Publication P8

J. Martikainen and S. J. Ovaska
"Comparison of a fuzzy EP algorithm and an AIS in dynamic optimization tasks"
in
Proc. of the IEEE Mountain Workshop on Adaptive and Learning Systems
Logan, UT, 2006, pp. 7-12.

Comparison of a Fuzzy EP Algorithm and an AIS in Dynamic Optimization Tasks

Jarno Martikainen and Seppo J. Ovaska
Helsinki University of Technology
Institute of Intelligent Power Electronics
Espoo, FI-02015 Finland
E-mail: jkmartik@cc.hut.fi, ovaska@ieee.org

***Abstract*—In this paper we compare a specific evolutionary programming algorithm with a basic artificial immune system-based method in a dynamic combinatorial optimization task. Evolutionary algorithms are known to produce competitive results in optimization tasks, where only a single best solution is desirable. Artificial immune systems, however, can simultaneously find many different competitive solutions, and this property makes them an interesting choice in dynamic optimization environments. The performance of these two algorithms is compared using a non-parametric statistical framework that does not require any knowledge regarding the output distribution of the algorithms.**

I. Introduction

Biologically inspired computational methods have been studied for decades and remarkable results have been achieved using, among others, artificial neural networks, fuzzy logic, and evolutionary algorithms. Natural evolution offers us an ever-increasing number of metaphors to be transformed in computational procedures as we learn more and more about how nature itself works.

Artificial Immune System (AIS) [1, 2] is a relatively new computational approach to tackling demanding problems not conveniently solvable using traditional optimization techniques. AIS in static optimization problems have been studied up to some extent, but there seems to be little research on the performance of AIS in dynamic problems. From the optimization point of view, the AIS scheme shares many similarities with evolutionary algorithms (EAs) [3, 4]. It has, however, been observed that AIS has better capabilities of preserving the diversity in the solution pool and thus producing many good solutions instead of just one, as evolutionary algorithms usually do. People working with complex dynamic optimization problems can find the comparison between EAs and AIS-based algorithms interesting. The aim of this paper is to compare a standard AIS-based optimization method, a clonal selection algorithm, and a specialized evolutionary algorithm in optimizing a dynamic traveling salesman problem (DTSP). DTSPs have been recently studied using various soft computing methods, such as neural networks [5], evolutionary computation [6, 7, 8, 9], and ant colony optimization [10]. AIS has been applied for solving TSP before in [11], and it has also been applied for dynamic optimization in [12].

To conduct reliable and sound comparison between the candidate algorithms we employ a statistical scheme introduced in [13]. This framework relies on bootstrap resampling and multiple hypothesis testing procedures and requires the user to make no distributional assumptions regarding the algorithm's output. This is very beneficial since it is difficult, if not impossible, to define accurately the distribution of the output of an evolutionary algorithm.

Our paper is divided as follows. Section II describes the AIS-based system used in this paper and section III briefly explains the evolutionary algorithm used for comparison. In section IV the principle of the statistical comparison framework is explained and Section V deals with the TSP problems used in this paper. Section VI contains the results, Section VII examines the effect of parameters in the statistical framework, and Section VIII concludes the paper.

II. Artificial Immune Systems

Artificial immune system approach to optimization is more recent exploitation of natural phenomena in computer science than evolutionary algorithms. EAs and AISs share many common aspects, but whereas EAs try to model the natural evolution, AISs try to benefit from the characteristics of a human immune system. Basic algorithm for AIS-based optimization is called the Clonal Selection Algorithm (CSA) and it works as follows [1, 2].

1. Create *N antibodies* (candidate solutions) randomly to form an initial population.

2. Determine the *affinity* (fitness or cost) of each antibody.

3. Select n_b fittest antibodies to form a group N_b.

4. For each member of the group N_b, create a number of clones independently and proportionally to their affinity values. The better the affinity the higher the number of clones

1-4244-0166-6/06/$20.00 ©2006 IEEE.

generated for each of the selected antibodies. The clones form a group *C*.

