A Hybrid Approach to Vehicle Routing using Neural Networks and Genetic Algorithms

Jean-Yves Potvin
Danny Dubé
Christian Robillard

Centre de recherche sur les transports
Université de Montréal
C.P. 6128, Succ. Centre-Ville,
Montréal (Québec),
Canada H3C 3J7

Abstract. A competitive neural network model and a genetic algorithm are used to improve the initialization and construction phase of a parallel insertion heuristic for the vehicle routing problem with time windows. The neural network identifies seed customers that are suitably distributed over the entire geographic area during the initialization phase, and the genetic algorithm finds good parameter settings in the route construction phase that follows. Computational results from a standard set of problems are also reported.

Keywords. Vehicle Routing, Time Windows, Neural Networks, Genetic Algorithms.

Section 1. Introduction

This paper discusses a competitive neural network model and genetic algorithm aimed at improving a route construction heuristic for the Vehicle Routing Problem with Time Windows (VRPTW). The VRPTW is currently the focus of very intensive research, and is used to model many realistic applications such as retail distribution, school bus routing, and mail delivery [1].

The overall objective is to service a set of customers at minimum cost with a fleet of vehicles of finite capacity operating out of a central depot (i.e., each route starts and ends at the depot). In this application, the objective is to minimize the number of routes and, for the same number of routes, to minimize the total route time. To be feasible, the routes must also satisfy three different types of constraints. First, each customer has a certain demand, like a quantity of goods to be delivered. Since the capacity of each vehicle is finite, the total quantity of goods to be delivered on a route cannot exceed the capacity of the vehicle assigned to this route. Second, there is a time window or time interval associated with each customer. No vehicle is allowed to arrive too late at a customer location (i.e., after the end of the time window). However, a vehicle can wait if it arrives too early. Finally, a time window is also included at the depot or garage. Each route must start and end within its bounds. This window acts similarly as a capacity constraint: by expanding either the vehicle capacity or the time window at the depot, more customers can be serviced by the same vehicle.

Figure 1 depicts a typical solution to this problem. In this figure, the central square represents the depot and the circles are customers to be serviced. There are eight customers, each identified by a letter from *a* to *h*. In this example, the quantity of goods to be delivered to each customer is set at 10 units, and the capacity of each vehicle is set at 30 units. Since the total quantity of goods to be delivered is equal to $8 \times 10 = 80$ units, it is clear that at least three vehicles must be used. The time window constraints, however, must also be satisfied. In Figure 1, the time window at each customer location and at the depot is a time interval $[t_i,t_j]$, where t_i and t_j are the lower and upper bounds, respectively. The travel time between two customers is also shown on each link.

This solution is clearly feasible. First, there are at most three customers on each route. Consequently, the capacity constraints are satisfied. Second, the time constraints are satisfied at each customer location and at the depot. For example, one vehicle leaves the depot at time 0 and arrives at customer *a* at time 3. This arrival time falls within the time window of customer *a*. Similarly, the arrival time of the vehicle at customers *b* and *c* is equal to 5 and 6, respectively. In both cases, the arrival time falls within the time window. Finally, the vehicle returns to the depot at time 7, before the upper bound, which is 8. The routes for the two other vehicles are feasible as well.

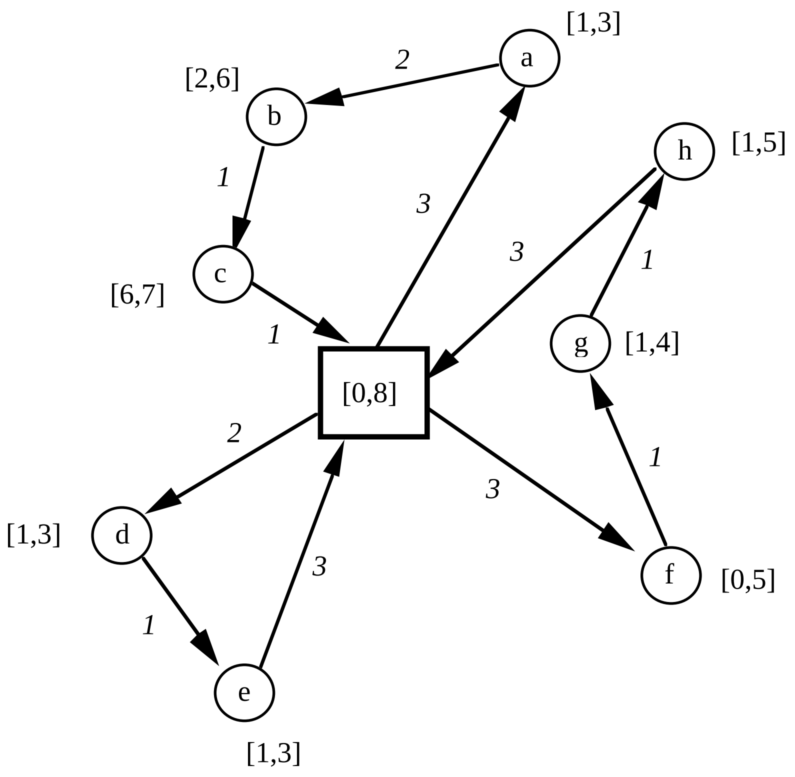

Figure 1. The vehicle routing problem with time windows

Many different heuristics for solving the VRPTW are described in the literature. Most of them are route improvement heuristics, or composite heuristics with a mix of route construction and route improvement procedures [2-5]. Only two "pure" route construction heuristics are described in the literature [6,7]. They are pure in the sense that they involve neither backtracking mechanisms to undo previous commitments nor reordering procedures to improve the final routes. Customers are simply inserted one by one into the routes, until all customers are serviced. These route construction heuristics are useful because they produce good feasible solutions with little computation. These solutions can then be used as starting points for more complex route improvement or composite heuristics.

In this paper, a competitive neural network model and a genetic algorithm are used to improve the parallel route construction heuristic described by Potvin and Rousseau [7]. Section 2 first introduces Solomon's classical insertion heuristic for the VRPTW [6]. Then, the parallel route construction heuristic of Potvin and Rousseau, which is derived from Solomon's work, is presented in Section 3. Sections 4 and 5 describe the neural network model and the genetic algorithm. Finally, computational results using Solomon's set of problems are reported.

