A COMPARATIVE STUDY TO EVOLUTIONARY ALGORITHMS

Eva Volna
Martin Kotyrba
Department of Informatics and Computers
University of Ostrava
70103, Ostrava, Czech Republic
E-mail: eva.volna@osu.cz
E-mail: martin.kotyrba@osu.cz

KEYWORDS

Genetic Algorithms (GA), Simulated Annealing (SA), Differential Evolution (DE), Self Organising Migrating Algorithms (SOMA), Travelling Salesman Problem (TSP).

ABSTRACT

Evolutionary algorithms are general iterative algorithms for combinatorial optimization. The term evolutionary algorithm is used to refer to any probabilistic algorithm whose design is inspired by evolutionary mechanisms found in biological species. These algorithms have been found to be very effective and robust in solving numerous problems from a wide range of application domains. In this paper we perform a comparative study among Genetic Algorithms (GA), Simulated Annealing (SA), Differential Evolution (DE), and Self Organising Migrating Algorithms (SOMA). These algorithms have many similarities, but they also possess distinctive features, mainly in their strategies for searching the solution state space. The four heuristics are applied on the same optimization problem - Travelling Salesman Problem (TSP) and compared with respect to (1) quality of the best solution identified by each heuristic, (2) progress of the search from an initial solution until stopping criteria are met.

INTRODUCTION TO EVOLUTIONARY ALGORITHMS

Evolutionary algorithms (EAs) have many interesting properties and have been widely used in various optimization problems from combinatorial problems such as job shop scheduling to real valued parameter optimization (Back et al. 1997). In computer science, evolutionary computation is a subfield of artificial intelligence (more particularly computational intelligence) that involves combinatorial optimization problems. Evolutionary computation uses iterative progress, such as growth or development in a population. This population is then selected in a guided random search using parallel processing to achieve the desired end. Such processes are often inspired by biological mechanisms of evolution. As evolution can produce highly optimised processes and networks, it has many applications in computer science. Problem solution using evolutionary algorithms is shown in Figure 1.

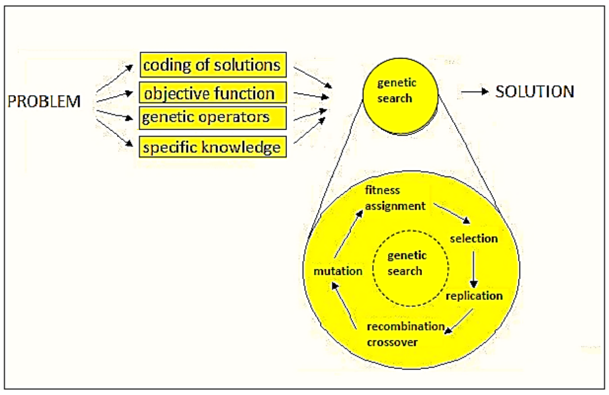

Figure 1: Problem solution using evolutionary algorithms (adapted from http://jpmc.sourceforge.net)

Genetic Algorithms

A genetic algorithm is a type of a searching algorithm. It searches a solution space for an optimal solution to a problem. The key characteristic of the genetic algorithm is how the searching is done. The algorithm creates a "population" of possible solutions to the problem and lets them "evolve" over multiple generations to find better and better solutions. The generic form of the genetic algorithm is shown in Figure 2. The items in bold in the algorithm are defined here (Volna 2013).

```
1. Create a population of random candidate solutions
   named pop.
2. Until the algorithm termination conditions are met,
   do the following:
   (a) Create an empty population named new-pop.
   (b) While new-pop is not full, do the following:
       i.  Select two individuals at random from pop so
           that individuals, which are more fit are more
           likely to be selected.
       ii. Cross-over the two individuals to produce
           two new individuals.
   (c) Let each individual in new-pop have a random
       chance to mutate.
   (d) Replace pop with new-pop.
3. Select the individual from pop with the highest
   fitness as the solution to the problem.
```

Figure 2: The genetic algorithm

The **population** consists of the collection of candidate solutions that we are considering during the course of the algorithm. Over the generations of the algorithm, new members are "born" into the population, while others "die" out of the population. A single solution in the population is referred to as an **individual**. The **fitness** of an individual is a measure of how "good" is the solution represented by the individual. The better solution has a higher fitness value – obviously, this is dependent on the problem to be solved. The **selection** process is analogous to the survival of the fittest in the natural world. Individuals are selected for "breeding" (or **cross-over**) based upon their fitness values. The crossover occurs by mingling two solutions together to produce two new individuals. During each generation, there is a small chance for each individual to **mutate**.

