Capacitated Arc Routing Problem Solver based on Evolutionary Simulated Annealing

Shengli Zhou (12212232), CSE, SUSTech

Abstract—**In this paper, we provide a method for solving the Capacitated Arc Routing Problem (CARP) by Evolutionary Simulated Annealing (ESA). In our method, we first use Path Scanning to generate an initial solution and then optimize the solution with 5 move operators, i.e., Flip, Single Insertion, Double Insertion, Swap and 2-opt. We determine whether a move will be applied to optimize current solution based on the strategy of Evolutionary Simulated Annealing.**

Index Terms—**Artificial Intelligence, Capacitated Arc Routing Problem (CARP), Combinatorial Optimization, Local Search, Evolutionary Simulated Annealing (ESA).**

I. Introduction

THE Capacitated Arc Routing Problem (CARP) is a combinatorial optimization problem that arises in the field of transportation and logistics. In CARP, the objective is to find the most efficient set of routes for a fleet of vehicles to serve a set of edges in a network while respecting capacity constraints on each vehicle.

The key elements of CARP include an undirected graph representing the transportation network, each arc having a cost associated with traversing it, and capacity constraints for each vehicle. Initially, a fleet of vehicles are located at a specific node called depot. For each vehicle, it traverse through some routes and return to the depot for arbitrary times. When every traverse begins, the capacity of a vehicle is reset to a specific value Q as initial capacity. Every time a vehicle traverse through an edge, it has to pay for the cost. Also, the vehicle can choose to satisfy the demand of the edge if the remaining capacity of the vehicle is no less than the demand.

The goal of CARP is to minimize the total cost of routing while ensuring all edges with demand are satisfied without exceeding the capacity of the vehicles.

CARP has numerous real-world applications in areas such as waste collection, street sweeping, postal delivery, and public transportation. Efficient solutions to CARP can lead to significant cost savings, improved service quality, and reduced environmental impact.

In this project, we propose a method for solving CARP based on **Evolutionary Simulated Annealing (ESA)**. We first use **Path Scanning** to generate an initial solution and then optimize the solution with 5 move operators, i.e., **Flip, Single Insertion, Double Insertion, Swap and 2-opt**. Finally, we also conduct experiments to evaluate the performance of different sets of parameters.

This project was supported by Department of Computer Science and Engineering, Southern University of Science and Technology (SUSTech). The project was posted in course CS311 Artificial Intelligence (H) as the second course project on April 12, 2024. All works of the project was finished on May 10, 2024.

S. Zhou is a sophomore student in Southern University of Science and Technology (SUSTech), Shenzhen, Guangdong Province, China. (e-mail: zhousl2022@mail.sustech.edu.cn).

II. Preliminary: Problem Formulation

The Capacitated Arc Routing Problem can be formulated as a combinatorial optimization problem on an undirected graph $G = (V, E)$. For each edge $e \in E$, it is characterized by its endpoints (u, v) and its cost c. For some edges $t \in E$, they also have a demand d. We denote the edge set containing all edges with demands as T. Obviously, $T \subseteq E$. CARP requires the solver to generate a set of routes, each route starts and ends at a specific node called depot, denoted as v_0 ($v_0 \in V$). The cost of route R is defined by

$$c_R = \left(\sum_{i=1}^{|R|} \text{dis}(v_{i-1}, u_i) + c_{R,i} \right) + \text{dis}(v_{|R|}, v_0) \tag{1}$$

while the total demand on route R is defined by

$$d_R = \sum_{e \in S_R} d_e \tag{2}$$

where $S_R \subseteq R \cap T$.

A valid solution to CARP is constrained by the following requirements:

- $\forall R_i.\ d_{R_i} \leq Q$, where Q is the capacity of each vehicle.
- All S_{R_i}'s form a partition of T, i.e., each route with demand is served for exactly once.

The goal of CARP is to construct a valid solution with the lowest cost, i.e., minimize $\sum_{R_i} c_{R_i}$.

Here we also define P_R as the endpoint of route R, i.e., route R starts at depot v_0 and ends at P_R. When we finish constructing route R, we should have $P_R = v_0$.

III. Methodology

A. General Workflow

The framework is illustrated in Fig. 1.