Section 2. Solomon's I1 insertion heuristic [6]

Solomon's I1 insertion heuristic is a classical heuristic that constructs routes one by one. A seed customer is first selected to create an initial route servicing only this customer (i.e., the vehicle leaves the depot, services the customer, and comes back to the depot). The remaining customers are then inserted one by one into the route until no more customer with feasible insertions can be found. At this point, a seed customer is selected to create a second route, and this route is in turn filled using the remaining unrouted customers. The procedure is repeated until all customers are routed.

At each step, the next customer to be inserted, and the place for inserting this customer on the route, must be selected. This is done as follows:

Step 1. Compute the minimum feasible insertion cost $c_1{*}(u)$ for each unrouted customer u, where the insertion cost c_1 of customer u between two consecutive customers i and j on the route is computed as follows:

$$c_1(i,u,j) = \alpha_1 \times c_{11}(i,u,j) + \alpha_2 \times c_{12}(i,u,j),\quad \alpha_1+\alpha_2=1,\ \alpha_1\geq 0,\ \alpha_2\geq 0,$$

where

$c_{11}(i,u,j) = d_{iu} + d_{uj} - \mu \times d_{ij}$, d_{ij} = distance (in time units) between i and j.

$c_{12}(i,u,j) = b_{uj} - b_j$, b_j = current service time at j.
b_{uj} = new service time at j, given that u is inserted between i and j.

Since c_{11} is the detour (in time units) and c_{12} is the service delay at customer j due to the insertion of customer u, the insertion cost c_1 is a weighted sum of detour and delay. The objective is to determine a feasible insertion point such that this cost is minimized (see Figure 2). Note that the parameters α_1 and α_2 are used to weight the two components of the sum. An additional parameter μ modifies the detour formula, by placing more or less emphasis on the distance between customers i and j.

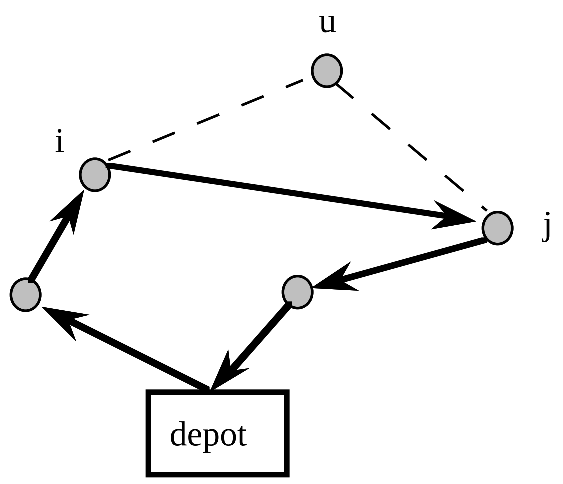

Figure 2. An insertion place for customer u

Step 2. Select the next unrouted customer among the customers that can be feasibly inserted into the route. The selected customer maximizes the generalized savings measure c_2, where:

$$c_2(u) = \lambda \times d_{depot,u} - c_1^*(u), \quad \lambda \geq 0.$$

The classical savings [8] are obtained from c_2 by setting $\alpha_1=1$, $\alpha_2=0$, $\mu=1$, and $\lambda=2$. Assuming that each unrouted customer u is serviced by an individual route (depot,u,depot), the classical saving is the distance saved by inserting u at its best feasible insertion place in the current route, and by eliminating its individual route. In this case, the saving is twice the distance between u and the depot, minus the detour caused by inserting u into the current route.

Step 3. Insert the selected customer at its best feasible insertion point (as computed in Step 1).

In [6], solutions were obtained by running this route construction heuristic eight times for each problem, using different parameter settings. The best solution was then selected.

Section 3. The parallel insertion heuristic [7]

Our parallel insertion heuristic is largely inspired by the work of Solomon. The main difference is that routes are built in parallel rather than one by one. Accordingly, many different seed customers must be selected to create the initial set of routes (with each route servicing a single customer). In Figure 3, for example, customers 1, 2 and 3 are selected to create three initial routes. Then, the remaining unrouted customers are inserted one by one into these routes until all customers are routed. Starting from the three initial routes of Figure 3, Figures 4a, 4b, 4c and 4d show how the solution develops when customers are inserted into these routes. Note that the next customer to be routed optimizes a selection cost measure (to be defined later), and is identified with an arrow. Figure 4 depicts the three first iterations of the route construction phase.

Obviously, it is not possible to know a priori the minimum number of routes required to service all customers. In Figure 3, for example, four routes may be required to service all customers. Similarly, a feasible solution may possibly be found with only two routes. Such a difficulty does not arise with Solomon's I1 heuristic, because routes are created and filled one by one, as needed, until all customers are routed.

To get an estimate of the initial number of routes, Solomon's I1 heuristic is applied to each problem, using a single parameter setting. Hence, only one run is performed on each problem (rather than eight different runs). The number of routes in the solution is then used to initiate the parallel insertion heuristic.