Simulated Annealing

Simulated Annealing (SA) was introduced by (Kirkpatrick et al. 1983) for the first time. SA starts off from a randomly selected point. Then a certain number of points is generated in the neighbourhood. The principle of acceptation solution during run of SA is following. If the new cost value is better than the old one new one is accepted immediately. It means that the difference between these two cost values is negative. If the difference is positive (the new cost value is worse than the old one) a number from interval <0, 1> is generated. If it is lower than the probability according to equation (1) the new point is accepted, otherwise the old one continues in the process. This is called Metropolis criterion (Kirkpatrick et al. 1983).

$$p(T) = e^{-\frac{\Delta E}{T}} \tag{1}$$

where $p(T)$ probability of transition for temperature T, ΔE is a difference between cost values of previous and current solution, and T is a current temperature that is a control parameter for cooling schedule.
The algorithm starts with high temperature T, which is decreased in steps. Equation (2) shows standard cooling function.

$$T_{n+1} = \alpha\, T_n \tag{2}$$

where T_{n+1} is a temperature in the next step, T_n is a temperature in the current step, and α is a coefficient from interval <0, 1>.

Differential Evolution

Differential Evolution (DE) is a population-based optimization method that works on real-number-coded individuals (Price 1999). For each individual $\vec{x}_{i,G}$ in the current generation G, DE generates a new trial individual $\vec{x}'_{i,G}$ by adding the weighted difference between two randomly selected individuals $\vec{x}_{r1,G}$ and $\vec{x}_{r2,G}$ to a randomly selected third individual $\vec{x}_{r3,G}$. The resulting individual $\vec{x}'_{i,G}$ is crossed-over with the original individual $\vec{x}_{i,G}$. The fitness of the resulting individual, referred to as a perturbed vector $\vec{u}_{i,G+1}$, is then compared with the fitness of $\vec{x}_{i,G}$. If the fitness of $\vec{u}_{i,G+1}$ is greater than the fitness of $\vec{x}_{i,G}$, then $\vec{x}_{i,G}$ is replaced with $\vec{u}_{i,G+1}$; otherwise, $\vec{x}_{i,G}$ remains in the population as $\vec{x}_{i,G+1}$. DE is quite robust, fast, and effective, with global optimization ability. It does not require the objective function to be differentiable, and it works well even with noisy and time-dependent objective functions. Figure 3 shows a two-dimensional example that illustrates the different vectors that are used in DE.

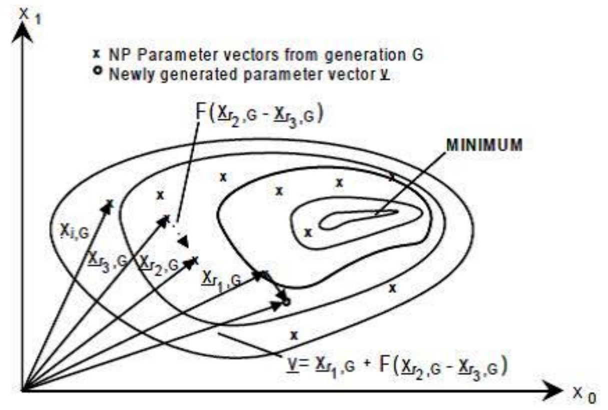
x 1
x NP Parameter vectors from generation G
o Newly generated parameter vector v
F(x_r2,G - x_r3,G)
MINIMUM
x_i,G
x_r3,G
x_r2,G
x_r1,G
v = x_r1,G + F(x_r2,G - x_r3,G)
x 0

