows (5 to 10% of the horizon), problems of type L with large time windows (20 to 25% of the horizon), and problems of type M with a 50:50 mix of tight and large time windows. Problem sets T1, L1 and M1 have a narrow scheduling horizon, and allow only a few customers per route, while problem sets T2, L2 and M2 have a large horizon and allow more customers per route.

The results are presented in Tables 1, 2 and 3. In these tables, Or-opt/2-opt*(1) refers to

TABLE 1. *Computational results with tight time windows*

	No. routes	Travel time	Waiting time	Route time	Comput. time (min:sec)
T1					
Initial solution	13.95	2505.9	602.6	3108.5	0:01
2-opt*	13.95	2474.1	574.9	3049.0	0:06
2-opt/2-opt*	13.95	2468.4	575.0	3043.4	0:14
Or-opt-1	13.90	2467.8	575.0	3042.8	0:06
Or-opt	13.90	2461.7	574.6	3036.3	0:19
2-opt*/Or-opt(1)	13.85	2447.9	562.3	3010.2	0:45
2-opt*/Or-opt(2)	13.85	2443.9	569.1	3013.0	0:53
2-opt*/Or-opt(1) + (2)	13.85	2441.6	565.6	3007.2	1:38
3-opt	13.90	2471.4	548.1	3019.5	17:32
T2					
Initial solution	4.05	2734.7	910.5	3645.2	0:05
2-opt*	4.05	2747.2	866.8	3614.0	0:02
2-opt/2-opt*	4.05	2723.1	877.6	3600.7	0:07
Or-opt-1	4.05	2726.3	880.1	3606.4	0:03
Or-opt	4.05	2723.1	877.6	3600.7	0:09
2-opt*/Or-opt(1)	4.05	2716.3	872.1	3588.4	0:24
2-opt*/Or-opt(2)	4.05	2710.4	875.7	3586.1	0:26
2-opt*/Or-opt(1) + (2)	4.05	2711.6	870.7	3582.3	0:50
3-opt	4.05	2725.7	862.6	3598.3	12:36

TABLE 2. *Computational results with large time windows*

	No. routes	Travel time	Waiting time	Route time	Comput. time (min:sec)
L1					
Initial solution	9.40	1855.7	245.1	2100.8	0:01
2-opt*	9.40	1842.8	213.1	2055.9	0:04
2-opt/2-opt*	9.40	1808.9	213.9	2022.8	0:16
Or-opt-1	9.40	1768.5	228.6	1997.1	0:13
Or-opt	9.40	1757.2	226.7	1983.9	0:37
2-opt*/Or-opt(1)	9.35	1742.5	211.2	1953.7	1:20
2-opt*/Or-opt(2)	9.35	1748.6	206.3	1954.9	1:38
2-opt*/Or-opt(1) + (2)	9.35	1748.9	202.5	1951.4	2:58
3-opt	9.35	1765.5	190.6	1956.1	35:52
L2					
Initial solution	3.00	2037.2	412.5	2449.7	0:04
2-opt*	3.00	2059.7	368.8	2428.5	0:01
2-opt/2-opt*	3.00	2006.9	370.4	2377.3	0:11
Or-opt-1	3.00	1993.3	352.8	2346.1	0:08
Or-opt	3.00	1998.6	344.2	2342.8	0:22
2-opt*/Or-opt(1)	3.00	1991.0	321.8	2312.8	0:52
2-opt*/Or-opt(2)	3.00	1995.6	323.9	2319.5	1:07
2-opt*/Or-opt(1) + (2)	3.00	1990.1	316.8	2306.9	1:59
3-opt	2.95	1980.9	327.1	2308.0	23:50

