A Comparative Study of Genetic Algorithms for Vehicle Routing with Time Constraints

Wan-Rong Jih
jih@robot.csie.ntu.edu.tw

Ying-Ping Chen
ypchen@solab.csie.ntu.edu.tw

Jane Yung-Jen Hsu
yjhsu@csie.ntu.edu.tw

Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan

Abstract

In this paper we investigate genetic algorithms as a search technique for obtaining near optimal solutions to the "single-vehicle pickup and delivery problem with time windows and capacity constraints" (single-vehicle PDPTW). This work compares four crossover operators, order crossover (OX), uniform order-based crossover (UOX), Merge Cross #1 (MX1) and Merge Cross #2 (MX2). The results of our experiments show that UOX and MX1 are suitable for the single-vehicle PDPTW, but OX and MX2 are not. In the future, we will lay more emphasis on UOX and MX1 for searching better solutions.

1 Introduction

In the real life, a variety of vehicle routing problems emerge in transportation systems, mail delivery routing, and jobshop vehicle scheduling situations. Consider an autonomous vehicle in an automated factory. There are usually many different requests for transporting materials to and from a number of locations. Moreover, each request may be associated with time constraints on its pickup or delivery locations. In general, the vehicle has a load capacity that cannot be exceeded. The vehicle routing problem is to find the optimal route such that the vehicle can complete all the requests while satisfying all the constraints.

The problem was first formulated by Dantzig and Ramser[2], and has been extensively studied by researchers on distribution management problem in operations research. Golden and Assad[6], Laporte[9], Gendreau and Laporte and Séguin[4] provided the surveys of this problem. It is a hard combinatorial problem, and to this day only relatively small VRP instances can be solved to optimality. Heuristic algorithms, are quite often faster and capable of obtaining optimal or near-optimal solutions to much larger problems in a reasonable amount of computer time.

In this paper we introduce genetic algorithms (GAs) for solving the vehicle routing problem. We compared four different crossover operators, including the order crossover[12, 14], uniform order-based crossover[3], and merge crossover #1 and #2[8]. The rest of this paper is organized as follows. Section 2 gives the problem formulation, and Section 3 describes our implementation of genetic algorithms. The experimental results and discussion are in Section 4, followed by the conclusion in Section 5.

2 Problem formulation

In this paper, we consider the single-vehicle pickup and delivery problem with time windows and capacity constraints (single-vehicle PDPTW). Suppose that the vehicle is at a depot initially, from which the vehicle departs for its first destination and will visit a number of locations. For each problem, there are N customer requests, each of which is specified by a pickup point and a delivery point. Let v_i and v_{N+i} denote the pickup point and the delivery point of customer i respectively, and

$$V = \{v_0, v_1, \ldots, v_{2N}\}$$

be the set of vertices in the problem, where v_0 is the depot. Each pickup or delivery point is associated with a time window $[a_i, b_i]$ which is a time interval representing the earliest and the latest processing time of v_i.

In the present work, we have the following assumptions:

- If the vehicle arrives at a destination earlier than the lower bound of its time window, it has to wait.
- The time needed to travel from one location to another location is known a priori.

- Each location has an a priori time window for service.

Our goal is to find a vehicle route starting from a predefined depot, serving all N customers, and ending at one of the delivery points, such that both the traveling time and the waiting time of the vehicle are minimized.

A solution to the single-vehicle PDPTW can be represented as an ordered list of vertices such that

$$\mathcal{S} = (v_{p(0)}, v_{p(1)}, v_{p(2)}, \ldots, v_{p(2N)}),$$

where $p(\cdot)$ is a permutation of integers from 1 to $2N$, and $p(0) = 0$. We define $\mathcal{S}(i)$, $i = 0, \ldots, 2N$, as the ith component of $\mathcal{S}$ (i.e. $\mathcal{S}(i) = v_{p(i)}$) for convenience. Thus, $\mathcal{S}$ is a vehicle route from $v_{p(0)}$ to $v_{p(2N)}$ through $v_{p(1)}$, $v_{p(2)}$, ..., $v_{p(2N-1)}$.