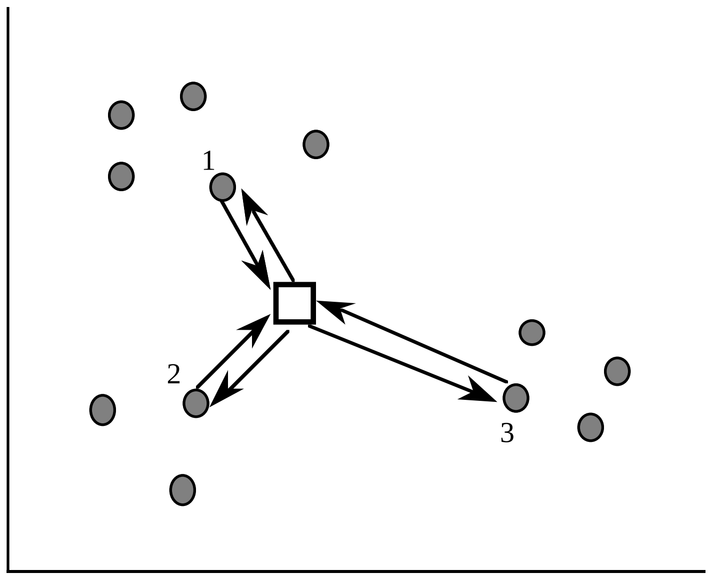

Figure 3. Route initialization phase

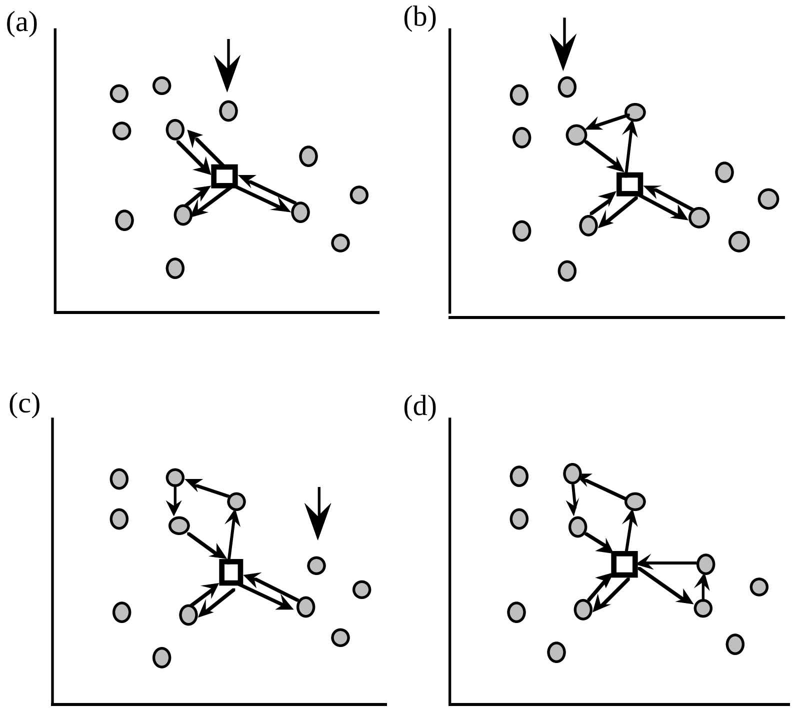

Figure 4. Route construction phase

The parallel insertion heuristic can now be introduced more formally.

Step 1. Estimate the initial number of routes *m* by running Solomon's I1 heuristic once.

Step 2. Create *m* routes by selecting *m* seed customers (one customer for each route). In [7], the farthest customer from the depot in each route produced by Solomon's I1 heuristic in Step 1 was selected as a seed customer. In this paper, a more sophisticated selection scheme based on a neural network model is proposed (see Section 4).

Step 3. Compute the minimum feasible insertion costs $c_1*(u,r)$ for each unrouted customer u in each route r. This is done in the same way as in Solomon's I1 heuristic, using the parameters α_1, α_2 and μ (see Section 2).

Step 4. Select the unrouted customer that maximizes the generalized regret measure c_2, where:

$$c_2(u) = \Sigma_{r=1,...,m,\ r\neq r'} (c_1^*(u,r) - c_1^*(u,r')) ,$$

where $c_1^*(u,r')$ corresponds to the best feasible insertion point of customer u over all routes.

Step 5. Insert the selected customer at its best feasible insertion point in route r'.

Step 6. Repeat Steps 3 to 5 until all customers are routed (feasible solution) or until one or more unrouted customers have no feasible insertion point (no feasible solution).

Once the best feasible insertion point for each unrouted customer u in each route is known, the next customer to be inserted is the one that maximizes c_2, the generalized regret measure. Basically, this measure is a kind of look-ahead that indicates what can be lost later, if a given customer is not immediately inserted within its best route. Here, a large regret measure means that there is a large gap between the best insertion point for a given customer and the best insertion points in the other routes. Hence, unrouted customers with large regrets must be considered first, since the number of good alternative routes for inserting them is small. On the other hand, customers with a small regret can easily be inserted into alternative routes without losing much, and are considered later for insertion. Broadly speaking, this new generalized measure improves upon the classical regret measure by extending the look-ahead to all available routes, whereas the classical regret is rather short sighted, and merely looks at the best and second best routes [9,10].

In [7], this parallel insertion heuristic was applied to each problem with three different parameter settings, namely $(\alpha_1, \alpha_2, \mu) = (0.5,0.5,1.0)$, $(0.75,0.25,1.0)$, $(1.0,0.0,1.0)$, and the best solution was selected at the end. These parameter settings were determined empirically by experimenting with different parameter settings on Solomon's test problems.

Typically, the parallel insertion heuristic is applied more than three times to each problem, because feasible solutions with fewer routes than the initial estimate of Solomon are also considered. Basically, the procedure terminates as soon as a feasible solution is produced for a given number of routes m. The entire procedure is then repeated for $m-1$ routes. The number of routes is reduced in this way until no feasible solution is found with the three parameter settings.

The neural network model for selecting the seed customers in Step 2 is introduced in the next section. A genetic algorithm aimed at finding better parameter settings for α_1, α_2 and μ is then described in Section 5.

Section 4. The neural network initialization

Competitive neural networks are now widely used to cluster data. A large body of neural network training algorithms are described in the literature [11-15] and can be adapted to our initialization problem. In particular, these models can be applied to identify geographical clusters of customers. This is a nice feature, since customers that are members of the same cluster naturally fit into the same route. Therefore, it would be desirable to select a seed customer in each cluster to create a different route for each. If no specific pattern in the geographical distribution of the customers is found, then good practice would be to select the seed customers widely apart so as to cover the whole geographic area, and create routes to serve different regions. The competitive neural network model exhibits such behavior.

Figure 5 is an example of a competitive network with two input units I_1 and I_2, encoding the (x,y) coordinates of each customer location, and three output units O_1, O_2, O_3 associated to three different clusters (or routes). Connections between the two input units and any given output unit j is weighted by the weight vector of cluster j. For example,

$w_1=(w_{11}, w_{21})$ is the weight vector associated with output unit O_1 and cluster 1.