TABLE 3. *Computational results with mixed time windows*

	No. routes	Travel time	Waiting time	Route time	Comput. time (min:sec)
M1					
Initial solution	11.65	2205.7	413.5	2619.2	0:01
2-opt*	11.65	2184.6	377.6	2562.2	0:05
2-opt/2-opt*	11.65	2178.0	369.1	2547.1	0:15
Or-opt-1	11.65	2158.7	376.5	2535.2	0:09
Or-opt	11.50	2152.5	370.2	2522.7	0:25
2-opt*/Or-opt(1)	11.50	2137.1	349.6	2486.7	0:59
2-opt*/Or-opt(2)	11.50	2138.6	357.2	2495.8	1:12
2-opt*/Or-opt(1) + (2)	11.50	2134.6	351.0	2485.6	2:11
3-opt	11.55	2175.5	343.2	2518.7	27.31
M2					
Initial solution	3.50	2512.2	571.9	3084.1	0:06
2-opt*	3.50	2482.2	556.4	3038.6	0:02
2-opt/2-opt*	3.50	2449.7	554.1	3003.8	0:10
Or-opt-1	3.50	2470.4	532.1	3002.5	0:04
Or-opt	3.50	2463.3	530.2	2993.5	0:13
2-opt*/Or-opt(1)	3.50	2448.7	516.7	2965.4	0:34
2-opt*/Or-opt(2)	3.50	2438.6	518.8	2957.4	0:38
2-opt*/Or-opt(1) + (2)	3.50	2436.8	519.5	2956.3	1:12
3-opt	3.50	2470.1	518.0	2988.1	18:35

the first implementation of the hybrid exchange heuristic with the alternative exploration of each individual neighbourhood, whereas Or-opt/2-opt*(2) refers to the second implementation with the exploration of a unique enlarged neighbourhood. Or-opt/2-opt*(1) + (2) is the best solution produced by the two previous heuristics on each problem.

Each number is the average taken over 20 problems. For each exchange heuristic, the average numbers of routes, travel time, waiting time, route time and computation time (in minutes and seconds) are shown. Note that the computation time of Or-opt/2-opt*(1) + (2) is the sum of the computation times of Or-opt/2-opt*(1) and Or-opt/2-opt*(2). Note also that the initial solutions were produced by running Solomon's I1 heuristic[2].

The headings in the tables are self-explanatory, but the followings should be emphasized:

(a) *Comput. time* is the overall time to get to the final solution from the initial solution;

(b) *Route time* is the sum of travel time, waiting time and service time.

From the results reported in Tables 1, 2 and 3, the following observations can be made.

(1) The 2-opt*/Or-opt heuristic outperforms 3-opt with respect to solution quality and computation time. The 3-opt is superior to the hybrid heuristic on problem set L2 only (allowing an additional route to be saved). Generally speaking, the 3-opt performs better on problems of type L. This result is not surprising because our heuristic was specifically designed for problems with time windows. Accordingly, it is not as effective when these constraints are relaxed.

(2) The 2-opt* exchange heuristic is very fast, and is quite good on problems with tight time windows. However, it is not as effective on problems of type L and M. For example, the 2-opt/2-opt* heuristic is better than 2-opt* on the latter problems, because reversing some portion of a route with a 2-opt is more likely to be feasible when the time windows are larger.

(3) Or-opt outperforms 2-opt/2-opt* on all problem sets. However, 2-opt/2-opt* is more competitive on problems with tight time windows. Given that Or-opt dominates 2-opt*, the use of 2-opt* within the hybrid 2-opt*/Or-opt heuristic can be viewed as a way of adding some additional power to Or-opt. This additional power is obtained at the cost of additional computation time, given that the local optimum is now found over a larger neighbourhood.

(4) Both implementations of the hybrid exchange heuristic are equally good. Hence, the methodology for exploring the neighbourhood of the current solution has no significant impact on the quality of the final solution.

These results indicate that the hybrid 2-opt*/Or-opt exchange heuristic should be considered when time windows are the main concern. It generates high quality solutions in much less computation time than a 3-opt. Accordingly, the hybrid heuristic is really focusing on useful exchanges for problems with time windows.

Solomon's test problems

In a second experiment, we applied the above exchange heuristics to Solomon's standard set of VRSPTWs[2]. The design of these problems highlights factors that can affect the behaviour of routeing and scheduling heuristics, like geographical data, number of customers serviced by a vehicle and time window characteristics (e.g. percentage of time-constrained customers, tightness and positioning of the time windows, width of the time horizon). Accordingly, this set of problems is aimed at testing the robustness of the exchange heuristics on a wide variety of problems.

