CRT-938

Parameter Optimization for a Vehicle Routing Heuristic Using Genetic Algorithms

by

Jean-Yves Potvin
Danny Dubé

C‿ntre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale A, Montréal, Canada H3C 3J7.

Centre de recherche sur les transports — Publication #938

September 1993

Résumé. Dans cet article, un algorithme génétique est utilisé afin de découvrir de bonnes valeurs pour les paramètres d'une heuristique d'insertion classique pour le problème de tournées de véhicules avec fenêtres de temps. Les valeurs identifiées par l'algorithme génétique permettent d'améliorer les résultats obtenus précédemment avec cette heuristique, sur un ensemble de problèmes tests. L'article contient également une introduction aux algorithmes génétiques.

Mots Clés. Algorithmes Génétiques, Tournées de Véhicules, Fenêtres de Temps

Abstract. In this paper, a genetic algorithm is used to search the space of parameter settings of a classical insertion heuristic for the vehicle routing problem with time windows. The parameter settings, as identified by the genetic search, greatly improve previous results obtained with the insertion heuristic on a standard set of test problems. The paper also provides a comprehensive introduction to genetic algorithms.

KeyWords. Genetic Algorithms, Vehicle Routing, Time Windows.

Section 1. Introduction

The Vehicle Routing Problem with Time Windows (VRPTW) is currently the focus of very intensive research, and is used to model many realistic applications [Solomon and Desrosiers 88]. The overall objective is to serve a set of customers at minimum cost with a fleet of vehicles of finite capacity housed at a central depot. Each customer has a known demand for service (either for pick-up or delivery but not both), as well as a time window or time interval for this service. The time window at the depot defines the scheduling horizon. Accordingly, each route must start and end within the bounds of this horizon. It is worth noting that the scheduling horizon acts similarly as a capacity constraint: when the horizon or the capacity is enlarged, a larger number of customers can be served by the same vehicle, and conversely.

Since the hard time window case is considered, a vehicle must wait if it arrives too early at a customer location. Hence, the total routing and scheduling costs include not only the total travel distance (or total travel time), but also the waiting time.

Interesting results are reported in the literature for route building and route improvement heuristics [Or 76, Solomon 87, Solomon et al. 88, Potvin and Rousseau 90, Savelsbergh 90, Potvin and Rousseau 93]. An exact algorithm for the VRPTW is also described in [Desrochers et al. 92]. In this work, a few 100-customer problems were solved to optimality with a column generation technique. However, due to its exponential nature, the algorithm did not find the optimum on many other problems of the same size.

Of particular interest is a paper by Solomon describing a variety of route construction heuristics for the VRPTW [Solomon 87]. In this paper, a sequential time-space based insertion algorithm outperformed all other tested approaches on a standard set of problems. Various parameter settings were suggested by Solomon to tune this heuristic. In this paper, we show that it is possible to greatly improve the results reported in the paper via a careful search in the space of parameter settings. To this end, a genetic algorithm is used to search the parameter space, and find "good" parameter settings for Solomon's heuristic.

In the following, Solomon's heuristic is first introduced. Then, Section 3 briefly describes the basic principles at the core of the genetic search. Section 4 provides additional details about our implementation of the genetic search for the parameter tuning problem. Finally, computational results are reported in Section 5.

Section 2. Solomon's Insertion Heuristic

In this section, we briefly introduce Solomon's approach to the VRPTW [Solomon 1987]. Here, routes are created one at a time. A route is first initialized with a "seed" customer, and the remaining unrouted customers are added to this route until it is full with respect to the time window or capacity constraints. If unrouted customers remain, the initialization and insertion procedures are repeated until all customers are served.

After initializing a route with a seed customer, the method uses two cost measures $c_2(i,u,j)$ and $c_1(i,u,j)$ to guide the selection and insertion of the next unrouted customer u between adjacent customers i and j. Let $(i_0, i_1, i_2, \ldots, i_m)$ be the current route with i_0 and i_m standing for the depot. For each unrouted customer, the best feasible insertion place in the current route is first computed as follows:

$$c_1^*(i(u),u,j(u)) = \min_{p=1,\ldots,m} [c_1(i_{p-1},u,i_p)].$$

$$c_1(i,u,j) = \alpha_1 \, c_{11}(i,u,j) + \alpha_2 \, c_{12}(i,u,j),$$

$$\alpha_1+\alpha_2=1, \; \alpha_1 \geq 0, \; \alpha_2 \geq 0.$$

where

$$c_{11}(i,u,j) = d_{i,u} + d_{u,j} - \mu \, d_{i,j},$$

$d_{k,l}$ = distance in time units between k and l.

$$c_{12}(i,u,j) = b_{u,j} - b_j,$$

b_l = current service time at l.
$b_{k,l}$ = new service time at l, given that k is in the route.