5. The clone group *C* undergoes an *affinity maturation*, in which the clones are mutated inversely proportionally to their affinities: the better the affinity the smaller the mutation rate.

6. The affinities of the affinity-matured clones are calculated.

7. If an affinity-matured clone has a better affinity value than the parent antibody, replace the parent antibody with the affinity-matured clone.

8. Replace the *d* lowest affinity antibodies with randomly created new antibodies.

9. Go to 3 if run time constraints have not been met, otherwise exit.

In this paper the size of the antibody population was 30. Three best antibodies were selected for cloning during each generation and 5, 4, or 3 clones were created out of the selected antibodies based on their affinity values. At the end of each generation, the worst antibody was replaced with a randomly created new antibody. The choice of parameter values in this case is a trade-off between the algorithm's exploration and exploitation capabilities: larger initial population and more clones offer better exploration of the search space, but simultaneously the increased computational requirements slow down the exploitation pace of the best solutions. So, these algorithms are parameter-sensitive and the values used were found suitable for this paper based on extensive testing. Single mutation was applied on high affinity clones, whereas lower affinity clones suffered two separate mutations per generation.

III. Fuzzy Logic Controlled Evolutionary Programming Algorithm

Evolutionary algorithms have for long been credited as tools for solving challenging optimization problems. According to the notorious no free lunch (NFL) theorem [14], there does not exist a single optimization algorithm that would deliver superior performance in a wide variety of problem types. Thus, to get most out of EAs performance, the algorithms need to be specifically tailored to the task at hand. In this paper, we use a specialized evolutionary programming algorithm specifically designed for solving large-scale TSPs. The details of the algorithms can be found in [15]. In general, the algorithm works as follows.

1. Create *N* solutions randomly to form an initial population.

2. Evaluate the cost of each solution.

3. Operate the following stages for 5 generations at a time (i.e. G_w and G_d equal 5, see Fig. 1).

Stage 1:

1. Create a single offspring per parent through a mutation mechanism.

2. Evaluate the cost of each offspring.

3. Select the fittest half of the parents and offspring to form the population for the next generation.

Stage 2:

1. Select N_p best solutions for decomposition and divide these solutions into N_d sub solutions.

2. Create N_o offspring for each sub solution through a mutation mechanism.

3. Replace the parent with offspring if the cost of the offspring is less than that of the parent.

4. Go to 3 if run time requirements have not been met, otherwise exit.

Fuzzy logic is used to control the parameters N_p, N_o, and N_d. Two inputs for both the fuzzy systems are used. These inputs describe the state of the solution population. We define dynamics *D* as the difference between f_b, the best, and f_w, the worst solution in the current population:

$$D(n) = f_b(n) - f_w(n) \tag{1}$$

The second input to the fuzzy systems is D_{std}, the standard deviation of three consecutive *D*s:

$$D(n)_{std} = \mathrm{STD}\left[D(n) \quad D(n-1) \quad D(n-2)\right] \tag{2}$$

The rule base for determining the number of best individuals selected for partitioning, N_p, and the number of offspring created out of each selected parent, N_o, is shown in Table I and the defuzzified output is then transformed into N_p and N_o using Table II. In the rule base tables *VS*, *S*, *M*, *L*, and *VL* stand for very small, small, medium, large, and very large. The actual membership functions are defined in [15].

