EVOLUTIONARY SCHEDULING AND ROUTING

Edmund Burke, chair

Balance between Genetic Search and Local Search in Hybrid Evolutionary Multi-Criterion Optimization Algorithms

Hisao Ishibuchi
Dept. of Industrial Engineering
Osaka Prefecture University
Sakai, Osaka 599-8531, JAPAN
hisaoi@ie.osakafu-u.ac.jp
Phone: +81-72-254-9350

Tadashi Yoshida
Dept. of Industrial Engineering
Osaka Prefecture University
Sakai, Osaka 599-8531, JAPAN
yossy@ie.osakafu-u.ac.jp
Phone: +81-72-254-9351

Tadahiko Murata
Dept. of Informatics
Kansai University
Takatsuki, Osaka 569-1095, JAPAN
murata@res.kutc.kansai-u.ac.jp
Phone: +81-726-90-2429

Abstract

The aim of this paper is to clearly demonstrate the importance of finding a good balance between genetic search and local search in the implementation of hybrid evolutionary multi-criterion optimization (EMO) algorithms. We first modify the local search part of an existing multi-objective genetic local search (MOGLS) algorithm. In the modified MOGLS algorithm, the computation time spent by local search can be decreased by two tricks: to apply local search to only selected solutions (not all solutions) and to terminate local search before all neighbors of the current solution are examined. Next we show that the local search part of the modified MOGLS algorithm can be combined with other EMO algorithms. We implement a hybrid version of a strength Pareto evolutionary algorithm (SPEA). Using the modified MOGLS algorithm and the hybrid SPEA algorithm, we examine the balance between genetic search and local search through computer simulations on a two-objective flowshop scheduling problem. Computer simulations are performed using various specifications of parameter values that control the computation time spent by local search.

1. INTRODUCTION

One promising trick for improving the search ability of evolutionary multi-criterion optimization (EMO) algorithms is the hybridization with local search. Such a hybrid EMO algorithm was first implemented as a multi-objective genetic local search (MOGLS) algorithm in Ishibuchi & Murata (1996) together with a simple idea of elitism. The MOGLS algorithm was successfully applied to multi-objective flowshop scheduling problems in Ishibuchi & Murata (1998). While their MOGLS algorithm implemented two promising tricks for improving the search ability of EMO algorithms (i.e., hybridization and elitism), its search ability is not high if compared with recently proposed EMO algorithms such as a strength Pareto evolutionary algorithm (SPEA) of Zitzler & Thiele (1999) and a revised non-dominated sorting genetic algorithm (NSGA-II) of Deb et al. (2000).

Jaszkiewicz (1998) and Jaszkiewicz et al. (2001) improved the performance of the MOGLS algorithm by modifying its selection mechanism of parent solutions. While his MOGLS algorithm uses a scalar fitness function with random weight values for selection and local search as in the original MOGLS algorithm in Ishibuchi & Murata (1996), it does not use the roulette wheel selection. A pair of parent solutions is randomly selected from a pre-specified number of the best solutions (i.e., a kind of subpopulation) with respect to the scalar fitness function with the current weight values. The weight values are randomly updated whenever a pair of parent solutions is selected as in the original MOGLS algorithm. In the above-mentioned two MOGLS algorithms, local search is applied to all solutions generated by genetic operations in every generation. In some hybrid EMO algorithms, local search is used only when the execution of EMO algorithms is terminated. Deb & Goel (2001) applied local search to final solutions obtained by EMO algorithms for decreasing the number of non-dominated solutions (i.e., for decreasing the variety of final solutions). On the other hand, Talbi (2001) intended to increase the variety of final solutions by the application of local search.

The performance of the original MOGLS algorithm in Ishibuchi & Murata (1996) can be improved by carefully addressing the following issues:

Choice of initial solutions for local search: Local search was applied to all solutions in the current population in the original MOGLS algorithm. Its performance can be improved by choosing only good solutions from the current population as initial solutions for local search.

Specification of local search directions: The local search direction for each solution was specified by the scalar fitness function used in the selection of its parent solutions in the original MOGLS algorithm. Its performance can be improved by specifying an appropriate local search direction for each solution independent of the scalar fitness function used in the selection of its parent solutions.

Balance between genetic search and local search: If we simply combine local search with EMO algorithms, almost all the available computation time is spent by local search. This is because a large number of solutions are usually examined by local search for a single initial solution until a locally optimal solution is found. As a result, the global search ability of EMO algorithms is deteriorated by the hybridization with local search. In the original MOGLS algorithm, the balance between genetic search and local search was controlled by the number of neighbors examined by local search around the current solution. Local search was terminated if a better solution was not found among a pre-specified number of neighbors examined around the current solution. The balance can be also controlled by the number of solutions in the current population to which local search is applied. The performance of the original MOGLS algorithm can be improved by finding a good balance between genetic search and local search.

In this paper, first we briefly discuss the first two issues: choice of initial solutions for local search and specification of a local search direction for each initial solution. Then the balance between genetic search and local search is discussed through computer simulations on a two-objective flowshop scheduling problem.

2. MULTI-CRITERION OPTIMIZATION

Let us consider the following n-objective minimization problem:

$$\text{Minimize } \boldsymbol{z} = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x})), \quad (1)$$

$$\text{subject to } \boldsymbol{x} \in \boldsymbol{X}, \quad (2)$$

where $\boldsymbol{z}$ is the objective vector, $\boldsymbol{x}$ is the decision vector, and $\boldsymbol{X}$ is the feasible region in the decision space. Usually, there is no optimal solution $\boldsymbol{x}^*$ that satisfies the following inequality condition:

$$f_i(\boldsymbol{x}^*) \le f_i(\boldsymbol{x}) \text{ for } \forall i \in \{1, 2, \ldots, n\} \text{ and } \forall \boldsymbol{x} \in \boldsymbol{X}. \quad (3)$$

Thus the task of EMO algorithms is not to find a single final solution but to find all solutions that are not dominated by any other solutions. Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be two decision vectors ($\boldsymbol{a}, \boldsymbol{b} \in \boldsymbol{X}$). Then $\boldsymbol{b}$ is said to be dominated by $\boldsymbol{a}$ (i.e., $\boldsymbol{a} \prec \boldsymbol{b}$) if and only if the following two conditions hold:

$$f_i(\boldsymbol{a}) \le f_i(\boldsymbol{b}) \text{ for } \forall i \in \{1, 2, \ldots, n\}, \quad (4)$$

$$f_i(\boldsymbol{a}) < f_i(\boldsymbol{b}) \text{ for } \exists i \in \{1, 2, \ldots, n\}. \quad (5)$$

When $\boldsymbol{b}$ is not dominated by any other solutions in $\boldsymbol{X}$, $\boldsymbol{b}$ is said to be a Pareto-optimal solution. That is, $\boldsymbol{b}$ is a Pareto-optimal solution when there is no solution $\boldsymbol{a}$ in $\boldsymbol{X}$ that satisfies the above two conditions.

While the task of EMO algorithms is to find all Pareto-optimal solutions, it is impractical to try to find true Pareto-optimal solutions of large problems. Thus EMO algorithms usually present non-dominated solutions among examined ones to decision makers as a result of their execution. In this case, the task of EMO algorithms is to drive populations to true Pareto-optimal solutions as close as possible.

3. HYBRID EMO ALGORITHMS

3.1 MOGLS ALGORITHM

In the original MOGLS algorithm, local search is applied to all solutions in every generation. The following scalar fitness function was used for both the selection of a pair of parent solutions and the local search for their offspring.

$$f(\boldsymbol{x}) = w_1 f_1(\boldsymbol{x}) + w_2 f_2(\boldsymbol{x}) + \cdots + w_n f_n(\boldsymbol{x}), \quad (6)$$

where w_i is a non-negative weight. The point is to randomly specify the weight values whenever a pair of parent solutions is selected. This weight specification mechanism generates various search directions in the n-dimensional objective space. The MOGLS algorithm also uses a kind of elitism where all non-dominated solutions obtained during its execution are stored as a secondary population separately from the current population. A few non-dominated solutions are randomly selected from the secondary population and their copies are added to the current population.

The main characteristic feature of local search in the MOGLS algorithm is that all neighbors of the current solution are not examined. For decreasing the computation time spent by local search, only k neighbors of the current solution are randomly chosen and examined. If no better solution is found among the examined k neighbors, local search for the current solution is terminated. The first move strategy is used in local search. That is, the current solution is replaced as soon as a better neighbor is found.

Figure 1 shows the general outline of hybrid EMO algorithms discussed in this paper. In hybrid EMO algorithms, a new population is generated by genetic operations in the EMO algorithm part. Then the new population is improved by local search. The improved population is handled as the current population in the

EMO algorithm part. In this manner, the population update is iterated by genetic operations and local search until a pre-specified stopping condition is satisfied.

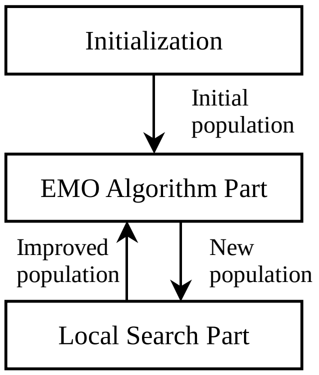


Figure 1: Outline of hybrid EMO algorithms.

3.2 MODIFICATION OF LOCAL SEARCH PART

In the original MOGLS algorithm, local search was applied to all solutions in the current population. The drawback of this scheme is computational inefficiency. That is, the application of local search to poor solutions seems to be mere waste of computation time. The efficiency of the original MOGLS algorithm can be improved by applying local search to only good solutions in the current population. The local search direction for each solution was specified by the scalar fitness function used in the selection of its parents. The drawback of this scheme is that the local search direction for each solution is not always appropriate.

As a remedy for these two drawbacks, we modify the local search part of the original MOGLS algorithm as follows:

[Modified Local Search Part]

Step 1. Iterate the following two procedures for constructing a local search pool of N_{pop} solutions:
(a) Randomly specify the weight values $w_1, ..., w_n$.
(b) Select a solution to be included in the local search pool from the current population (i.e., new population generated by genetic operations in Fig. 1) using the size four tournament selection with replacement based on the scalar fitness function with the current weight values specified in (a). That is, four solutions are randomly selected from the current population and a copy of the best one is added to the local search pool. The four solutions are returned to the current population for further selection to construct the local search pool.

Step 2. Randomly select N_{LS} solutions from the local search pool without replacement. Local search is applied to only the selected N_{LS} solutions. The local search direction of each solution is specified by the weight values used in the selection of that solution for constructing the local search pool. The next population consists of the improved N_{LS} solutions and the other $(N_{\text{pop}} - N_{\text{LS}})$ solutions in the local search pool.

3.3 HYBRIDIZATION WITH EMO ALGORITHMS

In the modified local search part, the local search direction of each solution is not inherited from its parent solutions. The local search direction is specified in the local search part independent of the EMO algorithm part in Fig. 1. Thus the modified local search part can be combined with any EMO algorithms even if they do not use the scalar fitness function in (6) for the selection of parent solutions. As shown in Fig. 1, the local search part of hybrid EMO algorithms receives a new population updated in the EMO algorithm part and returns an improved population by local search. It is an advantage of the modified MOGLS algorithm over the original one that the modified local search part can be combined as a module with any EMO algorithms.

In this paper, we examine the original MOGLS algorithm and its modified version. We also examine a hybrid version of the SPEA algorithm because high search ability of the SPEA algorithm to find Pareto-optimal solutions has been reported in the literature (see Zitzler & Thiele 1999 and Zitzler et al. 2000).

4. COMPUTER SIMULATIONS

4.1 EFFECT OF MODIFICATION OF MOGLS

We examined the effect of the modification of the local search part on the performance of the MOGLS algorithm. In the same manner as in Ishibuchi & Murata (1998), we generated a 40-job and 20-machine flowshop scheduling problem with two objectives: to minimize the makespan and to minimize the maximum tardiness. We applied the original MOGLS algorithm and its modified version to this test problem. Each algorithm was terminated when 60000 solutions were examined. As in Ishibuchi & Murata (1998), we used the position-based two-point crossover and the shift mutation as genetic operations. The neighborhood structure was defined by the shift mutation in local search.

We used the following parameter specifications. Population size: 20, crossover probability: 0.9, mutation probability for each string: 0.3, the number of elite solutions: 3, the number of neighbors examined for improving the current solution in local search (i.e., k): 2. These specifications are almost the same as Ishibuchi & Murata (1998). In the modified MOGLS algorithm, the tournament size was specified as four for constructing the local search pool from the current population. The number of selected initial solutions for local search was specified as $N_{\mathrm{LS}} = 20$ (i.e., the same as the population size).

Each algorithm was applied to the test problem 150 times. Fig. 2 and Fig. 3 show all solutions obtained by the 150 runs of each algorithm. From the comparison between Fig. 2 and Fig. 3, we can see that the modified MOGLS algorithm in Fig. 3 outperformed the original one in Fig. 2. That is, the performance of the original MOGLS algorithm was improved by the modification of the local search part.

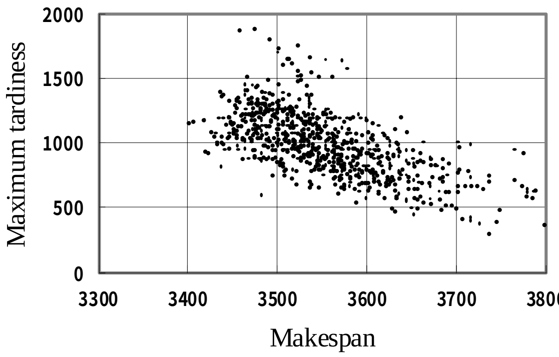


Figure 2: Obtained solutions by 150 runs of the original MOGLS algorithm.