Let function $T_t(v_i, v_j)$ be the traveling time for the vehicle to move from v_i to v_j, and function $T_a(\mathcal{S}, i)$, $i = 1, \ldots 2N$, denote the arriving time of the vehicle at v_i according to the specific route $\mathcal{S}$. We can define the single-vehicle PDPTW as

$$\min \phi(\mathcal{S}),$$

where the objective function $\phi(\cdot)$ is defined as

$$\begin{aligned} \phi(\mathcal{S}) &= w_1 \sum_{i=1}^{2N} T_t(\mathcal{S}(i-1), \mathcal{S}(i)) \\ &+ w_2 \sum_{i=1}^{2N} f(a_i - T_a(\mathcal{S}, i)), \end{aligned} \tag{1}$$

and function f is

$$f(x) = \begin{cases} x & \text{if } x > 0. \\ 0 & \text{Otherwise.} \end{cases}$$

The first part of Equation (1) is the total traveling time needed by the vehicle for completing the route, and the second part of Equation (1) is the total waiting time. Where a_i is the lower bound of time window $[a_i, b_i]$, and w_1, w_2 are the weights reflecting the relative importance of these two parts. This objective function assumes that the vehicle ends at $\mathcal{S}(2N)$.

Besides minimizing $\phi(\cdot)$, solutions are subject to three types of constraints associated with the single-vehicle PDPTW:

- *capacity constraints*: The vehicle has a capacity of C. The capacity constraint cannot be violated (i.e. the total load allocated to a vehicle cannot exceed its capacity) at any time. That is, the equation

$$\sum_{i=0}^{2N} f(cap(\mathcal{S}, i) - C) = 0 \tag{2}$$

has to be satisfied.

- *time window constraints*: Whenever the vehicle arrives at vertex v_i at time t_i, the criterion $t_i \leq b_i$ must be satisfied. It can be formulated as

$$\sum_{i=1}^{2N} f(T_a(\mathcal{S}, i) - b_i) = 0. \tag{3}$$

- *precedence constraints*: Vertex v_i may have to be visited before vertex v_j. For example, the pickup point must be visited before the delivery point if they belong to the same customer. Such constraints are written as

$$\begin{array}{ll} \textbf{If} & p(i) = k \quad \textbf{and} \quad p(j) = N + k, \\ \textbf{then} & i < j. \qquad\quad \forall k = 1, \ldots, N. \end{array} \tag{4}$$

We have modeled the single-vehicle PDPTW as an optimization problem that minimizes $\phi(\mathcal{S})$ with constraints. To avoid minimizing the objective function under the constraints directly, some of the constraints in the problem are relaxed as follows.

The main idea is to permit violations of constraints but with some penalties. First, we define functions ψ_1 and ψ_2 as

$$\psi_1(\mathcal{S}) = \sum_{i=0}^{2N} f(cap(\mathcal{S}, i) - C), \text{ and}$$

$$\psi_2(\mathcal{S}) = \sum_{i=1}^{2N} f(T_a(\mathcal{S}, i) - b_i).$$

The objective function of the relaxed problem can now be defined as

$$\Phi(\mathcal{S}) = \phi(\mathcal{S}) + \gamma_1 \psi_1(\mathcal{S}) + \gamma_2 \psi_2(\mathcal{S}), \tag{5}$$

where γ_1 and γ_2 are penalty coefficients for Equations (2) and (3) respectively.

And then, the relaxed single-vehicle PDPTW is

$$\min \Phi(\mathcal{S})$$

subject to

$$\begin{array}{ll} \textbf{If} & p(i) = k \quad \textbf{and} \quad p(j) = N + k, \\ \textbf{then} & i < j. \qquad\quad \forall k = 1, \ldots, N. \end{array}$$

The precedence constraint will be eliminated by the chromosome adjustment scheme discussed later in this paper. In what follows, all the experiments are designed to solve the relaxed single-vehicle PDPTW.