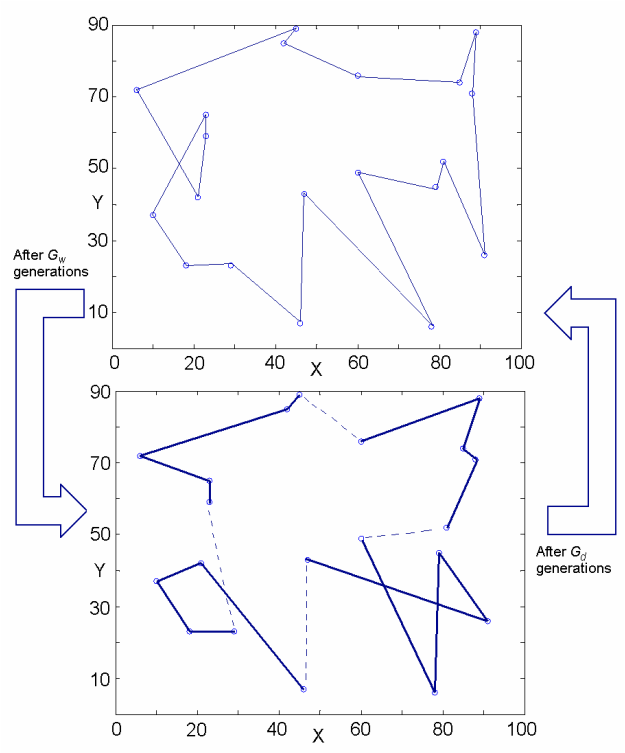

Fig 1. The principle of fuzzy logic controlled evolutionary programming algorithm.

TABLE I
RULE BASE FOR DETERMINING N_p AND N_o

Dynamics/Std	VS	S	M	L	VL
S	M	VS	VS	S	S
M	VS	S	S	M	M
L	VS	S	M	M	M
VL	S	M	M	L	VS

TABLE II
DEFUZZIFIED OUTPUT INTERVALS AND CORRESPONDING VALUES FOR N_p AND N_o

Defuzzified output	N_p	N_o
≥ 0 and < 0.2	1	30
≥ 0.2 and < 0.4	3	10
≥ 0.4 and < 0.6	5	6
≥ 0.6 and < 0.8	6	5
≥ 0.8 and ≤ 1	10	3

The rule base for determining the number of sub paths, N_d, is shown in Table III and the defuzzified output is the transformed into N_d Table IV.

TABLE III
RULE BASE FOR DETERMINING N_d

Dynamics/Std	VS	S	M	L	VL
S	S	L	M	M	M
M	M	M	L	L	VL
L	M	M	L	VS	VS
VL	M	L	VL	VS	VS

TABLE IV
DEFUZZIFIED OUTPUT INTERVALS AND CORRESPONDING VALUES FOR N_d

Defuzzified output	N_d
≥ 0 and < 0.25	5
≥ 0.25 and < 0.5	3
≥ 0.5 and < 0.75	1
≥ 0.75 and ≤ 1	1

The initial population size of 30 individuals was used for this algorithm. Both the fuzzy EP and CSA are rather sensitive to parameter values. To make the comparison of the algorithms objective, the best effort was made to find competitive parameters for the test schemes used in this paper.

IV. Framework for Statistical Comparison

The soft computing field is rather young, and it lacks generally accepted guidelines for statistical testing of the differences between the performances of two or more algorithms. A few tens of runs using different initial populations for each run and then comparing the achieved average fitnesses is not enough to reliably state one algorithm to perform better than another. Also, the much-used student's *t*-test may deliver misleading results if the user neglects the requirement of normal distribution for the input samples of the test. After all, it is difficult, if not impossible, to estimate the output distribution of an evolutionary algorithm. Detailed description of the statistical framework for proper comparison of two optimization algorithms can be found in [13]. However, in the following the procedure is briefly explained.

The applied statistical comparison scheme has two separate parts: first as much as possible data is collected and then, based on this data, multiple testing procedure is used to see if null hypotheses can be rejected. The null hypotheses have to be formulated beforehand, and in this case we select the null hypothesis to be that there exists no performance difference between the two algorithms. This means that if we reject the null hypothesis the opposite is true, that is, there exists difference in the performance of the two algorithms. It is appropriate to collect as much data as possible, so for each two algorithms to be compared, 50 random initial populations are created and 50 one-minute runs for each algorithm per initial population are conducted. So, for a single algorithm the calculations required 2500 minutes per algorithm. The number of null hypothesis is also 50, a single hypothesis for each of the initial populations. The amount of data collected was decided based on the available computational power.