Next, the best customer u* is selected as follows:

$$c_2(i(u^*),u^*,j(u^*)) = \max_u [c_2(i(u),u,j(u))], \text{ u unrouted and feasible.}$$

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

Client u* is then inserted in the route between i(u*) and j(u*). When there is no feasible insertion place in the route for the remaining unrouted customers, the method initializes a new route. This procedure is repeated until all customers are routed.

The insertion cost c_1 is a weighted sum of detour et delay, while the selection cost c_2 is a generalized savings measure (the classical savings are obtained by setting $\alpha_1=1$, $\alpha_2=0$, $\mu=1$, $\lambda=2$ [Clarke and Wright 64]).

Solomon uses two different initialization criteria to select seed customers, namely the farthest customer from the depot and the customer with earliest deadline (earliest time window's upper bound). In addition, he uses four different parameter settings to tune his heuristic, namely $(\alpha_1,\alpha_2,\mu,\lambda) = \{(1,0,1,1), (1,0,1,2), (0,1,1,1), (0,1,1,2)\}$. Consequently, Solomon's heuristic is run eight times on each problem, and the best solution is selected as the final result.

In the next sections, we show that much better results can be obtained with the same heuristic, via a proper setting of the parameter values. Since the search in the parameter space is performed with a genetic algorithm, the next section first introduces the genetic algorithms in general terms. Then, Section 4 describes our implementation for tuning Solomon's parameters.

Section 3. Genetic Algorithms

At the origin, evolution algorithms were randomized search techniques aimed at simulating the natural evolution of asexual species [Fogel 66]. The work of Holland and his students extended this model by allowing the combination or crossover of information taken from two parents to create a new offspring. These algorithms were called "genetic algorithms" [Bagley 67], and the introduction of the crossover operator proved to be a fundamental ingredient in the success of this search technique.

Broadly speaking, genetic algorithms differ from traditional search techniques in the following ways [Goldberg 89]:

(a) Genetic algorithms manipulate bit strings or chromosomes encoding useful information about the problem, but they do not manipulate the information as such (no decoding or interpretation).

(b) Genetic algorithms uses the evaluation of a chromosome, as returned by the fitness function, to guide the search. They do not use any other information about the fitness function or the application domain.

(c) The search is run in parallel from a population of chromosomes

(d) The transition from one chromosome to another in the search space is done stochastically

Points (a) and (b), in particular, explain the robustness of the genetic algorithms, and their wide applicability as meta-heuristics in many application domains. In the following, we briefly introduce the reader to the basic principles at the core of the genetic search.

3.1 Genetic Search

In order to get some insight about the nature of the genetic search, we consider the following initial population of 6-bit chromosomes and their respective evaluation or fitness (the column "Interval" is derived from the running sum of the fitness values, and is related to the proportional selection scheme, as described in Section 3.2.1):

		Fitness	Interval
Chromosome 1:	110001	1.80	]0.00, 1.80]
Chromosome 2:	010101	0.20	]1.80, 2.00]
Chromosome 3:	111001	2.00	]2.00, 4.00]
Chromosome 4:	100101	0.10	]4.00, 4.10]
Chromosome 5:	000011	1.90	]4.10, 6.00]
Chromosome 6:	010111	1.80	]6.00, 7.80]
Chromosome 7:	001100	0.10	]7.80, 7.90]
Chromosome 8:	101010	0.10	]7.90, 8.00]

By looking at the similarities and differences between the chromosomes, and by comparing their fitness values, it is possible to hypothesize that chromosomes with high fitness values have two 1's in the first two positions, or two 1's in the last two positions.

These similarities are exploited by the genetic search via the concept of schemata or similarity templates. A schema is composed of 0's and 1's, like the original chromosomes, but with the additional "wild card" or "don't care" symbol * (standing either for 0 or 1). Via the don't care symbol, schemata represent subsets of chromosomes in the population. For example, the schema 11**** stands for chromosomes 1 and 3 in the above population, while schema ****11 stands for chromosomes 5 and 6.