In our method, we first compute the shortest path of between each pair of nodes in G using Floyd-Warshall Algorithm, the information of shortest path will be used in the Path Scanning process. Then we use five strategies (namely Maximize Distance, Minimize Distance, Maximize Ratio, Minimize Ratio and Capacity Determined, which will be discussed in Sec. III-B1) during Path Scanning to generate a series of initial solutions. The generated solutions are valid solutions but may not be optimal. After Path Scanning, we try to optimize

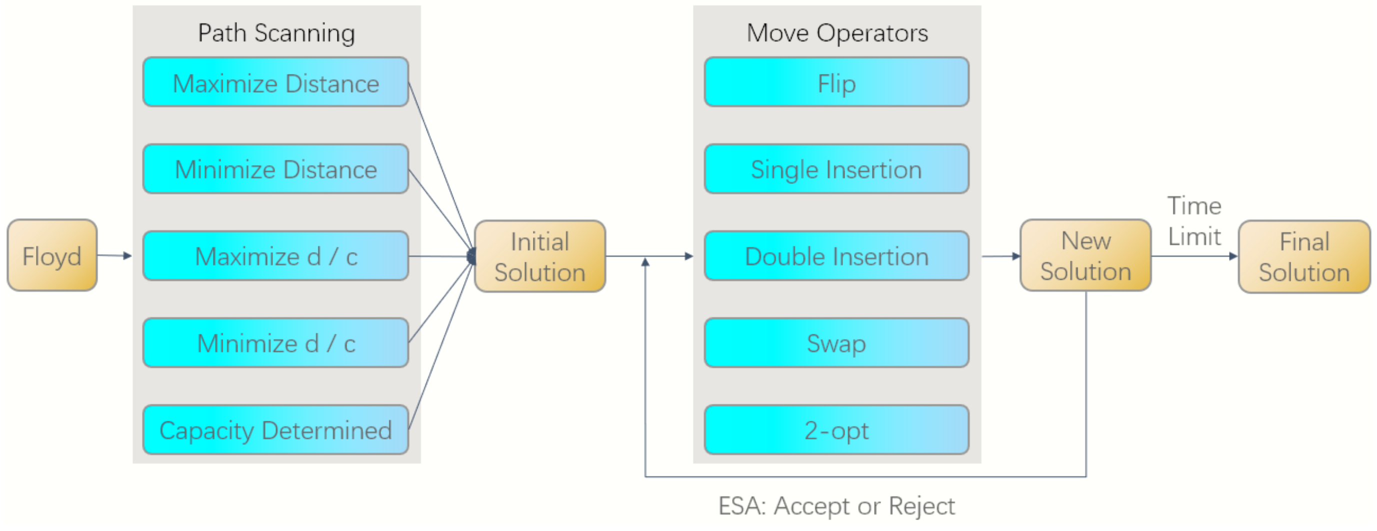

Fig. 1. General Workflow.

these solutions by five move operators, namely Flip, Single Insertion, Double Insertion, Swap and 2-opt (which will be discussed in Sec. III-B2). After each trial of optimization, we determine whether to accept the move or not by Evolutionary Simulated Annealing (discussed in Sec. III-B3). The algorithm terminates when the given time is used up.

B. Detailed Algorithm Design

1) Path Scanning: The Path Scanning algorithm for constructing an initial solution is as follows.

- Maintain an edge set $F \subseteq T$ containing all edges with demand but not satisfied yet.
- For each route, start from depot v_0.
- Let $F' \subseteq F$ be the set of all edges in F with capacity not exceeding the remaining capacity of the current route.
- Choose an edge from F' which is nearest to the current endpoint P_{R_k}. Here we define the distance between a node and an edge as the minimum value of the shortest path between the node and one of the nodes incident to the edge.
- If there are multiple edges having the same distance to the endpoint while the distance is optimal, we can decide on which path to choose based on the five rules below.
  - **Maximize Distance**: Choose the edge that can maximize the distance from the new endpoint of the route to depot v_0.
  - **Minimize Distance**: Choose the edge that can minimize the distance from the new endpoint of the route to depot v_0.
  - **Maximize Ratio**: Choose the edge that has the greatest ratio between its demand and its cost.
  - **Maximize Ratio**: Choose the edge that has the smallest ratio between its demand and its cost.
  - **Capacity Determined**: If the remaining capacity of the vehicle exceeds half of its original capacity, apply ”Maximize Distance”, otherwise apply ”Minimize Distance”.
- When F' is empty, we stop extending the current route by adding the distance of traversing back to depot v_0 and start a new route.
- When F is empty, the path scanning algorithm terminates.

