Publication P6

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.

USING FUZZY EVOLUTIONARY PROGRAMMING TO SOLVE TRAVELING SALESMAN PROBLEMS

Jarno Martikainen
Helsinki University of Technology
Institute of Intelligent Power Electronics
P. O. Box 3000 FIN-02015 HUT
FINLAND
E-mail: jkmartik@cc.hut.fi
URL: http://powerelectronics.hut.fi

Seppo J. Ovaska
Helsinki University of Technology
Institute of Intelligent Power Electronics
P. O. Box 3000 FIN-02015 HUT
FINLAND
E-mail: ovaska@ieee.org
URL: http://powerelectronics.hut.fi

ABSTRACT
In this paper we present an efficient decomposition technique to speed up evolutionary algorithms when dealing with large scale optimization problems. Divide and conquer methods aim to solving problems in smaller entities and then combining the sub-solutions to form complete solutions. Often the optimal way to divide the problem varies as the evolutionary algorithm proceeds, thus making a static decomposition not the best approach. In this paper, we present a fuzzy controlled decomposition algorithm and show how it outperforms a traditional evolutionary algorithm.

KEY WORDS
Evolutionary computation, traveling salesman problem, fuzzy logic, divide and conquer.

1. INTRODUCTION

Intuitively, humans tend to cluster large problems into several sub-problems that are solved separately and these solutions are then combined to form the overall problem solution. Especially different research and development processes are iterative in nature, so that at first a picture of the whole problem is formulated after which the problem is divided into less demanding sub-problems. Further, the sub-problems are elaborated separately after which the parts are connected and the problem is inspected as a whole. After a while, the problem can be fine tuned in smaller entities and the elements can be combined to produce a solution to the whole problem and so on. This kind of iterative loop can continue as long as a satisfactory solution is found. We rarely tackle a large problem as a whole, but split it into pieces of suitable size. These kind of divide and conquer methods are not new in the area of evolutionary algorithms (EAs) either [1].

In this paper, we introduce an efficient divide and conquer method that throughout the run time of the algorithm in turns optimizes the whole problem and the very same problem in less demanding sub-problems. Our goal is to develop an algorithm that is more reliable, converges faster, and is persistent to stagnation. The introduced algorithm has various tunable parameters, so after being convinced that our algorithm outperforms a competitive reference algorithm, we construct a fuzzy control mechanism to adaptively adjust the parameters during the runtime.

As a test case we use the traveling salesman problem (TSP). Traveling salesman problem (TSP) is a well known and a much studied minimization problem [1,2,3]. The problem describes a salesman who has to visit a certain number of cities without having to visit the same city twice, in other words, the shortest closed path connecting all the cities has to be found. A number of common problems, among others, printed circuit board design [4] and robot path planning [5], can be converted into a TSP. A 20-city map for the TSP is shown in Fig 1. The benchmark test in this paper was a demanding 500-city TSP.

Our paper is structured as follows. Section 2 discusses evolutionary programming and introduces our approach. Section 3 presents the simulation results. Section 4 introduces fuzzy logic based control to our decomposition algorithm and section 5 concludes the article.

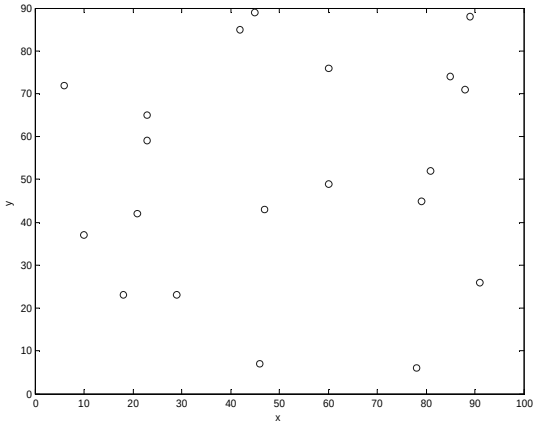

Fig 1. A 20-city TSP example.

2. EVOLUTIONARY PROGRAMMING

Evolutionary programming (EP) is a branch of evolutionary computing along with genetic algorithms (GAs) and evolution strategies (ESs) [6,7]. Each method, quite similar to one another, tries to find an optimal solution by ap-

plying basic operators, such as selection, mutation and reproduction, to a pool of candidate solutions. The structure of an evolutionary programming based algorithm is formulated in [6]:

1. Create a random initial population.

2. Mutate each solution to produce an offspring.

3. Select the fittest half of all the parents and offspring to become parents for the next generation.

4. Got to 2 if exiting conditions have not been met.

In our case, solutions were coded as an array, and each city was assigned an individual number between 1 and 500. Cities were visited in the order they appeared in the solution array and the fitness of a solution was the total path length visiting all the cities.