Two fundamental properties of schemata are the order and defining length, namely:

(a) The *order* is the number of positions with fixed values (the schema 11**** is of order 2, the schema 110*00 is of order 5)

(b) The *defining length* is the distance between the first and last positions with fixed values (e.g. the schema 11**** is of length 1, the schema 1****1 is of maximal length 5)

A "building block" is a schema of low order, short defining length and above-average fitness (the fitness of a schema is defined as the average fitness of its members in the current population). Generally speaking, the genetic algorithm moves in the search space by combining building blocks from two parent chromosomes into a single offspring. Consequently, the basic assumption at the core of the genetic algorithm is that a better chromosome is likely to be generated by combining the best features of two good chromosomes. In the example above, the genetic algorithm would combine substrings of bits taken from a chromosome with two 1's in the first two positions and a chromosome with two 1's in the last two positions, to create an offspring with two 1's in the first two positions *and* two 1's in the last two positions (in the hope that this chromosome is better fit than both of its parents)

3.2 A Simple Genetic Algorithm

Based on the above principles, a simple "pure" genetic search algorithm is defined. In the following description, many new terms are introduced. These terms will be defined more precisely in sections 3.2.1 to 3.2.4.

1. Create an initial population of N chromosomes (Generation 0)
2. Evaluate the fitness of each chromosome
3. Select N parents from the current population via proportional selection (i.e. the selection probability is proportional to the fitness)
4. Choose at random a pair of parents for mating. Exchange substrings of bits with the one-point crossover to create two offsprings.

5. Process each bit in the offsprings by the mutation operator, and insert the resulting offsprings in the new population
6. Repeat Steps 4 and 5 until all parents are selected and mated (N offsprings are created).
7. Replace the old population of chromosomes by the new one (new generation)
8. Evaluate the fitness of each chromosome in the new population
9. Go back to Step 3 if the number of generations is less than the maximum number of generations. Otherwise, the final result is the best chromosome generated during the search.

The above algorithm introduced many new concepts, like the probability of a chromosome to be parent, the one-point crossover operator to exchange bit strings, and the mutation operator to introduce random perturbations in the search. These concepts are now defined more precisely.

3.2.1 Selection Probability (selection pressure)

The parent chromosomes are selected for crossover via proportional selection, also known as "roulette wheel selection". It is defined as follows:

1. Sum up the fitness values of all chromosomes in the population
2. Generate a random number between 0 and the sum of the fitness values
3. Select the first chromosome whose fitness value, added to the sum of the fitness values of the previous chromosomes, is greater or equal to the random number.

Hence, the interval $]\Sigma_{i=1,\ldots,n-1}\text{Fitness}_i, \Sigma_{i=1,\ldots,n}\text{Fitness}_i]$ is associated to chromosome n, and the chromosome is selected if the random number falls in this interval. In the population of chromosomes of Section 3.1, the total fitness value is 8.00, and the interval assigned to each chromosome is shown in the last column, under the heading "Interval". Hence, the first chromosome is chosen when the random number falls in the interval [0,1.80]. Similarly, chromosomes 2 to 8 are chosen if the random number falls in the intervals]1.80, 2.00],]2.00, 4.00],]4.00, 4.10],]4.10, 6.00],]6.00, 7.80],]7.80, 7.90] and]7.90,8.00], respectively. Obviously, a chromosome with high fitness has a greater probability of being selected as a parent (assimilating the sum of the fitness values to a roulette-wheel, a chromosome with high fitness covers a larger fraction of the roulette). Chromosomes with high fitness contain more above-average building blocks, and are thus favored during the selection process. In this way, good solution features are propagated to the next generation.

It is worth noting that proportional selection has also some drawbacks. In particular, a "super-chromosome" with a very high fitness value can quickly become dominant in the population, and cause premature convergence. When such a situation occurs, the population does not evolve anymore, because all its members are derived from the super-chromosome (and are similar). To alleviate this problem, the ranks of the raw fitness values can be used to evaluate the chromosomes [Whitley 89]. In this case, the selection probability of a chromosome is related to its rank in the population, rather than its absolute fitness value. Hence, the selection probability of a super-chromosome becomes identical to the selection probability of any other chromosome of rank 1 in a given population.

