Journal of Heuristics, 8: 43–58, 2002
© 2002 Kluwer Academic Publishers. Manufactured in The Netherlands.

Using Constraint-Based Operators to Solve the Vehicle Routing Problem with Time Windows

LOUIS-MARTIN ROUSSEAU AND MICHEL GENDREAU
Département informatique et recherche opérationnelle, University of Montreal, C.P. 6128 Succ. Centre-Ville, Montréal, Canada, H3C 3J7; Centre for Research on Transportation, University of Montreal, C.P. 6128 Succ. Centre-Ville, Montréal, Canada, H3C 3J7
email: louism@crt.umontreal.ca
email: michelg@crt.umontreal.ca

GILLES PESANT
Département de génie électrique et de génie informatique, École Polytechnique de Montréal, C.P. 6079 Succ. Centre-Ville, Montréal, Canada, H3C 3A7; Centre for Research on Transportation, University of Montreal, C.P. 6128 Succ. Centre-Ville, Montréal, Canada, H3C 3J7
email: pesant@crt.umontreal.ca

Abstract

This paper presents operators searching large neighborhoods in order to solve the vehicle routing problem. They make use of the pruning and propagation techniques of constraint programming which allow an efficient search of such neighborhoods. The advantages of using a large neighborhood are not only the increased probability of finding a better solution at each iteration but also the reduction of the need to invoke specially-designed methods to avoid local minima. These operators are combined in a variable neighborhood descent in order to take advantage of the different neighborhood structures they generate.

Key Words: constraint programming, local search method, metaheuristic, hybrid method, vehicle routing problem, variable neighborhood descent, combinatorial optimization

1. Introduction

The operations research (OR) field has produced in the last decades numerous algorithms and optimization methods that are both effective and efficient. These methods, based on Mathematical Programming or Metaheuristics, are being used to solve most of today's logistic problems. Recently issued from the field of artificial intelligence (AI), the constraint programming (CP) paradigm provides very good modeling flexibility, which combined with the complete separation between the model and the search, creates a tool to solve real world problems. However, the usual search method used in CP often shows prohibitive execution times for problems of reasonable size, despite recent research on improving search techniques that minimize this drawback (Harvey and Ginsberg, 1995, Meseguer and Walsh 1998).

The combination of optimization methods issued from operations research, artificial intelligence and constraint programming thus seems to be fertile, as these research area show complementary advantages that could be combined while minimizing drawbacks. Pesant and Gendreau have proposed a general framework for this integration (Pesant and Gendreau, 1996, 1999) and shown how it could be applied to a vehicle routing problem (Pesant, Gendreau and Rousseau, 1997). Baptiste, Le Pape and Nuijten (1995) used algorithms taken from OR to build a constraint scheduling system. Caseau and Laburthe (1998) defined a language for the design of hybrid algorithms. The Ilog team, following the work of Pascal Van Hentenryck (1999) has recently commercialized OPL studio a tool for the rapid development of algorithms integrating mathematical and constraint programming.

This paper presents operators that make use of constraint programming to perform local search. These operators are combined in a variable neighborhood descent (VND) framework to solve the vehicle routing problem with time windows (VRPTW).

1.1. The VRPTW

Vehicle routing problems (VRP) are omnipresent in today's industries, ranging from distribution problem to fleet management. They account for a significant portion of the operational cost of many companies. The VRP can be described as follows: given a set of customers C, a set of vehicles V, and a depot d, find a set of routes, starting and ending at d, such that each customer in C is visited by exactly one vehicle in V. Each customer having a specific demand, there are usually capacity constraints on the load that can be carried by a vehicle. In addition, there is a maximum amount of time that can be spent on the road. The time window variant of the problem (VRPTW) imposes the additional constraint that each customer c must be visited after time b_c and before time e_c. One can wait in case of early arrival, but late arrival is not permitted.

The objective function for this class of problem varies from one instance to the other. The primary objective might be to reduce the number of vehicles, the total travel distance or even the time spent on the road. But most of the times these objectives, are considered simultaneously, in a hierarchical fashion.

1.2. Hybrid approach

Some hybrid algorithms using constraint programming have already proven the efficiency of the paradigm on this problem. De Backer and Furnon (1997) have proposed to use constraint programming to validate solutions generated by some heuristics, and Shaw (1998) proposes to use CP to re-optimize an insertion-based restriction of the VRPTW. Caseau and Laburthe have developed a method that makes use of incremental local optimization as customers are inserted one by one in the solution. Constraint programming is then used to solve the individual routes as traveling salesman problems and thus validate the insertion (Caseau and Laburthe, to appear). They also proposed, in Caseau, a set of operators using constraint-based insertions and ejections, which they combine in different metaheuristics using learning techniques.

Our method is somewhat similar to the latter, since we developed a set of constraint-based operators that are later on assembled to generate a solution method. We also follow the line of thought of Pesant and Gendreau (1999), in the sense that we propose to use constraint programming to search for improving solutions in a neighborhood.