The activation of each output unit is related to the distance between its weight vector and the current input vector. In particular, the activation level is higher when the distance is smaller. If output unit O_j is the most highly activated unit when the coordinates of customer i are presented to the network (i.e., its weight vector is the closest to customer i), then this unit "wins" the competition over the other output units, and customer i is associated with cluster j. Of course, cluster membership is a function of the weights on the connections, and the network must adjust these weights through a "learning algorithm" so as to minimize some objective function. This function is typically based on the sum of the distances between the weight vector of each output unit and the coordinate vectors of their associated customers.

The initial weight vectors are arbitrarily chosen, but as the learning algorithm proceeds, these weights are adjusted so as to move towards the clusters of customers (if such clusters exist). In Figure 6, for example, the circles are customers to be serviced and the three w's are weight vectors (assuming three output units). The figure illustrates how the three weight vectors evolve in the plane, so as to place themselves approximately at the center of the three clusters of customers. Figure 6d depicts the three clusters identified by the neural network. In particular, the weight vector w_i of output unit i is the closest weight vector to each customer within its box. Note also that the three black circles in Figure 6d are the closest customers to the final weight vectors, and can be used as seed customers in Step 2 of the parallel insertion heuristic.

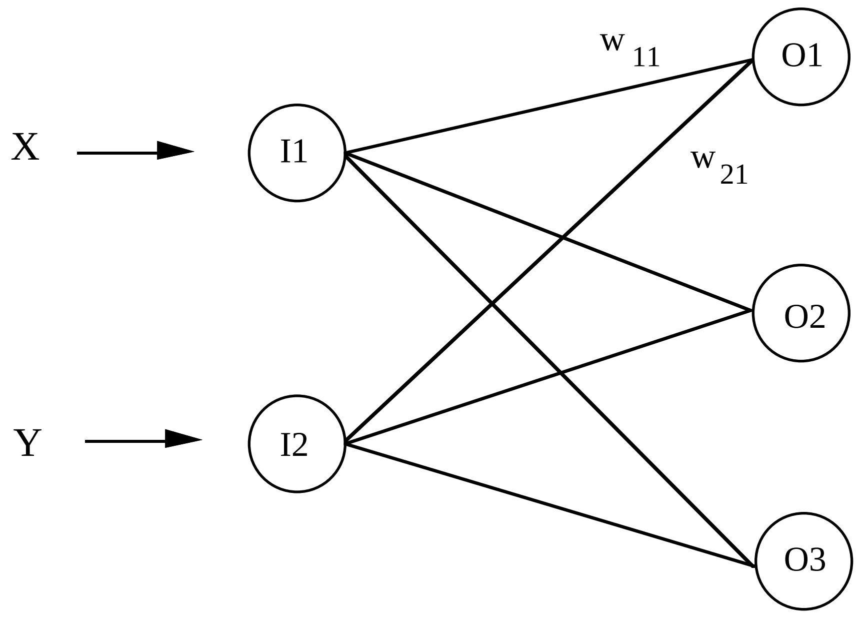

Figure 5. A competitive neural network

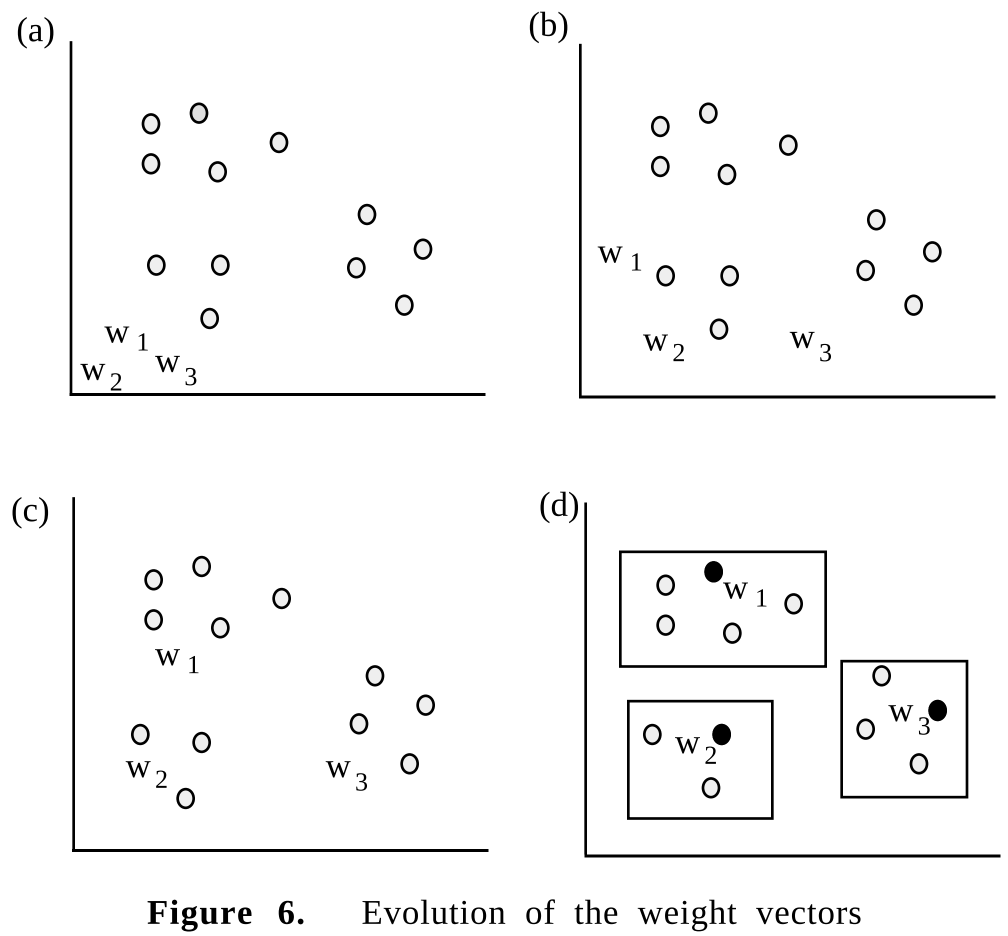

Figure 6. Evolution of the weight vectors

The training algorithm for the competitive neural network can be summarized as follows. We consider two input units and *m* output units. Each output unit j has an initial weight vector $w_j(0) = (w_{1j}(0), w_{2j}(0))$, where $w_{ij}(0)$ is the weight on the connection between input unit i and output unit j. Starting with these initial weight vectors, the training algorithm goes through the set of coordinates many times (referred to as "passes" through the data), and slightly modifies the weights on the connections after the presentation of each input.