Another drawback of proportional selection is the large variance associated to the number of selections of a single chromosome. For a population of size N, any given chromosome can be selected between zero and N times. In the latter case, the same chromosome is selected at each trial, and consequently, all offsprings are derived from a single chromosome! To alleviate this problem, many selection schemes now provide bounds on the number of selections associated to each chromosome, like Stochastic Universal Sampling (SUS) [Baker 87]. As opposed to roulette-wheel selection, SUS guarantees that each chromosome i will be selected at least Floor(Fitness_i) times and at most Ceiling(Fitness_i) times. Moreover, this procedure is very fast, with a time complexity of O(N) for a population of N chromosomes (as opposed to O(N^2) for a naive implementation of the roulette-wheel selection, and O(NlogN) for a clever implementation with B-trees).

On a classical roulette-wheel, there is a single random number for selecting the winning chromosome. In stochastic universal sampling, there are N equally spaced numbers indicating N winners. When the sum of the fitness values in the population is equal to N, the numbers are exactly 1.0 apart. The first number is randomly chosen between zero and one, and each successive number is obtained by adding one to the previous number. For the population of Section 3.1, we get eight numbers (0.70, 1.70, 2.70, 3.70, 4.70, 5.70, 6.70, 7.70), if we assume that the initial random number is

0.70. The two first values fall within the interval of chromosome 1, and this chromosome is selected twice. In the same way, the third and fourth numbers fall in the interval of chromosome 3, and this chromosome is also selected twice. The fourth and fifth numbers indicate chromosome 5, while the two last numbers indicate chromosome 6. Hence, chromosomes 2, 4, 7 and 8 are not selected, and do not contribute to the next population.

3.2.2 One-point crossover

The one-point crossover operator is aimed at exchanging bit strings between two parent chromosomes. A random position between 1 and L-1 is chosen along the two parent chromosomes, where L is the chromosome's length. The chromosomes are then cut at the selected position, and their end parts are exchanged to create two offsprings. In the example below, the parent chromosomes are cut at position 3.

Example:	parent1 :	**1**	**1**	**0**	|	**0**	**0**	**1**
	parent2 :	0	1	0	|	1	1	1
	offspring1 :	**1**	**1**	**0**		1	1	1
	offspring2 :	0	1	0		**0**	**0**	**1**

A probability is associated to the application of the crossover operator, and it is usually set to 0.6. If the operator is not applied to the selected parents, they are both copied in the new population without any modification.

Extensions of this operator, like two-point crossover, M-point crossover and uniform crossover are reported in the literature, but their description would be beyond the scope of this paper [Goldberg 89].

3.2.3 Mutation

The bits of the two offsprings generated by the one-point crossover are then processed by the mutation operator. This operator is applied in turn to each bit with a small probability (e.g. 0.01). When it is applied at a given position, the bit value switches from 0 to 1 or from 1 to 0. The aim of the mutation operator is to introduce random perturbations into the search process. It is useful, in particular, to introduce diversity in homogeneous populations, and to restore bit values that cannot be recovered via crossover (e.g. when the bit value at a given position is the same for every chromosome in the population).

3.2.4 Generation replacement

In the simple genetic algorithm, the whole population is replaced by a new population at each generation. Other approaches maintain a fraction of the population from one generation to the next. For example, the elitist approach maintains the best member of the old population in the new population.

Section 4. Parameter Tuning with a Genetic Algorithm

In this application, the genetic search is used to find good parameter settings for Solomon's heuristic (each chromosome encoding one or more parameter settings). The parameter settings encoded on a chromosome are provided to Solomon's heuristic, and the solutions produced with these settings are used to evaluate the chromosome's fitness.

In this section, we first describe the encoding of the parameters as bit strings or chromosomes. Then, we give details about the exact implementation of the genetic algorithm.

4.1 Encoding the Parameters

The domain of values to be explored for each numerical parameter value is the following :

$\alpha_1 \in [0,1]$ (note that α_2 is determined through α_1, since $\alpha_1+\alpha_2=1$ in Solomon's algorithm)

$\mu \in [0,1]$

$\lambda \in [1,2]$

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 real domain of the parameter. For example, if the bit string is 1010101 for parameter α_1 (that is, 85 in decimal notation), the α_1 value encoded by this string is 85/127 or approximately 0.67. In general, if the integer value corresponding to the bit string is x, the real value of α_1 is x/127. The same transformation applies to parameter μ. For λ, the transformation is $x/127 + 1$, because the domain of values for this parameter is [1,2].