The pseudo code of path scanning is shown below. For simplicity, we consider d_f for edges without demand as 0.

Algorithm 1 Path Scanning

```
$k \leftarrow 0$                                  ▷ $k$ is the number of routes.
$F \leftarrow T$                    ▷ $F$ is all unsatisfied edges with demand.
while $F \neq \emptyset$ do
    $k \leftarrow k + 1$
    $R_k \leftarrow \emptyset, c_{R_k} \leftarrow 0, d_{R_k} \leftarrow 0, P_{R_k} \leftarrow v_0$
    while True do
        $F' \leftarrow$ all edges $f$ in $F$ with $d_f \leq Q - d_{R_k}$
        if $F' == \emptyset$ then
            $c_{R_k} \leftarrow c_{R_k} + dis(P_{R_k}, v_0)$
            break
        end if
        $f \leftarrow$ the edge nearest to $P_{R_k}$
                              ▷ Also apply the five rules above here.
        if $dis(P_{R_k}, f.u) < dis(P_{R_k}, f.v)$ then
            $R_k \leftarrow R_k + (f.u, f.v)$
            $c_{R_k} \leftarrow c_{R_k} + dis(P_{R_k}, f.u) + c_f$
            $P_{R_k} \leftarrow f.v$
        else
            $R_k \leftarrow R_k + (f.v, f.u)$
            $c_{R_k} \leftarrow c_{R_k} + dis(P_{R_k}, f.v) + c_f$
            $P_{R_k} \leftarrow f.u$
        end if
        $d_{R_k} \leftarrow d_{R_k} + d_f$
        $F \leftarrow F - \{f\}$
    end while
end while
```

2) Move Operators: After path scanning, we have obtained some solutions to the problem. Here we try to optimize the solutions by applying five kinds of move operators to construct new valid solutions to the instance. For every iteration, we randomly choose one of the move operators to apply based on the probability given by the parameters. If the capacity of a route exceeds the restriction, the solution generated by the move operator will not be used to update the current solution.

Flip This operator randomly choose a route in the solution and reverse the direction of an randomly chosen edge with demand in the route. The effect of Flip operator is illustrated in Fig. 2 below.

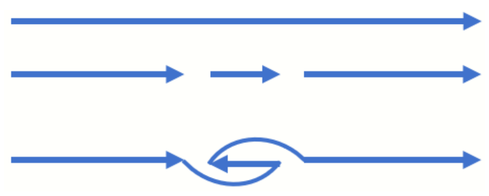

Fig. 2. Flip.

Single Insertion This operator randomly choose an edge e and a route R, the operator will remove e from its original position and randomly insert it after the depot or an edge (with demand) in R. Since the capacity of R will increase after the operation and can easily break the capacity constraint, we modify the method of choosing R to have half the probability of choosing R as the same route e is in and half the probability of choosing R other than the route e is in.

Double Insertion The Double Insertion operator is similar to the Single Insertion operator. The only difference is that it choose two contiguous required edges, remove them together and insert them together in another place.

Compared to Single Insertion, Double Insertion can also generate low-cost solutions as the cost between the chosen edges remain unchanged. However, as we move more edges, the generated solution is more likely to exceed the capacity restriction, thus we do not consider to move more (i.e., greater than 2) edges together.

Swap The swap operator randomly choose two different edges with demand and swap their position. Similarly, our method decide to choose the same route or different routes for two edges with the same probability. The effect of Swap operator is illustrated in Fig. 3 below.

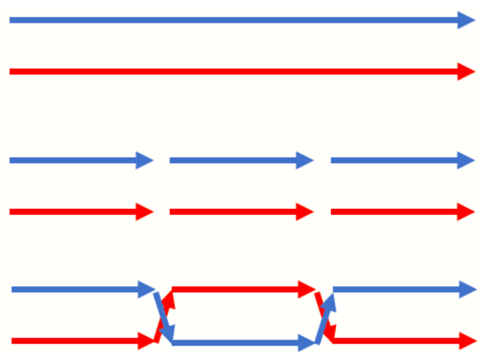

Fig. 3. Swap.