Weight vectors are updated as follows for a given input vector I = (x,y). Each output unit j first computes the Euclidean distance between its weight vector w_j and the input vector. The output unit j* closest to the input vector is selected as the winner. Hence, output unit j* claims that the current customer is member of cluster j*. Its weight vector w_{j*} is then adjusted to move even closer to the current customer, while the other weight vectors stay at their current location. The update of the weight vector of unit j* is based on the following formula:

$$w_{j*}(t+1) = w_{j*}(t) + \eta\,(I - w_{j*}(t)) \quad .$$

The parameter η in this formula is the learning rate. Its initial value is smaller than 1 and is progressively reduced as learning proceeds. Hence, the magnitude of the moves is progressively reduced, to allow the weight vectors to stabilize in their respective regions of Euclidean space.

Note that the closest weight vector to a given customer can change from one pass to another through the set of customers, because each unit can win many times during a single pass and the weight vector is moved each time. Cluster membership, however, stabilizes after a few passes through the data.

Customer coordinates are presented to the network until a stopping condition is satisfied. Typically, the following objective function is used to evaluate the neural network performance at a given iteration *t* (for *n* customers and *m* output units):

$$E(t) = 1/n \ \Sigma_{i=1,\ldots,n} \Sigma_{j=1,\ldots,m} M_{ij}\ d(I_i, w_j(t))^2 \ ,$$

where $d(I_i, w_j(t))$ is the Euclidean distance between the input vector I_i and the current weight vector w_j, and M is the (n x m) cluster membership matrix, that is, M_{ij} is set to 1 if customer i is a current member of cluster j and 0 otherwise. Hence, this function computes the average squared Euclidean distance between the weight vectors and the coordinate vectors of their current customers. This value decreases from one pass to another through the set of coordinates, and training stops when the improvement falls below a given threshold.

One problem with the above learning algorithm is its sensitivity to the initial location of the weight vectors. Quite often, some weight vectors remain idle at their initial location, because they never win a competition. To alleviate this problem, the Euclidean distance between the weight vector of output unit j and the input vector I_i is biased according to the formula [16]:

$$d'(I_i,w_j(t)) = d(I_i,w_j(t)) \times u_j(t) \quad ,$$

where $u_j(t)$ is the number of times unit j has won in the past (i.e., before iteration t). Consequently, if unit j was a frequent winner, the distance d' will increase and the likelihood of being a winner in the near future will be reduced. Conversely, a unit is more likely to win in the near future if it did not win very often in the past. This modification allows the network to be fully active.

As mentioned before, the weight vectors tend to settle approximately at the centroids of clusters of customers, when such clusters are present. Otherwise, they distribute themselves in the space of coordinates, to get an equal share of customers. At the end, the seed customers for the parallel insertion heuristic are identified by selecting the closest customer to each weight vector (as depicted in Figure 6d).

The next section will now describe the application of the genetic algorithm during the following insertion phase.

Section 5. Parameter optimization using a genetic algorithm

This section briefly introduces the reader to the basic principles of genetic search and describes the parameter optimization problem.

5.1 Overview of genetic algorithms

Evolutionary algorithms began as randomized search techniques designed to simulate the natural evolution of populations of asexual species [17]. Holland [18] extended this model by allowing the recombination of pieces of information taken from two parents to create new offspring. These algorithms were called "genetic algorithms", and the recombination phase proved to be a fundamental ingredient in the success of this search technique.

To apply a genetic algorithm to a given problem, solutions to the problem must first be encoded as chromosomes (typically, bit strings). Then, an evaluation function that returns the fitness or quality of each chromosome must be defined. Using these two components, a simple genetic algorithm can be summarized as follows.

Step 1 (Initialization). Create an initial random population of chromosomes and evaluate each chromosome. Set the current population to this initial population.

Step 2 (Reproduction). Select two parent chromosomes from the current population. The selection process is stochastic, and a chromosome with high fitness is more likely to be selected.

Step 3 (Recombination). Generate two offspring from the two parent chromosomes by exchanging bits (crossover).

Step 4 (Mutation). Apply a random mutation to each offspring with a small probability.

Step 5. Repeat Steps 2, 3 and 4, until the number of offspring in the new population is the same as the number of chromosomes in the old population. Evaluate each new chromosome. Then, set the current population to the new population.

This procedure is repeated for a fixed number of generations, or until convergence to a population of similar chromosomes is obtained. The best chromosome generated during the search is the final result of the genetic search. Through this process, it is expected that an initial population of randomly generated chromosomes with low fitness values will improve as parents are replaced by better offspring.

In the following, the main components of the genetic search are briefly described. First, the encoding transforms a solution to a problem into a bit string. For example, if the problem is to maximize a numerical function f whose domain is a set of integers, then each chromosome can represent an integer in base-2 notation. In this case, the fitness of each chromosome is equal to f(x), where x is the integer encoded in the chromosome.

The reproduction phase favors the best chromosomes via a bias in the selection process. Namely, a chromosome with high fitness is more likely to be selected as a parent. Once the parents have been selected, the recombination phase creates two new offspring by exchanging bit strings taken from two parent chromosomes. Here, the underlying assumption is that a better chromosome can be produced by combining bit patterns taken from two good chromosomes. An example of a one-point crossover is shown in Figure 7 for two parent bit strings of length 5. Here, the cross point is (randomly) chosen between the second and third bit. As we can see, the end parts of both parents are swapped to create two new offspring.