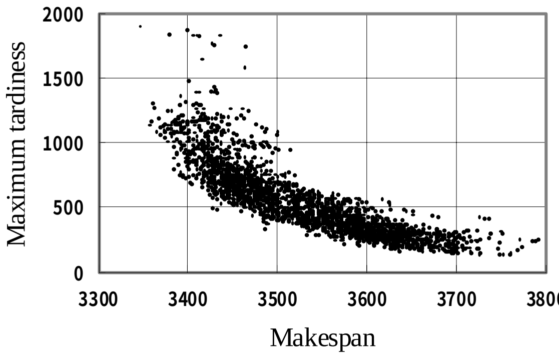


Figure 3: Obtained solutions by 150 runs of the modified MOGLS algorithm. Only the choice of initial solutions for local search is different from the original MOGLS algorithm.

4.2 EFFECT OF LOCAL SEARCH

In the same manner as in the previous subsection, we examined the effect of the hybridization with local search on the performance of EMO algorithms. In Fig. 4, we show simulation results by a simple EMO algorithm implemented by removing the local search part from the original MOGLS algorithm. Since the performance of this algorithm was very poor, many solutions are out of the range of Fig. 4. From the comparison of Fig. 4 with Fig. 2 and Fig. 3, we can see that the hybridization with local search significantly improved the performance of the simple EMO algorithm.

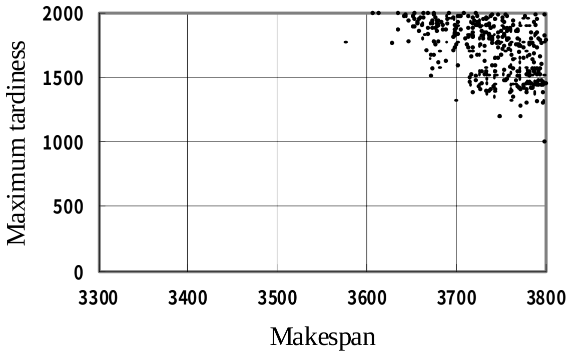


Figure 4: Obtained solutions by 150 runs of the original MOGLS algorithm with no local search. Many non-dominated solutions are out of the range of this figure.

While the comparison between Fig. 3 and Fig. 4 clearly shows that the simple EMO algorithm was significantly improved by the hybridization with local search, one may think that the improvement is mainly due to the poor performance of the simple EMO algorithm in Fig. 4. So we also implemented a hybrid version of the SPEA in the same manner as the modified MOGLS algorithm. Simulation results are summarized in Fig. 5 and Fig. 6. The maximum number of stored non-dominated solutions was specified as 20 in the computer simulations. The other parameters were specified in the same manner as in the previous subsection.

From the comparison between Fig. 5 and Fig. 6, we can see that the performance of the SPEA was slightly improved by the hybridization with local search. For example, more solutions were obtained in the region $[3300, 3400]\times[500, 1500]$ of Fig. 6 by the hybrid SPEA than the original SPEA in Fig. 5.

For further examining the effect of the hybridization with local search on the performance of the SPEA, a solution set obtained by the SPEA was compared with another solution set obtained by the hybrid SPEA. In this comparison, solutions obtained by one algorithm were examined whether they were dominated by other solutions obtained by the other algorithm. This comparison was performed over 150 runs of these two algorithms. Then the average number of non-dominated solutions was calculated. Simulation results are summarized in Table 1. This table shows the average number of obtained solutions by each algorithm, the average number of solutions that were not dominated by other solutions obtained by the other algorithm, the ratio of non-dominated solutions to obtained solutions, and the average CPU time. From this table, we can see that the hybrid SPEA outperformed the original SPEA in terms of the ratio of non-dominated solutions. We can also see from Table 1 that the CPU time was decreased by the hybridization with local search. This is because local search can be executed more efficiently than genetic search. If these two algorithms are compared under the same CPU time, it is more clearly shown that the hybrid SPEA outperforms the original SPEA (compare Fig. 7 with Fig. 5).

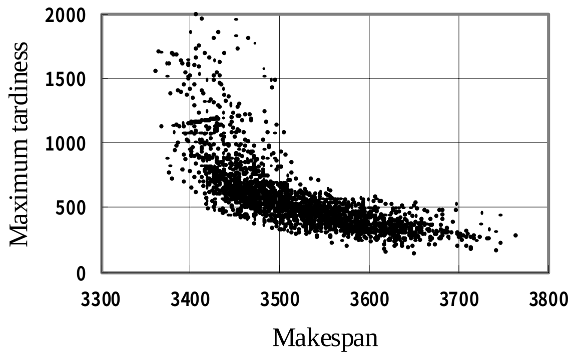


Figure 5: Obtained solutions by 150 runs of the original SPEA.

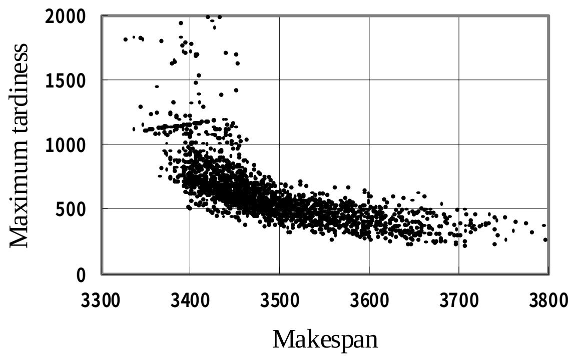


Figure 6: Obtained solutions by 150 runs of the hybrid SPEA.

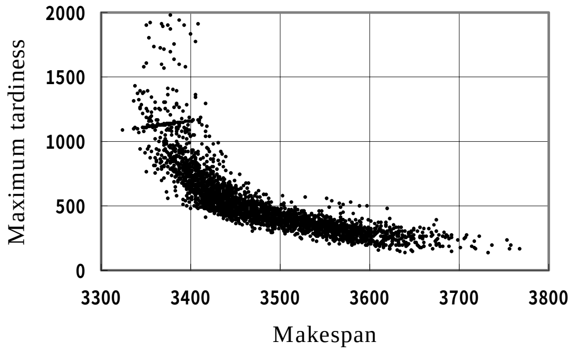


Figure 7: Obtained solutions by 150 runs of the hybrid SPEA with $k = 3$ using the same CPU time as the original SPEA.

Table 1: Comparison between SPEA and its hybrid version.

Algorithm	Obtained solutions	Non-dominated	Ratio of non-dominated	CPU time (Sec.)
SPEA	17.34	10.08	58.13%	17.47
Hybrid	14.69	9.85	67.05%	13.26

4.3 BALANCE BETWEEN GENETIC SEARCH AND LOCAL SEARCH IN THE MODIFIED MOGLS

The performance of hybrid EMO algorithms depends on parameter specifications. The point is to find a good balance between genetic search and local search. The balance is controlled by two parameters *k* and N_{LS} in the modified MOGLS algorithm (*k*: the number of neighbors examined for improving the current solution by local search, N_{LS}: the number of solutions in each population to which local search is applied). Table 2 shows simulation results with various values of *k*. In this table, N_{LS} was specified as $N_{\text{LS}} = 20$. Good results were not obtained from large values of *k* (see the column labeled as "Non-dominated"). Good specifications of *k* in Table 2 are $k = 1$~5.

Table 2: Simulation results by the modified MOGLS algorithm with various values of *k*. The value of N_{LS} was specified as $N_{\text{LS}} = 20$. Good results are highlighted by boldface letters. "Generation updates" means the number of generations.

k	Generation updates	Obtained solutions	Non-dominated	CPU time (seconds)
0	3000.00	17.78	1.08	19.81
1	1490.56	18.09	**4.29**	14.75
2	952.24	17.01	**4.60**	13.21
3	696.80	16.90	**3.41**	12.46
4	543.65	16.99	**3.44**	12.10
5	440.84	17.08	**3.13**	11.78
10	212.56	17.78	1.99	10.84
20	92.43	17.03	1.57	10.40
30	55.25	17.19	1.41	10.25
40	37.90	16.88	1.02	10.19
50	28.39	17.03	1.36	10.16
100	11.83	14.91	1.08	10.09
1521	1.00	5.28	1.36	13.75

On the other hand, Table 3 shows simulation results with various values of N_{LS}. In this table, *k* was specified as $k = 2$. Good results were not obtained from small values of N_{LS}. Good specifications of N_{LS} in Table 3 are $N_{\text{LS}} = 8$~20.

For further examining the balance between genetic search and local search, we examined various combinations of *k* and N_{LS}. A solution set obtained from each combination was compared with other solution sets obtained from other combinations in the same manner as in the previous computer simulations. Simulation results are summarized in Table 4. This table shows the number of solutions that were not dominated by any other solutions obtained from other parameter specifications. Table 4 shows average results over 150 trials as in the previous computer simulations. From this table, we can see that appropriate specifications of N_{LS} and k are related to each other. An appropriate value of N_{LS} decreases as the specified value of k increases in Table 4. In general, larger values of these two parameters mean longer computation time spent by local search. Thus the increase of one parameter value needs the decreases of the other parameter value for keeping a good balance between genetic search and local search.

Table 3: Simulation results by the modified MOGLS algorithm with various values of N_{LS}. The value of k was specified as $k = 2$. Good results are highlighted by boldface letters.

N_{LS}	Generation updates	Obtained solutions	Non-dominated	CPU time (seconds)
0	3000.00	17.35	0.77	18.90
2	2467.09	17.37	1.46	17.26
4	2094.59	17.77	2.12	16.26
6	1819.94	17.93	2.56	15.53
8	1611.89	16.93	**3.33**	14.92
10	1444.13	16.81	**3.11**	14.49
12	1306.81	17.70	2.71	14.14
14	1193.71	17.51	**3.27**	13.85
16	1100.39	17.27	2.93	13.59
18	1019.59	17.58	**3.09**	13.37
20	952.24	17.01	**3.62**	13.21

Table 4: The average number of solutions that were not dominated by any other solutions from other combinations of parameter values. The execution of the modified MOGLS algorithm was terminated when 60000 solutions were examined. Good results are highlighted by boldface letters.

N_{LS}	The value of k										
	0	1	2	3	4	5	10	20	30	40	50
0	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17
2	0.17	0.06	0.33	0.43	0.45	**0.67**	**0.58**	0.35	**0.58**	**0.51**	0.47
4	0.17	0.39	0.45	0.41	**0.59**	**0.55**	**0.73**	0.24	0.40	0.31	0.37
6	0.17	0.35	**0.55**	**0.51**	**0.51**	**0.81**	**0.50**	0.29	0.32	0.29	0.23
8	0.17	0.48	**0.83**	**0.61**	**0.65**	0.37	**0.52**	0.27	0.11	0.25	0.17
10	0.17	**0.50**	**0.66**	0.39	**0.64**	**0.59**	0.31	0.21	0.21	0.20	0.19
12	0.17	0.49	0.47	**0.57**	**0.57**	**0.64**	0.30	0.17	0.16	0.13	0.13
14	0.17	**0.75**	**0.62**	0.42	0.42	0.46	0.31	0.25	0.17	0.15	0.09
16	0.17	**0.61**	**0.74**	**0.53**	**0.67**	0.29	0.21	0.12	0.11	0.06	0.13
18	0.17	**0.68**	**0.58**	**0.65**	0.41	0.42	0.35	0.15	0.09	0.04	0.04
20	0.17	**0.59**	**0.61**	0.35	0.45	0.37	0.32	0.15	0.09	0.09	0.07

We also performed the same computer simulation using different specifications of the stopping condition. In one specification, we decreased the available computation time from the examination of 60000 solutions to 20000 solutions. Simulation results are shown in Table 5. In the other specification, it was increased to 120000 solutions. Simulation results are summarized in Table 6. Table 5 and Table 6 show that appropriate values of N_{LS} and k are related to each other as in Table 4. From the comparison among the three tables, we can see that larger values of k can be used when the available computation resource is larger (i.e., Table 6). This means that we can use a larger portion of the computation time for local search when the available computation time is longer.

Table 5: The average number of solutions that were not dominated by any other solutions from other combinations of parameter values. The execution of the modified MOGLS algorithm was terminated when 20000 solutions were examined. Good results are highlighted by boldface letters.

N_{LS}	The value of k										
	0	1	2	3	4	5	10	20	30	40	50
0	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
2	0.02	0.04	0.37	0.39	**0.51**	**0.52**	0.44	**0.63**	**0.68**	0.49	**0.92**
4	0.02	0.11	0.23	0.47	0.48	**0.57**	0.49	0.37	0.39	0.43	0.39
6	0.02	0.35	0.45	0.49	0.43	**0.51**	0.39	0.33	0.30	0.35	0.23
8	0.02	0.18	**0.71**	**0.54**	0.35	0.23	0.24	0.25	0.22	0.18	0.29
10	0.02	0.43	0.43	0.43	0.29	0.41	0.24	0.20	0.16	0.17	0.24
12	0.02	0.40	**0.52**	0.30	0.45	0.41	0.17	0.11	0.15	0.11	0.18
14	0.02	0.41	**0.52**	0.48	0.27	0.28	0.11	0.06	0.16	0.17	0.05
16	0.02	0.32	**0.53**	0.41	0.32	0.26	0.20	0.05	0.05	0.07	0.09
18	0.02	0.35	0.33	**0.53**	0.31	0.23	0.12	0.04	0.04	0.05	0.02
20	0.02	0.49	**0.53**	0.32	0.40	0.29	0.11	0.07	0.01	0.01	0.07

Table 6: The average number of solutions that were not dominated by any other solutions from other combinations of parameter values. The execution of the modified MOGLS algorithm was terminated when 120000 solutions were examined. Good results are highlighted by boldface letters.