Three different types of mutations were allowed [6]. Each mutation took place with equal probability. First type of mutation resulted in a simple exchange of the places of two cities in the solution array. Second type of mutation selected a random number of consecutive cities and reversed their order in the solution array. Third type of mutation selected a random number of consecutive cities and moved this block to a random location. A single solution could be mutated only once using one type of mutation per generation.

The method described above was used as a reference algorithm for our study. To outperform the performance of the reference approach we created two divide and conquer – type approaches to improve the convergence characteristics of the reference approach.

2.1. GEOGRAPHICAL DECOMPOSTION ALGORITHMS

Our Geographical Decomposition Algorithm is based on geographically decomposing the traveling salesman problem. These kinds of decomposition and clustering algorithms have been studied extensively in the literature [8]. After the initialization, the map containing all the cities was divided into four equally large areas, as shown in Fig. 2. The reference evolutionary programming algorithm was then used to find as short as possible a path connecting all the cities within each area. The division into four separate areas lasted for 5000 generations, after which the individual parts were connected and the algorithm was run further 45 000 generations according to the principle of the reference algorithm described above. The simulations were carried out using MATLAB 6 software and a P4 (2.6 GHz) processor. Average run time for a single 50 000 run was three hours.

2.2 REPETITIVE DECOMPOSITION ALGORITHM

Geographical decomposition is quite intuitive, but speeds up the convergence of the reference algorithm only in the beginning. Therefore, in our Repetitive Decomposition Algorithm (RDA), we optimize the TSP in turn as a single route and in turn as several sub-routes. The principle of Continuous Decomposition is shown in Fig. 3.

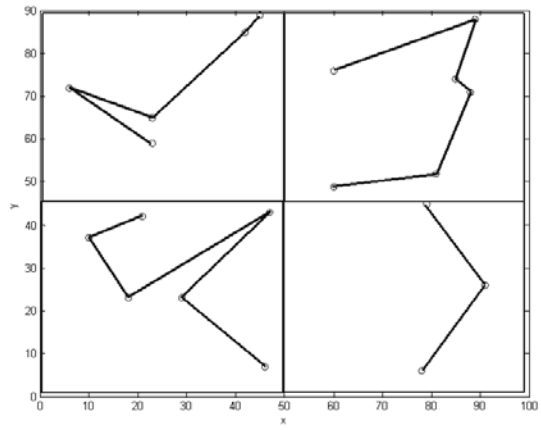

Fig 2. The principle of geographical decomposition.

At first the solution pool is initialized. In this case we use the population size of 20 individuals. After initialization, the reference algorithm is applied to the whole population for G_w generations. In this phase, each solution produces one offspring per generation.

After this the population is decomposed into N_d equally long paths, and these are optimized separately for G_d generations. Since it is important to keep the number of solution evaluations constant throughout the run time of the algorithm, we must select N_p best parents from each path and produce N_o offspring for each of them. The product of N_p and N_o has to equal 20 in order to keep the number of solution evaluations constant all the time. So, when the problem is decomposed into les demanding sub-problems, only a few solutions are selected, but they altogether produce the same amount of offspring as when the problem was dealt as a whole.

In short, our algorithm divides a few of the best solutions containing all the cities into shorter paths with smaller number of cities and tries to optimize these parts separately before composing them together again. The algorithm is also elitist in the sense, that combining the smaller part to form a complete solution cannot produce longer path than the one that was originally decomposed.

The RDA has a variety of tunable parameters and the effect of different values and combinations was experimented.

3. SIMULATION RESULTS

All the algorithms were run using an initial population size of 20 found suitable using extensive testing and taking into account the run time requirements. This means that our algorithms evaluated 20 solutions per generation. Also, each algorithm was run 15 times for 50 000 generations to ensure some statistical reliability. The averaged results of the reference algorithm and the Random Decomposition algorithm are shown in Table 1.