2. Operators

In a traditional metaheuristic context, an operator is defined as a recipe to modify a solution and obtain a potentially better one. An operator defines a neighborhood, that is the set of solutions that can be produced by applying that operator on one solution. A move is a transition from one solution to another one in its neighborhood. It is easy to understand that larger neighborhoods will tend to include better solutions, but the time needed to explore those neighborhoods, if we have to look at every solution, grows rapidly and substantially limits the scope of an operator. The idea, originally introduced by Pesant and Gendreau in (1996, 1999), is to explore the set of neighboring solutions with a branch-and-bound search in a constraint programming framework. Propagation and pruning are thus implicitly used to eliminate subsets of neighbors, which limits the number of solutions that are actually visited.

2.1. Operator: LNS-GENI

The first operator introduced is inspired by ideas from large neighborhood search presented by Shaw (1998). The algorithm first removes a subset of customers from the solution, and then looks for a better way to reinsert them in the partial solution. LNS obtains very good results on benchmark problems that are quite constrained, which is the case for Solomon class 1 problems where the time windows and capacities are tight and the solution consists of a good number of small routes. We think that this is due to class 1 problems being mainly partitioning problems, that is for a given list of customers for each route, getting the optimal routing (solving the TSPTW for each route) is easy. Solving problems where the routing component is more present like Solomon's class 2 problems is still difficult using LNS. And indeed no computational results were given for that class.

So the idea is to perform LNS but instead of performing simple insertion we use a constraint-programming version of the GENI algorithm. The GENeralized Insertion algorithm (Gendreau, Hertz and Laporate, 1992) and its time window variant (Gendreau et al., 1995) try to insert a customer in a given route between any two customers of that route: if the chosen customers are not consecutive, a local optimization is performed to make the insertion possible. The constraint programming version developed by Pesant, Gendreau and Rousseau (1997) differs from the original by the fact that instead of looking at all possible local optimizations, it looks for the best one using branch and bound with constraint propagation.

The customers to be removed are chosen randomly but a good bias towards customers generating the longest detour is introduced. The idea is that removing a customer situated far away from the rest of the route will create temporal space in that route and thus facilitate future insertions. The selection strategy also tends to reduce the total travel distance as it tends to replace long arcs by shorter ones. To select the set of customers to be removed, we first order them according to the detour they generate in their route, and then select them randomly with the ith customer having a selection probability of $2i/(n^2+n)$, where n is the number of customers.

To reintroduce the removed customers, we proceed following a *first fail* strategy, attempting the reinsertion of the most constrained customers first. Customer constrainedness can be defined by various means, usually depending on the problem instance that is addressed.

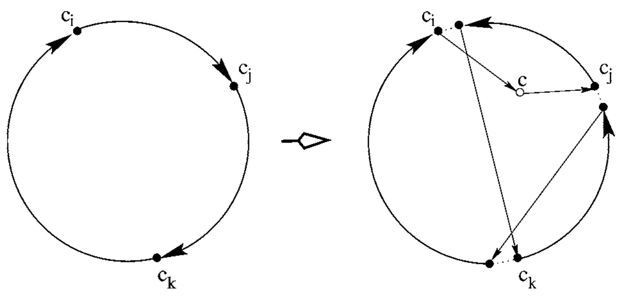

Figure 1. A GENeralized Insertion.

Choices include demand, relative position or time window width: in our case the key constraints appeared to be time windows, the insertion order is thus based on time window width. Figure 1 illustrates a GENI where the optimal insertion point for c is between c_i in c_j.

2.2. *Operator NEC*

Ejection chains are well known in the vehicle routing field as they provide a powerful and complex way of moving customers from one vehicle to another. The principle behind ejection chains relies on the fact that some customer c_i of vehicle v_k could possibly be advantageously transferred to vehicle v_l if some customer c_j of v_l was not there. So the idea is to remove c_j of v_l in order to introduce c_i and place c_j elsewhere, which might cause another customer to be ejected and replaced, and so on. The process is called an ejection chain, as it is a series of insertion and ejection that ends with some customer being inserted without the need of an ejection.

The tricky part is to search for such a chain that has negative total differential cost, that is, when all customers have been reinserted, the total travel distance is less than the one before the chain was started. To do so, several methods exist. Glover (1996), Rego and Roucairol (1996), Thompson and Psaraftis (1993) and Gendreau et al. (1998) have all worked on this problem, suggesting different techniques using graph theory. For instance, Gendreau et al. (1998) propose to use a graph, where nodes would be the different vehicles and the arcs would be the possible customer transfers, and to solve a negative cycle problem to find an improving ejection chain.

The naive ejection chain (NEC) operator we proposed is defined in the same way as the general algorithm except for the ejection chain search process. In fact, NEC does not really search for an ejection chain that will reduce the total travel distance but rather for a chain that will permit to remove a given customer from a given vehicle. The idea behind such a choice is to try to remove all the customers from a given vehicle in order to eliminate it.