3 Genetic algorithm

Genetic algorithms (GAs) are search algorithms based on the mechanics of natural selection and natural genetics. "The survivals of the fittest" among string structures are recombined with a structured yet randomized information exchange to form such a search algorithm. In every generation, a new set of artificial creatures (strings) is created using bits and pieces of the fittest of the old; an occasional new part is tried for good measure.

John Holland and his collaborators at the University of Michigan have been the key developers of genetic algorithms. Their approach has led to important discoveries in both natural and artificial systems science. Detailed and further information can be found in many previous publications [5, 10, 11].

A typical genetic algorithm contains

- some individuals which represent the solutions to a particular problem coded by a proper chromosome encoding scheme (i.e. the *population*),
- a *fitness function* to determine the fitness of each individual,
- a *selection* operator to select individuals for applying genetic operators, and
- *recombination* operators (*crossover* and *mutation*) handling the evolution.

We summarize the genetic algorithm used in our work in Algorithm 1.

3.1 Representation

Given a single-vehicle PDPTW with N customer requests, a solution is encoded as a chromosome, which is represented by a permutation of integers from 1 to $2N$. For example, an individual I for a problem of size $N = 4$ may look like

$$I = \ 3\ 2\ 4\ 8\ 7\ 1\ 6\ 5.$$

The corresponding route of the vehicle starts at the initial depot v_0, travels through v_3, v_2, ..., and stops at v_5.

The representation does not preclude infeasible solutions that violate Equation (4). In all the genetic operators defined below, a simple algorithm is used to maintain the feasibility of the corresponding solutions. Algorithm 2 below describes the chromosome adjustment procedure.

The procedure removes all violations of the precedence constraints by making the pickup point to appear before the delivery point for any specific customer. Algorithm 2 runs in $O(N)$ time using $O(N)$ space.

Algorithm 1 The genetic algorithm for the single-vehicle PDPTW

```
Initialize the parameters;
Generate a population P randomly;
Let generation = 1;
repeat
    Clear the new population P';
    Use Φ(·) as the fitness function to evaluate each individual in P;
    while |P'| < population_size do
        Select two parents from P using the tournament selection ([7, 1]) with the tournament size = 2;
        Perform crossover;
        /* Four crossover operators are tested and compared in this paper. */
        Place the offsprings into P';
    end while
    Replace P with P' (i.e. P' → P);
    Let generation = generation + 1;
until (generation = max_generation) or (σ(P)/m(P) ≤ 0.005);
/* σ(P) and m(P) are the standard deviation and the mean value of the fitness of P respectively. */
Output the solutions;
```

Algorithm 2 Chromosome adjustment scheme

Require: $p(\cdot)$ is the permutation representing the solution to be adjusted.
Ensure: $p'(\cdot)$ is the permutation that forms a feasible solution.

```
Create an array A with 2N elements;
for i = 1 to 2N do
    A(p(i)) = i;
end for
for i = 1 to N do
    if A(i) > A(N + i) then
        Swap(A(i), A(N + i));
    end if
end for
for i = 1 to 2N do
    p'(A(i)) = i;
end for
```

3.2 Crossover operators

The chromosomes representation we designed is order-based and research has been done on crossover operators for such chromosomes ([5, 12]). In the present work, we compare the performance of four crossover operators on the single-vehicle PDPTW problem. These operators are order crossover (OX), uniform order-based crossover (UOX), Merge Cross #1 (MX1) and Merge Cross #2 (MX2)[8]. The first two operators are traditional crossover operators, and the last two operators use a global precedence vector to be the guidance of crossover. The crossover operators are described below.

3.2.1 Order crossover (OX)

Order crossover operator was developed in [12] and studied in [14]. The algorithm for OX is shown in Algorithm 3, and an example is given as follows:

Algorithm 3 Order crossover

1: Choose two cut points randomly;
2: Copy genes between cut points of P_2 to C_1;
3: Start from the point immediately after the second cut point and fill the missing genes into C_1 according to their order in P_1;
4: Repeat line 2 to line 3 to generate C_2;

Cut points:			*			*		
P_1:	1	2	3	4	5	6	7	8
P_2:	3	5	1	8	4	7	2	6
After step 2:								
C_1:	-	-	1	8	4	7	-	-
C_2:	-	-	3	4	5	6	-	-
After step 3:								
C_1:	5	6	1	8	4	7	2	3
C_2:	8	7	3	4	5	6	2	1

3.2.2 Uniform order-based crossover (UOX)

Uniform order-based crossover operator is originally described in [3], pp.79–81. UOX is the analog of uniform crossover, translated into the order-based realm. We show it in Algorithm 4.