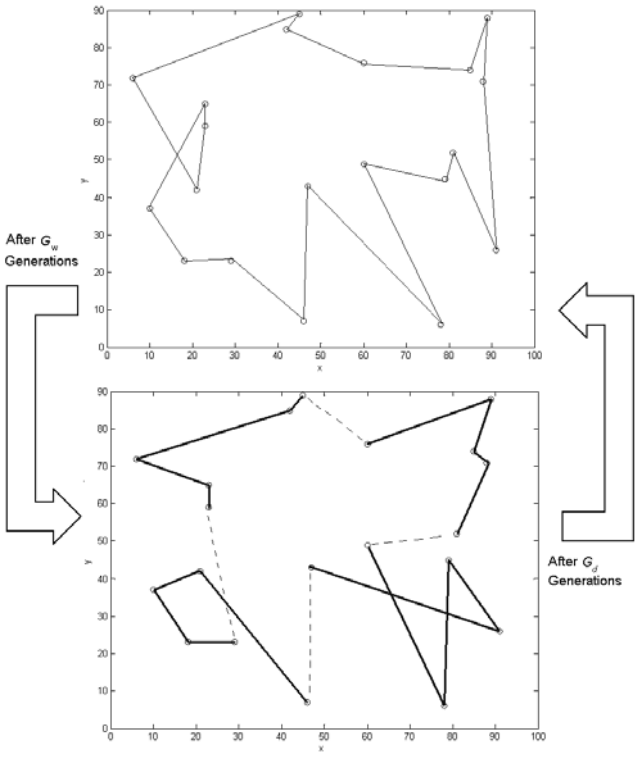

Fig 3. Example of Repetitive Decomposition Algorithm with N_d=4.

Table 1. The reference approach performance

Algorithm	Average	Std
Reference algorithm	2393.11	40.99
Geographical Decomposition	1834.98	17.80

The Geographical Decomposition algorithm very reliably outperforms the reference approach in both terms of average value and standard deviation. Similar performance results are also documented in the literature [8].

The performance of the RDA is shown in Table 2. The averages and standard deviations suggest that the Continuous decomposition algorithm is superior to the algorithms presented previously in this paper. This, however, is not a surprise since the divide and conquer principle is applied to the problem throughout the run time of the algorithm, where as in the previously discussed algorithms it is implemented only once after initialization.

Our experiments suggest that the decomposition of the solutions should take place quite frequently for a short time. Optimizing the whole problem for five generations and then decomposing it for another five generations outperformed 25 and 50 generation intervals.

Selecting only a few best solutions to be decomposed and producing several offspring for these few parents outperformed the option of selecting more parents and producing fewer offspring.

Finally, in our tests, the number of decompositions was not found so critical, all the options performing in a relatively similar way.

Table 2. The Continuous decomposition algorithm performance.

G_w	G_d	N_d	N_p	N_o	Average	Std.
5	5	10	5	4	1698.92	24.60
25	25	10	5	4	1709.55	19.07
50	50	10	5	4	1721.59	23.26
20	20	10	1	20	1705.96	24.76
20	20	10	4	5	1729.80	28.85
20	20	10	10	2	1734.09	30.24
20	20	5	5	4	1714.87	16.59
20	20	10	5	4	1711.84	32.31
20	20	25	5	4	1713.81	25.61

Despite the final results shown in Table 2, the convergence characteristics of the algorithm using different parameter sets varied quite interestingly.

Figs. 4 and 5 show the convergence characteristics of different decomposition interval settings G_w and G_d. It is clearly visible from Figs. 4 and 5 that setting G_w=5 and G_d=5 outperforms other parameter sets throughout the entire optimization run. So, our results suggest that optimizations should be decomposed into smaller sub problems quite frequently and for a short time.

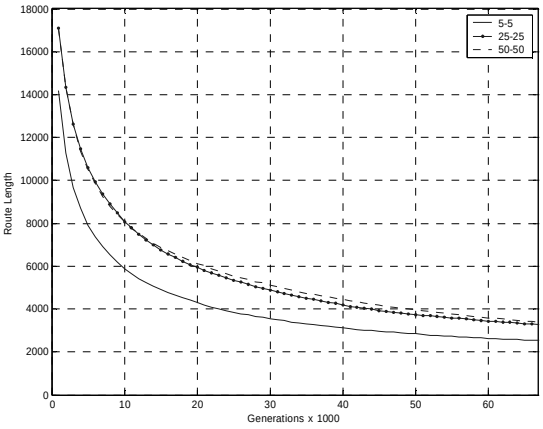

Fig 4. Convergence characteristics of RDA using different G_w and G_d. (Solid line: G_w=5 and G_d=5, dash dotted line: G_w=25 and G_d=25, dashed line: G_w=50 and G_d=50).