The algorithm is defined as follows. To eliminate a vehicle v_l we first remove one of its customer c_i (which we will call the floating customer) and we try to insert it in another route. If this succeeds then the ejection chain is complete. If no insertion is possible, we try to identify another customer c_j in vehicle v_k such that c_i could be inserted in v_k if c_j was ejected. If such c_j exist we perform the insertion-ejection and restart the process with c_j as the floating customer. In the case where no ejection allows the insertion of the floating customer,

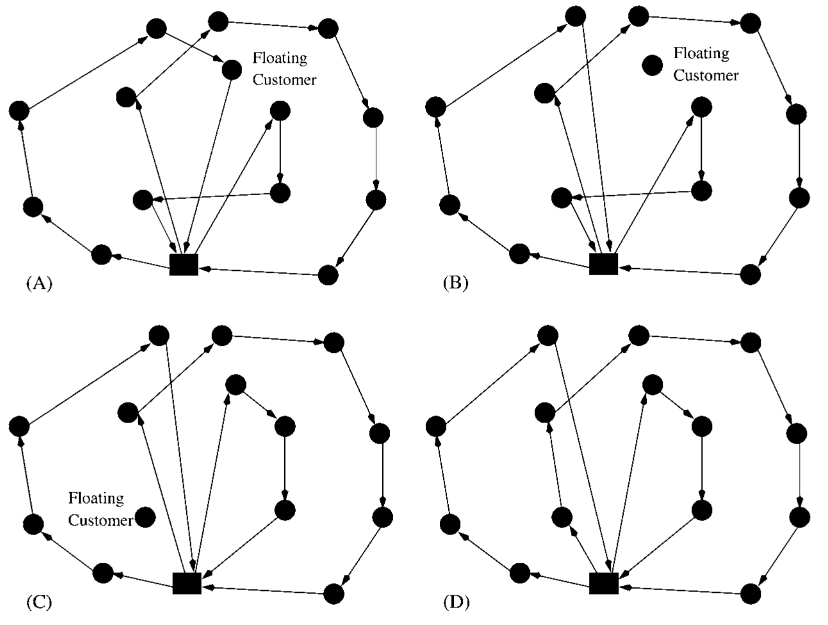

Figure 2. NEC (A) original solution (B) *Floating customer* ejected (C) An insertion-ejection and new *Floating customer* (D) new solution.

we backtrack and chose another floating customer. In order to limit the search space, we always accept the first completed ejection chain we find. Figure 2 illustrates this process.

Some restrictions have to be applied in order to prevent the algorithm from cycling and to generate the desired effect of vehicle reduction. Firstly, if we want to eliminate one vehicle we must forbid any insertion in that vehicle anywhere along the chain. Secondly, there are several strategies that could be used to prevent cycling, we chose to simply store the value (Customer) and thus prevent each customer from being moved more then once.

We could have also stored the couple (Customer,Vehicle) and prevent the same customer from being inserted twice in the same vehicle, which would have been a more refined strategy. Unfortunately, this generates a neighborhood too large to be searched efficiently.

Finally, we need an ejection-candidate selection algorithm for which several strategies can be considered. We could decide, for instance, to look for ejection-candidate by looking at only one route at a time, and if no possible ejection is found, then consider another route. We could also evaluate each customer independently from their route, sorting them with a particular criterion. A logical criterion could be the distance between the candidate and the floating customer, the similitude of their time windows or, in instances where capacities are a key factor, the demand of the candidate. We experimented with those strategies without much success, therefore we considered a criterion similar to one we already had used in LNS-GENI. In the final implementation, we choose the ejection-candidate that generates the maximum detour in its route, hoping that its ejection will create sufficient temporal space for the insertion of the floating customer, as well as reduce the total travel distance.

The whole search process is performed using the natural backtracking mechanism of a CP solver. The operations of inserting and removing a customer are modeled as constraints on a working copy of the current solution and the ability to validate (prune) feasible (infeasible) solutions is left to the original model. To find an ejection chain, starting with a given customer c already removed from its vehicle, we simply need to solve a constraint satisfaction problem (CSP) similar to:

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

where T and C are variables representing any tour or customer.

2.3. Operator SMART

The SMAll RouTing operator bears its name because it actually solves a smaller VRP that is a restriction of the original one. The general concept is that, instead of removing customers, we remove arcs from the problem thus creating an incomplete solution to the original problem. This incomplete solution could be interpreted as a smaller VRP in two different ways. The first one is to suppose that the freed arcs are all consecutive. The last customer before and the first customer after the freed sequences of all routes would then become the new depots, and all we would need to do is solve (exactly or not) the smaller problem. A second way of interpreting the relaxation is to consider the general case where the removed arcs are randomly distributed across the solution: we could then replace each of the remaining sub-sequences by a single customer. After adjusting the distance table we would then have a smaller asymmetric VRP.