The geographical data were either randomly generated with a uniform distribution (problem sets R1 and R2), clustered (problem sets C1 and C2) or mixed with randomly distributed and clustered customers (problem sets RC1 and RC2). Problem sets R1, C1 and RC1 have a narrow scheduling horizon, and allow only a few customers per route. Conversely, problem sets R2, C2 and RC2 have a large scheduling horizon and allow more customers per route. For problems of type R and RC, the time windows have a uniformly distributed, randomly generated centre and a normally distributed random width. For problems of type C, the time windows are positioned around the arrival times of customers after a 3-opt, cluster by cluster, routeing solution. Finally, all problems are 100-customer Euclidean problems and include capacity constraints, in addition to the time windows.

The results are shown in Tables 4, 5 and 6, using the format of Tables 1 to 3 (remember that a service time value of 1000 must be added to the sum of travel time and waiting time for the problems of type R and RC, and a value of 9000 for the problems of type C). In the Tables, the initial solutions were produced by running Solomon's I1 heuristic[2].

We can see that the 3-opt exchange heuristic is still outperformed by 2-opt*/Or-opt(1) + (2) on these problems. In particular, the hybrid heuristic outperforms 3-opt on four problem sets, namely R1, R2, C2 and RC1, and it is much less computationally expensive.

Generally speaking, the results on Solomon's test set support the previous results obtained

TABLE 4. *Computational results on problem set R*

	No. routes	Travel time	Waiting time	Route time	Comput. time (min:sec)
R1					
(12 problems)					
Initial solution	13.6	1436.7	258.8	2695.5	0:01
2-opt*	13.5	1403.6	223.0	2626.6	0:08
2-opt/2-opt*	13.5	1427.6	140.6	2568.2	0:21
Or-opt-1	13.5	1422.1	162.3	2584.4	0:09
Or-opt	13.4	1413.1	155.0	2568.1	0:26
2-opt*/Or-opt(1)	13.4	1394.3	128.5	2522.8	1:17
2-opt*/Or-opt(2)	13.4	1389.9	138.0	2527.9	1:24
2-opt*/Or-opt(1) + (2)	13.3	1381.9	127.6	2509.5	2:41
3-opt	13.3	1401.5	130.3	2531.8	27:03
R2					
(11 problems)					
Initial solution	3.3	1402.4	175.6	2578.1	0:06
2-opt*	3.3	1403.6	124.9	2528.5	0:02
2-opt/2-opt*	3.3	1374.6	111.6	2486.2	0:15
Or-opt-1	3.3	1349.1	88.4	2437.5	0:17
Or-opt	3.3	1339.8	89.7	2429.5	0:40
2-opt*/Or-opt(1)	3.3	1299.9	91.6	2391.5	1:47
2-opt*/Or-opt(2)	3.3	1307.4	84.5	2391.9	2:01
2-opt*/Or-opt(1) + (2)	3.3	1293.4	84.1	2377.5	3:48
3-opt	3.3	1316.5	74.8	2391.3	42:19

TABLE 5. *Computational results on problem set C*

	No. routes	Travel time	Waiting time	Route time	Comput. time (min:sec)
C1					
(9 problems)					
Initial solution	10.0	951.9	152.3	10104.2	0:01
2-opt*	10.0	952.5	44.4	9996.9	0:03
2-opt/2-opt*	10.0	944.4	45.7	9990.1	0:08
Or-opt-1	10.0	904.3	70.8	9975.1	0:05
Or-opt	10.0	907.9	66.1	9974.0	0:16
2-opt*/Or-opt(1)	10.0	918.6	29.0	9947.6	0:34
2-opt*/Or-opt(2)	10.0	906.3	19.3	9925.6	0:37
2-opt*/Or-opt(1) + (2)	10.0	902.9	11.7	9914.6	1:11
3-opt	10.0	898.4	6.7	9905.1	20:38
C2					
(8 problems)					
Initial solution	3.1	692.7	228.6	9921.4	0:04
2-opt*	3.1	757.1	45.9	9803.0	0:01
2-opt/2-opt*	3.1	741.1	47.4	9788.5	0:06
Or-opt-1	3.1	701.9	3.8	9705.7	0:12
Or-opt	3.1	683.6	3.2	9686.8	0:31
2-opt*/Or-opt(1)	3.1	669.3	3.2	9672.5	1:04
2-opt*/Or-opt(2)	3.1	662.3	5.7	9668.0	1:27
2-opt*/Or-opt(1) + (2)	3.1	653.2	3.2	9656.4	2:31
3-opt	3.1	675.3	6.7	9682.0	18:17