Generalizations of the one-point crossover are also proposed in the literature, like the two-point crossover illustrated in Figure 8. Here, two cross points are (randomly) chosen between the second and third bit, and the fourth and fifth bit, respectively. Then, the substrings located between the two cross points on both parents are swapped.

1 1 | 1 0 0 (parent 1)
0 **0** | **1** **1** **1** (parent 2)

___

1 1 **1** **1** **1** (offspring 1)
0 **0** 1 0 0 (offspring 2)

Figure 7. One-point crossover on two bit strings

1 1 | 1 0 | 0 (parent 1)
0 **0** | **1** **1** | **1** (parent 2)

___

1 1 **1** **1** 0 (offspring 1)
0 **0** 1 0 **1** (offspring 2)

Figure 8. Two-point crossover on two bit strings

Finally, the mutation phase is designed to maintain a minimum level of diversity in the population via random modifications to the bit values. Specifically, the mutation operator flips bit values from 0 to 1 (or from 1 to 0) within each offspring, with a small probability at each position. A reasonable level of diversity in the population is crucial to the effectiveness of the search. If a population is composed of similar chromosomes, the search cannot make any progress because the offspring look like their parents.

At this point, it is useful to note that a "pure" genetic algorithm does not make any particular assumption about the fitness function. It only exploits the value returned by the fitness function to guide the search (c.f., reproduction phase). Accordingly, pure genetic algorithms are robust problem-solving methods that can be applied to the optimization of complex multimodal functions.

5.2 Parameter optimization

In this application, the genetic search is used to find good values for the parameters α_1, α_2 and μ in Step 3 of the parallel insertion heuristic (see Section 3). The parameters α_1 and α_2 weight the detour and delay introduced in the route by inserting a new customer, while the parameter μ is used within the detour formula. By performing a careful search of the parameter space, it is expected that better parameter settings than those obtained through empirical means in [7] will be found.

In the following, a particular implementation of the genetic search for this parameter optimization problem is described.

Representation

The domain of values to be explored for each parameter is $\alpha_1 \in [0,1]$ and $\mu \in [0,1]$ (note that α_2 is determined through α_1, since $\alpha_1+\alpha_2=1$). Seven bits are used to encode each parameter value. To decode a substring as a numerical value, the integer represented by the substring, namely a value between 0 and 2^7-1 or 127, is mapped to the appropriate real domain. For example, if the bit string is 1010101 for parameter α_1, or 85 in decimal notation, the α_1 value encoded by this string is 85/127 (approximately 0.67). In general, if the integer value of the bit string is x, the real value of α_1 is x/127. The same transformation applies to parameter μ. With 127 values mapped to a real interval of width 1, a precision to the second decimal point is obtained. Greater precision can be achieved by adding more bits to the encoding, but the length of each chromosome increases, and the search space grows. Two substrings of length 7 encode a parameter setting (α_1, μ), and three different settings are concatenated on each chromosome, in order to perform three different runs of the parallel insertion heuristic for each problem, as in [7]. A typical chromosome is depicted below.

1000111	0011011	0000000	1111111	0101011	1000000
α_1	μ	α_1	μ	α_1	μ

In this example, the parameter settings are :

$$(\alpha_1,\mu) = \{(71/127,\ 27/127),\ (0/127,\ 127/127),\ (43/127,\ 64/127)\}$$
$$\cong \{(0.56,\ 0.21),\ (0.00,\ 1.00),\ (0.34,\ 0.50)\}.$$

Reproduction

The fitness of each chromosome is derived from the quality of the solution produced by the parallel insertion heuristic, using the parameter settings encoded on the chromosome. Solution quality is based on the number of routes and, for the same number of routes, on total route time. In order to assign a numerical fitness value to a chromosome, its rank in the population is used [19]. In the example below, the population is composed of four chromosomes (encoding four different parameter settings). The value of the solution associated with each chromosome is shown on the same line. In this example, chromosome 3 gets rank 1 because its parameter settings produce the minimum number of routes, while chromosomes 2, 1 and 4 get ranks 2, 3 and 4, respectively.

	Number of Routes	Route Time	Rank
chromosome 1	12	1612.0	3
chromosome 2	12	1588.1	2
chromosome 3	11	1660.0	1
chromosome 4	13	1380.0	4

With these ranks, the fitness of each chromosome can be computed as follows:

$$\text{fitness}_i = \text{Max} - [(\text{Max} - \text{Min}) \times (i-1)/(n-1)],$$

where i is the rank of the chromosome, and n is the number of chromosomes in the population. Hence, the best ranked chromosome gets fitness value Max and the worst chromosome gets fitness value Min. In the current implementation, Max and Min are set to 1.5 and 0.5, respectively.

A fitness proportional selection scheme is then applied to these values. Specifically, the selection probability for a chromosome of rank i is:

$$p_i = \frac{\text{fitness}_i}{\Sigma_{j=1,\ldots,n} \text{fitness}_j} = \frac{\text{fitness}_i}{n}$$

It is worth noting that the sum over all fitness values in a population of size n is equal to n, because Min + Max = 2, and the average fitness is 1. Since n different selections (with replacement) must be performed on the current population in order to select n parents and generate n offspring, the expected number of selections E_i for a chromosome of rank i is:

$$E_i = n \times p_i = \text{fitness}_i \ .$$

Hence, the fitness of each chromosome is also equal to its expected number of selections. For example, the best chromosome with fitness Max=1.5, is expected to be selected 1.5 times on average over n different trials.

To reduce the variance associated with proportional selection (i.e., each chromosome has a non null probability of being selected between 0 and n times over n different trials), stochastic universal selection or SUS was applied to the fitness values. This approach guarantees that the number of selections for any given chromosome is at least the floor of its fitness, and at most the ceiling of its fitness [20].

Recombination

The crossover operator is the two-point crossover. The crossover rate is set to 0.60. Hence, about 40% of the parent chromosomes are copied from one generation to the next without any modification.

Mutation

The classical mutation operator is applied to each new offspring at a fixed rate of 0.01. Hence, each bit has one chance out of 100 to change from 0 to 1 or from 1 to 0.

Generation Replacement.

Each new generation replaces the old one. However, elitism is used, where the best chromosome is preserved from one generation to the next.