With 127 values mapped to real intervals of width 1, we get a precision to the second decimal point. Greater precision can be achieved by adding more bits to the encoding, but the length of the chromosomes increases and the size of the search space grows up.

The three substrings of length 7 are concatenated on a single chromosome to encode a parameter setting, which is then provided to Solomon's heuristic. An additional bit is put at the end of the chromosome to select the initialization criterion: if this bit is one, route initialization is done with the farthest customer from the depot, otherwise, initialization is done with the customer with earliest deadline. A typical chromosome is depicted in Figure 1. In this case, the parameter values are $(\alpha 1, \mu, \lambda) = (71/127, 27/127, 70/127) \cong (0.56, 0.21, 0.55)$, and route initialization is done with the farthest customer from the depot.

1000111	0011011	1000110	1
α_1	μ	λ	initialization

Figure 1. Decoding a chromosome

4.2 Implementation of the genetic search

Generally speaking, our implementation follows the guidelines provided in Section 3. In the following, we provide additional details about the components of our genetic search.

4.2.1 Fitness Values

The fitness of a chromosome is related to the quality of the solutions generated by Solomon's heuristic, with the parameter settings encoded on the chromosome. Solution quality is based first on the number or routes, and second, on total route time (i.e. travel time + waiting time + service time).

To assign a numerical fitness value to a chromosome, we use the rank of the chromosome in the population, as suggested in [Whitley 89]. In the example below, the population is composed of five chromosomes, encoding five different parameter settings. The solution generated by Solomon's heuristic with each parameter setting (for a given problem) is shown besides the corresponding chromosome. In this example, chromosome 3 gets rank 1 because it generated the minimum number of routes, while chromosomes 2, 1, 4, and 5 get ranks 2, 3, 4, and 5, respectively.

	Number of Routes	Route Time
chromosome 1	12	1612.0
chromosome 2	12	1588.1
chromosome 3	11	1660.0
chromosome 4	12	1644.0
chromosome 5	13	1928.0

With these ranks, it is possible to determine the fitness value of a chromosome via the following formula:

$$\text{Min} + (\text{Max} - \text{Min})\ (i\text{-}1/N\text{-}1)$$

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. Note that the sum of the fitness values is equal to the number of chromosomes in the population (N), because Min+Max=2.0.

4.2.2 Selection Probability

Chromosomes are selected for crossover according to the above fitness values. The selection scheme is Stochastic Universal Sampling (SUS), as described in Section 3.2.1.

4.2.3 Crossover Operator

The crossover operator is the classical one-point crossover. The crossover rate is set to 0.6. Hence, about 40% of the parent chromosomes are not modified by the crossover operator.

4.2.4 Mutation Operator

The mutation operator is applied to each new offspring at a fixed rate of 0.01.

4.2.5 Generation Replacement

Each new generation replaces the old one. However, elitism is used, and the best chromosome at generation M is preserved in generation M+1.

Section 5. Computational Results

For the computational tests, we used the standard set of problems of Solomon, which are all 100-customer Euclidean problems. In these problems, the travel times between customers are equal to the corresponding Euclidean distances. The geographical data were either randomly generated using a uniform distribution (problem sets R1 and R2), clustered (problem sets C1 and C2) or semi-clustered with a mix of randomly distributed and clustered customers (problem sets RC1 and RC2). Problem sets R1, C1 and RC1 have a narrow scheduling horizon, and only a few customers can be served by the same vehicle. Conversely, problem sets R2, C2 and RC2 have a large scheduling horizon, and many customers can be served by the same vehicle. Additional details about these problems, in particular details relating to the characteristics of the time windows, may be found in [Solomon 1987].

The objective is first to minimize the number of routes, and second, to minimize route time or schedule time, that is, the sum of travel time, waiting time and service time (note that a fixed service time value of 9,000 must be added to the sum of travel time and waiting time in sets C1 and C2, and a value of 1,000 in the other sets). Table 1 shows the results obtained by specializing the genetic algorithm to each set of problems. In this case, a different genetic search is performed on each set of problems. Consequently, the best parameter settings are not necessarily the same from one problem set to another.