After this, test statistics are calculated for the observed values as in [13]. The data is then resampled using bootstrap resampling technique and test statistics are calculated for the resampled values. The tests produce as a result adjusted *p*-values that define the minimum confidence level to reject an individual hypothesis. In this paper a confidence level of 0.05 was used to reject an individual hypotheses and the Bonferroni [13] testing procedure was applied. This procedure ranks in the middle when comparing different multiple testing procedures in terms of conservativeness.

V. Static and Dynamic TSP

The traveling salesman problem is a well-known combinatorial optimization problem. In brief, the goal in TSP is to find a shortest path connecting a number of given cities. In static TSPs the distances between the cities stay the same throughout the runtime of an algorithm. In dynamic TSPs the connections between the cities can be expressed for example as costs and these costs vary as a function of time [16]. Real-world DTSPs can include, among others, routing problems in which the cost or time to take a certain route from a place to another varies with time, e.g., roads may be blocked or ticket prices may change.

To evaluate the performances of a specialized EA and CSA in static optimization problem a 225-city tsp225 from TSPLIB [17] was used. This problem has a known minimum path length at 3919. However, in this case we were not primarily interested in the minima the two algorithms would eventually achieve, rather, each algorithm was given one minute of CPU time on MATLAB 6 software and a P4 (2.6 GHz) processor to find as short a path as possible.

In [9] Kang et al. correctly stated that the changes in the cost basis for evaluating dynamic TSP should change rapidly, for if the changes are too far apart the problem could be separated into independent sub problems. Two different dynamic schemes were used to evaluate the performances of the candidate algorithms. Figures 7a and 7b explain the structure of the test cases. In the first case the initial cost matrix changes every 10 seconds. The new cost matrices are not completely random, but a significant random component is added to the initial cost matrix. In the second case there are two stages of different length: 20 and 30 seconds separated by a 5 second interval. The test schemes were designed to test the performance of the algorithms with rapidly changing conditions with different intervals. The first scheme has constant intervals for optimization whereas in the second scheme the time available for optimization varies from 5 to 30 seconds.

For mutation, a single exchange of cities, a multiple-city block relocation or reversal could be implemented with equal probability in both CSA and fuzzy EP.

VI. Results

A. Static TSP

The results were evaluated based on the average final tour lengths of the algorithms. The performance difference between the algorithms was obvious as shown in Fig. 2, where the fuzzy EP clearly outperforms the CSA. This notion was confirmed by the statistical frameworks, since all of the 50 null hypotheses were rejected with the confidence of 0.95 (α=0.05).

To confirm the notion of the fuzzy EP outperforming the CSA, we conducted the statistical test described in Section IV. The time used for optimization was not enough to achieve the global minimum of the problem, but in this paper the emphasis was on the dynamic behaviour of the cost function, i.e., the capabilities of the algorithms in rapidly changing environments with a small amount of time for optimization were studied.

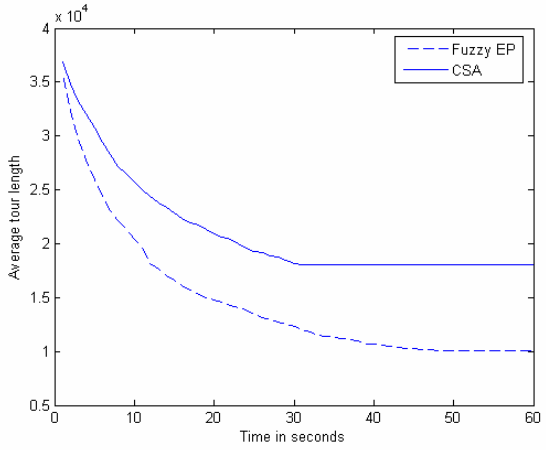

Fig. 2. The average performance of the candidate algorithms in the static test scheme.

B. Dynamic TSP

Figure 3 shows the average performance in the case of the first dynamic test scheme. The fuzzy EP does slightly better during the first cost matrix, and as the first change occurs, the algorithms perform rather similarly. After consecutive changes in the cost matrix, however the CSA outperforms the fuzzy EP algorithm.