2-opt 2-opt is a move operator proposed in [1]. There are two kinds of 2-opt operations, we set the probability of choosing either type as 0.5.

The first type of 2-opt choose a route and flip a sub-route, i.e., applying Flip operator from the i-th demanded edge to the j demanded edge where $i \leq j$.

The effect of the second type of 2-opt operator is illustrated in Fig. 4 below.

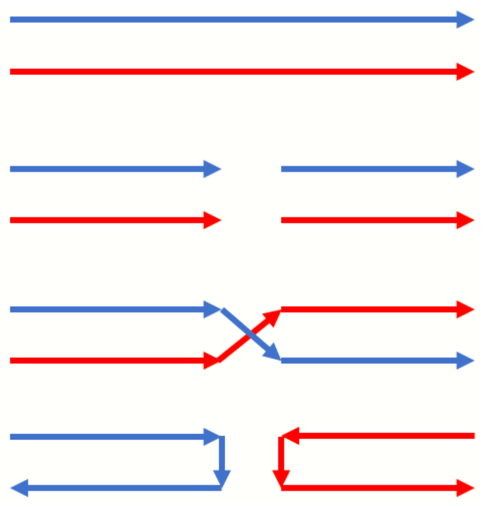

Fig. 4. 2-opt.

This type of 2-opt operator choose two different routes and split each of them into two sub-routes, here we denote the sub-routes as $R_{1A}, R_{1B}, R_{2A}, R_{2B}$ respectively. The first method of applying 2-opt is to concatenate R_{1A}, R_{2B} and R_{2A}, R_{1B} respectively to form two new routes (as shown in the third row of Fig. 4). The second method of applying 2-opt is to concatenate R_{1A} with the reverse of R_{2A} and the reverse of R_{1B} with R_{2B} to form two new routes (as shown in the fourth row of Fig. 4). In our method, we will choose the one with less cost if both of them does not break the capacity constraint.

3) Evolutionary Simulated Annealing: After applying move operators to current solution S, we can get a new solution S'. In our method, we use the rule of Evolutionary Simulated Annealing (ESA) to determine whether we should replace our current solution with the new one.

During this process, we first calculate the magnitude of optimization by the difference of cost, i.e.,

$$\Delta c = c_{S'} - c_S \tag{3}$$

If $\Delta c \leq 0$, S' is better than S, thus we will always accept S'. When $\Delta c > 0$, we accept the new solution by probability

$$p(\text{Accept}) = \exp\left(-\frac{\Delta c}{T}\right) \tag{4}$$

Where T is a dynamic parameter of the algorithm representing the temperature during annealing.

During our method, we set an initial value T_0 and a lower bound for temperature T, for every ΔI iterations, T will be multiplied by a decay coefficient δ until T reaches the lower bound.

Since when $T < 1$, $p(\text{Accept})$ is too small for worse solutions to be accepted, we set the lower bound of the temperature as 1.

C. Analysis

During Path Scanning, we can generate solutions that are valid but generally far from optimal. Also since the Path Scanning algorithm apply greedy strategies when constructing solutions, the initial solutions may be hard to optimize without causing extra cost. Thus, we generate several agents using different strategies of Path Scanning to seek for more possibilities of acquiring better solutions.

For Flip operator, simply flipping an edge on the solution constructed by greedy algorithm tends to increase the cost. However, the operator is important since it not only provides a disturbance to the solution (which may lead to better solutions), but is also a small-step operator that does not cause the solution to change too much.

Swap operators also make minor adjustments to the solution and have similar effect as Flip operations.

2-opt operators make major changes to the solution while minimizing the extra cost due to these operators. The 2-opt operator can help the solver jump out of local minimum, but may also cause the solver to miss the local minimum nearby.

Insertion operators make larger adjustments than Flip and Swap operators but smaller adjustments than 2-opt operator. We choose these operators since they may form a balance between finding local minimum and jumping out of local minimum. However, since inserting edges in a different route is not likely to decrease the cost, the performance of Insertion operators are not satisfying practically.

Generally speaking, there is no move operator that is optimal for all cases. Thus, we randomly apply different methods to the current solution to seek for local minimum while try to jump out of it when we reach a relatively good solution.

IV. Experiments

During this part, we conduct two experiments to determine the optimal set of parameters to solve CARP.