N_{LS}	The value of k										
	0	1	2	3	4	5	10	20	30	40	50
0	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11
2	0.11	0.12	0.37	0.48	**0.65**	**0.57**	**0.52**	**0.52**	0.42	**0.61**	**0.51**
4	0.11	0.41	**0.61**	**0.52**	**0.61**	**0.57**	**0.53**	0.47	0.47	0.25	0.29
6	0.11	0.27	**0.70**	**0.55**	**0.54**	**0.73**	**0.51**	0.37	0.35	0.31	0.35
8	0.11	**0.51**	**0.54**	**0.65**	**0.59**	**0.55**	0.45	0.27	0.28	0.27	0.21
10	0.11	0.38	**0.51**	**0.51**	**0.71**	**0.77**	**0.54**	0.39	0.19	0.29	0.27
12	0.11	**0.53**	**0.85**	**0.77**	**0.51**	**0.73**	0.41	0.18	0.15	0.26	0.15
14	0.11	**0.71**	**0.53**	**0.69**	**0.56**	**0.55**	0.38	0.31	0.15	0.14	0.19
16	0.11	**0.56**	**0.62**	**0.69**	**0.73**	0.45	0.38	0.15	0.10	0.13	0.14
18	0.11	**0.61**	**0.53**	**0.72**	**0.58**	**0.53**	0.31	0.28	0.15	0.11	0.11
20	0.11	**0.69**	**0.57**	**0.70**	**0.74**	**0.60**	0.11	0.13	0.16	0.13	0.16

4.4 BALANCE BETWEEN GENETIC SEARCH AND LOCAL SEARCH IN THE HYBRID SPEA

We also examined the balance between genetic search and local search using the hybrid SPEA in the same manner as in the previous subsection. Table 7 shows simulation results by the hybrid SPEA with $N_{\mathrm{LS}} = 20$ and various values of *k*. Good results were not obtained from large values of *k*. Good specifications of *k* in Table 7 are $k = 2$~5. On the other hand, Table 8 shows simulation results by the hybrid SPEA with $k = 2$ and various values of N_{LS}. Good results were not obtained from small values of N_{LS}. Good specifications of N_{LS} in Table 8 are $N_{\mathrm{LS}} = 18$~20. We can also see that the simulation results by the hybrid SPEA in Table 7 and Table 8 are similar to those by the modified MOGLS algorithm in Table 2 and Table 3, respectively.

We also examined various combinations of *k* and N_{LS} using the hybrid SPEA. Simulation results are summarized in Table 9 ~ Table 11. As in the previous subsection, these tables show simulation results using different stopping conditions. From these tables, we can see that appropriate specifications of N_{LS} and *k* are related to each other. An appropriate value of N_{LS} decreases as the specified value of *k* increases. From the comparison between the simulation results in this subsection by the hybrid SPEA and those in the previous subsection by the modified MOGLS algorithm, we can see that appropriate specifications of N_{LS} and *k* depend on the algorithm. For example, appropriate values of *k* for the hybrid SPEA are larger than those for the modified MOGLS algorithm. This is observed from the comparison between Table 6 and Table 11.

Table 7: Simulation results by the hybrid SPEA with various values of *k*. The value of N_{LS} was specified as $N_{\mathrm{LS}} = 20$. Good results are highlighted by boldface letters.

k	Generation updates	Obtained solutions	Non-Dominated	CPU time (seconds)
0	3000.00	15.46	1.26	19.03
1	1483.57	15.69	2.91	14.91
2	976.18	14.69	**3.12**	13.26
3	719.59	15.85	**3.73**	12.59
4	565.81	15.43	**3.50**	12.21
5	462.69	15.35	**3.95**	11.99
10	227.53	15.68	2.93	10.95
20	97.62	16.69	1.77	10.47
30	57.25	16.68	1.62	10.31
40	38.53	16.76	1.29	10.23
50	28.95	16.38	1.15	10.19
100	11.58	15.35	1.11	10.13
1521	2.00	5.10	1.35	13.89

Table 8: Simulation results by the hybrid SPEA with various values of N_{LS}. The value of *k* was specified as $k = 2$. Good results are highlighted by boldface letters.

N_{LS}	Generation Updates	Obtained solutions	Non-Dominated	CPU time (seconds)
0	3000.00	15.46	0.78	19.03
2	2486.16	15.64	1.99	17.50
4	2123.08	15.13	1.99	16.41
6	1850.80	16.26	2.33	15.74
8	1641.54	15.63	2.49	15.06
10	1474.84	16.01	2.56	14.63
12	1338.12	15.45	2.53	14.28
14	1223.87	15.80	2.62	14.06
16	1128.85	15.39	2.70	13.77
18	1046.31	15.75	**3.19**	13.54
20	976.18	15.44	**3.23**	13.26

Table 9: The average number of solutions that were not dominated by any other solutions. The execution of the hybrid SPEA was terminated when 60000 solutions were examined. Good results are highlighted by boldface letters.

N_{LS}	The value of *k*										
	0	1	2	3	4	5	10	20	30	40	50
0	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07
2	0.07	0.09	0.29	0.23	0.38	0.37	**0.76**	**1.03**	**0.57**	**0.76**	**0.61**
4	0.07	0.25	0.37	0.29	0.37	0.29	**0.63**	**0.55**	**0.53**	**0.50**	0.35
6	0.07	0.29	0.49	0.34	0.46	**0.62**	0.48	**0.50**	0.39	0.33	0.33
8	0.07	0.29	0.31	0.49	0.26	**0.72**	0.49	0.30	0.41	0.41	0.35
10	0.07	0.23	0.43	0.31	**0.59**	**0.56**	**0.57**	0.29	0.41	0.22	0.21
12	0.07	0.32	0.35	0.41	0.33	**0.66**	**0.58**	0.49	0.15	0.18	0.16
14	0.07	0.19	0.37	0.47	0.49	**0.57**	**0.57**	0.37	0.19	0.17	0.12
16	0.07	0.32	0.31	0.44	**0.65**	0.39	0.37	0.39	0.22	0.06	0.05
18	0.07	0.31	0.34	0.44	**0.65**	**0.51**	0.33	0.23	0.14	0.17	0.07
20	0.07	0.24	**0.67**	0.43	0.41	0.34	0.33	0.12	0.15	0.08	0.08

Table 10: The average number of solutions that were not dominated by any other solutions. The execution of the hybrid SPEA was terminated when 20000 solutions were examined. Good results are highlighted by boldface letters.

N_{LS}	The value of *k*										
	0	1	2	3	4	5	10	20	30	40	50
0	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06
2	0.06	0.10	0.28	0.17	0.23	0.26	**0.61**	**0.68**	0.47	**0.69**	**0.72**
4	0.06	0.22	0.21	0.23	0.39	0.32	**0.73**	0.47	0.41	**0.53**	0.34
6	0.06	0.25	0.29	0.29	0.41	**0.56**	**0.58**	0.33	0.35	0.28	0.29
8	0.06	0.38	0.37	0.47	0.24	**0.65**	0.44	0.15	0.25	0.27	0.33
10	0.06	0.23	0.23	0.31	0.24	0.35	0.38	0.14	0.25	0.14	0.13
12	0.06	0.37	0.29	0.38	0.28	0.37	0.41	0.25	0.13	0.16	0.10
14	0.06	0.18	0.46	0.45	0.40	0.35	0.23	0.12	0.11	0.11	0.09
16	0.06	0.36	0.30	0.29	0.40	0.41	0.20	0.14	0.09	0.07	0.04
18	0.06	0.43	**0.51**	0.19	0.39	0.35	0.13	0.11	0.03	0.06	0.04
20	0.06	0.41	0.43	0.42	0.24	0.40	0.17	0.03	0.07	0.06	0.03

Table 11: The average number of solutions that were not dominated by any other solutions. The execution of the hybrid SPEA was terminated when 120000 solutions were examined. Good results are highlighted by boldface letters.

N_{LS}	The value of k										
	0	1	2	3	4	5	10	20	30	40	50
0	0.09	0.09	0.09	0.09	0.09	0.09	0.09	0.09	0.09	0.09	0.09
2	0.09	0.11	0.23	0.25	0.21	0.21	**0.63**	**0.62**	**0.65**	**0.77**	**0.56**
4	0.09	0.17	0.28	0.21	0.38	0.34	0.43	**0.61**	**0.65**	**0.53**	0.48
6	0.09	0.27	**0.57**	0.36	0.45	**0.57**	0.45	**0.57**	0.46	**0.57**	**0.52**
8	0.09	0.37	0.45	0.39	0.43	0.45	**0.57**	0.27	**0.51**	0.39	0.39
10	0.09	0.39	0.45	0.31	0.43	0.47	**0.54**	**0.51**	0.44	**0.51**	0.41
12	0.09	0.35	0.42	0.49	0.28	**0.54**	**0.53**	0.40	0.31	0.31	0.24
14	0.09	0.32	0.36	0.36	**0.56**	**0.53**	**0.69**	0.34	0.33	0.27	0.13
16	0.09	0.33	0.37	0.47	**0.67**	**0.56**	0.49	**0.55**	0.29	0.19	0.15
18	0.09	0.34	0.47	**0.58**	**0.58**	**0.54**	0.47	0.25	0.21	0.23	0.16
20	0.09	0.42	0.45	**0.59**	0.49	0.45	**0.50**	0.20	0.13	0.21	0.19

5. CONCLUSIONS

In this paper, we first modified the local search part of the MOGLS algorithm of Ishibuchi & Murata (1996) for applying local search only to good solutions in the current population and assigning an appropriate local search direction to each solution. The local search direction of each solution is specified in the modified local search part independent of genetic operations in the EMO algorithm part. Thus the modified local search part can be combined with other EMO algorithms. We implemented a hybrid SPEA by combining local search with the SPEA. Using the modified MOGLS algorithm and the hybrid SPEA, we examined the balance between genetic search and local search. Simulation results in this paper showed that the performance of the hybrid EMO algorithms strongly depends on this balance. When a good balance is achieved by appropriate parameter specifications, the hybrid EMO algorithms outperform the corresponding non-hybrid EMO algorithms.

It was also shown through computer simulations with different stopping conditions that appropriate parameter specifications for achieving a good balance between genetic search and local search depend on the amount of the available computation time. When long computation time was available, good results were obtained from parameter specifications that increase the ratio of the computation time spent by local search. On the other hand, good results were obtained in the case of a small ratio of the computation time spent by local search when we did not have long computation time. Simulation results also showed that different hybrid EMO algorithms require different parameter specifications for achieving a good balance. An appropriate ratio of the computation time spent by local search in the hybrid SPEA was larger than that in the modified MOGLS algorithm. Implication of this observation is not clear. One possible explanation is that genetic search in the hybrid SPEA may require shorter computation time than that in the modified MOGLS algorithm because the EMO algorithm part of the hybrid SPEA is more powerful than that of the modified MOGLS algorithm (compare Fig. 5 by the original non-hybrid SPEA with Fig. 4 by the EMO algorithm part in the modified MOGLS algorithm).

REFERENCES

K. Deb and T. Goel, “A hybrid multi-objective evolutionary approach to engineering shape design,” *Proc. of 1st International Conference on Evolutionary Multi-Criterion Optimization*, pp. 385-399, Zurich, Switzerland, March 7-9, 2001.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast elitist multi-objective genetic algorithm: NSGA-II,” *KanGAL Report* 200001, Indian Institute of Technology Kanpur, 2000.

H. Ishibuchi and T. Murata, “Multi-objective genetic local search algorithm,” *Proc. of 3rd IEEE International Conference on Evolutionary Computation*, pp. 119-124, 1996.

H. Ishibuchi and T. Murata, “A multi-objective genetic local search algorithm and its application to flowshop scheduling,” *IEEE Trans. on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, vol. 28, no. 3, pp. 392-403, 1998.

A. Jaszkiewicz, “Genetic local search for multiple objective combinatorial optimization,” *Working Paper*, RA-014/98, Poznan University of Technology, 1998.

A. Jaszkiewicz, M. Hapke, and P. Kominek, “Performance of multiple objective evolutionary algorithms on a distribution system design problem - Computational experiment,” *Proc. of 1st International Conference on Evolutionary Multi-Criterion Optimization*, pp. 241-255, 2001.

E. Talbi, M. Rahoual, M. H. Mabed, and C. Dhaenens, “A hybrid evolutionary approach for multicriteria optimization problems: Application to the flow shop,” *Proc. of 1st International Conference on Evolutionary Multi-Criterion Optimization*, pp. 416-428, Zurich, Switzerland, March 7-9, 2001.

E. Zitzler and L. Thiele, “Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach,” *IEEE Trans. on Evolutionary Computation*, vol. 3, no. 4, pp. 257-271, 1999.

E. Zitzler, K. Deb, and L. Thiele, “Comparison of Multiobjective Evolutionary Algorithms: Empirical Results,” *Evolutionary Computation*, vol. 8, no. 2, pp. 173-195, 2000.

A Hybrid Genetic Algorithm for the Vehicle Routing Problem with Time Windows

Soonchul Jung and Byung-Ro Moon
School of Computer Science and Engineering
Seoul National University
Seoul, 151-742 Korea
{samuel,moon}@soar.snu.ac.kr

Abstract

This paper suggests a new hybrid genetic algorithm for the 2D Euclidean vehicle routing problem with time windows. The natural crossover, proposed for the 2D Euclidean traveling salesman problem, was adopted with some modification in the suggested genetic algorithm. The most notable feature of the natural crossover is that it uses the 2D image of a solution itself for chromosomal cutting. We also investigate the usefulness of parents' decision variables such as arrival times during recombination. The suggested genetic algorithm found optimal solutions for 26 out of 31 instances with known optimal solutions.

1 Introduction

The vehicle routing problem (VRP) is the problem of finding a set of minimum-cost vehicle routes which start at a central depot, serve a set of customers with known demands, and return to the depot without any violation of constraints [6], [24]. There are several variants of VRP depending on their constraints. The vehicle routing problem with time windows (VRPTW) is an extension of VRP. In VRPTW, time-window constraints are added to the basic constrains of VRP. Each customer must be served only once by one vehicle, and the total demands of the customers served by a particular vehicle must not exceed the capacity of the vehicle. Each customer must be served within his/her time window. A vehicle must wait until the service is possible if the vehicle arrives at a customer earlier than the lower bound of his/her time window – the earliest arrival time. The depot also has a time window, and all the vehicles must return by the latest arrival time of the depot. In VRPTW, the objective can be the minimization of the travel distance, the travel time, the number of vehicles, or their combinations.

VRPTW has shown its usefulness in the area of distribution-related systems — school bus routing, newspaper delivery, garbage collection, fuel oil delivery, dial-a-ride service, etc. If a new routing plan of vehicles is more efficient than before, we can save fuel, money, and/or time.