TABLE 6 *Computational results on problem set RC*

	No. routes	Travel time	Waiting time	Route time	Comput. time (min:sec)
RC1					
(8 problems)					
Initial solution	13.5	1596.5	178.5	2775.0	0:01
2-opt*	13.5	1567.1	116.2	2683.3	0:12
2-opt/2-opt*	13.5	1576.6	98.8	2675.4	0:19
Or-opt-1	13.5	1567.2	109.9	2677.1	0:14
Or-opt	13.4	1557.0	98.5	2655.5	0:38
2-opt*/Or-opt(1)	13.3	1542.0	80.4	2622.4	1:38
2-opt*/Or-opt(2)	13.3	1551.4	73.3	2624.7	2:15
2-opt*/Or-opt(1) + (2)	13.3	1545.3	70.4	2615.7	3:53
3-opt	13.4	1550.8	67.7	2618.5	52:12
RC2					
(8 problems)					
Initial solution	3.9	1682.1	273.2	2955.4	0:05
2-opt*	3.9	1713.3	197.6	2910.9	0:04
2-opt/2-opt*	3.9	1675.9	197.0	2872.9	0:14
Or-opt-1	3.9	1652.9	159.7	2812.6	0:16
Or-opt	3.9	1624.0	151.0	2775.0	0:46
2-opt*/Or-opt(1)	3.9	1621.2	105.0	2726.2	1:41
2-opt*/Or-opt(2)	3.9	1633.1	93.4	2726.5	2:05
2-opt*/Or-opt(1) + (2)	3.9	1595.1	116.0	2711.1	3:46
3-opt	3.8	1635.1	99.2	2734.3	36:17

with randomly generated problems. These numbers also indicate that the time windows are the main concern in Solomon's problems, given that our hybrid exchange heuristic does not offer any specific advantage with respect to the capacity constraints.

CONCLUSIONS

Overall, and with respect to solution quality, the results presented in this paper show the superiority of the hybrid 2-opt*/Or-opt heuristic over the classical 3-opt heuristic on

Solomon's test problems and on randomly generated problems. A new ingredient within the hybrid heuristic is the 2-opt* which focuses on a subset of exchanges that are particularly powerful when the time constraints are the main concern.

The hybrid heuristic is simple, easy to implement and produces good quality solutions within reasonable computation time. However, it is clear that by using more complex (and more computationally expensive) problem-solving methods, better solutions can be produced. For example, different methods based on cyclic transfers, tabu search, simulated annealing, genetic algorithms and hybrids have recently been proposed for solving different types of vehicle routeing problems[12–14].

Acknowledgements—We would like to thank Lucie Bibeau, Patrice Bergeron and Tanguy Kervahut who spent long hours running the computational experiments. Also, financial support for this research has been provided by the Natural Sciences and Engineering Research Council of Canada (NSERC).

APPENDIX

Pseudo-code for the exchange heuristics

Note 1. Node 1 is always the starting point of the tour; suiv(i) is the successor of node i, and node 1 is the successor of node N; reverse(i, j) reverses the orientation of the tour between nodes i and j.

Note 2. In order to simplify the description of the exchange heuristics and avoid special cases that occur for small N, a sufficiently large value for N is assumed in each case.