A. Experimental Setup

1) Environment: The code for performing the experiments is written in Python 3.9.7 and can be executed under an environment with numpy==1.24.4.

The code is both tested locally on Dell Inspiron 15 3530 with 13th Gen Intel(R) Core(TM) i7-1355U processor (1.70 GHz) and on sustech-lms platform with Debian 10 and two 2.2GHz and 8-core CPUs.

Since some of our moves are based on ramdomization, the performance may vary when executed on different platforms.

2) Datasets: We conduct the first experiment on the dataset provided with the project specifications (which can be downloaded here). The detailed information of the instances are shown in Table I below.

TABLE I
DETAILED INFORMATION OF DATASETS

Instance	Abbreviation	Vertices	Edges	Tasks
egl-e1-A	e1	77	98	51
egl-s1-A	s1	140	190	75
gdb1	g1	12	22	22
gdb10	g10	12	25	25
val1A	v1	24	39	39
val4A	v4	41	69	69
val7A	v7	40	66	66

Our second experiment is based on the Eglese Dataset (which can be downloaded here).

B. Results

1) Experiment 1: Initially, we choose a set of parameters that has relatively good performance on sustech-lms platform. The parameters are shown in Table II. The detailed information of the instances are shown in Table I.

TABLE II
INITIAL PARAMETERS

Attribute	Explanation	Value
p_1	p(Flip)	0.3
p_2	p(Single Insertion)	0.05
p_3	p(Double Insertion)	0.05
p_4	p(Swap)	0.2
p_5	p(2-opt)	0.4
N	Number of Agents	5
T_0	Initial Temperature	100
δ	Decay Coefficient	0.999
ΔI	Period of Applying δ	40

In experiment cases $2 \sim 28$, we adjust the parameters of each category to determine the effect. The detailed adjustment is shown in Table III and Table IV.

TABLE III
DETAILED ADJUSTMENTS

No.	Modified Parameter	Value
2		10
3	N	15
4		20
5		0.9993
6	δ	0.9995
7		0.9997
8		0.9999
9		1
10	T_0	10
11		50
12		150
13		1
14		10
15	ΔI	20
16		30
17		50
18		100

TABLE IV
DETAILED ADJUSTMENTS (CONTD.)

No.	Modified Parameter	Value
19		$(0.2, 0.1, 0.1, 0.2, 0.4)$
20		$(0.3, 0.05, 0.05, 0.25, 0.35)$
21		$(0.3, 0.05, 0.05, 0.15, 0.45)$
22		$(0.2, 0.05, 0.05, 0.3, 0.4)$
23	$(p_1, p_2, p_3, p_4, p_5)$	$(0.25, 0.05, 0.05, 0.25, 0.4)$
24		$(0.3, 0.05, 0.05, 0.4, 0.2)$
25		$(0.3, 0.05, 0.05, 0.3, 0.3)$
26		$(0.3, 0.05, 0.05, 0.35, 0.25)$
27		$(0.3, 0.05, 0.05, 0.25, 0.35)$
27		$(0.3, 0.1, 0.1, 0.2, 0.3)$

The result of the experiment is shown in Table V below. It can be seen that the 3rd set of parameters perform well on large instances like e1 and s1 while the 21st set of parameters perform well on small instances like g1, g10, v1, v4 and v7.

TABLE V
RESULTS OF EXPERIMENT 1

No.	e1	s1	g1	g10	v1	v4	v7
1	**3548**	5138	329	285	181	471	324
2	3564	5226	340	306	186	457	**310**
3	3552	**5128**	334	303	184	467	321
4	3572	5183	331	308	185	467	363
5	3561	5176	327	306	185	444	326
6	**3548**	5147	**316**	299	180	469	339
7	**3548**	5153	335	298	185	443	336
8	3569	5165	323	304	177	463	320
9	**3548**	5153	**316**	307	183	436	339
10	**3548**	5153	333	307	183	455	325
11	**3548**	5169	323	309	174	435	343
12	3576	5176	330	299	185	456	336
13	3564	5169	324	300	185	470	337
14	**3548**	5153	329	294	185	437	363
15	**3548**	5152	322	297	180	453	351
16	**3548**	5154	329	285	**173**	465	337
17	**3548**	5185	323	291	185	466	324
18	3552	5173	328	298	**173**	474	324
19	**3548**	5136	323	302	180	454	334
20	3552	5375	360	325	187	543	363
21	3614	5197	322	**283**	180	**425**	337
22	**3548**	5159	325	301	185	461	321
23	**3548**	5151	330	302	180	444	332
24	3591	5329	337	335	205	478	375
25	3581	5371	337	326	214	459	363
26	3592	5270	362	335	202	459	339
27	3552	5330	360	325	202	459	339
28	3560	5385	342	335	187	507	339
best	3548	5128	316	283	173	425	310
avg.	3560	5202	332	306	185	461	338