The number of rejected hypotheses in Table VI confirms the notion that at the start the fuzzy EP performs slightly better than the CSA. As Table VI shows, 23 out of 50 hypotheses are rejected. This is almost half of the hypotheses and it gives some indication to the fact that the fuzzy EP could be performing better. At the 20 second mark, however, none of the hypotheses could be rejected, so it is reasonable to consider the two algorithm's performance similar. From 30 seconds onward nearly all the hypotheses are rejected and this indicates that the CSA clearly outperforms the fuzzy EP.

Figure 4 shows the average performance characteristics of the candidate algorithms in the second dynamic scheme. Similar to the first dynamic scheme, the fuzzy EP seems to be doing better than the CSA in the early stages of the algorithm, that is, until the first change of the cost matrix. It is safe to say that the fuzzy EP outperforms the CSA at the time of the first change since all of the 50 null hypotheses are rejected. From then on, nearly all the hypotheses are rejected and the CSA outperforms very reliably the fuzzy EP. The number of rejected null hypotheses is presented in Table VI.

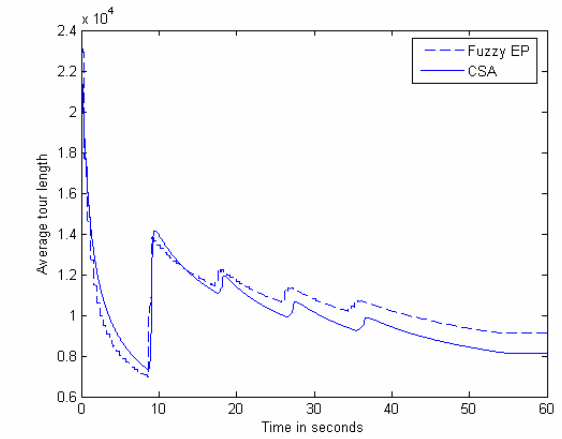

Fig. 3. The performance of the candidate algorithms on the first dynamic environment.

TABLE VI
REJECTED HYPOTHESES IN THE DYNAMIC SCHEMES

First Dynamic Scheme		Second Dynamic Scheme	
Time [s]	Rejected hypothesis	Time [s]	Rejected hypothesis
10	23	5	50
20	0	25	49
30	49	30	50
40	50	60	50
50	50		
60	50		

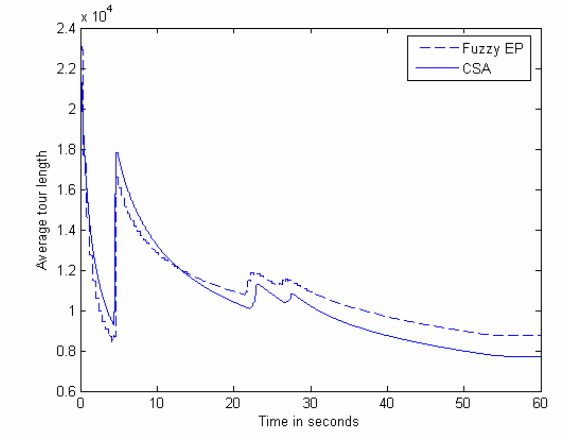

Fig. 4. The performance of the candidate algorithms on the second dynamic environment.

VII. Effect of Parameters of the Statistical Scheme

The statistical framework used to evaluate the results in this paper contains a few selectable parameters. The effect of two of them, the number on initial populations and the number of runs per initial population were studied. Figures 5 and 6 show the percentage of rejected hypotheses as the function of initial populations and runs per population.

In statistical testing the more data we have the better. In Fig. 5, small number of runs per initial population gives different results than using a larger number of runs. This indicates that using a small amount of runs one could easily make false conclusions about the relative performances of the algorithms. This is the case especially in situation where the difference in performances of the algorithms is not clear. Figure 5 was calculated using the values from the first dynamic optimization scheme at 10 seconds. For 50 initial populations and 50 runs each population 23 of the initial 50 null hypotheses were rejected, as shown in Table VI.