Various algorithms for VRP and its variants have been studied intensively for decades. There are some of exact methods to solve VRPs and VRPTWs to the optimality [10], [12], [19], [21]. Although the speed-up techniques for exact methods have been introduced, the NP-hardness of VRPTW [27] still makes the required computational time prohibitive. Local heuristic methods often produce good near-optimal solutions in short computational time. They are divided into two classes: route construction heuristics [13], [11] and route improvement heuristics [7], [32]. Solomon [26] designed and reviewed several route construction heuristics. In [18], A number of route improvement heuristics are clearly described. Although these heuristics were able to run separately to solve VRPTW, they have also been incorporated in meta heuristics like tabu search, simulated annealing, genetic algorithm (GA), etc [4], [5], [30], [23], [15].

Blanton and Wainwright [2] introduced a genetic algorithm for VRPTW that used a sequence of customers as a chromosome. A greedy insertion-based heuristic interprets a chromosome (sequence), and calculates the fitness of the chromosome. The sequence means the insertion order of customers for the heuristic. GIDEON, suggested by Thangiah *et al.* [31], [30], is a framework for VRPTW, adopting a cluster-first route-second strategy. Their GA was used in the clustering phase. The chromosome represents angles whose origin is the depot, in order to define sectors to which customers will belong. Customers within a sector are assigned to one vehicle, and routed by the cheapest insertion method

[14]. In another system of Thangiah [29], a chromosome represents circles (by defining an origin and a radius). Customers within or near a circle are assigned to one vehicle. GENEROUS of Potvin and Bengio [23] uses a set of routes themselves as a chromosome. The crossover merges two parents heuristically, then the repair operator is applied to the offspring. Most approaches use a set of routes themselves as a chromosome after Potvin and Bengio's work [23]. Recently, Tan *et al.* [28] introduced a messy genetic algorithm that a chromosome is a sequence of (customer number, vehicle number) pairs.

The natural crossover, introduced by Jung and Moon [17], [16] is a crossover manipulating chromosomes in which genes are laid on a 2D space. Because 2D chromosomes can preserve problem information with less distortion, two-dimensional chromosomes are often used in genetic algorithms for 2D problems [8], [1], [3]. The natural crossover was originally devised for the 2D Euclidean traveling salesman problem (TSP), and produced better experimental results than state-of-the-art GAs for TSP [17]. In most VRPTW instances including Solomon's benchmark instances [26], customers are located on a 2D Euclidean space. Therefore, only if a 2D form of chromosomes are defined for VRPTW, it is possible for a genetic algorithm to use the natural crossover. This paper provides an extension of the natural crossover to VRPTW, and investigates its competence.

There are variables created during the evaluation of a route. Waiting times, arrival times, and travel-so-far distances are some of such decision variables. Almost all genetic algorithms for VRPTW did not utilize parents' decision variables in the course of recombination and mutation. In this paper, we utilize parents' decision variables during crossover.

The paper is organized as follows. Section 2 describes the mathematical formulation of VRPTW. Section 3 explains the genetic operators used in the proposed genetic algorithm. Section 4 presents the experimental results. Finally Section 5 makes conclusions.

2 Formulation of VRPTW

Table 1 represents the meanings of terms related to VRPTW. For a given route $R_k = (v_1, v_2, ..., v_m), v_1 = c_0$, decision variables are calculated as follows:

$$a_{v_i} = \begin{cases} 0, & i = 1 \\ tt_{v_{i-1}} + t_{v_{i-1}v_i}, & i > 1 \end{cases}$$

$$w_{v_i} = \begin{cases} 0, & i = 1 \\ \max(0, e_{v_i} - a_{v_i}), & i > 1 \end{cases}$$

$$tt_{v_i} = \begin{cases} 0, & i = 1 \\ a_{v_i} + w_{v_i} + s_{v_i}, & i > 1 \end{cases}$$

Table 1: Terminologies

constants	meaning
N	number of customers
Q	capacity of vehicles
C	set of all customers including the depot
c_i	customer i, $0 \le i \le N$ (c_0 is the depot.)
d_{ij}	distance from customer i to customer j
t_{ij}	travel time from customer i to customer j
q_i	demand of customer i
s_i	service time of customer i
e_i	earliest arrival time to customer i
l_i	latest arrival time to customer i
variables	**meaning**
$n(R)$	number of routes
$n(R_k)$	number of customers in route k
v_{jk}	j^{th} customer of route k
R_k	route $k = (v_{1k}, v_{2k}, ...)$
S_k	set of customers in route k
RD_k	travel distance of route k
RT_k	travel time of route k
RL_k	total load of route k
a_i	arrival time at c_i
al_i	adjusted latest arrival time at c_i
w_i	waiting time before servicing c_i
tt_i	travel-so-far time after servicing c_i
td_i	travel-so-far distance when arriving at c_i
ad_i	accumulated demands of customers after servicing c_i

$$td_{v_i} = \begin{cases} 0, & i = 1 \\ td_{v_{i-1}} + d_{v_{i-1}v_i}, & i > 1 \end{cases}$$

$$ad_{v_i} = \begin{cases} 0, & i = 1 \\ ad_{v_{i-1}} + q_{v_i}, & i > 1 \end{cases}$$

$$RT_k = tt_{v_m} + t_{v_m v_1}$$

$$RD_k = td_{v_m} + d_{v_m v_1}$$

$$RL_k = ad_{v_m}$$

The objective of our genetic algorithm is to find a set of routes having the minimal travel distance. In other words, we have to minimize

$$\sum_{k=1}^{n(R)} RD_k$$

subject to

$$S_1 \cup S_2 \cup \cdots \cup S_{n(R)} = C, \quad (1)$$

$$S_i \cap S_j = \{c_0\}, \quad 1 \le i, j \le n(R), i \ne j \quad (2)$$

$$v_{ik} \ne v_{jk}, \quad 1 \le i, j \le n(R_k), i \ne j \quad (3)$$

$$a_i \le l_i, \quad 1 \le i \le N \quad (4)$$

```
GA()
{
    initialize population $P$ of size $N$;
    while ( stopping condition is unsatisfied ) {
        select $parent_1$ and $parent_2$ from $P$;
        offspring ← crossover($parent_1, parent_2$);
        if ( random number is larger than mutation rate )
            mutate offspring;
        local-optimize offspring;
        replace an individual in $P$ with offspring;
    }
    return the best individual;
}
```

Figure 1: A typical steady-state hybrid genetic algorithm

$$RT_k \le l_0, \qquad 1 \le k \le n(R) \qquad (5)$$
$$RL_k \le Q, \qquad 1 \le k \le n(R). \qquad (6)$$

Restriction (1) ensures that all the customers are necessarily visited. Restriction (2) means that all the customers must be partitioned disjoint, and the depot is included in all routes. Restriction (3) ensures that every customer is visited only once. Restriction (4) and (5) take care of time constraints. Restriction (6) prevents the overload of vehicles.

3 Hybrid Genetic Algorithm

We use a typical steady-state hybrid genetic algorithm (Figure 1). Local optimization algorithms help GAs fine-tuning around local optima. The following subsections describe our genetic algorithm in detail.

3.1 Initialization of Population

A lot of route construction heuristics have been proposed for VRPTW. In [26], several heuristics are carefully designed and compared with one another. According to [26], the insertion heuristic 1 (I1) overall beat other route construction heuristics such as savings, nearest neighbor, etc. The core part of I1 is the routine inserting a new unrouted customer into the current route, between two adjacent customers on the route. If there is no feasible customer to insert, a new route is created. I1 repeats the loop until there are no unrouted customers.

We create a population of solutions using a stochastic version of I1. The existence of adjustable weights in the cost function of I1 makes the creation of various solutions possible. In calculating an insertion cost of a customer, the weights determine the balance between the spatial aspect and the temporal aspect of the problem instance. The values of weights are changed at random in the ranges of $\mu \in [0.1, 0.9], \lambda \in [0, 1], \alpha_1 \in [0.1, 0.9]$, and $\alpha_2 \in [0.1, 0.9] (\alpha_1 + \alpha_2 = 1)$. This is expected to be helpful in creating a robust population whose solution qualities do not depend on specific aspects of the problem instance. The first customer for a new route is chosen at random among the farthest unrouted customer, the unrouted customer with the earliest deadline, and a random unrouted customer.

3.2 Selection and Crossover

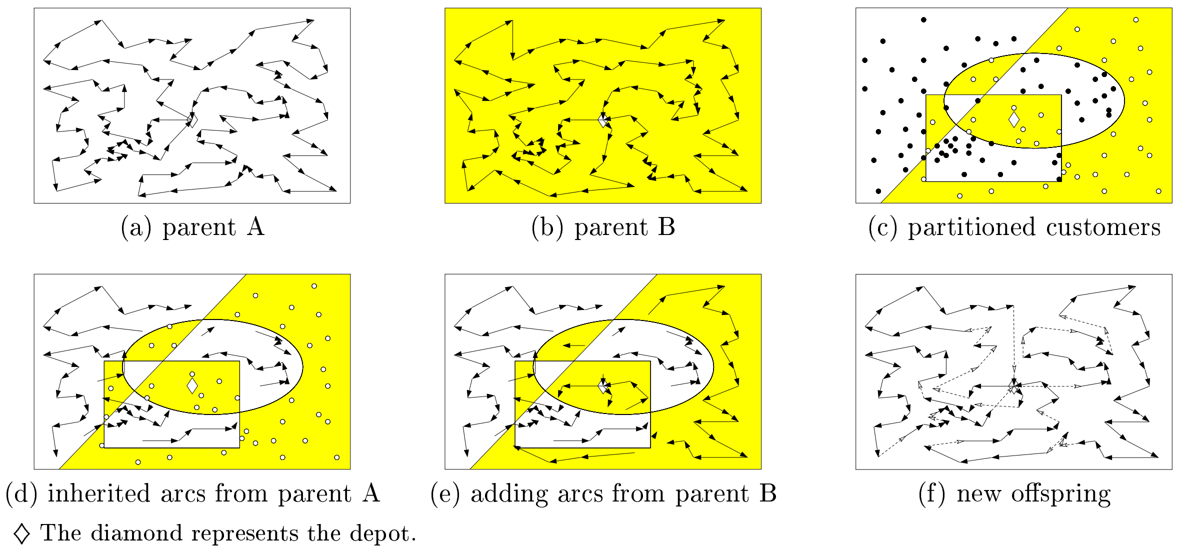
We use the typical binary tournament selection.

Most genetic algorithms for VRPTW do not consider the representation of chromosomes as important, and they recombine the new offspring by heuristically interpreting the two parents. In recombining the offspring, they consider a number of criteria like distances between customers, ranges of time windows, sizes of routes, distribution of distances, etc; but they do not consider the physical locations of customers. In GIDEON system of Thangiah *et al.* [31], [30], each chromosome contains a set of numbers representing the angles defining sectors (centered at the depot) instead of routes themselves. Customers in a sector basically belong to the same route. Namely, this system considers the locations of customers to be more important than other elements.

Multi-dimensional chromosomes were suggested for problems with multi-dimensional characteristics. A two-dimensional crossover, introduced by Cohoon and Paris [8], chooses a small rectangle from one parent and then copies the genes in the rectangle into the offspring with the rest of genes copied from the other parent. Anderson *et al.* [1] suggested a block-uniform crossover which tessellates a 2D chromosome into $i \times j$ blocks, and copies the genes block by block from a uniformly selected parent. Bui and Moon [3] proposed a generalization of crossovers to n dimensions. Jung and Moon [17], [16] introduced an encoding/crossover pair for the 2D Euclidean traveling salesman problem which uses a 2D image as a chromosome, and performs crossover on the chromosome. Since 2D Euclidean VRPs and 2D Euclidean TSPs share a lot of characteristics, we inherit the natural encoding/crossover pair with some modification.

In this paper, we use the 2D image of routes as a chromosome, where each gene is located at the coordinate of the corresponding customer. We describe the natural crossover for the 2D Euclidean VRPTW in the following:

1. The 2D image of two solutions are selected as parents (Figure 2 (a),(b)).


(a) parent A (b) parent B (c) partitioned customers

(d) inherited arcs from parent A (e) adding arcs from parent B (f) new offspring

◇ The diamond represents the depot.

Figure 2: An example of the natural crossover for VRPTW

2. Free curves or figures are drawn on the 2D space where customers are located[1]. It is proven that they always partition the chromosomal space into two equivalent classes [17] (marked white and gray in Figure 2 (c)). Every customer belongs to one of the classes. Customers in the white class are marked black and customers in the other gray class are marked white.

3. For every arc of the parent A, if both of the start-point and the end-point are marked black[2], it survives in the offspring (Figure 2(d)); for every arc of the parent B, if both are marked white, it survives in the offspring (Figure 2(e)). Then we have a number of disconnected segments.

4. The decision variables of the parent A such as a_i, w_i, tt_i, and td_i are saved as a_i^A, w_i^A, tt_i^A, and td_i^A, respectively. The decision variables of the parent B are saved in the same way. They are used in repairing the offspring later.

5. A valid solution is made by adding arcs by the repair algorithm in Section 3.2.1 (Figure 2(f)).

Because we only have to calculate the class of every customer, the time complexity of the crossover grows linearly with respect to the number of customers.

[1]We do not have an efficient implementation for drawing fully free curves. Instead, we use four types of curves — straight line, triangle, quadrangle, and ellipse. Two curves are chosen at random among them allowing multiple occurrences. Refer to [16] for more information.

[2]It is possible that the arc passes through the gray region even when both points are marked black; there are a few arcs in Figure 2 (d). For efficient implementation, we ignore classes of arcs.

3.2.1 Repair Algorithm

The step 5 in the previous section repairs the intermediate offspring to a valid solution. We utilize the parents' decision variables in this process. Figure 3 represents the repair algorithm. Its key routine is connecting the last customer on the current partial route to the minimum-cost start-point of a segment in a nearest-neighbor manner.