The following is an example of UOX:

Algorithm 4 Uniform order-based crossover

1: Generate a bit string that is the same length as the parents;
2: Fill in some of the positions on C_1 by copying them from P_1 wherever the bit string contains a "1"; (Now we have C_1 filled in wherever the bit string contained a "1" and we have gaps wherever the bit string contained a "0".)
3: Make a list of the elements from P_1 associated with a "0" in the bit string;
4: Permute these elements so that they appear in the same order they appear in on P_2;
5: Fill these permuted elements in the gaps on C_1 in the order generated in line 4;
6: Carry out a similar process to make C_2.

Binary string:	0	1	1	0	1	1	0	0
P_1:	1	2	3	4	5	6	7	8
P_2:	3	5	1	8	4	7	2	6
After step 2:								
C_1:	-	2	3	-	5	6	-	-
C_2:	3	-	-	8	-	-	2	6
After step 5:								
C_1:	1	2	3	8	5	6	4	7
C_2:	3	1	4	8	5	7	2	6

3.2.3 Merge cross #1 (MX1)

Most traditional crossover operators for order-based GAs do not have strong connection to the constraints of the problems they are applied to. If we apply traditional crossover operators to single-vehicle PDPTW, it may not be conducive to the searching process of optimal solutions because the information of the constraints is not used by these operators. On the other hand, the merge crossover operators are based upon the notion of a global precedence among genes independent of any chromosome, rather than defining a local precedence among genes specific to a chromosome as the traditional recombination operator. That is, each gene in the chromosome has a precedence relationship to every other gene. From the characteristics of constraints of single-vehicle PDPTW, a global precedence relation probably exists among genes. The global precedence vector is formed by such relationship and could be the guidance of the generation of offsprings.

In our single-vehicle PDPTW, each gene is either a delivery point or a pickup point and has an associated time window. Therefore, we can make use of the time window $[a_i, b_i]$ of v_i, and a precedence relationship among the earliest processing time a_i. Given a_i and a_j, if a_i less than a_j, let v_i appear before v_j in the

global precedence vector because it will be a reasonable solution to serves v_i before v_j. In other words, we can sort all a_i in ascending order, and the sorting result will be a global precedence vector.

In the following example, it is suppose that v_8 has the highest precedence and v_1 is the lowerest. The MX1 operator on two chromosomes (say P_1 and P_2) results in a single offspring, as shown below:

P_1:	$\underline{4}$	2	8	6	1	3	7	5
P_2:	$\underline{5}$	3	1	6	8	2	7	4
Global Precedence Vector:								
	8	7	6	5	4	3	2	1

The underlined genes are the ones under consideration in the current step. At the first step, we compare the precedence of v_4 and v_5 according to the global precedence vector. By definition of this vector, it shows that the precedence of v_5 is higher than that of v_4. Therefore, the first gene of the offspring inherits from the gene of P_2. As for P_1, the first gene was swapped with v_5 so that we can maintain the validity. We show the genes swapped in bold face.

P_1:	**5**	$\underline{2}$	8	6	1	3	7	**4**
P_2:	5	$\underline{3}$	1	6	8	2	7	4
C:	**5**	-	-	-	-	-	-	-

Again the gene with the earlier precedence is placed into C, and genes are swapped to maintain validity if necessary. We continue the process until C is filled with genes. The final result is shown below.

C:	5	3	8	6	1	2	7	4

Effectively, it produces a child which is close to the order of the global precedence.