2) Experiment 2: In this experiment, we evaluate the performance of our model by comparing with MAENS proposed in [1]. Results can be seen in Table VI and VII. Here we define

$$\Delta = \frac{\text{Ours} - \text{MAENS}}{\text{MAENS}} \tag{5}$$

TABLE VI
RESULTS OF EXPERIMENT 2

Instance	Vertices	Edges	Tasks	Ours	MAENS	Δ
egl-e1-A	77	98	51	3548	3548	0.00
egl-e1-B	77	98	51	4567	4498	0.02
egl-e1-C	77	98	51	5687	5595	0.02
egl-e2-A	77	98	72	5036	5018	0.00
egl-e2-B	77	98	72	6415	6317	0.02
egl-e2-C	77	98	72	8502	8335	0.02

TABLE VII
RESULTS OF EXPERIMENT 2 (CONTD.)

Instance	Vertices	Edges	Tasks	Ours	MAENS	Δ
egl-e3-A	77	98	87	5998	5898	0.02
egl-e3-B	77	98	87	8059	7775	0.04
egl-e3-C	77	98	87	10412	10292	0.01
egl-e4-A	77	98	98	6657	6456	0.03
egl-e4-B	77	98	98	9256	8998	0.03
egl-e4-C	77	98	98	11933	11561	0.03
egl-s1-A	140	190	75	5138	5018	0.02
egl-s1-B	140	190	75	6583	6388	0.03
egl-s1-C	140	190	75	8743	8518	0.03
egl-s2-A	140	190	147	10483	9895	0.06
egl-s2-B	140	190	147	13978	13147	0.06
egl-s2-C	140	190	147	17381	16430	0.06
egl-s3-A	140	190	159	11043	10257	0.08
egl-s3-B	140	190	159	14829	13749	0.08
egl-s3-C	140	190	159	18113	17207	0.05
egl-s4-A	140	190	190	13579	12341	0.10
egl-s4-B	140	190	190	17434	16337	0.07
egl-s4-C	140	190	190	22032	20538	0.07

The results show that our model is generally more inefficient than MAENS but the gap is not big.

V. CONCLUSION

In this paper, we design an agent for solving CARP based on Path Scanning and Evolutionary Simulated Annealing. We apply five strategies (Maximize Distance, Minimize Distance, Maximize Ratio, Minimize Ratio and Capacity Determined) during Path Scanning and utilize Flip, Single Insertion, Double Insertion, Swap and 2-opt as move operators to optimize solutions during Evolutionary Simulated Annealing. We also use experiments to determine the best set of parameters to solve CARP.

During this project, we are more aware of the efficiency of implementing algorithms in python. When we start developing our project, we first use quite a few deep copies to avoid mistakes caused by python's "parse by reference" rules. However, deep copies are too time-consuming and greatly limited the number of optimization trials in a relatively short time limit. Therefore, we optimized our code to decrease the number of deep copies and increase the number of trials, the cost of our solution to Eglese's instance egl-e1-A dropped from around 4100 to around 3900 after our optimization.

Apart from out method, there are still several ways to improve the performance of the agent, including Ulusoy Split and MAENS. We hope to try these methods in the future and compare them with our current model.

ACKNOWLEDGMENTS

I would like to thank Professor Bo Yuan for his informative lectures and guidance through the project.

I would also like to thank laboratory teacher Yao Zhao for the inspiring suggestions to this project.

In addition, I would like to thank Teaching Assistants Tingjing Zhang, Yibo Tang and Xiang Yi for the tutorial sessions and patient question-answering.

REFERENCES

[1] Ke,Tang, Yi Mei, and Xin Yao. *Memetic algorithm with extended neighborhood search for capacitated arc routing problems.* IEEE Transactions on Evolutionary Computation 13.5 (2009): 1151-1166.