We now examine in detail the version of the SMART operator where sequences of consecutive arcs are removed. In order to make the move useful, we need to insure that some exchange will be possible, which means that the freed sequences should be close to each other either geographically, temporally or both. The removing policy we use is the following. First identify randomly a *primary pivot*, which will define the neighborhood. Then, remove a certain number of customers before and after that *pivot* thus creating a hole in its tour. The next step is to identify the one customer in each other tour that can be visited in that hole while generating a minimal detour: such customers are named *secondary pivots*. Apply the same treatment to the *secondary pivots*, removing a number of preceding and following customers. Figure 3 illustrates this procedure.

As for the second variant of the SMART procedure, arcs are freed randomly with a good bias toward the longer arcs. This procedure insures that intact sequences (those remaining after all chosen arcs have been removed) are short and thus movable; it also helps to accelerate the descent process of reducing the total distance traveled as it replaces longer arcs by shorter ones.

If we choose the size of a SMART problem correctly, we can manage to solve it exactly. To do so we use a modified version the TSPTW model developed by Pesant et al. (1998). For upper bound, we use the previous total distance, before the operator was invoked, and for lower bounds three different methods. A nearest neighbor bound and minimum spanning tree are calculated at each node, adding tightness to a more complex regret based nearest neighbor described by Caseau and Laburthe (1997), which is calculated only once per

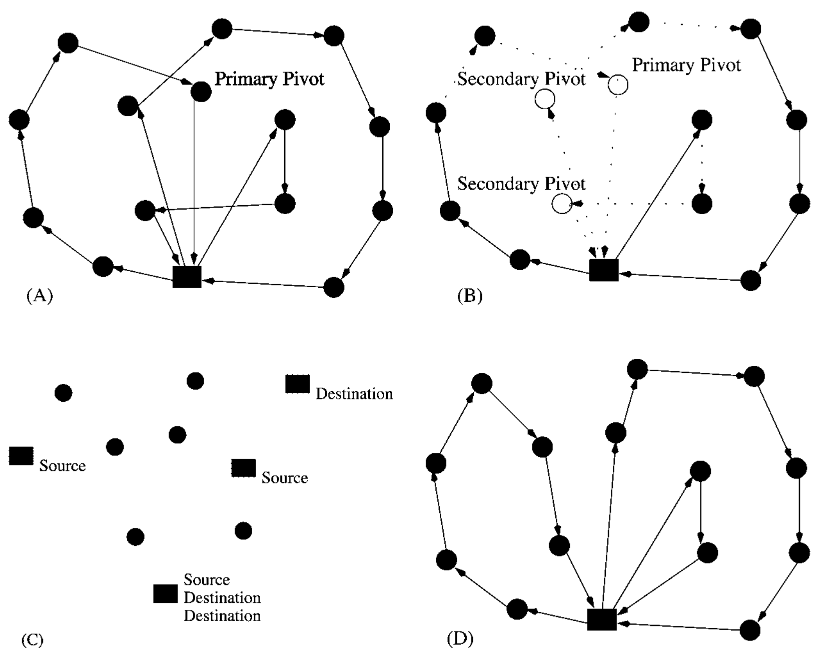

Figure 3. SMART (A) original solution (B) *Pivots* and arcs to remove (C) New SMART problem (D) new solution.

problem. In order to increase the neighborhood size we consider, we choose to explore it only partially using limited discrepancies search (LDS) with a bounded number of discrepancies. In combination with a good value selection algorithm, we can explore much more rapidly a large search space without a significant decrease in solution quality.

It is important to understand that the two variants presented above are only representations of the same problem. We could generalize the procedure by looking at its implementation: to each arc in a solution corresponds a certain variable (in a CP sense). What we do is simply fix the variable associated with the remaining arcs and solve the problem over the variable associated with the ones that are freed. This method allows the treatment of complex constraints since the intact sequences are not really replaced by single customers or depots. It also permits the use of several different arc-removing strategies since the solving process is independent from the arcs that have been removed. Therefore we could use one of the two variants presented, designed a new one or combine multiple strategies during the search. For simplicity we will restrict ourselves to the two variants presented.

3. Method

We want to use the operators presented above to solve the vehicle routing problem with time window. We thus need to find a way to combine them in order to take advantage of their diverse properties and neighborhood searching capabilities. We propose a construction algorithm, a minimum escaping strategy a and diversification procedure all using the described operators. We also propose to use post-optimization on the best-found solution to furthermore increase its quality.

3.1. Construction method

The main problem encountered while solving routing problems is the reduction of the number of vehicles needed to visit all customers. This problem arises from the fact that it is hard to represent this objective in a cost function since two solutions using the same number of vehicles cannot be easily differentiated. Furthermore, this objective is usually in conflict with the attempt to minimize the travel distance. Therefore, apart for the case where an operator gives a solution using fewer vehicles, it is quite hard to guide the search toward vehicle reduction.