Figs. 6 and 7 show how the RDA converges using different parameter settings for N_d. It is noticeable that in the

beginning all the parameter settings perform almost equally well. Using extensive testing, we came to conclusion that in the very early stages of the algorithm, the whole problem should be divided into very small pieces. After the algorithm has converged for a short while, there should be medium size decompositions, as Fig. 6 shows. Finally towards the end, in Fig 7. larger decompositions seem to have the best convergence.

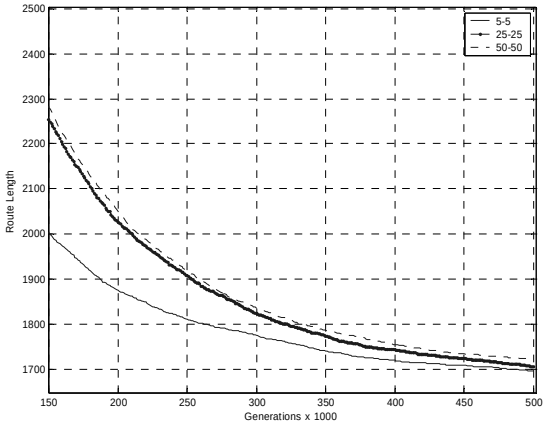

Fig 5. Convergence characteristics of RDA using different G_w and G_d. (Solid line: G_w=5 and G_d=5, dash dotted line: G_w=25 and G_d=25, dashed line: G_w=50 and G_d=50).

Fig. 8 shows the convergence characteristics using different values for N_p and N_o. Fig. 8 clearly shows that by selecting a few promising solutions to be decomposed and producing a couple of offspring for each of them produces fast convergence in the beginning. However, after a while, when the algorithm has converged more, selecting only the best solution and producing the maximum number of offspring for it produces better performance towards the end. This, also, is quite intuitive. Early in the algorithm we cannot be sure which candidates eventually lead to good solutions, so we should try a few of them. When the algorithm has converged for some time, we know that, for this run, the best candidate is likely to produce better solutions than the other candidates. Thus, we should concentrate on the best solution when decomposing the problem.

4. FUZZY REPETITIVE DECOMPOSITION ALGORITHM

Since having different parameter sets produces different performance, it is tempting to develop an adaptive system to control the RDA. Fuzzy logic [9] is known to be capable of transforming human intuition easily into control structures. In the following, we construct a fuzzy logic based adaptive control structure [10,11], FRDA, to enhance the convergence of the standard RDA.

Fuzzy logic system controls three RDA parameter, N_d, N_p, and N_o. Since G_w=5 and G_d=5 produced superior performance in Table 2, there is no need to adjust these parameters on-line. We have altogether two fuzzy controllers in the system, one controlling N_d, and the other controlling both N_p and N_o.

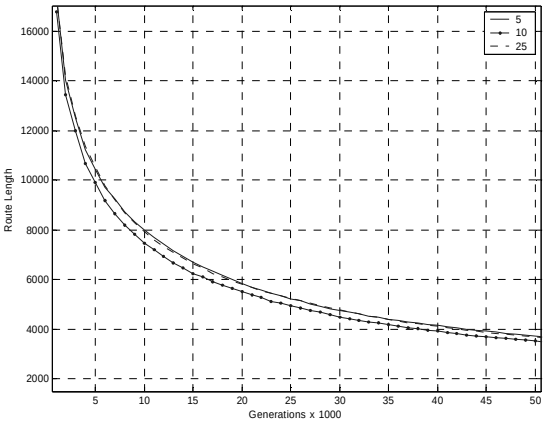

Fig 6. Convergence characteristics of RDA using different N_d. (Solid line: N_d =5, dash dotted line: N_d =10, dashed line: N_d =25).

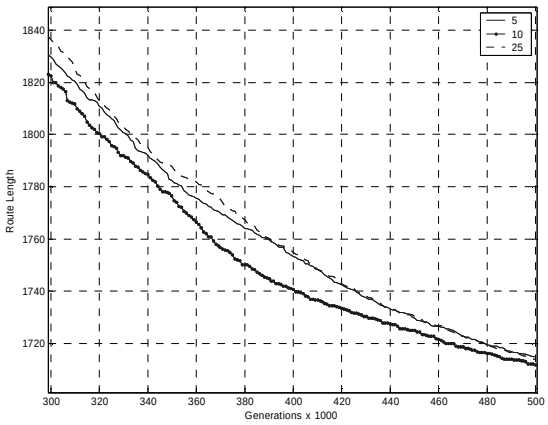

Fig 7. Convergence characteristics of RDA using different N_d. (Solid line: N_d =5, dash dotted line: N_d =10, dashed line: N_d =25).

