Improving a Vehicle Routing Heuristic through Genetic Search

Jean-Yves Potvin
Danny Dubé

Centre de Recherche sur les Transports
Université de Montréal
C.P. 6128, Succ. A,
Montréal (Québec)
Canada H3C 3J7

Abstract. A genetic algorithm is applied to the search of good parameter settings for a vehicle routing heuristic. The parameter settings identified by the genetic search allow the insertion heuristic to generate solutions that are much better than the solutions previously reported on a standard set of routing problems.

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. 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. The route of each vehicle starts and ends at the depot. Also, each customer has a known demand (i.e. some quantity of goods to be picked-up), as well as a time window or time interval that constrains the vehicle's arrival time. The time windows are defined within a scheduling horizon, so that each route starts and ends within the bounds of this horizon. 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, the arrival time at a given customer cannot be larger than the time window's upper bound at this customer. On the other hand, the vehicle can wait if it arrives before the lower bound. Accordingly, the total routing and scheduling costs include not only the total travel time, but also the waiting time.

Many interesting problem-solving methodologies for the VRPTW are reported in the literature [Solomon 87, Desrochers et al. 92, Potvin and Rousseau 93]. In this paper, it is shown that the parallel insertion heuristic of Potvin and Rousseau can be greatly improved by performing a careful search in the space of parameter settings, using a genetic algorithm.

The next section will first briefly describe the parallel insertion heuristic. Then, Section 3 will describe our genetic algorithm, and computational results will be reported in Section 4.

Section 2. The Parallel Insertion Heuristic [Potvin and Rousseau 93]

Potvin and Rousseau's heuristic is a parallel version of heuristic I1 described in [Solomon 87]. As opposed to Solomon's sequential approach, where routes are built one by one, the parallel heuristic initializes and builds many routes in parallel. First, each route is initialized with a different "seed" customer (i.e., each initial route serves a single customer). Then, the remaining unrouted customers

are sequentially inserted into any one of the initial routes until all customers are routed.

Let l_{r0}-l_{r1}-l_{r2}-...-$l_{rn(r)}$ be route r, $1 \le r \le m$, with l_{r0} and $l_{rn(r)}$ standing for the depot. For each unrouted customer u, the best feasible insertion place between two consecutive customers in each route r is computed as follows:

$$c_{1r}^*(i_r(u),u,j_r(u)) = \min_{p=1,...,n(r)} [c_{1r}(l_{r(p-1)},u,l_{rp})], \quad r=1,...,m.$$

$$c_{1r}(l_{r(p-1)},u,l_{rp}) = \alpha_1 c_{11r}(l_{r(p-1)},u,l_{rp}) + \alpha_2 c_{12r}(l_{r(p-1)},u,l_{rp}),$$

$$\alpha_1+\alpha_2=1, \ \alpha_1 \ge 0, \ \alpha_2 \ge 0,$$

where

$c_{11r}(k,u,l) = d_{k,u} + d_{u,l} - \mu d_{k,l}$

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

$c_{12r}(k,u,l) = b_{u,l} - b_l$,

b_l = current service time at l,

$b_{u,l}$ = new service time at l, given that u is inserted between k and l.

Next, the best customer u* is selected according to the formula:

$$c_2(u^*) = \max_u c_2(u)$$

$$c_2(u) = \Sigma_{r \ne r'} [c_{1r}^*(i_r(u),u,j_r(u)) - c_{1r'}^*(i_{r'}(u),u,j_{r'}(u))],$$

where

$$c_{1r'}^*(i_{r'}(u),u,j_{r'}(u)) = \min_{r=1,...,m} c_{1r}^*(i_r(u),u,j_r(u))$$

Finally, client u* is inserted in route r' between $i_{r'}(u^*)$ and $j_{r'}(u^*)$. This procedure is repeated until all customers are routed. Note that the insertion cost c_{1r} is a weighted sum of detour and delay, while the selection cost c_{2r} is a generalized regret measure over all routes. The regret estimates what could be lost later if customer u is not immediately inserted in his best route.

This heuristic was applied to Solomon's test problems [Solomon 87] with three different parameter settings, namely, $(\alpha_1,\alpha_2,\mu) = \{(0.5,0.5,1), (0.75,0.25,1), (1,0,1)\}$, and the best solution was selected as the final result. These parameter settings were chosen after experimenting with different values. However, a systematic search of the parameter space was not performed. In the next section, a genetic algorithm is used to identify better parameter settings for this heuristic.

Section 3. Parameter Tuning using a Genetic Algorithm

Genetic algorithms are randomized search techniques that are well adapted to parameter optimization problems [Davis 91]. In this application, each chromosome encodes different parameter settings. These parameter settings are provided to the parallel insertion heuristic, and the average solution produced with these settings on Solomon's test problems is used to evaluate the fitness of the chromosome.

3.1 Parameter Encoding

The domain of values to be explored for each parameter is: $\alpha_1 \in [0,1]$, $\mu \in [0,1]$. It is worth noting 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 (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.

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 each chromosome increases and the search space grows up. Two substrings of length 7 encode a parameter setting, and three different settings are concatenated on each chromosome. A typical chromosome is depicted below. In this case, the parameter settings are (α_1,μ) = {(71/127, 27/127), (0/127,127/127), (43/127,64/127)} ≅ {(0.56, 0.21), (0.00,1.00), (0.34,0.50)}.

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

3.2 Implementation of the Genetic Search

In the following, additional details are provided about the main components of our genetic search.