The idea of this method is to concentrate on constructing an initial solution that has a minimal number of vehicles instead of a rather small total travel distance. To do so we use the naive ejection chains (NEC) presented in the previous section. This operator tries to find a feasible solution where one vehicle visits one less customer. Therefore, if applied repeatedly to the same vehicle, it has a good chance of removing it completely. Our construction method thus can be defined as follows. First start by constructing a solution where each vehicle visits only one customer (number of vehicles = number of customers). Then, successively apply the NEC operator to all vehicles, by increasing order of customers visited, in order to maximize the chance of removing the route. The number of times the NEC operator is applied is a parameter of the search construction method, but basically very good solutions are obtained after about 5 executions on all routes.

3.2. VND framework

The SMART and LNS-GENI operators presented above are used in a local descent strategy, as they never allow the objective function to increase; it is known that such strategies get trapped in local minimum when no better solution can be found in a neighborhood. To circumvent such a problem, let us consider the variable neighborhood descent (VND) scheme introduced by Mladenovic and Hansen (1997). The VND strategy is to use one operator until we are sufficiently sure that we are trapped in a local minimum, then to start applying the second one until itself can no longer improve the solution. A VND will oscillate in this way between two or more operators, hoping that changes in neighborhood structure will permit an escape from most local minimums. For this principle to work well, we think that the operators used should be very different with respect to the neighborhood structure they generate. Fortunately, it is the case with the operators we proposed, one being centered on customer insertion and the other on arc exchange. We do not include the NEC operator in the VND process as it possesses no mechanism to force local descent.

3.3. Improvement and diversification phases

Even though the VND scheme permits the search to escape local minimums, some solutions are local minimums for all the VND operators. Such solutions entrap the search process and no escape is possible. When this situation occurs we propose to use a diversification strategy to reposition the search in another area of the solution space. To diversify, we propose to use the NEC operator again, but in a more limited way. When an ejection chain is completed, a small number of customers usually change vehicle. This is a desirable effect because it permits to “shuffle” a certain number of customers around. However because we don’t want to loose too much of the good solution at hand, we limit the number of successful ejection chains completed, for instance we accept only the first 5 or 10 completed chains.

The search process thus becomes a two-phase process. The first phase is an Improvement procedure that uses the VND scheme in a descent strategy until it reaches a local minimum (for all the VND operators). The second phase, being the diversification procedure described above, permits the Improvement phase to become effective again, as the solution it produces is no longer local minimums. The global search process alternates between those two phases a predefined number of times, but always keeping the best solution found after the Improvement phase.

3.4. Post-optimization

During the search process we consider multiple exchanges that involve customers belonging to different routes. Partitioning being a key factor in a Vehicle Routing Problem, it needs to be addressed constantly. However at the end of the search process, we can assumed that nothing more can be gained by looking at inter-route exchanges and concentrate on intra-route moves, which are much more simple. In fact, solving a VRP route by route is like solving a series of independent travelling salesman problems (TSP). By regarding each route as TSP and trying to improve them, two cases arise.

The first is when the considered route or TSP as a rather small number of customers (typically less than 20) in which case we can attempt to solve it exactly. To do so we use the original version of TSPTW model developed by Pesant et al. (1998). As the execution time distribution tends to have a very long tail, we allow a maximum time limit after which we consider that the TSP belongs to the case.

The second case includes all routes that contain too many customers to be solved exactly. To try to improve those routes we use the US part of the GENIUS-CP (Pesant, Gendreau and Rousseau, 1997), which tries to remove a customer and reinsert it in better place. This method is very fast and allows some final local optimization to be performed.

4. Results

We use the 56 Solomon problems as benchmarks for our method, firstly because they provide a good variety of instances in terms of constraint tightness, and secondly because they are probably the most popular benchmark for algorithms solving the VRPTW, making it easier to compare with other approaches. All tests were performed on a Sun Ultra10 workstation using Ilog Solver 3.2 as our constraint solver.

Table 1. Averages by class of best initial solutions obtained by compared methods.

	Construction method			
	Solomon's Insertion I1 (1987)		NEC operator	
Class	Distance	Vehicles	Distance	Vehicles
R1	1393.92	13.42	1419.85	12.17
C1	909.81	10.00	1525.72	10.00
RC1	1561.98	13.50	1575.35	11.75
R2	1280.15	3.18	1290.54	3.09
C2	696.57	3.13	883.75	3.00
RC2	1567.67	3.75	1578.46	.0375

4.1. Construction

We easily note, in Table 1, that the proposed algorithm is very good in generating a solution with a small number of needed vehicles. When we compare it with Solomon's insertion algorithm, we see that the NEC operator always finds a solution using fewer vehicles. However Solomon's method runs in a fraction of a second instead of minutes for our strategy, but running the Solomon method longer could not produce significantly better results.

Since the NEC operator is not designed to reduce the total travel distance, one could think that the solutions produced would have been of very poor quality. But in fact experimentation showed that this method produces solution comparable to those produced by other specialized algorithms. Except for the C1 and C2 classes, which are easy enough for the Solomon method to almost solve. We think that this is due to the ejection candidate selection algorithm that always tries to eject the customer generating the maximum detour and thus tends to reduce the total travel distance.