In calculating the cost of adding an arc, we consider terms about spatial and temporal closenesses of customers, and terms about parents. Let c_i be the last customer (point) on the current partial route, and let c_j be a candidate customer to connect c_i. The cost is the weighted sum of i) the distance between c_i and c_j, ii) the waiting time at c_j, iii) the slack time of delivery to c_j, and iv) the difference of the service completion between parents and the offspring at c_j:

$$c_{ij} = \delta_1 \cdot d_{ij} + \delta_2 \cdot W_j + \delta_3 \cdot S_j + \delta_4 \cdot P_j$$

where

$$\begin{aligned}
&\delta_1 + \delta_2 + \delta_3 + \delta_4 = 1, \\
A_j &= t_i + t_{ij}, \\
W_j &= \max(0, e_j - A_j), \\
S_j &= l_j - A_j, \text{ and} \\
P_j &= \min(|tt_j^A - (A_j + W_j + s_j)|, \\
&\qquad |tt_j^B - (A_j + W_j + s_j)|).
\end{aligned}$$

$\delta_1, \delta_2, \delta_3$, and δ_4 are reinitialized within respective specific ranges whenever the repair function is invoked.

To investigate the usefulness of parents' decision variables, we compare in Section 4 a GA version with $\delta_4 \neq 0$ against one with $\delta_4 = 0$.

```
repair()
{
    while( there are segments starting from the depot ) {
        randomly choose a segment among them as a partial route;
        complete_a_route(the last customer of the partial route, start-points of the remaining segments);
    }
    while( there are remaining segments )
        complete_a_route(depot, start-points of the segments);
}
complete_a_route(the last customer, candidate customers)
{
    $t \leftarrow$ the last customer;
    do {
        find a feasible customer, say $c^*$, among candidate customers and the depot,
            such that the cost from $t$ to $c^*$ is minimized;
        add an arc from $t$ to $c^*$;
        $t \leftarrow$ the end-point of the segment whose start-point is $c^*$;
    } while( $t$ is not the depot )
}
```

Figure 3: The pseudo-code of the repair algorithm

3.3 Mutation

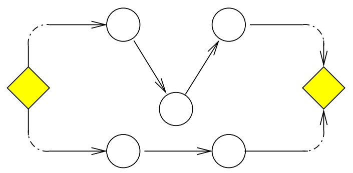
In mutation, each route of the offspring is split into at most three routes. Two cut-points are selected at random to split a route.

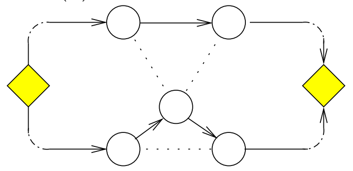
3.4 Local Optimization

A considerable number of local optimization algorithms have been proposed to improve routes. Most of them belong to edge-exchange neighborhoods [18]. Most edge-exchange neighborhoods can be viewed as special cases of the cyclic transfer algorithm introduced by Thompson and Psaraftis [32]. Although the cyclic transfer algorithm is a generalized edge-exchange algorithm, its performance is limited due to its computational cost.

We call three local optimization heuristics in sequence — Or-opt [22], crossover [25], relocation [25] — to optimize the offspring locally. These three heuristics have different characteristics from one another; they are thought to produce synergies. The Or-opt is a vertex-based algorithm trying to move a vertex to another place within a single route. The crossover is a special type of two-edge exchange which removes the cross links of two routes. The relocation is similar to the Or-opt; it is different in that it manipulates multiple routes. Its key routine is moving a vertex in a route to another place in other routes (an example in Figure 4).

In VRPTW, it consumes considerable CPU time to check the time-feasibility of a solution. Consider the routine which checks whether an unrouted customer u can be inserted between two specific adjacent customer v_{p-1} and v_p on the route $R_k = (v_1, v_2, ..., v_m)$. The routine must check the time-feasibility at $u, v_p, v_{p+1}, ...,$ and v_m, respectively. Solomon introduced *Push Forward* [26] to practically speed up this kind of operation. However, the time-feasibility checking takes $O(n)$ in the worst case even when using Push Forward (n is the number of customers in the route.).


(a) before relocation


(b) after relocation

For the sake of convenience, the depot was depicted as two diamonds.

Figure 4: An example of a relocation

We use the *adjusted latest arrival time* (al_{v_i}), instead [20]. The adjusted latest arrival times at customers on a route are computed as follows:

$$al_{v_i} = \begin{cases} l_0 - t_{v_i c_0}, & i = m \\ \min(l_{v_i}, al_{v_{i+1}} - t_{v_i v_{i+1}} - s_{v_i}), & i < m. \end{cases}$$

The adjusted latest arrival time of a customer is the time by which the vehicle must arrive at the customer to satisfy the time-feasibility with no further checking. Using the adjusted latest arrival times, the time-feasibility check is completed in constant time even in the worst case. In other words, when inserting an unrouted customer u between v_{p-1}, and v_p, it only have to check to see if $tt_{v_{p-1}} + t_{v_{p-1}u} + w_u + s_u + t_{uv_p} \leq al_{v_p}$.

The Or-opt algorithm and the relocation algorithm call the above routine very frequently, and thus the time-feasibility is checked fast using the adjusted latest arrival time described in Section 3.1.

3.5 Replacement

The offspring is compared with one of the parents. The parent is replaced to the offspring if the offspring is better. Otherwise, the other parent is replaced if the offspring is better than it. Otherwise, the worst in the population is replaced.

3.6 Stop Condition

Our GA stops when the best solution has not been broken during p consecutive generations. p was set to 2,000.

4 Experimental Results

We set a GA with $\delta_1 \in [0.4, 0.9], \delta_2 \in [0.2, 0.7], \delta_3 \in [0.1, 0.6]$, and $\delta_4 \in [0.3, 0.8]$[3] and call it VGA1.

We programmed our GA in C++ language. In the experiment, the population size and the mutation rate were set to 60 and 0.05, respectively. 100 runs were performed for each Solomon's VRPTW instance [26].

4.1 Performance

Table 2 shows the experimental results of VGA1 and TLOL [28], a recent representative paper from the field of the evolutionary computation. Inter-customer distances were calculated with real double-precision. VGA1-best and VGA1-average represent the best and the average results of VGA1 over 100 runs, respectively. For TLOL, only the best results are available [28]. The figures were rounded off to two decimal places. "#V" and "TD" mean the number of vehicles and the travel distance, respectively. "t" represents the average CPU seconds on Pentium III 1GHz.

VGA1 outperformed TLOL for 47 of the 56 instances; TLOL outperformed VGA1 for two of them; they tied for the other seven instances. Among the 47 cases that VGA1 outperformed, even the average results of VGA1 were better than TLOL (the best results) for all of them except one (R206).

In Table 3, we compared VGA1-best with the optimal solutions[4] reported in [9]. Inter-customer distances were truncated to the first decimal place to be consistent with [9]. The bold-faced numbers represent that the results of VGA1 equal the optimal solutions. VGA1 found optimal solutions for 26 out of 31 instances whose optimal solutions are known. VGA1 found most of the optimal solutions for the C and R groups, but it found the optimum for one of the five in the RC group.

4.2 The Usefulness of Parents' Decision Variables During Crossover

To test the usefulness of parents' decision variables, we set another GA with $\delta_1 \in [0.4, 0.9], \delta_2 \in [0.5, 1.0], \delta_3 \in [0.1, 0.6]$, and $\delta_4 = 0$ (VGA2). Because $\delta_4 \neq 0$, VGA1 utilized parents' decision variables (parents' travel-so-far times, in detail), while VGA2 did not.

Table 4 shows the results of VGA1 and VGA2 for each problem group. Each group of problems has about 10 instances, e.g., C1 has C101 through C109. The best ("Best") and average ("Average") results of each group are the averages of the best and average results for the corresponding instances, respectively.

According to the best results, VGA1 found better solutions more frequently than VGA2, but VGA2 was better on the average. In other words, the deviation of the best and average results in VGA1 was larger than in VGA2. VGA1 seems to be strong in instances that have long scheduling horizons and large vehicle capacities (C2, R2, and RC2 groups), although there are only slight differences compared to VGA2. Overall, the performances of VGA1 and VGA2 were comparable. It is notable that VGA1 was about 30% faster than VGA2.

5 Conclusion

In this paper, we suggested a new hybrid genetic algorithm for the 2D Euclidean vehicle routing problem with time windows. In our genetic algorithm, the 2D image itself of a solution becomes a chromosome; each gene corresponds to a customer in the 2D plane; the natural crossover cuts the chromosomal space with free curves. Because of its simplicity, the natural crossover may be applied to other variants of VRP with minor modification. The experimental results showed that the suggested hybrid genetic algorithm solved VRPTWs to

[3]If $\delta_1 + \delta_2 + \delta_3 + \delta_4 > 1$, then they are scaled down to satisfy that their sum is equal to 1.

[4]http://web.cba.neu.edu/~msolomon/problems.htm. After [9], some more optimal solutions were added.

Table 2: Experimental Results of VGA1

	TLOL		VGA1-best		VGA1-average				TLOL		VGA1-best		VGA1-average		
Instance	#V	TD	#V	TD	#V	TD	t	Instance	#V	TD	#V	TD	#V	TD	t
C101	10	**828.94**	10	**828.94**	10.00	**828.94**	6	C201	3	**591.56**	3	**591.56**	3.00	**591.56**	9
C102	10	**828.94**	10	**828.94**	10.00	**828.94**	12	C202	3	**591.56**	3	**591.56**	3.00	**591.56**	16
C103	10	859.78	10	**828.06**	10.00	**828.06**	18	C203	3	618.00	3	**591.17**	3.00	**591.17**	30
C104	10	893.23	10	**824.78**	10.00	**824.96**	29	C204	3	609.02	3	**590.60**	3.00	**591.18**	44
C105	10	**828.94**	10	**828.94**	10.00	**828.94**	7	C205	3	616.32	3	**588.88**	3.00	**588.88**	10
C106	10	**828.94**	10	**828.94**	10.00	**828.94**	7	C206	3	615.92	3	**588.49**	3.00	**588.49**	12
C107	10	**828.94**	10	**828.94**	10.00	**828.94**	7	C207	3	636.62	3	**588.29**	3.00	**588.29**	13
C108	10	830.94	10	**828.94**	10.00	**828.94**	8	C208	3	611.29	3	**588.32**	3.00	**588.32**	12
C109	10	849.03	10	**828.94**	10.00	**828.94**	9								
R101	18	1648.86	20	**1642.88**	20.00	**1643.53**	30	R201	8	1198.15	9	**1149.68**	8.29	**1153.04**	64
R102	17	1486.71	18	**1472.81**	18.50	**1479.19**	52	R202	9	1057.56	8	**1034.35**	7.40	**1038.40**	81
R103	14	1234.43	14	**1213.62**	14.81	**1222.29**	51	R203	5	922.38	6	**874.87**	6.00	**875.87**	84
R104	11	1024.38	11	**976.61**	11.70	**1001.44**	61	R204	5	791.78	4	**736.52**	4.46	**741.41**	92
R105	15	1372.71	15	**1360.78**	15.91	**1371.52**	34	R205	5	1015.99	5	**955.82**	6.05	**964.69**	70
R106	12	1271.11	13	**1240.47**	13.59	**1252.44**	48	R206	4	884.65	5	**879.89**	5.33	892.55	93
R107	12	1106.19	11	**1073.34**	11.73	**1083.10**	63	R207	4	875.76	4	**799.86**	4.66	**814.05**	97
R108	9	992.12	10	**947.55**	10.74	**959.65**	65	R208	3	778.38	4	**705.45**	3.50	**714.37**	99
R109	13	**1101.37**	13	1151.84	12.97	1157.27	41	R209	3	920.34	5	**859.39**	5.26	**867.52**	83
R110	11	1119.12	12	**1072.41**	12.00	**1082.72**	53	R210	4	961.18	5	**910.70**	6.10	**918.37**	102
R111	12	1083.05	12	**1053.50**	12.00	**1063.21**	56	R211	6	820.23	4	**755.96**	4.70	**765.64**	96
R112	11	1020.52	10	**953.63**	10.77	**971.89**	49								
RC101	14	1659.68	16	**1643.41**	16.46	**1658.34**	28	RC201	4	1354.96	9	**1265.56**	9.00	**1269.94**	50
RC102	15	1492.10	14	**1461.23**	14.65	**1480.82**	45	RC202	8	1151.46	8	**1095.64**	7.84	**1101.03**	74
RC103	11	**1249.86**	12	1277.54	12.11	1313.73	48	RC203	7	1018.09	5	**928.51**	5.29	**943.81**	104
RC104	11	1202.12	10	**1136.81**	10.56	**1154.26**	57	RC204	4	865.51	4	**786.38**	4.05	**799.19**	81
RC105	16	1585.34	16	**1518.58**	15.96	**1540.66**	42	RC205	9	1225.69	7	**1157.55**	7.80	**1164.43**	69
RC106	12	1449.30	13	**1381.23**	13.39	**1397.45**	33	RC206	5	1122.23	7	**1054.61**	6.39	**1067.49**	64
RC107	11	1303.36	12	**1212.83**	12.03	**1227.81**	32	RC207	6	1047.86	6	**966.08**	6.07	**975.24**	76
RC108	11	1197.13	11	**1117.53**	11.00	**1135.81**	32	RC208	4	854.75	4	**779.31**	4.98	**791.35**	68

the near-optimality.

We also tested the usefulness of parents' decision variables during crossover. In terms of solution qualities, the use of parents' decision variables did not give notable improvement; but, it shortened the running time.

We used the synergy of three local optimization heuristics. Some stronger local optimization heuristic may further improve the hybrid genetic algorithm. This part is left for future study.

Acknowledgements

This work was partly supported by KOSEF through Statistical Research Center for Complex Systems at Seoul National University and Brain Korea 21 Project. The RIACT at Seoul National University provided research facilities for this study.

References

[1] C. A. Anderson, K. F. Jones, and J. Ryan. A two-dimensional genetic algorithm for the Ising problem. *Complex Systems*, 5:327–333, 1991.