2-opt exchange heuristic

Step 0. Generate an initial tour (1, 2, . . ., N) and set it as the current tour.
Step 1. Start the search on the current tour.
Step 2. $i_1 \leftarrow 1$
do count$_1$ = 1, $N - 2$
if count$_1$ = 1 then limit $\leftarrow N - 1$ else limit $\leftarrow N$
$i_2 \leftarrow$ suiv(i_1)
$j_1 \leftarrow$ suiv(i_2)
do count$_2$ = count$_1$ + 2, limit
$j_2 \leftarrow$ suiv(j_1)
/2-opt exchange on current tour/
reverse(i_2, j_1)
replace edges (i_1, i_2), (j_1, j_2) by edges (i_1, j_1), (i_2, j_2),
if the new tour is feasible and better than the current tour then set it as the current tour and go to Step 1.
$j_1 \leftarrow j_2$
end do
$i_1 \leftarrow i_2$
end do
Step 3. Output the current tour.

2-opt exchange heuristic*

Step 0. Generate an initial tour (1, 2, . . ., N) and set it as the current tour.
Step 1. Start the search on the current tour.
Step 2. $i_1 \leftarrow 1$
do count$_1$ = 1, $N - 2$
if count$_1$ = 1 then limit $\leftarrow N - 1$ else limit $\leftarrow N$
$i_2 \leftarrow$ suiv(i_1)
$j_1 \leftarrow$ suiv(i_2)

```
      do count2 = count1 + 2, limit
        j2 ← suiv(j1)
        /2-opt* exchange on current tour/
        if edges (i1, i2) and (j1, j2) are not in the same route then
          replace edges (i1, i2), (j1, j2) by edges (i1, j2), (j1, i2),
          if the two subtours are feasible and better than the current tour then construct
            an equivalent single tour, set it as the current tour and go to Step 1.
        j1 ← j2
      end do
      i1 ← i2
    end do
```

Step 3. Output the current tour.

Mixed 2-opt/2-opt exchange heuristic*

Step 0. Generate an initial tour (1, 2, . . ., N) and set it as the current tour.
Step 1. Start the search on the current tour.
Step 2.

```
i1 ← 1
do count1 = 1, N − 2
  if count1 = 1 then limit ← N − 1 else limit ← N
  i2 ← suiv(i1)
  j1 ← suiv(i2)
  do count2 = count1 + 2, limit
    j2 ← suiv(j1)
    /2-opt exchange on current tour/
    reverse(i2, j1)
    replace edges (i1, i2), (j1, j2) by edges (i1, j1), (i2, j2),
    if the new tour is feasible and better than the current tour then set it as the
     current tour and go to Step 1.
    /2-opt* exchange on current tour/
    if edges (i1, i2) and (j1, j2) are not in the same route then
      replace edges (i1, i2), (j1, j2) by edges (i1, j2), (j1, i2),
      if the two subtours are feasible and better than the current tour then construct
        an equivalent single tour, set it as the current tour and go to Step 1.
    j1 ← j2
  end do
  i1 ← i2
end do
```

Step 3. Output the current tour.

3-opt exchange heuristic

Note. There are seven ways to connect three chains in order to get a new tour. In this set, three are 2-opt exchanges. On the four remaining 3-opt exchanges, only two are really distinct (the two other ones can be obtained by rotation). We use this observation in the algorithm below.

Step 0. Generate an initial tour (1, 2, . . ., N) and set it as the current tour.
Step 1. Start the search on the current tour.
Step 2.

```
i1 ← 1
do count1 = 1, N
  i2 ← suiv(i1)
  j1 ← i2
  do count2 = 2, N − 3
    j2 ← suiv(j1)
    k1 ← suiv(j2)
```

do $count_3 = count_2 + 2, N - 1$
$k_2 \leftarrow suiv(k_1)$
/3-opt exchange on current tour/
replace edges (i_1, i_2), (j_1, j_2), (k_1, k_2) by (i_1, j_2), (k_1, i_2), (j_1, k_2),
if the new tour is feasible and better than the current tour then set it as the current tour and go to Step 1.
/3-opt exchange on current tour/
reverse(k_2, i_1)
replace edges (i_1, i_2), (j_1, j_2), (k_1, k_2) by (j_1, i_1), (k_2, j_2), (k_1, i_2),
if the new tour is feasible and better than the current tour then set it as the current tour and go to Step 1.
$k_1 \leftarrow k_2$
end do
$j_1 \leftarrow j_2$
end do
$i_1 \leftarrow i_2$
end do

Step 3. Output the current tour.