4.2. Operators

We compare the SMART and LNS-GENI algorithms to determine their relative strengths and weaknesses. Looking at Table 2 we see that SMART outperforms LNS-GENI in average on all the problem classes. The SMART operator produces very good results considering the fact that it is used in simple descent; but what about LNS-GENI?

We can attribute the worse performance of the LNS-GENI operator to the fact that using GENeralized Insertions actually limits the size of the neighborhood it can search. Because GENIs are much more complicated to evaluate than simple insertions, we can afford to consider only the best insertion point for each removed customer, as opposed to the original LNS (Shaw, 1998) which could try to insert a customer in its second or third best insertion point. LNS-GENI is however very useful in combination with the SMART operator as it allows it to reposition in a significantly different area of the solution space, without a decrease in solution quality.

Table 2. Averages by class of average solutions obtained by compared operators.

	Operators and combination					
	LNS-GENI		SMART		VND	
Class	Distance	Vehicles	Distance	Vehicles	Distance	Vehicles
R1	1276.39	12.17	1245.41	12.17	1225.19	12.08
C1	977.60	10.00	860.11	10.00	834.30	10.00
RC1	1438.79	11.75	1426.80	11.63	1401.76	11.63
R2	999.43	3.09	971.17	3.09	954.07	3.00
C2	697.88	3.00	599.78	3.00	591.06	3.00
RC2	1232.18	3.38	1156.42	3.38	1124.46	3.38

Table 3. Averages by class of best solutions obtained by compared methods.

	Comparative table									
	Tabu (Taillard et al., 1997)		GLS (Kilby, Prosser and Shaw, 1999)		LNS (Shaw, 1998)		MH Factory (Caseau Laburthe and Silverstein, personal communication)		C^3PO	
	Dist	Vehi	Dist	Vehi	Dist	Vehi	Dist	Vehi	Dist	Vehi
R1	1209.35	12.17	1200.83	12.67	1209.80	12.00	1207.26	12.16	1210.21	12.08
C1	828.38	10.00	830.75	10.00	828.94	10.00	–	–	828.38	10.00
RC1	1389.22	11.50	1388.15	12.12	1366.40	11.75	1367.38	12	1382.78	11.63
R2	980.27	2.82	966.56	3.00	–	–	–	–	941.08	3.00
C2	589.86	3.00	592.24	3.00	–	–	–	–	589.86	3.00
RC2	11017.44	3.38	1133.42	3.38	–	–	–	–	1105.22	3.38

Moreover, it his hoped that, by combining these operators in VND, we can benefit from the strength of both operators while avoiding their weaknesses. It is also hoped that the local minimum escaping property of VND will permit the search to explore a better region of the solution space and thus produce better results. Again looking at Table 2, we see that this goal is achieved. Furthermore, the figures reported being averages of 20 executions, we note that the VND scheme is very robust, as it produces results very close to those reported in the literature (see Table 3).

Since the GENI operator is well known in the operations research community and many implementations of it exists, it could be interesting to compare the Constraint Programming implementation to a more straightforward version of the algorithm and measure the impact on our application. A very similar experimentation has been performed in Pesant, Gendreau and Rousseau (1997) and we report here some results. When tested on individual vehicle routes taken from good solutions of Solomon's problems, the CP version of the algorithm was on average four times slower than the specialized heuristic. Using Table 4 we can

Table 4. Approximate execution time for each component

Approximate execution times	
Component	CPU time
The Construction Phase (NEC)	200 seconds
A Local Descent (SMART or LNS-GENI)	500 seconds
Improvement Phase (DVV)	2000 seconds
Diversification Phase (NEC)	50 seconds
Post-Optimization (TSPTW or GENIUS)	100 seconds
Total Execution of C^3PO^1	11000 seconds

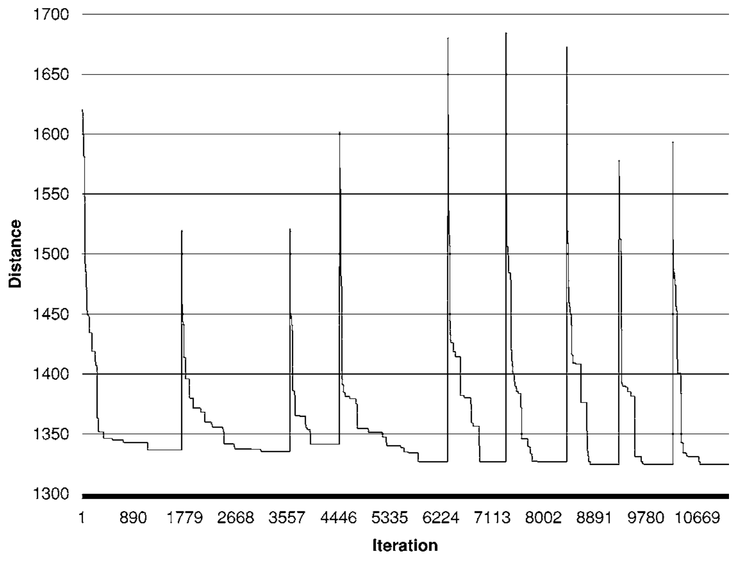

Figure 4. An example of the search process for a particular instance.