We use two inputs for both the fuzzy systems. 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:

$$D(n) = f_b(n) - f_w(n) \quad (1)$$

In addition, D is averaged over three consecutive generations to smooth rapid changes. D tends to be large when the population has not yet converged so much towards the final solution. As the population converges D decreases. If D equals to zero, all the solutions in the population are the same.

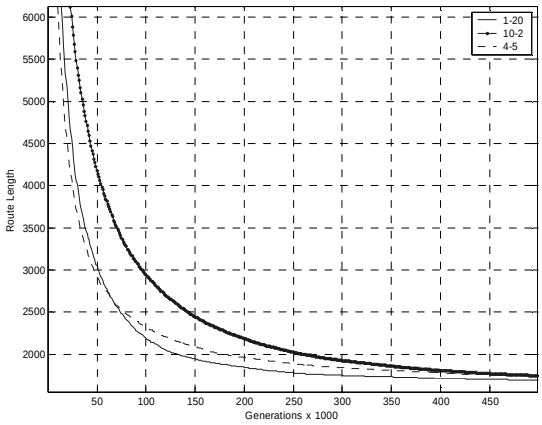

Fig 8. Convergence characteristics of RDA using different N_p and N_o. (Solid line: N_p and N_o=5, dash dotted line: N_p and N_o =10, dashed line: N_p and N_o =25).

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

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

where "STD" is a standard deviation operator. D_{std} also describes the state of the solution population. Not yet converged population has a higher D_{std} than a population that has somewhat converged.

The input membership functions for D are shown in Fig. 9 and for D_{std} in Fig. 10.

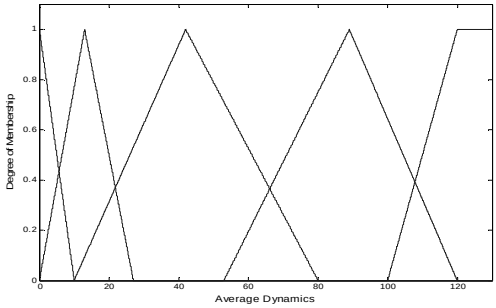

Fig. 9. The input membership functions for D from left to right: *VS, S, M, L, VL*.

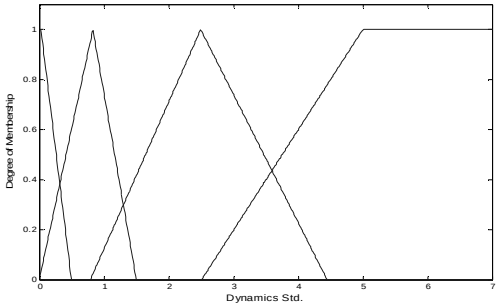

Fig. 10. The input membership functions for D from left to right: *S, M, L, VL*.

The used output membership functions are shown in Fig. 11.

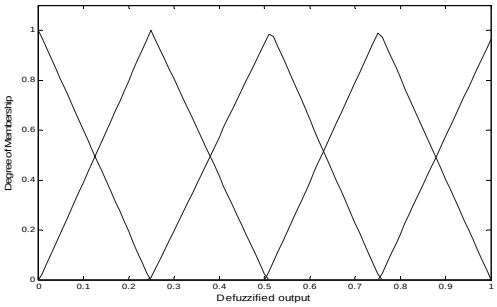

Fig. 11. The output membership functions for D from left to right: *VS, S, M, L, VL*.

The rule base for determining N_p and N_o is shown in Table 3 and the defuzzified output is then transformed into N_p and N_o using Table 4.

Table 3. 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 4. Defuzzified output intervals and corresponding values for N_p and N_o.

Defuzzified output	N_p	N_o
≥ 0 and < 0.2	1	20
≥ 0.2 and < 0.4	2	10
≥ 0.4 and < 0.6	4	5
≥ 0.6 and < 0.8	5	4
≥ 0.8 and ≤ 1	10	2

The purpose of the rule base in Table 3 is to a use small amount of parents and a lot of offspring when the algorithm has not yet converged at all. When some convergence has taken place, the number of parents in the decomposition is increased and the number of offspring is decreased. When the algorithm has converged a lot, again, only a few parents are used.

The rule base for determining N_d is shown in Table 5 and the defuzzified output is the transformed into N_d Table 6.

Table 5. 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 6. Defuzzified output intervals and corresponding values for N_p and N_o.