Figure 4: Differential Evolution (Price 1999)

Self-Organising Migrating Algorithm

Self-Organising Migrating Algorithm (SOMA) was developed by prof. Zelinka (Zelinka 2004). SOMA works with groups of individuals (population) whose behaviour can be described as competitive – cooperative strategy. The construction of new population of individuals is not based on evolution principle (two parents produce offspring) but on the behaviour of social group, e.g. a herd of animals looking for food. During one generation, in the case of SOMA this is called migration loop (ML), only the position of individuals in the search space is changed.
In every migration loop the best individual is chosen, i.e. individual with the minimum cost value, which is called leader. An active individual from the population moves in the direction to leader in the search space. At the end of the movement the position of the individual with minimum cost value is chosen. If the cost value of the new position is better than the cost value of an individual from the old population, the new one appears in new population. Otherwise the old one rests there. The movement is described by equation (3).

$$x_{i,j}^{ML+1} = x_{i,j,START}^{ML} + (x_{L,j}^{ML} - x_{i,j,START}^{ML}) \cdot t \cdot PRTVector_j \tag{3}$$

where $x_{i,j}^{MlL+1}$ is a value of i-individual's j-parameter, in step t (in the next migration loop $ML + 1$). $x_{i,j,START}^{ML}$ is a value of i-individual's j-parameter that is the $START$ position in the actual migration loop (ML). $x_{L,j}^{ML}$ is a value of leader's *j*-parameter in migration loop *ML*. Step *t* is from <0, *by Step to, PathLength*>. *PRTVector* is a vector of ones and zeros depended on *PRT*. If random number from interval <0, 1> is less than *PRT*, then 1 is saved to *PRTVector*, otherwise 0 is saved to *PRTVector*. There exist four versions of SOMA, but we use version All-To-One in this work because of least time-consuming computing.

THE TRAVELLING SALESMAN PROBLEM

Evolutionary algorithms can be used to solve the travelling salesman problem (TSP), see Figure 4. For those who are unfamiliar with this problem, it can be stated in two ways. Informally, there is a travelling salesman who services some number of cities, including his home city. He needs to travel on a trip such that he starts in his home city, visits every other city exactly once, and returns home. He wants to set up the trip so that it costs him the least amount of money possible. The more formal way of stating the problem casts it as a graph problem. Given a weighted graph with *N* vertices, find the lowest cost path from some city v that visits every other node exactly once and returns to v. For a more thorough discussion of TSP, see (Garey and Johnson 1979).
The problem with TSP is that it is an NP-complete problem. The only known way to find the answer is to list every possible route and find the one with the lowest cost. Since there are a total of (N – 1)! routes, this quickly becomes intractable for large N. There are approximation algorithms that run in a reasonable time and produce reasonable results – evolutionary algorithms belong to them.

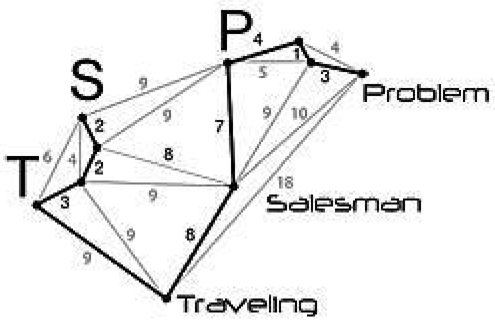

Figure 4: Travelling Salesman Problem (adapted from http://www.amdusers.com/wiki/tiki-index.php?page=TSP/)