thus calculate that the impact on the total run time of the entire algorithm would be a reduction in the order of 35%. However we would lose the flexibility provided by the constraint programming model. Each constraint of the problem would have to be separately implemented in the specialized GENI heuristic and we could even lose the ability to model and solve problems with complex and intricate constraints. This argument also stands in the case of the NEC and SMART operators, since the three operators share the same model to describe the problem.

4.3. *Improvement and Diversification*

If we look at the search process presented in figure 4, we note that, even though the diversification phases produce solutions which present distance values comparable to those of the construction algorithm, the solution produced after the improvement phase seems to decrease. This reflects the fact that the total distance traveled does not give a proper evaluation of an overall solution quality. The solution produced by the diversification procedure only differs by a few customer exchanges from the previous local minimum solution. The increase in distance is therefore only a result of those few exchanges while the rest of the solution structure is still of good quality. This explains the fact that the Improvement phase seems to be more effective towards the end of the search process, when solutions have a good structure, than at the beginning, when the solutions exhibit more chaotic structure elements.

4.4. *Post-optimization*

On the 56 problems we have tested, post-optimization allows the improvement of 15 solutions at the end of the search process. However the relative improvement is usually well under 1%.

4.5. *Comparison*

To better evaluate the overall performance of our method, we compare it to other recent methods solving the same problem. We choose to only compare to methods that consider the different distances to be real numbers. The figures presented for our method are the averages by class of the best solutions found for each problem over 10 executions. In the following

Table 5. New best solutions.

New best solutions					
	Previous best			New best	
Problem	Distance	Vehicles	Reference	Distance	Vehicles
R110	1135.07	10	Shaw, 1998	1124.40	10
R111	1096.73	10	Shaw, 1998	1096.72	10
RC105	1643.38	13	Taillard et al., 1997	1633.72	13
R201	1254.09	4	Kilby, Prosser and Shaw, 1999	1252.37	4
R202	1214.28	3	Taillard et al., 1997	1191.70	3
R205	998.96	3	Kilby, Prosser and Shaw, 1999	994.42	3
R206	932.47	3	Taillard et al., 1997	929.03	3
R209	923.96	3	Kilby, Prosser and Shaw, 1999	909.86	3
RC202	1162.80	4	Kilby, Prosser and Shaw, 1999	1161.28	4
RC203	1068.07	3	Kilby, Prosser and Shaw, 1999	1066.99	3
RC204	803.90	3	Kilby, Prosser and Shaw, 1999	801.40	3
RC206	1156.26	3	Kilby, Prosser and Shaw, 1999	1153.93	3
RC208	833.97	3	Rochat and Taillard, 1995	829.69	3

tables we refer to our combination of cooperating constraint-programming operators with the acronym C^3PO.

The solutions produced by our hybrid method seem to be competitive with those published in the literature: we note that both the number of vehicle used and the total traveled distance is around 0.5% of the other methods if not better. We also report new best solutions on 12 problems which are reported in Table 5 and detailed in annex.

However a note must be taken that C^3PO is somewhat slower than those with which it is compared to by a factor of at least three. More precisely, Table 5 shows an approximate execution time for each component. The aim of this project was not develop a very fast heuristic but rather to demonstrate that constraint programming could produce state-of-the-art results while providing increased flexibility.

5. Conclusion

We have presented a method using constraint programming as a neighborhood-searching algorithm for three operators. These operators, combined in variable neighborhood descent and a two phase search process, produce good results on all Solomon's benchmark problems while also proving extra modeling capability. This method is presently able to solve problems with multiple time windows and precedence constraints. Hybrids that are using CP to validate solutions or optimize individual tours would suffer from the introduction of complex and very tight constraints because solutions acceptable by the CP solver would become more rare. As for our method, it would gain in performance, propagation being used more extensively to prune the search space.

Constraint programming has also benefited us from the software developing point of view. Since we have developed all operators on top of a single constraint-programming model of our problem (the VRPTW), any future modification to this problem could simply be achieved by a modification of this model.

The next step consists of exploring incomplete search methods in order to limit the time needed to find improving solutions. Efforts should be made to speed up the search process in both the SMART and the LNS-GENI operators, which would benefit the overall search process.

Acknowledgments

Financial support for this research was provided by the Natural Science and Engineering Research Council of Canada and the "Fonds pour la Formation des Chercheurs et l'Aide à la Recherche (FCAR)" in transportation. We would like to thank the referees for their comments and suggestions.