Or-opt-1 exchange heuristic

Step 0. Generate an initial tour $(1, 2, \ldots, N)$ and set it as the current tour.
Step 1. Start the search on the current tour.
Step 2. $i_1 \leftarrow 1$
do $count_1 = 1, N$
$i_2 \leftarrow suiv(i_1)$
$i_3 \leftarrow suiv(i_2)$
$i_4 \leftarrow suiv(i_3)$
$k_1 \leftarrow i_4$
do $count_2 = 1, N - 3$
$k_2 \leftarrow suiv(k_1)$
/Or-opt-1 exchange on current tour/
replace edges (i_1, i_2), (i_2, i_3) (k_1, k_2) by (i_1, i_3), (k_1, i_2), (i_2, k_2),
if the new tour is feasible and better than the current tour, set it as the current tour, and go to Step 1.
$k_1 \leftarrow k_2$
end do
$i_1 \leftarrow i_2$
end do

Step 3. Output the current tour.

Or-opt exchange heuristic

Step 0. Generate an initial tour $(1, 2, \ldots, N)$ and set it as the current tour.
Step 1. Start the search on the current tour.
Step 2. $i_1 \leftarrow 1$ /Or-opt-3 exchanges on current tour/
do $count_1 = 1, N$
do $j = 1, 7$
$i_{j+1} \leftarrow suiv(i_j)$
end do
$k_1 \leftarrow i_8$
do $count_2 = 1, N - 9$
$k_2 \leftarrow suiv(k_1)$
replace edges (i_1, i_2), (i_4, i_5) (k_1, k_2) by (i_1, i_5), (k_1, i_2), (i_4, k_2)
if the new tour is feasible and better than the current tour, set it as the current tour, and go to Step 1.

$k_1 \leftarrow k_2$
end do
$i_1 \leftarrow i_2$
end do
$i_1 \leftarrow 1$ /Or-opt-2 exchanges on current tour/
do $\text{count}_1 = 1, N$
do $j = 1, 5$
$i_{j+1} \leftarrow \text{suiv}(i_j)$
end do
$k_1 \leftarrow i_6$
do $\text{count}_2 = 1, N - 6$
$k_2 \leftarrow \text{suiv}(k_1)$
replace edges (i_1, i_2), (i_3, i_4) (k_1, k_2) by (i_1, i_4), (k_1, i_2), (i_3, k_2),
if the new tour is feasible and better than the current tour, set it as the current tour, and go to Step 1.
$k_1 \leftarrow k_2$
end do
$i_1 \leftarrow i_2$
end do
Step 2 of Or-opt-1 heuristic /Or-opt-1 exchanges on current tour/

Step 3. Output the current tour.

2-opt/Or-opt exchange heuristic (1)*

Step 0. Generate an initial tour (1, 2, . . ., N). Set initial_tour and current_tour1 to this tour.
/2-opt*/
Step 1. Start the 2-opt* search on current_tour1.
Step 2. Step 2 of 2-opt* exchange heuristic with current_tour1.
Step 3. If some exchange has been performed in Step 2 or current_tour1 is initial_tour go to Step 4, otherwise, go to Step 7.
/Or-opt/
Step 4. Start the Or-opt search on current_tour1.
Step 5. Step 2 of Or-opt exchange heuristic with current_tour1 (replacing 'go to Step 1' by 'go to Step 4')
Step 6. If some exchange has been performed in Step 5, go to Step 1.
Step 7. Set current_tour2 to initial_tour
/Or-opt/
Step 8. Start the Or-opt search on current_tour2.
Step 9. Step 2 of Or-opt exchange heuristic with current_tour2 (replacing 'go to Step 1' by 'go to Step 8').
Step 10. If some exchange has been performed in Step 9 or current_tour2 is initial_tour, go to Step 11, otherwise go to Step 14.
/2-opt*/
Step 11. Start the 2-opt* search on current_tour2.
Step 12. Step 2 of 2-opt* exchange heuristic with current_tour2 (replacing 'go to Step 1' by 'go to Step 11').
Step 13. If some exchange has been performed in Step 12, go to Step 8.
Step 14. Output the best of current_tour1 and current_tour2.