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

[3] T. N. Bui and B. R. Moon. On multi-dimensional encoding/crossover. In *Sixth International Conference on Genetic Algorithms*, pages 49–56, 1995.

[4] W. Chiang and R. Russell. Simulated annealing metaheuristics for the vehicle routing problem with time windows. *Annals of Operations Research*, 63:3–27, 1996.

[5] W. Chiang and R. Russell. A reactive tabu search metaheuristics for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 9:417–430, 1997.

[6] N. Christofides. Vehicle routing. In E. Lawler, J. Lenstra, A. Rinnooy Kan, and D. Shmoys, editors, *The Traveling Salesman Problem*, pages 431–448. John Wiley & Sons, 1985.

[7] G. Clarke and J. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568–582, 1964.

[8] J. P. Cohoon and W. Paris. Genetic placement. *IEEE Trans. on Computer-Aided Design*, CAD-6(6):956–964, 1987.

[9] J. Cordeau, G. Desaulniers, J. Desrosiers, M. M. Solomon, and F. Soumis. The VRP with time windows. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, pages 157–193. SIAM Monographs on Discrete Mathematics and Applications, 2002.

[10] 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, 1992.

[11] M. Fisher and R. Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11:109–124, 1981.

[12] M. Fisher, K. Jornsten, and O. Madsen. Vehicle routing with time windows: Two optimization algorithms. *Operations Research*, 45:488–492, 1995.

[13] B. Gillet and L. Miller. A heuristic algorithm for the vehcile distpatch problem. *Operations Research*, 22:340–349, 1974.

[14] B. Golden and W. Stewart. Empirical analysis of heuristics. In E. Lawler, J. Lenstra, A. Rinnooy Kan, and D. Shmoys, editors, *The Traveling Salesman Problem*, pages 207–249. John Wiley & Sons, 1985.

[15] J. Homberger and H. Gehring. Two evolutionary metaheuristics for the vehicle routing problem with time windows. *INFOR*, 37:297–318, 1999.

Table 3: Comparison with Optimal Solutions

	Optimum		VGA1-best			Optimum		VGA1-best			Optimum		VGA1-best	
Instance	#V	TD	#V	TD	Instance	#V	TD	#V	TD	Instance	#V	TD	#V	TD
C101	10	827.3	10	**827.3**	R101	20	1637.7	20	**1637.7**	RC101	15	1619.8	16	1637.8
C102	10	827.3	10	**827.3**	R102	18	1466.6	18	**1466.6**	RC102	14	1457.4	14	1473.5
C103	10	826.3	10	**826.3**	R103	14	1208.7	14	**1208.7**	RC103	11	1258.0	11	1273.4
C104	10	822.9	10	**822.9**	R104	-	-	11	971.5	RC104	-	-	10	1132.8
C105	10	827.3	10	**827.3**	R105	15	1355.3	15	**1355.3**	RC105	15	1513.7	15	**1513.7**
C106	10	827.3	10	**827.3**	R106	13	1234.6	13	**1234.6**	RC106	-	-	13	1373.9
C107	10	827.3	10	**827.3**	R107	11	1064.6	11	**1064.6**	RC107	-	-	12	1209.3
C108	10	827.3	10	**827.3**	R108	-	-	10	935.1	RC108	-	-	11	1114.2
C109	10	827.3	10	**827.3**	R109	13	1146.9	13	**1146.9**					
					R110	12	1068.0	12	**1068.0**					
					R111	12	1048.7	12	1049.6					
					R112	-	-	10	948.6					
C201	3	589.1	3	**589.1**	R201	8	1143.2	8	**1143.2**	RC201	9	1261.8	9	1262.4
C202	3	589.1	3	**589.1**	R202	-	-	8	1029.6	RC202	-	-	8	1092.3
C203	3	588.7	3	**588.7**	R203	-	-	6	870.8	RC203	-	-	5	925.5
C204	-	-	3	588.1	R204	-	-	4	731.8	RC204	-	-	4	783.5
C205	3	586.4	3	**586.4**	R205	-	-	5	951.3	RC205	-	-	7	1154.0
C206	3	586.0	3	**586.0**	R206	-	-	5	875.9	RC206	-	-	7	1051.1
C207	3	585.8	3	**585.8**	R207	-	-	4	797.1	RC207	-	-	6	962.9
C208	3	585.8	3	**585.8**	R208	-	-	4	701.4	RC208	-	-	5	779.6
					R209	-	-	5	854.8					
					R210	-	-	6	901.8					
					R211	-	-	4	746.7					

Table 4: Results of VGA1 and VGA2

	VGA1					VGA2				
Group	Best		Average			Best		Average		
	#V	TD	#V	TD	t	#V	TD	#V	TD	t
C1	10.00	**828.38**	10.00	828.40	12	10.00	**828.38**	10.00	**828.38**	13
C2	3.00	**589.86**	3.00	**589.93**	18	3.00	**589.86**	3.00	589.95	21
R1	13.25	**1179.95**	13.73	1190.69	50	13.25	1180.20	13.70	**1189.28**	69
R2	5.36	**878.41**	5.61	**885.99**	87	5.27	878.46	5.64	886.60	109
RC1	13.00	1343.64	13.27	1363.61	40	12.88	**1340.53**	13.34	**1362.38**	54
RC2	6.25	**1004.20**	6.43	1014.06	73	6.38	1004.57	6.44	**1013.72**	88

[16] S. Jung and B. R. Moon. Toward minimal restriction of genetic encoding and crossovers for the 2D Euclidean TSP. *IEEE Trans. on Evolutionary Computation* accepted with minor revision.

[17] S. Jung and B. R. Moon. The natural crossover for the 2D Euclidean TSP. In *Genetic and Evolutionary Computation Conference*, pages 1003–1010, 2000.

[18] G. Kindervater and M. Savelsbergh. Vehicle routing: Handling edge exchanges. In E. Aarts and J. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 337–360. John-Wiley and Sons, Ltd., 1997.

[19] N. Kohl, J. Desrosiers, O. B. G. Madsen, M. M. Solomon, and F. Soumis. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33:101–116, 1999.

[20] G. Kontoravdis and J. Bard. A GRASP for the vehicle routing problem with time windows. *ORSA Journal on Computing*, 7:10–23, 1995.

[21] J. Larsen. *Parallelization of the Vehicle Routing Problem with Time Windows.* PhD thesis, Technical University of Denmark, 1999.

[22] I. Or. *Traveling Salesman-Type Combinatorial Problems and Their Relation to the Logistics of Regional Blood Banking.* PhD thesis, Northwestern University, 1976.

[23] J. Potvin and S. Bengio. The vehicle routing problem with time windows — part II: Genetic search. *INFORMS Journal on Computing*, 8:165–172, 1996.

[24] M. W. P. Savelsbergh. *Computer Aided Routing.* PhD thesis, Centrum voor Wiskunde en Informatica, 1988.

[25] M. W. P. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, 4:146–154, 1992.

[26] M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

[27] M. Solomon and J. Desrosiers. Time window constrained routing and scheduling problems. *Transportation Science*, 22(1):1–13, 1988.

[28] K. Tan, T. Lee, K. Ou, and L. Lee. A messy genetic algorithm for the vehicle routing problem with time window constraints. In *IEEE Congress on Evolutionary Computation*, pages 679–686, 2001.

[29] S. Thangiah. An adaptive method using a geometrical shape for vehicle routing problems with time windows. In *International Conference on Genetic Algorithms*, pages 536–543, 1995.

[30] S. Thangiah. Vehicle routing with time windows using genetic algorithms. In Lance Chambers, editor, *Application Handbook of Genetic Algorithms: New Frontiers*, pages 253–277. CRC Press, 1995.

[31] S. Thangiah, K. Nygard, and P. Juell. GIDEON: A genetic algorithm system for vehicle routing with time windows. In *Seventh Conference on Artificial Intelligence Applications*, pages 322–328, 1991.

[32] P. Thompson and H. Psaraftis. Cyclic transfer algorithms for multi-vehicle routing and scheduling problems. *Operations Research*, 41(5):935–946, 1993.

A Savings based Ant System for the Vehicle Routing Problem

Marc Reimann
Center for Business Studies
University of Vienna
Vienna, Austria
Tel. ++43 1 4277 38096
Fax: ++43 1 4277 38094
e-mail: marc.reimann@univie.ac.at

Michael Stummer
Center for Business Studies
University of Vienna
Vienna, Austria

Karl Doerner
Center for Business Studies
University of Vienna
Vienna, Austria

Abstract

In this paper we study the merit of using the well known Savings algorithm within the framework of an Ant System to tackle the Vehicle Routing Problem (VRP). First, we show the influence of the pheromone information on the solution quality, by comparing the Ant System with a randomized implementation of the Savings algorithm. Second, we evaluate our approach on benchmark data sets. Finally, we provide numerical results about the average and worst case behavior of our algorithm.

1 INTRODUCTION

In this paper we show how a powerful problem specific algorithm can be incorporated into the framework of an Ant System. This is exemplified for a problem from distribution logistics, namely the Vehicle Routing problem (VRP).

The VRP is a well known combinatorial optimization problem, which has been extensively studied for the last 40 years. It involves the construction of a set of vehicle tours starting and ending at a single depot and satisfying the demands of a set of customers. Constraints ensure that each customer is served by exactly one vehicle, vehicle capacities are not exceeded and a prespecified upper bound on the maximum tour length is respected.

The VRP belongs to the class of NP-hard problems (cf. Garey and Johnson, 1979). Therefore no efficient exact solution methods are available, and the existing solution approaches are of heuristic nature. Recently the focus of research on this problem was on the use of meta-heuristics such as Tabu Search, Simulated Annealing and Ant Systems.

The Ant System is a new meta-heuristic developed in the nineties (cf. Colorni et al., 1991). It's inspiration comes from the observation of trail laying - trail following behavior of some ant species. As they move in search for food, the individual ants of these species deposit an aromatic essence called pheromone on the ground. The amount deposited generally depends on the quality of the food sources found. Other ants, observing the pheromone are likely to follow the pheromone trail, with a bias towards stronger trails. Thus, the pheromone trails reflect the 'memory' of the ant population, and over time trails leading to good food sources will be reinforced while paths leading to remote sources will be abandoned.

Within the framework of the Ant System the above mentioned details were implemented in the following way. The artificial ants construct solutions for a given combinatorial optimization problem by taking a number of decisions probabilistically. First these decisions are only based on some local information (e.g. a heuristic rule), as there is no artificial pheromone available. Gradually the collective memory is built up, as some ants, depending on the solution quality found, are allowed to lay artificial pheromone on the paths they used. The amount of pheromone laid also depends on the solution quality. Other ants are then guided in their decision making. Over time paths with high pheromone concentration will attract more ants than paths with low concentration. Thus, these paths will be reinforced and the artificial ants are (hopefully) guided to promising regions of the search space.

This approach has been applied to a number of combinatorial optimization problems, such as the Graph Coloring Problem (c.f. Costa and Hertz, 1997), the Quadratic Assignment Problem (e.g. Stuetzle and Dorigo, 1999), the Travelling Salesman Problem (e.g. (Dorigo and Gambardella, 1997), (Bullnheimer et al., 1999a)), the Vehicle Routing Problem ((Bullnheimer et al., 1999b), (Bullnheimer et al., 1999c)) and the

Vehicle Routing Problem with Time Windows (Gambardella et al., 1999). Recently, a convergence proof for a generalized Ant System has been developed by Gutjahr (Gutjahr, 2002).

In Doerner et al. (Doerner et al., 2002), we have proposed the incorporation of a problem specific heuristic algorithm, namely the well known Savings algorithm into an Ant System for the VRP. The results there have shown the potential of the method. In this paper we present a modified version of the algorithm, where the modifications stem from observations made on the behavior of our original algorithm. These modifications have led to a significant improvement of the performance. Furthermore, we thoroughly evaluate the algorithm. First, we study the learning behavior by comparing cases with and without learning, respectively. Second, we show that the best results found by our approach are competitive to state of the art results. Third, we examine the average and worst case behavior of our algorithm and the effects of problem characteristics on these measures.

The remainder of this paper is organized as follows. In the next section we provide a problem formulation and an overview of related works on the VRP. After that we give a detailed description of our new approach. Section 4 contains the results of the computational study we performed. We conclude with a discussion of our findings.

2 PROBLEM FORMULATION AND RELATED WORKS

The VRP can be formulated in the following way[1]. Let $G = (V, E, c)$ be a complete graph, with $n+1$ nodes $(v_0, ..., v_N)$ corresponding to the customers $i = 1, ..., N$ and the depot $i = 0$, and the edge set $((v_i, v_j) \in E$ $\forall\, v_i, v_j \in V)$. With each edge $(v_i, v_j) \in E$ is associated a non-negative weight c_{ij}, which refers to the travel costs between nodes v_i and v_j and a non-negative weight t_{ij}, which refers to the travel time between the nodes. Furthermore, with each node $v_i, i = 1, ..., N$ is associated a non-negative demand d_i, which has to be satisfied, as well as a service time δ_i. The service time at the depot is set to $\delta_0 = 0$. At the depot a fleet of size K is available, where each vehicle has a capacity of Q^k and the maximum driving time for each vehicle is T^k.

[1]This formulation is closely related to the formulation presented in (Christofides, 1985).