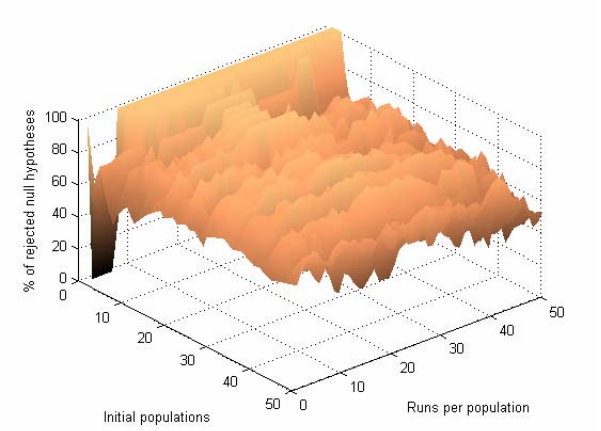

Fig. 5. The effect of parameter values in the statistical framework on the percentage of rejected hypotheses in a case where the two algorithms perform quite similarly.

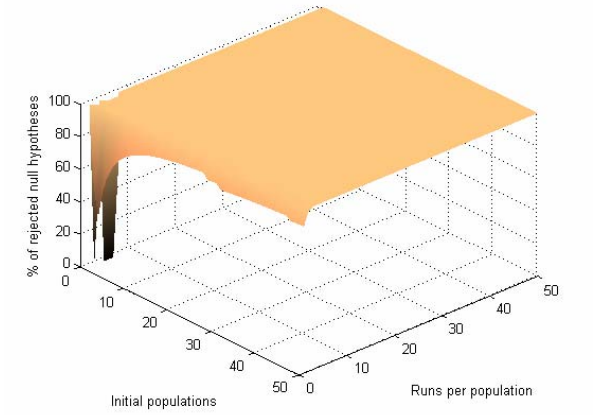

Fig. 6. The effect of parameter values in the statistical framework on the percentage of rejected hypotheses in a case where the two algorithms perform very differently.

In Fig. 6 the effect of parameters is shown in case in which the performance difference is clearer. Even at small number of runs a major number of the null hypotheses can be rejected and solid conclusion can be made. Figure 6 was calculated using results from the first dynamic scheme at 60 seconds. As shown in Table VI, for 50 initial populations and 50 runs each population, all the 50 initial null hypotheses were rejected.

VIII. Conclusions

In this paper, we have studied the performances of an application specific EA and a standard CSA in dynamic envi-

ronments. Based on the results it seems that the specialized EA is capable of outperforming the CSA in static environment. Certainly, EAs usually target for a single best solution and the whole population is structured online to achieve the best possible single solution. Then again, CSAs emphasize the goal of achieving multiple good solutions simultaneously. In static problems this means that in CSA the computational resources have to be divided between multiple competing good solutions, whereas in EAs all the computational resources benefit the currently best solution.

Aiming for a single best solution has drawbacks, the most critical of which is the degradation of the population diversity. This can be clearly seen in the results of the fuzzy EP in dynamic environments: the initial population covers the search space up to some degree, but after running for a while the population has biased considerably and it is difficult for the EAs to explore the search space rapidly. Certainly, EAs can find the new optima after a while, but this requires time since the biased population needs more diversity before such new optima can be found. In CSA-based systems a degree of diversity is contained in memory cells. This means that when the cost matrix changes the CSA has good chances of having a reasonably good solution to start the search from even without a sudden increase of diversity.

In this paper, a statistical framework was used to confirm the results. This non-parametric testing scheme has a few parameters and their effect was studied in terms of the number of initial populations and the number of runs per initial population. The more data we have the more confident we can be about the results. Analysis of the parameter values shows that using too few samples to draw conclusions may cause erroneous results.

ACKNOWLEDGMENT

This research work was funded by the Academy of Finland under Grant 214144.

REFERENCES

[1] L. N. de Castro and J. Timmis, *Artificial Immune Systems: A New Computational Intelligence Approach*. London, UK: Springer Verlag, 2002.

[2] L. N. de Castro and F. J. Von Zuben, "Learning and optimization using the clonal selection principle," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 3, 2002, pp. 239-251.