Fitness Values. The fitness of a chromosome is related to the quality of the solutions generated by the parallel insertion heuristic, using the parameter settings encoded on a chromosome. Solution quality is based first on the number of routes, and second, on total route time (including waiting time). In order to assign a numerical fitness value to a chromosome, its rank in the population is used [Whitley 89]. In the example below, the population is composed of three chromosomes (encoding three different sets of parameter settings). The best solution generated by the heuristic on a given problem with the parameter settings encoded on chromosome i, $1 \le i \le 3$, is shown on the same line. In this example, chromosome 3 gets rank 1 because its parameter settings generate the minimum number of routes, while chromosomes 2 and 1 get ranks 2 and 3, respectively.

	Number of Routes	Route Time
chromosome 1	12	1612.0
chromosome 2	12	1588.1
chromosome 3	11	1660.0

With these ranks, it is possible to determine the fitness value of a chromosome with the formula: Max - [(Max - Min) (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.

Selection Probability. The chromosomes are selected for crossover according to the above fitness values. The selection scheme is Stochastic Universal Sampling [Baker 87].

Crossover Operator. The crossover operator is the one-point crossover. The crossover rate is set to 0.60. Hence, about 40% of the parent chromosomes are not modified by this operator.

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

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

Section 4. Computational Results

For the computational tests, we used Solomon's standard set of problems, which are all 100-customer Euclidean problems. The geographical data were either randomly generated using a uniform distribution (problem sets R1 and R2), clustered (problem sets C1 and C2) or generated from 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 may be found in [Solomon 87].

The objective is first to minimize the number of routes, and second, to minimize total route time (including waiting time). Table 1 shows the results obtained by performing a different genetic search on each set of problems. Hence, the best parameter settings are not necessarily the same from one problem set to another.

In Table 1, "Solomon" corresponds to the solutions reported in [Solomon 87] for heuristic I1, using four different parameter settings and two different initialization criteria. "Paral" corresponds to the solutions reported in [Potvin and Rousseau 93] for the parallel insertion heuristic, using the three parameter settings suggested in their paper. Finally, "Paral-Gen" corresponds to the solutions generated by the parallel insertion heuristic, using the three best parameter settings identified by the genetic algorithm on each set of problems. The heading "Comput. Time" refers to the average computation time of the genetic algorithm in minutes and seconds on a SPARC10-41 workstation.

During the genetic search, the fitness and rank of each chromosome was determined through the average solution obtained over a particular set of problems, namely: "For each problem j is set X, run the parallel insertion heuristic three times on problem j, using 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". The results were obtained after 20 generations, on a population of size 30. In each case, the initial populations were seeded with chromosomes encoding parameter settings taken from [Solomon 87, Potvin and Rousseau 93]. More precisely, eight chromosomes were derived from these settings, and the remaining chromosomes were randomly generated. The three best parameter settings are shown for each problem set in Table 1. In each case, the values for α_1 and μ are shown as integer values, and must be divided by 2^7-1 or 127, in order to get the exact real values.

Table 1 shows that the best parameter settings identified by the genetic algorithm allow the parallel insertion heuristic to perform much better on each set of problems. In particular, the average number of routes is reduced in each case. Paral-Gen is now much closer to Solomon on problems of type C, while it is much better on the remaining sets. The genetic algorithm is computationally expensive, but once the best parameter settings are identified for a given problem set, they can be applied to other (new) problems with similar characteristics, to generate high quality solutions.

It is worth noting that the results on set R1, with $20*30=600$ randomly generated chromosomes, are provided between parentheses under the results of

Paral-Gen (for illustrative purposes). As expected, Paral-Gen performs better than the random search, since the latter does not reduce the average number of routes of Paral on set R1. Moreover, 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 without any modification from one generation to the next.

Section 5. Conclusion

This paper has shown that it is possible to greatly improve the results of a parallel insertion heuristic, via a careful search in the parameter space. By specializing the genetic search to each problem set, the original results, as reported in [Potvin and Rousseau 93] were greatly improved.

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

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

[Davis 91] L. Davis, Handbook of Genetic Algorithms, Van Nostrand Reinhold.

[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*, 342-354.

[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*, 331-340.

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

[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, 116-121.

R1 12 problems	Number of Routes	Route Time	Comput. Time (min:sec)	(α_1, μ)
Solomon	13.6	2695.5	---	---
Paral	13.33	2696.0	---	---
Paral-Gen	13.25 (13.33)	2659.6 (2669.8)	34:02 (49:01)	(120,093),(117,088),(067,117)

R2 11 problems	Number of Routes	Route Time	Comput. Time (min:sec)	(α_1, μ)
Solomon	3.3	2578.1	---	---
Paral	3.09	2513.3	---	---
Paral-Gen	3.00	2489.0	37:50	(123,115),(119,127),(111,125)

C1 9 problems	Number of Routes	Route Time	Comput. Time (min:sec)	(α_1, μ)
Solomon	10.0	10104.2	---	---
Paral	10.67	10610.3	---	---
Paral-Gen	10.11	10069.1	29:54	(122,021),(121,043),(122,112)

C2 8 problems	Number of Routes	Route Time	Comput. Time (min:sec)	(α_1, μ)
Solomon	3.1	9921.4	---	---
Paral	3.38	10477.6	---	---
Paral-Gen	3.25	9856.5	24:32	(121,125),(127,119),(064,126)

RC1 8 problems	Number of Routes	Route Time	Comput. Time (min:sec)	(α_1, μ)
Solomon	13.5	2775.0	---	---
Paral	13.38	2877.9	---	---
Paral-Gen	13.00	2799.5	23:28	(127,105),(097,095),(096,082)

RC2 8 problems	Number of Routes	Route Time	Comput. Time (min:sec)	(α_1, μ)
Solomon	3.9	2955.4	---	---
Paral	3.62	2807.4	---	---
Paral-Gen	3.50	2762.5	29:31	(084,120),(111,127),(109,118)

Table 1. Computational Results on Solomon's problems