Section 6. Computational Results

In this section, the set of test problems is briefly described. Then, we give some additional details about the implementation of the neural network model and the genetic algorithm. Finally, computational results on the test problems are reported.

6.1 The test set

For testing purposes, we used Solomon's standard set of 100-customer Euclidean problems [6]. For each problem, the travel times are equivalent to the corresponding Euclidean distances. The design of these problems highlights factors that can affect the behavior of vehicle routing 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 scheduling horizon).

The geographical data were either randomly distributed according to 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). The time window at the depot is narrow for sets R1, C1 and RC1. Hence, only a few customers can be serviced on the same route. Conversely, the time window at the depot is wide for sets R2, C2 and RC2, so that many customers can be serviced on the same route.

Each set includes problems with wide time windows, narrow time windows or a mix of wide and narrow time windows. For problems of type R and RC, the time windows have a uniformly distributed, randomly generated center and a normally distributed random width. For problems of type C, the time windows are positioned around the arrival times obtained from near optimal solutions of the corresponding problems with no time windows. The reader is referred to [6] for additional details about these problems.

6.2 The implementation

This section provides additional implementation details about the neural network model and the genetic algorithm.

6.2.1 The neural network model

In the current implementation, the following values and initial conditions are used:

(a) all the weight vectors are initially located at the origin (plus a small random perturbation)

(b) the learning rate η is initially set to 0.8 and its value is decreased from one pass to another through the set of coordinates, according to the following formula [16]:

$$\eta = 0.8 \times e^{-0.239 \times (p-1)}$$

In this formula, p is the number of passes through the coordinates, and is initially set to 1.

(c) the stopping criterion is

$$\Delta E < 0.001 ,$$

where E is the objective function that monitors the network's performance. Hence, the training algorithm stops when E decreases by less than 0.001 after one complete pass through the coordinates.

6.2.2 The genetic algorithm

The genetic search for the best parameter settings is applied to the parallel insertion heuristic with the neural network initialization. Although the genetic search is computationally expensive and runs for about 15 minutes on a SPARC 2 workstation for each problem set, the best parameter settings can be used to solve any new problem with similar characteristics (i.e., there is no need to run the genetic algorithm again). Also, a different search is performed on each set of problems. Consequently, the best parameter settings are not the same from one set of problems to another.

During the genetic search, the fitness and rank of each chromosome is determined through the average solution produced over a particular set of problems, according to the following rule: for each problem j in set X, run the parallel insertion heuristic with the three parameter settings encoded on the chromosome, and take the best solution $best_j$; then, compute the average of the $best_j$'s over set X. For these experiments, the size of the population was set to 30, and the number of generations was set to 20. In each case, the initial populations were seeded with chromosomes encoding the parameter settings suggested in [6,7]. More precisely, eight chromosomes were constructed from these settings, and the remaining chromosomes were randomly generated.

6.3 Numerical results

Table 1 shows the results of the three route construction heuristics using Solomon's test problems. I1 is Solomon's heuristic [6]. PARIS is the PARallel InSertion heuristic of Potvin and Rousseau [7], while PARIS+ is the new implementation with the neural network model and the genetic algorithm. Finally, a comparison is also provided with the cyclic transfer heuristic SP3 of Thompson and Psaraftis [5]. In the latter case, route improvement procedures are included in the heuristic (e.g., exchange of customers between routes). Therefore, SP3 is not a pure route construction heuristic, unlike the three other methods.

For each set of problems, Tables 1a, 1b and 1c show the average number of routes, route time, and computation time. For PARIS+, the three parameter settings suggested by the genetic algorithm are also shown. Note that each parameter value must be divided by 2^7-1 or 127 in order to get the exact real number, and that the α_2 value can be determined through the equation $\alpha_1 + \alpha_2 = 1$. Finally, note that a reduction of 0.1 in the average number of routes over a particular set of problems means that one route has been saved overall (since there are approximately 10 problems in each set).

R1 12 problems	Avg. Number of Routes	Avg. Route Time	Avg. Comput. Time (seconds)	(α_1, μ)
I1	13.6	2695.5	1.2*	---
PARIS	13.3	2696.0	4.2*	---
PARIS+	13.2	2660.1	5.2*	(127,119),(127,092),(063,127)
SP3	13.1	2484.0	65.0**	---

* Sparc2 workstation
** IBM PC-AT

R2 11 problems	Avg. Number of Routes	Avg. Route Time	Avg. Comput. Time (seconds)	(α_1, μ)
I1	3.3	2578.1	6.2	---
PARIS	3.1	2513.3	4.0	---
PARIS+	3.0	2490.6	4.4	(123,115),(109,102),(127,127)
SP3	3.1	2333.0	260.0	---

Table 1a. Random Problems

C1 9 problems	Avg. Number of Routes	Avg. Route Time	Avg. Comput. Time (seconds)	(α_1, μ)
I1	10.0	10104.2	1.2	---
PARIS	10.7	10610.3	3.9	---
PARIS+	10.0	10042.3	4.2	(121,043),(127,116),(126,114)
SP3	10.0	9965.0	31.0	---

C2 8 problems	Avg. Number of Routes	Avg. Route Time	Avg. Comput. Time (seconds)	(α_1, μ)
I1	3.1	9921.4	3.6	---
PARIS	3.4	10477.6	3.2	---
PARIS+	3.0	9706.6	3.7	(127,101),(122,116),(126,111)
SP3	3.0	9649.0	71.0	---

Table 1b. Clustered Problems

RC1 8 problems	Avg. Number of Routes	Avg. Route Time	Avg. Comput. Time (seconds)	(α_1, μ)
I1	13.5	2775.0	1.2	---
PARIS	13.4	2877.9	4.1	---
PARIS+	13.0	2779.7	6.1	(105,124),(123,114),(095,110)
SP3	13.0	2598.0	61.0	---

RC2 8 problems	Avg. Number of Routes	Avg. Route Time	Avg. Comput. Time (seconds)	(α_1, μ)
I1	3.9	2955.4	4.8	---
PARIS	3.6	2807.4	3.3	---
PARIS+	3.4	2701.1	3.9	(109,116),(118,122),(084,111)
SP3	3.7	2706.0	140.0	---

Table 1c. Mixed problems

The tables show that the neural network and genetic algorithm in PARIS+ provide substantial improvement over the results obtained with the original PARIS implementation. In fact, a total of 18 routes were saved over the 56 test problems. This reduction is important, because large acquisition and maintenance costs are associated with each vehicle. The route savings were achieved through the fine tuning of the parameter values, and through the application of the neural network-based route initialization procedure.