In the Tables, "Solomon" corresponds to the solutions reported in [Solomon 87]. In this case, the results were obtained by running the algorithm with Solomon's parameter settings, that is:

```
For each problem j is set X do
    run Solomon's heuristic eight times on problem j,
    with the eight parameter settings suggested in
    [Solomon 87], and take the best solution $best_j$.

Compute the average of the $best_j$'s over set X.
```

"Genetic-i" i=1,...,8, is the result of the genetic search with i different parameter settings encoded on each chromosome. Note that a chromosome of type i is obtained by concatenating i chromosomes of the type shown in Figure 1. The rank of a chromosome of type i is determined through the average solution obtained over a particular set of problems in Solomon's testbed, namely:

```
For each problem j is set X do
    run Solomon's heuristic i times on problem j, with
    the i parameter settings encoded on the
    chromosome, and take the best solution $best_j$.
Compute the average of the $best_j$'s over set X.
```

For each set of problems R1, R2, C1, C2, RC1, and RC2, the Tables show the minimum number of parameter settings needed to improve Solomon's results. We also show the results of Genetic-8 on each set of problems. For i=1, the results were obtained after 15 generations, on a population of chromosomes of size 30. For i=2,3, the results were obtained after 20 generations, with the same population size. Finally, for i=8, the results were obtained after 25 generations, on a population of size 40. In each case, the initial populations were seeded with chromosomes encoding parameter settings taken from [Solomon 87]. More precisely, eight chromosomes were derived from Solomon's settings, and the remaining chromosomes were randomly generated.

The experiments were performed on a SPARC10 workstation. Note that the heading "Comput. Time" in the Tables is the computation time for running the genetic algorithms (and not the time to run Solomon's algorithm with the parameter settings found by the genetic search).

The final parameter settings are shown under each set of problems. In these settings, init is equal to F or D. In the first case, route initialization is done with the Farthest customer from the depot, and in the second case, initialization is done with the customer with earliest Deadline. Also, the numerical values for α_1, μ, and λ are shown as integer values, and must be divided by 2^7-1 or 127, in order to get the exact real values.

R1 12 problems	Number of Routes	Distance	Waiting Time	Route Time	Comput. Time (min:sec)
Solomon	13.6	1436.7	258.8	2695.5	---
Genetic-2	13.3 (13.7)	1382.5 (1440.7)	286.2 (260.0)	2668.7 (2700.7)	5:14 (12:04)
Genetic-8	13.2	1407.3	248.0	2655.3	21:48

Genetic-2 : $(\alpha_1, \mu, \lambda, \text{init})$ = (122,123,161,F) ,(127,127,254,F)
Genetic-8 : $(\alpha_1, \mu, \lambda, \text{init})$ = (127,120,201,F) ,(127,127,144,F) ,(127,125,157,F)
(000,127,254,F) ,(112,127,144,F) ,(047,036,167,F)
(008,095,127,D) ,(120,035,225,D)

R2 11 problems	Number of Routes	Distance	Waiting Time	Route Time	Comput. Time (min:sec)
Solomon	3.3	1402.4	175.6	2578.1	---
Genetic-2	3.2	1408.8	161.7	2570.5	32:34
Genetic-8	3.2	1313.7	152.5	2466.2	58:03

Genetic-2 : $(\alpha_1, \mu, \lambda, \text{init})$ = (028,079,182,F) ,(108,124,185,D)
Genetic-8 : $(\alpha_1, \mu, \lambda, \text{init})$ = (046,120,232,F) ,(004,125,191,D) ,(032,123,252,D)
(083,127,224,D) ,(120,127,229,D) ,(103,127,251,D)
(100,012,235,D) ,(091,109,208,D)

Table 1a. Random Problems

C1 9 problems	Number of Routes	Distance	Waiting Time	Route Time	Comput. Time (min:sec)
Solomon	10.0	951.9	152.3	10104.2	---
Genetic-2	10.0	966.7	96.4	10063.1	4:28
Genetic-8	10.0	943.5	95.4	10038.9	16:19

Genetic-2 : $(\alpha_1, \mu, \lambda, \text{init})$ = (074,124,254,F) ,(032,108,172,D)
Genetic-8 : $(\alpha_1, \mu, \lambda, \text{init})$ = (013,068,225,F) ,(097,084,241,F) ,(000,126,213,F)
(003,035,163,D) ,(078,083,217,D) ,(122,039,158,D)
(073,092,155,D) ,(096,126,134,D)