Let x_{ij}^k denote the binary decision variables with the following interpretation:

$$x_{ij}^k = \begin{cases} 1 & \text{if vehicle } k \text{ visits node } v_j \\ & \text{immediately after node } v_i \\ 0 & \text{otherwise.} \end{cases}$$

Then the objective can be written as

$$minimize \sum_{i=0}^{N} \sum_{j=0}^{N} \sum_{k=1}^{K} c_{ij} x_{ij}^k \tag{1}$$

under the following restrictions

$$\sum_{i=1}^{N} \sum_{j=1}^{N} x_{ij}^k d_i \le Q^k \quad 1 \le k \le K \tag{2}$$

$$\sum_{i=0}^{N} \sum_{j=0}^{N} x_{ij}^k (t_{ij} + \delta_i) \le T^k \quad 1 \le k \le K \tag{3}$$

$$\sum_{i=0}^{N} x_{ij}^k - \sum_{l=0}^{N} x_{jl}^k = 0 \quad 1 \le k \le K, 0 \le j \le N \tag{4}$$

$$\sum_{i=0}^{N} \sum_{k=1}^{K} x_{ij}^k = \begin{cases} 1 & 1 \le j \le N \\ K & j = 0 \end{cases} \tag{5}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij}^k \le |S| - 1 \quad \forall S \subseteq \{1, ..., N\}, 1 \le k \le K \tag{6}$$

$$x_{ij}^k \in \{0, 1\} \quad 1 \le k \le K, 0 \le i, j \le N \tag{7}$$

The objective (1) is to minimize the total travel costs. Constraints (2) ensure that no vehicle is overloaded. Constraints (3) require that the maximum driving time for each vehicle is respected. Constraints (4) ensure that if a vehicle visits a customer it also leaves the customer. Constraints (5) require that all customers are visited once, and that the depot is left K times. Subtour elimination is ensured through constraints (6). Finally, constraints (7) are the usual binary constraints.

A large number of researchers have addressed this problem with different approaches. Early approaches dealt with simple heuristics for constructing solutions like the Savings algorithm (Clarke and Wright, 1964) or the Sweep algorithm (Gillett and Miller, 1974), as well as with the application of more or less sophisticated improvement mechanisms like the 2-opt algorithm (Croes, 1958). We use the idea of the Savings algorithm in our Ant System approach, so we will discuss this algorithm in more detail in the next section.

In the last decade the use of meta-heuristics was investigated. First approaches were based on Tabu Search (c.f. Osman, 1993, Gendreau et al., 1994 and Rego and Roucairol, 1996) and Simulated Annealing (Osman, 1993). Recently an Ant System approach for the VRP was proposed by Bullnheimer et al. ((Bullnheimer et al., 1999b),(Bullnheimer et al., 1999c)).

Overviews on exact and approximate methods can be found in Laporte (Laporte, 1999) and Laporte and Semet (Laporte and Semet, 1999), respectively. Meta-heuristics have been reviewed by Gendreau et al. (Gendreau et al., 1999).

As our approach is based on an Ant System, we will now briefly describe the algorithm proposed for the VRP by Bullnheimer et al. (Bullnheimer et al., 1999c). In their paper, the construction of solutions is done using the Nearest Neighbor algorithm. This algorithm starts with the assignment of an arbitrary customer to the first vehicle and in each decision step chooses to visit the customer closest to the current location until the capacity or time constraints of the vehicle are violated. At this point the vehicle returns to the depot, another vehicle is initialized and the procedure is repeated until all customers are assigned. In the case of the Ant System, this Nearest Neighbor algorithm is made stochastic, such that the closest location is not always chosen, but rather all unvisited locations have a positive probability to be chosen. In Bullnheimer et al. (Bullnheimer et al., 1999c) this probability was calculated via a parametrized Savings function. After an ant has constructed a solution the tours of this solution are improved using the 2-opt algorithm (Croes, 1958). At the end of an iteration, i.e. when all ants have generated their solutions, pheromone is updated according to a rank-based scheme with elitists. This means that not all ants are allowed to update the pheromone information, but only the m best ants. The amount of pheromone written depends on the solution quality found as well as on the rank of the ant. In addition to the best ants of the iteration, the global best solution found during the process is updated as if a number of elitist ants had used it in the current iteration.

3 THE SAVINGS BASED ANT SYSTEM ALGORITHM

In this section we propose our implementation of the Savings based Ant System. The Ant System framework of our algorithm is identical to the one proposed in Bullnheimer et al. (Bullnheimer et al., 1999c) and mainly consists of the iteration of three steps:

- Generation of solutions by ants according to private information and pheromone information
- Application of a local search to the ants' solutions
- Update of the pheromone information

Our approach differs in the actual implementation of the three steps as described below.

3.1 SOLUTION GENERATION

The solution generation technique we implemented is the main contribution of our work. As discussed above, solution construction in an Ant System for the VRP has so far been based on the Nearest Neighbor construction mechanism. Note, that in this constructive mechanism vehicles are filled one at a time.

As opposed to that, each of our ants constructs a solution based on the well known Savings algorithm (Clarke and Wright, 1964). We will now describe the main structure of this algorithm and propose the modifications we applied in order to be able to use it in the Ant System context.

The Savings algorithm starts from a solution where all customers are served on separate tours. After that for each pair of customers i and j the following savings measure is calculated:

$$s_{ij} = d_{i0} + d_{0j} - d_{ij}, \tag{8}$$

where d_{ij} denotes the distance between locations i and j and the index 0 denotes the depot. Thus, the values s_{ij} contain the savings of combining two customers i and j on one tour as opposed to serving them on two different tours.

In the iterative phase, customers or partial tours are combined according to these savings, starting with the largest savings, until no more combinations are feasible. A combination is infeasible if it violates either the capacity or the tourlength constraints.

The result of this algorithm is a (sub-)optimal set of tours through all customers.

Our modifications are related to the use of pheromone information in the decision making. Initially, we generate a sorted list of attractiveness values ξ_{ij} in decreasing order. These attractiveness values feature both the savings values as well as the pheromone information.

Thus the list consists of the following values

$$\xi_{ij} = [s_{ij}]^{\beta}[\tau_{ij}]^{\alpha} \tag{9}$$

where τ_{ij} denotes the pheromone concentration on the arc connecting customers i and j, and α and β bias the relative influence of the pheromone trails and the savings values, respectively. The pheromone concentration τ_{ij} contains information about how good the combination of two customers i and j was in previous iterations.

In each decision step of an ant, we consider the k best combinations still available, where k is a parameter of the algorithm which we will refer to as 'neighborhood' below.

Let Ω_k denote the set of k neighbors, i.e. the k feasible combinations (i, j) yielding the largest savings, considered in a given decision step, then the decision rule is given by equation (10), where $\mathcal{P}_{ij}$ is the probability of choosing to combine customers i and j on one tour.

$$\mathcal{P}_{ij} = \begin{cases} \frac{\xi_{ij}}{\sum_{(h,l)\in\Omega_k} \xi_{hl}} & \text{if } \xi_{ij} \in \Omega_k \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

The construction process is stopped when no more feasible combinations are possible.

3.2 LOCAL SEARCH

After the ants have constructed their solutions but before the pheromone is updated each ants' solution is improved by applying a local search. In the paper by Bullnheimer et al. (Bullnheimer et al., 1999c) as well as in our original algorithm (Doerner et al., 2002) the local search algorithm used was the 2-opt algorithm (c.f. Croes, 1958). The 2-opt algorithm was developed for the traveling salesman problem and iteratively exchanges two edges with 2 new edges until no further improvements are possible. In the context of the VRP it was applied separately to all vehicle routes built by the ants. The main idea of this algorithm is to improve the routing of each tour.

In preliminary tests we found that the solutions obtained using only the 2-opt algorithm are acceptable as reported in Doerner et al. (Doerner et al., 2002). While the individual routes were of high quality, the 'sub-optimal' clustering of customers led to the main deviation of these results from the best known solutions.

Thus, we modified the algorithm in the following way. In addition to the 2-opt algorithm, we first apply a local search based on *swap* moves to an ants solution. A *swap* move aims at improving the solution by exchanging two customers from different tours, i.e. a customer i from tour k is exchanged with a customer j from tour l. The idea of this local search is to improve the clustering of the solution. The use of *swap* moves was proposed by Osman (Osman, 1993) for the VRP.

So in our new approach, we first aim to improve the clustering and if no more improvements are possible, we subject each resulting cluster to a 2-opt algorithm in order to improve the routing.

3.3 PHEROMONE UPDATE

After all ants have constructed their solutions, the pheromone trails are updated on the basis of the solutions found by the ants. According to the rank based scheme proposed in Bullnheimer et al. (Bullnheimer et al., 1999a) the pheromone update is done as follows

$$\tau_{ij} := \rho\tau_{ij} + \sum_{\mu=1}^{m} \Delta\tau_{ij}^{\mu} + \sigma\Delta\tau_{ij}^{*} \tag{11}$$

where $0 \leq \rho \leq 1$ is the trail persistance and $\sigma = m + 1$ is the number of elitists. Using this scheme two kinds of trails are laid. First, the best solution found during the process is updated as if σ ants had traversed it. The amount of pheromone laid by the elitists is $\Delta\tau_{ij}^{*} = 1/L^{*}$, where L^{*} is the objective value of the best solution found so far. Second, the m best ants of the iteration are allowed to lay pheromone on the arcs they traversed. The quantity laid by these ants depends on their rank μ as well as their solution quality L^{μ}, such that the μ-th best ant lays $\Delta\tau_{ij}^{\mu} = (m - \mu + 1)/L^{\mu}$. Arcs belonging to neither of those solutions just lose pheromone at the rate $(1-\rho)$, which constitutes the trail evaporation.

After the pheromone information has been updated the attractiveness values ξ_{ij} are augmented with the new pheromone information as in equation (9).

Note, that in our original approach as proposed in (Doerner et al., 2002), this attractiveness list was re-sorted after each iteration. The intuition for this re-sorting was the following. In the beginning the attractiveness values are sorted according to the savings values, as the pheromone is equal on all arcs. As learning occurs, and some arcs are reinforced through the update of the pheromone information, the attractiveness values ξ_{ij} change, as they become more and more biased by the pheromone information. Thus, values that were initially high but turned out not to be in good solutions will decrease, while combinations with initially low values that appeared in good solutions will become more attractive. As the attractiveness values are re-sorted after each iteration, this leads to dynamic effects. In particular, 'good' arcs are reinforced twice. First, they receive more pheromone than others, and second as their attractiveness increases they are considered earlier in the constructive process.

However, this re-sorting lead to fast, and premature convergence and thus was left out of the modified algorithm presented here. This, as a positive side effect, also lead to a minor decrease in computation time which to some degree offset the additional effort needed for the new local search.

4 COMPUTATIONAL STUDY

In this section we will evaluate our proposed approach. First we will describe the standard benchmark problem instances for the VRP. Afterwards we will provide a comparison between a stochastic savings algorithm, where no learning occurs and our new approach, as well as with the approach described in Bullnheimer et al. (Bullnheimer et al., 1999c). Next we will compare our results with state of the art results of other meta-heuristic approaches. Finally, we present information on the average and worst case behavior of our algorithm.

4.1 THE BENCHMARK PROBLEM INSTANCES

All our computations were performed on a set of benchmark problems described in (Christofides et al., 1979). Information on these instances is collected in Table 1.

The instances $C1-C10$ are random problems, i.e. the customers are located randomly in the plane, while instances $C11-C14$ are clustered problems, i.e. the customer locations are clustered. All instances are capacity constrained. In addition to that, the instances $C6-C10$ and $C13-C14$ are restricted with respect to tourlength. In these instances, all customers have identical service times δ. Apart from the additional time constraints, instances 1-5 and 6-10 are identical. The same is true for instances 11-12 and 13-14.

Table 1: Characteristics Of The Benchmark Problem Instances

		Random Problems			
Instance	n	Q	L	δ	best publ.
C1	50	160	∞	0	524.61 (a)
C2	75	140	∞	0	835.26 (a)
C3	100	200	∞	0	826.14 (a)
C4	150	200	∞	0	1028.42 (a)
C5	199	200	∞	0	1291.45 (b)
C6	50	160	200	10	555.43 (a)
C7	75	140	160	10	909.68 (a)
C8	100	200	230	10	865.94 (a)
C9	150	200	200	10	1162.55 (a)
C10	199	200	200	10	1395.85 (b)

		Clustered Problems			
Instance	n	Q	L	δ	best publ.
C11	120	200	∞	0	1042.11 (a)
C12	100	200	∞	0	819.56 (a)
C13	120	200	720	50	1541.14 (a)
C14	100	200	1040	90	866.37 (a)

n ... number of customers
Q ... vehicle capacity
L ... maximum tour length
δ ... service time
best publ. ... best published solution
(a) Taillard, 1993
(b) Rochat and Taillard, 1995

4.2 EVALUATION OF THE LEARNING BEHAVIOR

Let us first analyse the learning behavior of our approach. This will give us a first insight into the performance of the Ant System. As stated above our Ant System is very similar to the Ant System described in (Bullnheimer et al., 1999c). Thus, we chose basically the same parameter settings, namely n artificial ants, $\alpha = \beta = 5$ and $\sigma = 6$ elitist ants.

In preliminary studies we found that for our approach an evaporation rate $\rho = 0.95$ is preferable to $\rho = 0.75$ (as proposed in (Bullnheimer et al., 1999c)). We also varied the population sizes and the number of iterations, but found that n ants and $2 \cdot n$ iterations provide a good compromise between computation time and solution quality. Finally, we tested different sizes of the neighborhood, i.e. different numbers of alternatives in each decision step of an ant and found that $k = \lfloor n/4 \rfloor$ again yields the best results with respect to both computation times and solution quality. More details about these results can be found in (Doerner et al., 2002).

To analyse the learning behavior of our approach and the performance of our new Ant System approach more generally, we compare three cases.