Note

1. Based on a main loop of 5 calls to the Improvement and Diversification phases.

References

Baptiste, P. C., Le Pape, and W. Nuijten. (1995). “Incorporating Efficient Operations Research Algorithms in Contraint-Based Scheduling.” In *Proceedings of the First International Joint Workshop on Artificial Intelligence and Operations Research*. Timberline Lodge, Oregon.

Caseau, Y. and F. Laburthe. (to appear). “Heuristics for Large Constrained Vehicle Routing Problems.” *Journal of Heuristics*.

Caseau, Y. and F. Laburthe. (1997). “Solving Small TSPs with Constraints.” In *Proceedings of the 14th International Conference on Logic Programming*. Cambridge MA: MIT Press, pp. 316–330.

Caseau, Y. and F. Laburthe. (1998). “SALSA: A Language for Search Algorithms.” In *Principle and Practice of Constraint Programming—CP98*. Pisa, Italy, Oct. 1998.

Caseau, Y., F. Laburthe, and G. Silverstein. “A Meta-Heuristic Factory for Vehicle Routing Problems.” Personal Communication.

De Backer, B. and V. Furnon. (1997). “Meta-heuristics in Constraint Programming Experiments with Tabu Search on the Vehicle Routing Problem.” In *Proceedings of the Second International Conference on Metaheuristics (MIC’97)*. Sophia Antipolis, France, July 1997.

Gendreau, M., F. Guertin, J.-Y. Potvin, and R. Seguin. (1998). “Neighborhood Search Heuristics for a Dynamic Vehicle Dispatching Problem with Pick-Ups and Deliveries.” Publication CRT-98-10, Centre de recherche sur les transports, Université de Montréal, Montréal.

Gendreau, M., A. Hertz, and G. Laporte. (1992). “New Insertion and Postoptimization Procedures for the Traveling Salesman Problem.” *Operations Research* 40, 1086–1094.

Gendreau, M., A. Hertz, G. Laporte, and M. Stan. (1995). “A Generalized Insertion Heuristic for the Traveling Salesman Problem with Time Windows.” *Operations Research* 43, 330–335.

Glover, F. (1996). “Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems.” *Discrete Applied Mathmatics* 65, 223–253.

Harvey, W. and M. Ginsberg. (1995). “Limited Discrepancy Search.” In *Proc. of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*. Montréal, Canada, San Mateo, CA: Morgan Kaufmann, pp. 607–615.

Kilby, P., P. Prosser, and P. Shaw. (1999). “Guided Local Search for the Vehicle Routing Problem with Time Windows.” In *META-HEURISTICS Advances and Trends in Local Search Paradigms for Optimization*. Dordrecht: Kluwer Academic Publishers, pp. 473–486.

Meseguer, P. and T. Walsh. (1998). “Interleave and Discrepancy Based Search.” In *ECAI 98*.

Mladenovic, N. and P. Hansen. (1997). “Variable Neighbourhood Search.” *Computer and Operations Research* 24, 1097–1100.

Pesant, G. and M. Gendreau. (1996). “A view of Local Search in Constraint Programming.” *In Principles and Practice of Constraint Programming—CP96: Proceedings of the Second International Conference*. Berlin: Springer-Verlag, pp. 353–366. LNCS, 1118.

Pesant, G. and M. Gendreau. (1999). “A Constraint Programming Framework for Local Search Methods.” *Journal of Heuristics* 5, 255–279.

Pesant, G., M. Gendreau, J.-Y. Potvin, and J.-M. Rousseau. (1998). “An Exact Constraint Logic Programming Algorithm for the Traveling Salesman Problem with Time Windows.” *Transportation Science* 32, 12–29.

Pesant, G., M. Gendreau, and J.-M. Rousseau. (1997). “GENIUS-CP: A Generic Vehicle Routing Algorithm.” In *Principles and Practice of Constraint Programming—CP97: Proceedings of the Third International Conference*. Berlin: Springer-Verlag, pp. 420–434. LNCS 1330.

Rego, C. and C. Roucairol. (1996). “A Parallel Tabu Search Algorithm Using Ejection Chain for the Vehicle Routing Problem.” *Metaheuristics: Theory and Applications*, pp. 661–675.

Rochat, Y. and É. Thaillard. (1995). “Probabilistic Diversification and Intensification in Local Search for Vehicle Routing.” *Journal of Heuristics* 1, 147–167.

Shaw, P. (1998). “Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems.” In *Principle and Practice of Constraint Programming—CP98*. Pisa, Italy, Oct. 1998. Berlin: Springer-Verlag. LNCS.

Solomon, M.M. (1987). “Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints.” *Operations Research* 35, 254–265.

Taillard, É., P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. (1997). “A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows.” *Transportation Science* 31, 170–186.

Thompson, P. and H.N. Psaraftis. (1993). “Cyclic Transfer Algorithms for Multi-Vehicle Routing and Scheduling Problems.” *Operations Research* 41, 935–946.

Van Hentenryck, P. (1999). *The OPL Optimization Programming Language*. Cambridge, MA: MIT Press.