[3] T. Bäck, *Evolutionary Algorithms in Theory and Practice*. New York, NY: Oxford University Press, 1996.

[4] D. B. Fogel, *Evolutionary Computation, Toward a New Philosophy of Machine Intelligence*. Piscataway, NJ: IEEE Press, 2000.

[5] K. Shinozawa, T. Uchiyama, and K. Shimohara, "An approach for solving dynamic TSPs using neural networks," in *Proc. of the IEEE International Joint Conference on Neural Networks*, Singapore, 1991, vol. 3, pp. 2450-2454.

[6] A. Zhou, L. Kang, and Z. Yan, "Solving dynamic TSP with evolutionary approach in real time," in *Proc. of the Congress on Evolutionary Computation*, Canberra, Australia, 2003, vol. 2, pp. 951-957.

[7] Z.-C. Huang, X.-L. Hu, and S.-D. Chen, "Dynamic traveling salesman problem based on evolutionary computation," in *Proc. of the Congress on Evolutionary Computation*, Seoul, Korea, 2001, vol. 2, pp. 1283-1288.

[8] X.-S. Yan, L.-S. Kang, Z.-H. Cai, and H. Li, "An approach to dynamic traveling salesman problem," in *Proc. of the International Conference on MachineLearning and Cybernetics*, Shanghai, China, 2004, vol. 4, pp. 2418-2420.

[9] L. Kang, A. Zhou, B. McKay, Y. Li, and Z. Kang, "Benchmarking algorithms for dynamic traveling salesman problems," in *Proc. of the Congress on Evolutionary Computation*, Portland, OR, 2004, vol. 2, pp. 1286-1292.

[10] R. Montemanni, L. M. Gambardella, A. E. Rizzoli, and A. V. Donati, "A new algorithm for a dynamic vehicle routing problem based on ant colony system," *Technical Report* IDSIA-23-02, Instituto Dalle Molle di Studi sull'Intelligenza Artificiale, Switzerland, 2003.

[11] S. Endo, N. Toma, and K. Yamada, "Immune algorithm for n-TSP," in *Proc. of the IEEE International Conference on Systems, Man, and Cybernetics*, San Diego, CA, 1998, vol. 4, pp. 3844-3849.

[12] Z. Li, J. Wu, and Z. Mao, "Application of artificial immune algorithm in the dynamic zoning of elevator traffic," in *Proc. of the 5th World Congress on Intelligent Control and Automation*, Hangzhou, China, 2004, vol. 3, pp. 2222-2226. In Chinese.

[13] D. Shilane, J. Martikainen, S. Dudoit, and S. J. Ovaska, "A general framework for statistical performance comparison of evolutionary computation algorithms," In *Proc. of the IASTED International Conference on Artificial Intelligence and Applications*, Innsbruck, Austria, 2006, pp. 7-12.

[14] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, 1997, pp. 67-82.

[15] J. Martikainen and S. J. Ovaska, "Using fuzzy evolutionary programming to solve traveling salesman problems," in *Proc. of the 9th IASTED International Conference on Artificial Intelligence and Soft Computing*, Benidorm, Spain, 2005, pp. 49-54.

[16] A. Zhou, L. Kang, and Z. Yan, "Solving dynamic TSP with evolution ary approach in real time," in *Proc. of the Congress on Evolutionary Computation*, Canberra, Australia, 2003, vol. 2, pp. 951-957.

[17] University of Heidelberg, Department of Computer Science, *Traveling Salesman Problem library, TSPLIB*. Cited 03/15/06. Available at <http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/>.

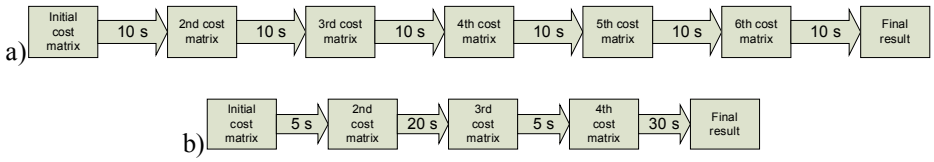

Fig 7. The dynamic schemes used for testing.