C2 8 problems	Number of Routes	Distance	Waiting Time	Route Time	Comput. Time (min:sec)
Solomon	3.1	692.7	228.6	9921.4	---
Genetic-3	3.0	798.7	54.0	9852.7	14:22
Genetic-8	3.0	728.9	50.8	9779.7	46:18

Genetic-3 : $(\alpha_1, \mu, \lambda, \text{init})$ = (127,127,156,F) ,(027,018,163,D) ,(040,031,134,D)
Genetic-8 : $(\alpha_1, \mu, \lambda, \text{init})$ = (000,126,129,F) ,(126,127,243,F) ,(098,122,254,D)
(060,063,130,D) ,(000,103,127,D) ,(124,122,130,D)
(125,127,127,D) ,(123,113,143,D)

Table 1b. Clustered Problems

RC1 8 problems	Number of Routes	Distance	Waiting Time	Route Time	Comput. Time (min:sec)
Solomon	13.5	1596.5	178.5	2775.0	---
Genetic-2	13.3	1661.5	132.2	2793.7	3:41
Genetic-8	13.1	1573.7	151.6	2725.3	13:04

Genetic-2 : $(\alpha_1, \mu, \lambda, \text{init})$ = (066,013,176,F) ,(008,127,147,F)
Genetic-8 : $(\alpha_1, \mu, \lambda, \text{init})$ = (074,059,191,F) ,(006,089,148,F) ,(046,064,171,F)
(127,119,206,F) ,(053,108,224,F) ,(033,026,206,D)
(120,087,127,D) ,(026,104,245,D)

RC2 8 problems	Number of Routes	Distance	Waiting Time	Route Time	Comput. Time (min:sec)
Solomon	3.9	1682.1	273.2	2955.4	---
Genetic-1	3.8	1721.8	160.9	2882.7	10:36
Genetic-8	3.5	1604.1	173.4	2777.5	72:21

Genetic-1 : $(\alpha_1, \mu, \lambda, \text{init})$ = (044,124,244,F)
Genetic-8 : $(\alpha_1, \mu, \lambda, \text{init})$ = (068,123,235,F) ,(044,043,190,F) ,(061,101,220,F)
(072,047,222,F) ,(122,087,209,F) ,(114,125,227,F)
(024,120,184,D) ,(071,013,160,D)

Table 1c. Mixed Problems

As we can see, the single parameter setting found by Genetic-1 outperforms the eight parameter settings of Solomon on set RC2 (c.f. Genetic-1 in Table 1c). For R1, R2, C1, and RC1, the minimum number of parameter settings is two, and for C2, it is three. Hence, it is possible to improve Solomon's results on all problem sets by running his algorithm three times (or less) on each problem, rather than eight times! Of course, the gap with Solomon's solutions gets larger with the eight parameter settings suggested by Genetic-8.

For illustrative purposes, the results on set R1, as obtained with $20*30=600$ randomly generated chromosomes of type 2, is provided under the results of Genetic 2 (between parentheses). As expected, Genetic-2 provides much better solutions than the random search! Also, the random search is more computationally expensive, because each new chromosome must be evaluated, as opposed to the genetic search, where a fraction of the chromosomes are copied unchanged from one generation to the next.

Tables 2a, 2b and 2c show similar results for a single genetic search aimed at finding the eight best parameter settings over all problems. In this case, each chromosome was evaluated by computing the average solution over all 56 test problems. It took 3 hours and 51 minutes on a SPARC10 workstation to run Genetic-8 in this case. The genetic search found the following parameter settings:

$$
\begin{aligned}
(\alpha_1,\mu,\lambda,\text{init}) = {} & (066,118,238,F),(111,126,215,F),(127,119,223,F) \\
& (127,127,127,F),(091,047,190,F),(126,122,175,F) \\
& (032,098,252,D),(001,119,150,D).
\end{aligned}
$$

R1 12 problems	Number of Routes	Distance	Waiting Time	Route Time
Solomon	13.6	1436.7	258.8	2695.5
Genetic-8	13.4	1406.8	273.3	2680.1

R2 11 problems	Number of Routes	Distance	Waiting Time	Route Time
Solomon	3.3	1402.4	175.6	2578.1
Genetic-8	3.2	1373.3	155.7	2529.0