3.2.4 Merge cross #2 (MX2)

The MX2 operator on two chromosome (say P_1 and P_2) also results in a single offspring, as shown below:

P_1:	$\underline{4}$	2	8	6	1	3	7	5
P_2:	$\underline{5}$	3	1	6	8	2	7	4
Global Precedence Vector:								
	8	7	6	5	4	3	2	1

The process is similar to merge two sorted vectors. The first gene of P_2 is found to be prior to the one of P_1 according to the global precedence vector. The prior gene is placed in offspring (C). Now the first gene of P_2 is removed in both individuals.

P_1:	$\underline{4}$	2	8	6	1	3	7	-
P_2:	-	$\underline{3}$	1	6	8	2	7	4
C:	5	-	-	-	-	-	-	-

The next gene of each chromosome is compared and the second gene of P_2 is found to be the prior gene. Again the prior gene is placed into the child, and removed from both chromosomes.

P_1:	-	$\underline{2}$	8	6	1	3	7	-
P_2:	-	$\underline{3}$	1	6	8	2	7	-
C:	5	4	-	-	-	-	-	-

And the next iteration produces

P_1:	-	$\underline{2}$	8	6	1	-	7	-
P_2:	-	-	$\underline{1}$	6	8	2	7	-
C:	5	4	3	-	-	-	-	-

Continuing the process in this fashion, we can get C shown below.

C:	5	4	3	2	8	6	1	7

Note that when gene 1 of P_2 is encountered, the rest gene of P_1 are filled into the offspring. This operator will move the gene with lowest precedence to near the end of the chromosome.

3.3 Mutation

We don't use the mutation operator in this work in order to study the pure relative performance of different crossover operators on the single-vehicle PDPTW.

4 Experimental results and discussion

We have tested the above crossover operators on five problems consisting of 10, 20, 30, 40 and 50 customers, respectively. In these problems, the pickup and delivery points are randomly placed in a rectangular grid, and for each point we randomly add time window constraints. The Euclidean distance is used to measure the distance between a pair of points. For any two points v_i and v_j, let $d(v_i, v_j)$ denotes the distance between v_i and v_j. We define the value of $T_t(v_i, v_j)$ same as the value of $d(v_i, v_j)$, without considering the units. This section describes the experiments, parameters and results.

4.1 Parameters and experiments

The parameters used in our algorithm are listed in Table 1. These parameters are not changed when we deal with different problems.

The population size of each problem is the number of customer multiplied by 5. For the five problems, we test the four crossover operators associated with three different crossover rates (0.45, 0.6, and 0.75). That is, we have $5 \times 4 \times 3 = 60$ experiments. Each experiment is executed for 30 times. Note that each customer is associated with two points, one pickup point and one delivery point.

Parameter	Value
max_generation	2000
tournament_size	2
w_1	1
w_2	2
γ_1	50
γ_2	100

Table 1: Parameters

4.2 Results and discussion

In our experiments, the global precedence vector of MX1 and MX2 is based on the lower bound of time windows. We implement our single-vehicle PDPTW in the programming language C, and the experiments are running on the operating system SunOS 4.1.4.

cus. no.	cr. rate	*crossover operators*				optimal value
		OX	UOX	MX1	MX2	
10	.45	1.117	1.000	1.000	1.000	811
	.60	-	1.000	1.000	1.000	
	.75	-	1.000	1.000	1.000	
20	.45	-	1.015	1.022	-	1926
	.60	-	1.010	1.018	-	
	.75	-	1.000	1.018	-	
30	.45	-	-	1.008	-	2785
	.60	-	1.000	1.008	-	
	.75	-	1.000	1.008	-	
40	.45	-	-	1.000	-	4063
	.60	-	-	1.000	-	
	.75	-	1.000	1.000	-	

Table 2: Relative optimality of the solutions

Table 2 describes the ratios of best costs (found in 30 runs) to the optimal cost of the 10, 20, 30 and 40 customer problems. The solution cost is based on Equation (1) where $w_1 = w_2 = 1$. A dynamic programming algorithm[13] produced the optimal solution of our single-vehicle PDPTW. The optimal value of 10, 20, 30 and 40 customers problems are listed in the last column. A '-' means that all solutions have been found (in 30 runs) are infeasible. That is, at least one of capacity and time window constraints is violated.