2-opt/Or-opt exchange heuristic (2)*

Step 0. Generate an initial tour (1, 2, . . ., N). Set initial_tour and current_tour1 to this tour.
/mixed 2-opt* and Or-opt/

Step 1. Start the mixed 2-opt* and Or-opt search on current_tour1.
Step 2. Step 2 of 2-opt* exchange heuristic with current_tour1.
Step 3. Step 2 of Or-opt exchange heuristic with current_tour1.
Step 4. Set current_tour2 to initial_tour
/mixed Or-opt and 2-opt*/
Step 5. Start the mixed Or-opt and 2-opt* search on current_tour2.
Step 6. Step 2 of Or-opt exchange heuristic with current_tour2 (replacing 'go to Step 1' by 'go to Step 5').
Step 7. Step 2 of 2-opt* exchange heuristic with current_tour2 (replacing 'go to Step 1' by 'go to Step 5').
Step 8. Output the best of current_tour1 and current_tour2.

REFERENCES

1. L. Bodin, B. Golden, A. Assad and M. Ball (1983) Routing and scheduling of vehicles and crews: the state of the art. *Comps & Opns Res.* **10** (2), 63–211.
2. M. M. Solomon (1987) Algorithms for the vehicle routing and scheduling problem with time window constraints. *Opns Res.* **35**, 254–265.
3. J. Y. Potvin and J. M. Rousseau (1993) A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *Eur. J. Opl Res.* **66**, 331–340.
4. I. Or (1976) Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking. PhD Thesis, Department of Industrial Engineering and Management Sciences, NorthWestern University.
5. S. Lin (1965) Computer solutions of the traveling salesman problem. *Bell Syst. Tech. J.* **44**, 2245–2269.
6. D. S. Johnson, C. H. Papadimitriou and M. Yannakakis (1985) How easy is local search? In *26th IEEE Annual Symposium on Foundations of Computer Science* pp. 39–42. IEEE Press, New York.
7. M. M. Solomon, E. K. Baker and J. R. Schaffer (1988) Vehicle routing and scheduling problems with time window constraints: efficient implementation of solution improvement procedures. In *Vehicle Routing: Methods and Studies*. (B. L. Golden and A. A. Assad, Eds) pp. 85–106. North-Holland, Amsterdam.
8. M. W. P. Savelsbergh (1986) Local search in routing problems with time windows. *Ann. Opns Res.* **4**, 285–305.
9. H. N. Psaraftis (1983) *k*-Interchange procedures for local search in a precedence-constrained routing problem. *Eur. J. Opl Res.* **13**, 391–402.
10. M. Bellmore and S. Hong (1974) Transformation of multi-salesman problem to the standard traveling salesman problem. *J. ACM* **21**, 500–504.
11. J. Svestka and V. Huckfeldt (1973) Computational experience with an *M*-salesmen traveling salesman algorithm. *Mgmt Sci.* **19**, 790–799.
12. P. M. Thompson and H. N. Psaraftis (1993) Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Opns Res.* **41**, 935–946.
13. J. Y. Potvin, T. Kervahut, B. L. Garcia and J. M. Rousseau (1993) A tabu search heuristic for the vehicle routing problem with time windows. Technical Report CRT-755, Centre de recherche sur les transports, Université de Montréal, Montréal, Canada.
14. S. R. Thangiah, I. H. Osman and T. Sun (1994) Hybrid genetic algorithms, simulated annealing and tabu search methods for vehicle routing problems with time windows. Technical report SRU-CpSc-TR-94-27, Slippery Rock University, Slippery Rock, PA.

Received September 1993; accepted March 1995 after one revision