- Savings based Ants: our new approach
- Stochastic Savings algorithm: this refers to our new approach with $\alpha = 0$, i.e. the influence of the pheromone information is deliberately set to zero and no learning occurs
- Standard Ant System: this refers to the algorithm by (Bullnheimer et al., 1999c)

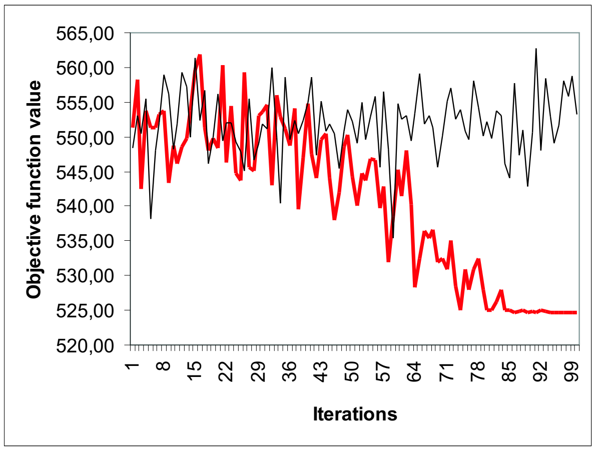


Figure 1: Comparing The Learning Behavior Of Our Savings Based Ants (bold line) With The Stochastic Savings Algorithm

Figure 1 shows the behavior of our new approach as compared to the Stochastic Savings algorithm for a typical run of problem $C1$. We can observe two important results. First, in the beginning the two algorithms are almost identical. The ants have not yet gathered enough pheromone, so the influence of the trails is small. Indeed, it seems that the Stochastic Algorithm finds better solutions more quickly as it is not disturbed by the emerging structure in the pheromone information. Second, the Savings based Ants start to gain from the pheromone memory and continually improve their solutions until they converge to an optimum (which in this case is slightly worse than the global optimum), while the Stochastic Savings algorithm does of course neither learn nor converge to a certain level, but rather oscillates around an average solution that is much worse than the solutions found by our new Savings based Ant System.

Let us now compare the three algorithms with respect to the best solution found over 10 runs for each of the 14 instances. Note, that for all three approaches we generated an identical number of solutions. The results for the three approaches are summarized in Table 2. For each instance we indicate the best solution found with any of the three algorithms in bold.

Table 2: Comparison Of Savings Based Ants With Stochastic Savings And A Standard Ant System

Instance	Savings based AS	SSA	SAS
C1	524.63	530.26	**524.61**
C2	**838.60**	851.63	844.31
C3	**828.67**	847.70	832.32
C4	**1040.09**	1077.57	1061.55
C5	**1303.53**	1356.71	1343.46
C6	**555.43**	560.35	560.24
C7	**909.68**	932.46	916.21
C8	866.87	901.52	**866.74**
C9	**1171.34**	1247.32	1195.99
C10	**1416.05**	1520.17	1451.64
C11	**1042.11**	1047.09	1065.21
C12	**819.56**	**819.56**	**819.56**
C13	**1545.12**	1566.96	1559.92
C14	**866.37**	866.79	**866.37**

SSA...Stochastic Savings algorithm
SAS...Standard Ant System

From Table 2 we can observe two interesting results. First, clearly our new Savings based Ant System outperforms the other two algorithms. In 10 out of the 14 instances it finds strictly better solutions than the other two approaches. For 2 more instances our new approach finds the same solution as one or both of the other algorithms. Only two instances can be solved more effectively with the standard Ant System. However, if we look at the results more closely, we see that our algorithm generally finds significantly better solutions than the other two approaches, whereas when it does not find the best solution, it is very close to the result obtained by the standard Ant System.

Second, for the clustered problems $C11 - C14$ a striking observation can be made. For these instances, the Stochastic Savings algorithm, which performs worst for the random problems, finds very good solutions. More specifically, it is competitive with the Standard Ant System. The structure of these problems seems to be strongly exploited by the Savings algorithm. In fact, as the Savings algorithm tends to combine customers that are close to each other and far from the depot and is able to build tours simultaneously, it is very likely to start building partial tours for each cluster. Thus, the assignment of a customer belonging to a certain cluster, to a tour for another cluster, leading to long unnecessary movements between clusters is unlikely.

4.3 EVALUATION OF OUR SAVINGS BASED ANTS AGAINST STATE OF THE ART RESULTS

Let us now turn the analysis of our algorithm with respect to absolute effectiveness. To that end we compare the results obtained with our Savings based Ant System with two Tabu Search approaches, which are currently the best meta-heuristic approaches for the problem. We will measure the performance in terms of deviation of the best solution obtained with each method from the best known solution available for each instance as presented in Table 1.

The algorithms we compare our Savings based Ant System with are the parallel tabu search algorithm (PTS) from (Rego and Roucairol, 1996) and the TABUROUTE algorithm (TS) from (Gendreau et al., 1994).

The last two rows of Table 3 show for all algorithms the relative percentage deviation (RPD) over the best known solution. More specifically, the second to last row gives the average RPD for the random problems (C1-C10), while the last row shows the average RPD for the clustered problems (C11-C14).

Table 3: Comparison Of Tabu Search And The Savings Based Ant System

Instance	PTS		TS		Savings based AS	
	RPD	min[1]	RPD	min[2]	RPD	min[3]
C1	0.00	1.05	0.00	6.0	0.00	0.06
C2	0.01	43.4	0.06	53.8	0.40	0.34
C3	0.17	26.3	0.40	18.4	0.31	1.37
C4	1.55	48.5	0.75	58.8	1.14	8.41
C5	3.34	77.1	2.42	90.9	0.94	33.2
C6	0.00	2.38	0.00	13.5	0.00	0.06
C7	0.00	20.6	0.39	54.6	0.00	0.40
C8	0.09	18.9	0.00	25.6	0.11	1.48
C9	0.14	29.9	1.31	71.0	0.76	9.91
C10	1.79	42.7	1.62	99.8	1.45	39.3
C11	0.00	11.2	3.01	22.2	0.00	3.44
C12	0.00	1.57	0.00	16.0	0.00	1.33
C13	0.59	1.95	2.12	59.2	0.26	7.22
C14	0.00	24.7	0.00	65.7	0.00	1.44
			RPD (avg.)			
C1-C10	0.71		0.70		0.51	
C11-C14	0.15		1.28		0.06	

[1]Minutes on 4 parallel Sun Sparc 4 machines.
[2]Minutes on a Silicon Graphics Workstation (36MHz).
[3]Minutes on a Pentium III (900 MHz).

From Table 3 it can be clearly seen, that our algorithm outperforms the two Tabu Search approaches with respect to solution quality. The average deviation of our Savings based Ant System over all instances is only 0.38%, while the parallel tabu search is on average 0.55% away from the best known solution, and the TABUROUTE algorithm has an average deviation of 0.81%. Particularly, on the clustered problems the superior performance of our algorithm can be observed. This is due to the fact, that the Savings algorithm itself is known to perform very well on the clustered problems. This was also confirmed by our results in the last section.

Looking at computation times, we have to consider the following issues: first, the machines used differ greatly. Moreover, the PTS algorithm, was performed on parallel machines. Second, the times given for the Tabu Search approaches denote the time to find the best solution, whereas our computation times denote the time to perform $2 \cdot n$ solutions. For most problems the best solution was found earlier in the search process. Keeping these points in mind, it seems that our approach finds competitive solutions very fast as compared to the other methods.

4.4 AVERAGE AND WORST CASE BEHAVIOR OF THE PROPOSED METHOD

So far we have evaluated our approach according to the best solutions found for each instance in 10 trials. In this section we will take a look at the average and worst case behavior of our algorithm in these 10 trials for each of the 14 instances.

What is mainly of interest, is the question whether the average or worst deviation depends on the problem size and characteristics. Therefore we will now provide results for the different problem sizes and for the different problem characteristics, namely random and clustered.

Let us first look at problem characteristics. In Table 4 we can see the average and worst case behavior of our Savings based Ant System, averaged over the random and clustered problems, respectively.

From these results it becomes once again obvious that our algorithm performs particularly well on the clustered problems. However, the worst case behavior of 1.92% for the random problem seem still to be more than reasonable.

While we have now confirmed the strong performance of our Savings based Ant System for the clustered problems let us finally turn to the question how av-

Table 4: Influence Of Problem Characteristics On The Average And Worst Case Behavior Of Our Savings Based Ant System

Deviation in %	**Random Problems**	**Clustered Problems**
Average	1.10	0.14
Worst Case	1.92	0.18

erage and worst case behavior relate to the problem size. In order to answer this question, we have only looked at problems $C1 - C10$. The intuition for this is the following. The clustered problems $C11 - C14$ have sizes of 100 and 120 customers. Thus, they are medium sized. However, due to their clustered structure they can be better solved than problems with random distribution of customers of equal and even smaller size. In order not to bias our results on the dependence of average and worst case behavior on problem size we have thus decided to ignore problems $C11 - C14$.

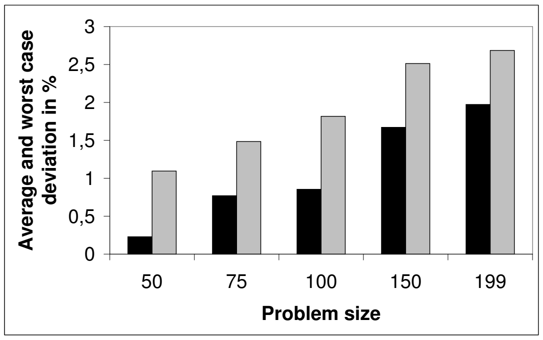


Figure 2: Average(Dark Columns) And Worst Case Behavior Of Our Savings Based Ants For Different Problem Sizes

Figure 2 shows that both average and worst case results increase with problem size however at a decreasing rate. This result is particularly encouraging as it suggests that even for larger problems our Savings based approach should be able to find robust results. However, the validity of this statement needs to be tested in future work.

5 CONCLUSIONS

In this paper we have investigated the merit of the incorporation of a powerful problem specific algorithm for the VRP, namely the Savings algorithm, into an Ant System framework.

We have first presented a mathematical formulation, followed by an overview of existing research in the area. Afterwards we have described our approach in detail.

Through tests on the standard benchmark problem instances we were able to show, that our algorithm exhibits adaptive learning, outperforms existing Ant System as well as state of the art Tabu Search approaches and shows satisfying average and worst case behavior.

Finally, we should add, that our approach features two important issues concerning real world problems. First, these problems are generally clustered, as in cities the density of customer locations is higher than in rural areas. We showed, that for clustered problems our algorithm works particularly well. Second, computation time is often crucial in real world applications, and our results suggest, that our Savings based Ant System compares favorably to other techniques with respect to this objective.

Acknowledgments

This work was supported by the Austrian Science Foundation under grant SFB #010 'Adaptive Information Systems and Modelling in Economics and Management Science' and by the Oesterreichische Nationalbank (OeNB) under grant #8630. We are grateful to Herbert Dawid and four anonymous referees for their valuable comments on the paper.

References

B. Bullnheimer, R. F. Hartl and Ch. Strauss (1999a): A new rank based version of the ant system: a computational study. *Central European Journal of Operations Research* **7**(1):25–38.

B. Bullnheimer, R. F. Hartl and Ch. Strauss (1999b): Applying the ant system to the vehicle routing problem. In: S. Voss et al. (eds.), *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer, Boston, 285–296.

B. Bullnheimer, R. F. Hartl and Ch. Strauss (1999c): An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research* **89**:319–328.

N. Christofides, A. Mingozzi and P. Toth (1979): The vehicle routing problem. In: N. Christofides et al. (eds.), *Combinatorial Optimization*, Wiley, Chicester, 315–338.

N. Christofides (1985): Vehicle Routing. In. E. L. Lawler et al. (eds.), The Traveling Salesman Problem, Wiley, Chicester, 431–448.

G. Clarke and , J. W. Wright (1964): Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research* **12**:568–581.

A. Colorni, M. Dorigo and V. Maniezzo (1991): Distributed Optimization by Ant Colonies. In: F. Varela and P. Bourgine (eds.), *Proceedings of the First European Conference on Artificial Life*, Elsevier, Amsterdam, 134–142.

D. Costa, A. Hertz (1997). Ants can colour graphs. *Journal of the Operational Research Society* **48**(3):295–305.

G. A. Croes (1958). A method for solving Traveling Salesman Problems. *Operations Research* **6**:791–801.

K. Doerner, M. Gronalt, R. F. Hartl, M. Reimann, Ch. Strauss and M. Stummer (2002): SavingsAnts for the Vehicle Routing Problem. In S. Cagnoni et al.(eds.), *Applications of Evolutionary Computing*, Springer LNCS 2279, Berlin/Heidelberg, 11–20.

M. Dorigo and L. M. Gambardella (1997): Ant Colony System: A cooperative learning approach to the Travelling Salesman Problem. *IEEE Transactions on Evolutionary Computation* **1**(1):53–66.

L. M. Gambardella, E. Taillard and G. Agazzi (1999): MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows. In D. Corne et al.(eds.), *New Ideas in Optimization*, Mc Graw-Hill, London, 63–73.

M. R. Garey, D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP Completeness.* W. H. Freeman & Co., New York.

M. Gendreau, A. Hertz and G. Laporte (1994): A tabu search heuristic for the vehicle routing problem. *Management Science* **40**:1276–1290.

M. Gendreau, G. Laporte and Y. Potvin (1999): Metaheuristics for the vehicle routing problem. GERAD Technical report G-98-52.

B. E. Gillet and L. R. Miller (1974): A heuristic algorithm for the vehicle-dispatch problem. *Operations Research* **22**:340–349.

W. J. Gutjahr (2002): ACO algorithms with guaranteed convergence to the optimal solution. *Information Processing Letters* **82**:145–153.

G. Laporte (1999): Exact algorithms for the travelling salesman problem and the vehicle routing problem. GERAD Technical report G-98-37.

G. Laporte and F. Semet (1999): Classical heuristics for the vehicle routing problem. GERAD Technical report G-98-54.

I. H. Osman (1993): Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research* **41**:421–451.

C. Rego and C. Roucairol (1996): A parallel tabu search algorithm using ejection chains for the vehicle routing problem. In: I. H. Osman and J. Kelly (eds.), *Meta-Heuristics:Theory and Applications*, Kluwer, Boston, 661–675.

Y. Rochat and E. D. Taillard (1995): Probabilistic Diversification and Intensification in Local Search for Vehicle Routing. *Journal of Heuristics* **1**:147–167.

T. Stützle, M. Dorigo (1999): ACO Algorithms for the Quadratic Assignment Problem. In D. Corne et al. (eds.), *New Ideas in Optimization*, Mc Graw-Hill, London, 33–50.

E. D. Taillard (1993): Parallel iterative search methods for vehicle routing problems. *Networks* **23**:661–673.