Defuzzified output	N_d
≥ 0 and < 0.25	25
≥ 0.25 and < 0.5	10
≥ 0.5 and < 0.75	5
≥ 0.75 and < 0.1	2

Here, purpose of the rule base in Table 5 is to use small sub-problems when the algorithm is still at its early stages. After some convergence, medium sub-problems are used. Finally, towards the end large sub problems are used. Towards the end the algorithm may suffer stagnation, so when D and D_{std} are both very low, medium sub problems are used to relieve the situation.

As the results in Table 7 show, the fuzzy controlled RDA outperforms the standard RDA in terms of average shortest path over 15 runs. Fig. 12 shows the improved convergence performance of the FRDA compared to that of the RDA:

Table 7. The reference approach performance.

Algorithm	Average	Std
RDA	1698.92	24.59
FRDA	1679.89	33.39

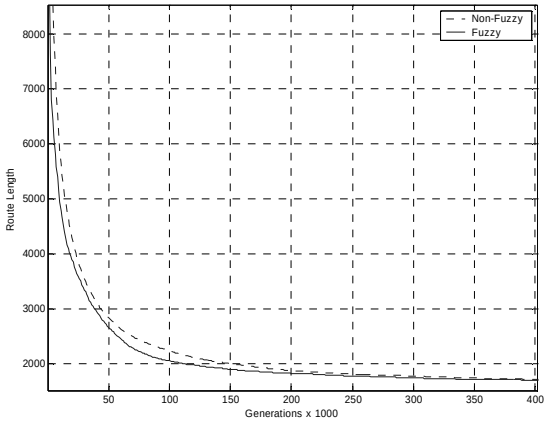

Fig.12. Comparison of the fuzzy and non-fuzzy RDA's convergence characteristics.

5. CONCLUSION

In this paper we have presented a new divide and conquer based evolutionary algorithm, RDA. To our best knowledge, this kind of repetitive decomposition/composition algorithm has newer been used before. In this paper we also presented a fuzzy controlled RDA, the FRDA. In our approach fuzzy controller adjusts the decomposition parameters based on dynamics characteristics of the solution population. By adaptively changing the decomposition parameters our algorithm is capable of producing competitive results in TSP optimization compared to an algorithm using static parameters.

REFERENCES

[1] C.L. Valenzuela & A.J. Jones, A parallel implementation of evolutionary divide and conquer for the TSP. *Proc. First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, 446*, Galesia, Sep 1995, 499 – 504.

[2] H.-K. Tsai, J.-M. Yang & C.-Y. Kao, Solving traveling salesman problems by combining global and local search mechanisms. *Proc. 2002 Congress on Evolutionary Computation, 2*, May 2002, 1290 – 1295.

[3] H. Watabe & T. Kawaoka, Application of multi-step GA to the traveling salesman problem. *Proc. Fourth International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies*, vol. 2, Honolulu, HI, 2000, 510 – 513.

[4] K. Fujimura, O.-C. Kwaw & H. Tokutaka, Optimization of surface component mounting on the printed circuit board using SOM-TSP method. *Proc. 6th International Conference on Neural Information Processing*, Perth, Australia, 1999, 131 – 136.

[5] W. Sheng, N. Xi, M. Song & Y. Chen, Optimization in automated surface inspection of stamped automotive parts. *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2, Lausanne, Switzerland, 2002, 1850 – 1855.

[6] D. B. Fogel, *Evolutionary computation, toward a new philosophy of machine intelligence* (Piscataway, NJ: IEEE Press, 2000).

[7] T. Bäck, T.: *Evolutionary algorithms in theory and practice* (New York, NY: Oxford University Press, 1996).

[8] N. Aras, I.K. Altinel, J. Oommen, A Kohonen-like decomposition method for the Euclidean traveling salesman problem-KNIES_DECOMPOSE. *IEEE Transactions on Neural Networks 14*(4), Jul 2003, 869 – 890.

[9] L. Wang, *A course in fuzzy systems and control* (Upper Saddle River, NJ: Prentice-Hall International, 1997).

[10] S. McClintock, T. Lunney, A. Hashim, A Fuzzy Logic Controlled Genetic Algorithm Environment. *Proc. IEEE International Conference on Computational Cybernetics and Simulation*, vol. 3, Orlando, FL, 1997, 2181 – 2186.

[11] K. Pytel, G. Kluka, A. Szymonik, Fuzzy methods of driving genetic algorithms. *Proc. Fourth International Workshop on Robot Motion and Control*, Puszczykowo, Poland, 2004, 339 – 343.