Table 2a. Random Problems

C1 9 problems	Number of Routes	Distance	Waiting Time	Route Time
Solomon	10.0	951.9	152.3	10104.2
Genetic-8	10.0	926.8	153.5	10080.3

C2 8 problems	Number of Routes	Distance	Waiting Time	Route Time
Solomon	3.1	692.7	228.6	9921.4
Genetic-8	3.1	692.5	96.7	9789.3

Table 2b. Clustered Problems

RC1 8 problems	Number of Routes	Distance	Waiting Time	Route Time
Solomon	13.5	1596.5	178.5	2775.0
Genetic-8	13.3	1600.6	161.5	2762.1

RC2 8 problems	Number of Routes	Distance	Waiting Time	Route Time
Solomon	3.9	1682.1	273.2	2955.4
Genetic-8	3.6	1611.8	204.7	2816.5

Table 2c. Mixed Problems

As we can see, the results are better on each set of problems. Apart from reducing route time, these parameter values also save many routes. Hence, the fine tuning of Solomon's heuristic via a genetic search proved to be very beneficial.

Section 6. Conclusion

This paper has shown that it is possible to greatly improve the results of Solomon's heuristic, via a careful search in the parameter space. By specializing the genetic search to each problem set, it was also possible to improve Solomon's results with only three different parameter settings (or less), as opposed to the eight different parameter settings suggested in [Solomon 87].

Acknowledgments. Financial support for this work was provided by the Natural Sciences and Engineering Research Council of Canada (NSERC) and by the Fonds pour la Formation de Chercheurs et l'Aide à la Recherche of the Quebec government (FCAR).

References

[Bagley 67] J.D. Bagley, "The Behavior of Adaptive Systems which employ Genetic and Correlation Algorithms", Doctoral Dissertation, University of Michigan, *Dissertation Abstracts International 28(12), 5106B.*

[Baker 87] J.E. Baker, "Reducing Bias and Inefficiency in the Selection Algorithm", in Proceedings of the Second Int. Conf. on Genetic Algorithms, pp. 14-21.

[Clarke and Wright 64] G. Clarke G. and W. Wright, "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points", *Operations Research 12*, pp. 568-581.

[Desrochers et al. 88] M. Desrochers, J.K. Lenstra, M.W.P. Savelsbergh and F. Soumis, "Vehicle Routing with Time Windows: Optimization and Approximation", in Vehicle Routing: Methods and Studies, B.L. Golden and A.A. Assad (Eds), North-Holland, pp. 65-84.

[Desrochers et al. 92] M. Desrochers, J. Desrosiers and M.M. Solomon, "A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows", *Operations Research 40*, pp. 342-354.

[Fogel et al. 66], L.J. Fogel, A.J. Owens, M.J. Walsh, Artificial Intelligence through Simulated Evolution, Wiley.

[Goldberg 89] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley.

[Holland 75] J. H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor. Reprinted by MIT Press (1992).

[Or 76] I. Or, "Traveling Salesman-type Combinatorial Problems and their relation to the Logistics of Blood Banking", Ph.D. Thesis, Dept. of Industrial Engineering and Management Sciences, Northwestern University.

[Potvin and Rousseau 90] J.Y. Potvin and J.M. Rousseau, "A New Exchange Heuristic for Routing Problems with Time Windows", Technical Report #729, Centre de Recherche sur les Transports, Université de Montréal.

[Potvin and Rousseau 93] J.Y. Potvin and J.M. Rousseau, "A Parallel Route Building Algorithm for the Vehicle Routing and Scheduling Problem with Time Windows", *European Journal of Operational Research 66*, pp. 331-340.

[Savelsbergh 90] M.W.P. Savelsbergh (1990), "An Efficient Implementation of Local Search Algorithms for Constrained Routing Problems", *European Journal of Operational Research 47*, pp. 75-85.

[Solomon 87] M.M. Solomon, "Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints", *Operations Research 35*, pp. 254-265.

[Solomon and Desrosiers 88] M.M. Solomon and J. Desrosiers, "Time Window Constrained Routing and Scheduling Problems", *Transportation Science 22*, pp. 1-13.

[Solomon et al. 88] M.M. Solomon, E.K. Baker and J.R. Schaffer, "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, pp. 85-105.

[Whitley 89] D. Whitley, "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 (ICGA'89), pp. 116-121.