According to Table 2, OX is not a good operator for the single-vehicle PDPTW on the three crossover rates, neither is MX2. OX creates offspring which inherits subtours of its parents. Such a new route is analogous to its parent routes. The solutions were traped to similar routes, and therefore feasible solutions may not be found. As to MX2, a point with the lowerest precedence is moved to near the end of the routes, and the solution may be stuck to a local minimum. Higher crossover rates are more suitable for UOX, especially on larger problems. Only MX1 obtains feasible solutions in all of the four problems.

crossover rate	*crossover operators*			
	OX	UOX	MX1	MX2
0.45	-	-	4904	-
0.60	-	-	4909	-
0.75	-	4867	4902	-

Table 3: Solution cost of 50 customers problem

Values in Table 3 are solution costs, and each of them is the minimal cost in 30 runs. Most of exact algorithms applied to vehicle routing problems can not solve relatively larger problems, and we can not obtain the optimal values either. Hence, the optimal values are not listed in this table. UOX on crossover rate 0.75 obtains the minimal cost in Table 3, while MX1 can find feasible solutions on all the three crossover rates. OX and MX2 can not produce any feasible solution.

Figure 1 shown the convergent behavior of UOX. Offsprings of OX and UOX both keep subtours of their parents. However, offsprings of OX tend to reserve only a subtour, which is relatively longer, of one parent, but offsprings of UOX are composed of several shorter subtours from both of their parents. UOX enables new routes to find other feasible solutions. For UOX, it is possible that the higher crossover rates will lead to the more feasible solutions.

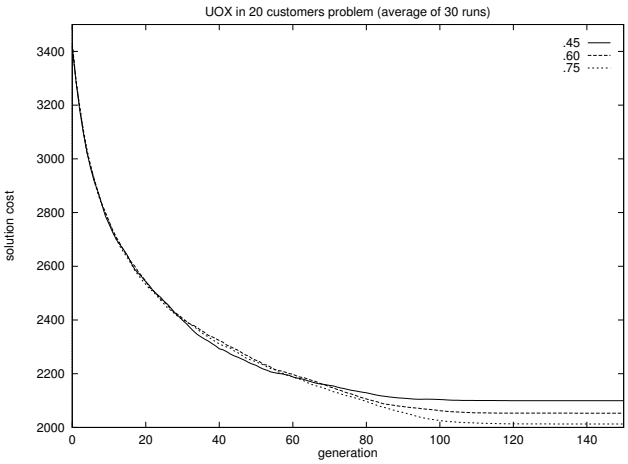

Figure 1: A comparison of crossover rate (UOX)

In Figure 2, MX1 converges faster than the other

operators. MX1 produces a new route which is closer to the order of the global precedence vector. The new route has a great possibility that it can be a feasible solution. It is why MX1 alway produces feasible solutions. After operation of MX1, more feasible solutions are produced and the average cost drops down dramatically. So MX1 converges faster.