Table 1 also shows that PARIS+ outperforms I1 on all problem sets. It is important to remember that PARIS, PARIS+, and I1 are all pure route construction heuristics. That is, they insert customers one by one into routes until all customers are serviced. There are no reordering procedures to modify the sequence of customers within each route, or exchange of customers between routes to improve the final solution. Even if PARIS+ does not incorporate any reordering or exchange procedures, it is competitive with SP3 in regard to the average number of routes. Overall, two additional routes are saved over SP3 on the 56 test problems. However, the reordering and exchange procedures within SP3 significantly improve total route time with the exception of problem set RC2.

Section 7. Concluding Remarks

In this paper, a competitive neural network model and a genetic algorithm were used to improve a parallel insertion heuristic for the VRPTW. Overall, the combination of neural network, genetic algorithm and operations research techniques proved to be very useful, and provided a way to reduce the number of routes significantly.

We note two possible avenues for further research. First, the neural network initialization was based on spatial considerations only. Better results could possibly be achieved by considering both spatial and temporal issues during the initialization phase. In this case, a third input unit relating to the time window at each customer would be added to the neural network (e.g., the upper bound on the time window).

A second avenue of research would focus on the ART network [13]. In the competitive network described in this paper, the number of output units is fixed. Since each output unit stands for a cluster (route), we need to obtain an initial estimate of the number of routes with Solomon's I1 heuristic. The ART network does not require such a priori knowledge. It determines the number of routes by itself, to obtain an acceptable clustering of customers. With the ART model, our parallel insertion heuristic would not rely on Solomon's heuristic to estimate this initial number of routes.

Acknowledgments. We would like to thank Tanguy Kervahut for his help during the computational experiments. Also, this research would not have been possible without the financial support of the Natural Sciences and Engineering Council of Canada (NSERC) and the Quebec Fonds pour la Formation de Chercheurs et l'Aide a la Recherche (FCAR).

References

1. Solomon M.M. and Desrosiers J. (1988), "Time Window Constrained Routing and Scheduling Problems", *Transportation Science 22,* pp. 1-13.
2. Blanton J.L., and Wainwright R.L. (1993), "Multiple Vehicle Routing with Time and Capacity Constraints using Genetic Algorithms", in Proceedings of the Fifth International Conference on Genetic Algorithms, Champaign, IL, pp. 452-459.
3. Solomon M.M., Baker E.K. and Schaffer J.R. (1988), "Vehicle Routing and Scheduling Problems with Time Window Constraints: Efficient Implementations of Solution Improvement Procedures", in Vehicle Routing: Methods and Studies, B.L. Golden and A.A. Assad Eds, North-Holland: Amsterdam, pp. 85-105.
4. Savelsbergh M.W.P. (1990), "An Efficient Implementation of Local Search Algorithms for Constrained Routing Problems", *European Journal of Operational Research 47,* pp. 75-85.
5. Thompson P. and Psaraftis H. (1993), "Cyclic Transfer Algorithms for Multivehicle Routing and Scheduling Problems", *Operations Research 41*, pp. 935-946.
6. Solomon M.M. (1987), "Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints", *Operations Research 35,* pp. 254-265.
7. Potvin J.Y. and Rousseau J.M. (1993), "A Parallel Route Building Algorithm for the Vehicle Routing and Scheduling Problem with Time Windows", *European Journal of Operational Research 66,* pp. 331-340.
8. Clarke G. and Wright J. (1964), "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points", *Operations Research 12*, 568-581.

9. Tillman F. and Cain T. (1972), "An Upper Bounding Algorithm for the Single and Multiple Terminal Delivery Problem", *Management Science 18*, pp. 664-682.

10. Martello S. and Toth P. (1981) "An Algorithm for the Generalized Assignment Problem", in *Operational Research'81: Proceedings of the Ninth IFORS Int. Conf. on Operational Research*, J.P. Brans (Eds), North-Holland, 589-603.

11. Grossberg S. (1976a), "Adaptive Pattern Classification and Universal Recoding I. Parallel Development and Coding of Neural Feature Detectors", *Biological Cybernetics 23*, pp. 121-134.

12. Grossberg S. (1976b), "Adaptive Pattern Classification and Universal Recoding II. Feedback, Expectation, Olfaction, Illusions", *Biological Cybernetics 23*, pp. 187-202.

13. Grossberg S. (1987), "Competitive Learning: From Interactive Activation to Adaptive Resonance", *Cognitive Science 11*, pp. 23-63.

14. Kohonen T. (1988), Self Organization and Associative Memory, Second Edition, Springer: Berlin.

15. Rumelhart D.E. and Zipser D. (1985), "Feature Discovery by Competitive Learning", *Cognitive Science 9*, pp. 75-112.

16. Ahalt S.C., Krishnamurthy A.K., Chen P. and Melton D.E. (1990), "Competitive Learning Algorithms for Vector Quantization", *Neural Networks 3*, pp. 277-290

17. Fogel L.J., Owens A.J., Walsh M.J. (1966), Artificial Intelligence through Simulated Evolution, Wiley: New-York.

18. Holland J.H. (1975), Adaptation in Natural and Artificial Systems, The University of Michigan Press: Ann Arbor. Reprinted by MIT Press (1992).

19. Whitley D. (1989), "The Genitor Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best", in Proceedings of the Third Int. Conf. on Genetic Algorithms, Fairfax, VA, pp. 116-121.

20. Baker J.E. (1987), "Reducing Bias and Inefficiency in the Selection Algorithm", in Proceedings of the Second Int. Conf. on Genetic Algorithms, Cambridge, MA, pp. 14-21.