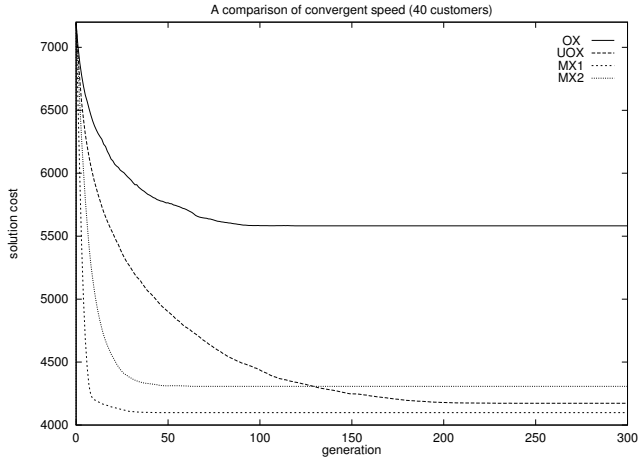

Figure 2: A comparison of convergent speed

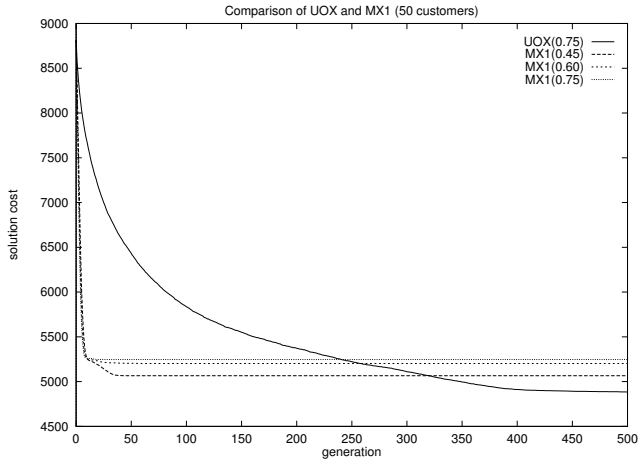

Figure 3: A comparison of UOX and MX1

Since UOX on 0.75 crossover rate and MX1 on the three crossover rates obtain feasible solution of our single-vehicle PDPTW. We make a comparison of these two operators in Figure 3.

From Figure 3, it is clear that the convergence speed of UOX is slower than that of MX1. If the computation time is the major concern, MX1 will be a good operator. The near-optimal or optimal solutions can be produced by MX1, and this operator converges fast.

5 Conclusion

In this paper, a genetic algorithm to solve the vehicle routing problem is proposed, and four crossover operators are compared. Genetic algorithms can solve the single-vehicle PDPTW well if the right genetic operators are employed. From the experiments, it is clear that UOX and MX1 are more suitable for our single-vehicle PDPTW problem, but OX and MX2 are not. We can lay more emphasis on UOX and MX1 in the future research because they outperform OX and MX2 overwhelmingly in almost all our experiments. However, since MX1 converges more rapidly than UOX, if the computation time is mainly concerned, MX1 will be a better choice.

Further experiments on UOX and MX1 with problems of different sizes should be done in the future to understand the relative performance of UOX and MX1 with various problem sizes. The appropriate crossover rate is another important subject for research, which is dependent on different problems and different crossover operators. Additionally, we don't apply mutation operators in our work, and it is also another research direction. The performance of some of the crossover operators may be improved if mutation operators are applied.

References

[1] T. Blickle and L. Thiele. A mathematical analysis of tournament selection. In *Proceedings of The Sixth International Conference on Genetic Algorithms*, pages 9–16, 1995.

[2] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6:80–91, 1959.

[3] L. Davis, editor. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, New York, 1991.

[4] M. Gendreau, G. Laporte, and R. Séguin. Stochastic vehicle routing. *European Journal of Operational Research*, 88:3–12, 1996.

[5] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, Inc., Reading, MA, USA, 1989.

[6] B. L. Golden and A. A. Assad, editors. *Vehicle Routing: Methods and Studies*. North-Holland, Amsterdam, 1988.

[7] G. R. Harik. Finding multimodal solutions using restricted tournament selection. In *Proceedings of*

The Sixth International Conference on Genetic Algorithms, pages 24–31, 1995.

[8] J. L. Blanton Jr. and R. L. Wainwright. Multiple vehicle routing with time and capacity constraints using genetic algorithms. In *Proceedings of The Fifth International Conference on Genetic Algorithms*, pages 452–459, 1993.

[9] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:345–358, 1992.

[10] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, New York, USA ; Berlin, 1992.

[11] M. Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, Cambridge, Massachusetts; London, England, 1996.

[12] I. M. Oliver, D. J. Smith, and J. R. C. Holland. A study of permutation crossover operators on the traveling salesman problem. In *Proceedings of The Second International Conference on Genetic Algorithms*, pages 224–230, 1987.

[13] Harilaos N. Psaraftis. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154, 1980.

[14] T. Starkweather, S. McDaniel, K. Mathias, D. Whitley, and C. whitley. A comparison of genetic sequencing operators. In *Proceedings of The Fourth International Conference on Genetic Algorithms*, pages 69–76, 1991.