COMPARATIVE EXPERIMENTAL STUDY

There are selected evolutionary algorithms tested on TSP. To assess the quality of a solution, individual cities are arranged in an orthogonal grid of *N*x*N* points (each point represents one city). Distance between cities is designed as Manhattan distance, i.e. the distance is calculated as an absolute value of a sum of differences between the two coordinates. Manhattan distance *d* between two cities $P_1=[x_1,y_1]$ a $P_2=[x_2,y_2]$ in plane is calculated as follows (4):

$$d(P_1,P_2) = |x_1 - x_2| + |y_1 - y_2| \quad (4)$$

The advantage of arrangement of cities in the grid lies in the easy comparison, how good is the result. In this arrangement, we are able to easily determine a length of the shortest path, as follows:
If *N* is even, the shortest path $d_{e\text{-}min}$ is calculated as follows (5):

$$d_{e\text{-}min} = N^2 \quad (5)$$

If *N* is odd, the shortest path $d_{o\text{-}min}$ is calculated as follows (5):

$$d_{o\text{-}min} = N^2 + 1 \quad (6)$$

Fig. 5 represents format of outputs. Individual routes between cities are drawn in different colours. The principle of these colours is the following: (1) black colour represents the shortest distances; (2) pink colour represents distances equal half of the size *N* of the grid; and (3) blue colour represents all other routes. In the ideal case, the window should contain only black lines. On the top bar of the window (Fig. 5), there are two buttons, which allow switching between results of calculations. Each algorithm was tested on a grid of size 5x5, 7x7 and 10x10, which gives 25, 49 and 100 cities. The number of different combinations that can be created in the grid of 10x10 is approximately $9.3 \cdot 10^{157}$. It shows how is hard to find the right solution for permutation problems.

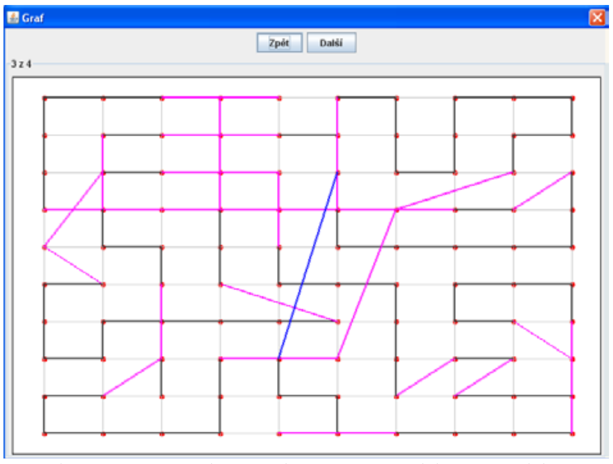

Figure 5: Found route in 10 x 10 grid – 100 cities

Comparative experimental study involves the following evolutionary algorithms:

- Genetic Algorithms (GA).
- Simulated Annealing (SA).
- Differential Evolution (DE).
- Self-Organising Migrating Algorithms (SOMA).

It is necessary to set values of all parameters for each of these algorithms. A minimal number of calculations for each parameters setting was 5000.

Genetic Algorithms

There is a process of reproduction very important in GA. It consists of two parts - the *cross-over* and *mutation*. It

was convenient to separate, for which part of the reproduction was responsible only cross-over and only mutation during experimental study. This was achieved, if we have tested the probability of cross-over, probability of mutation was set to zero and vice versa.
Testing the following parameters *population* and *generation* were merged into one unit, because both parameters determine, how much evaluations were done during calculation. It was an effort to evaluate similar numbers of calculations. For example, if we have reduced a population size to one half, the number of generations was raised twice.

To test CA should be set the following parameters:
Size of population is an integer that specifies a number of chromosomes that occur in a population. *Used values:* 10 – 2000.
Probability of cross-over is a number from the interval <0; 1>, which indicates the probability that new individuals will be from two randomly selected parents *Used values:* 0.5 – 1.
Probability of mutation is a number from the interval <0; 1>, which indicates the probability with which occurs mutation at individual positions in a chromosome. *Used values:* 0 – 0.1.
Threshold of termination is a number from the interval <0; 1> that indicates how many percent of occurrence of the same chromosome in the population means that a calculation will be terminated. *Used values:* 0.75 – 1.
Maximal number of generations is an integer greater than one indicating how many generations could be done, if some terminal criteria are not fulfilled. *Used values:* 50 – 10000.

Summary:
We could make the following conclusions regarding our experimental study of Genetic Algorithms:

- *Size of population*: > 80. If the population was smaller, GA started too soon to converge. For example: If the population contains ten individuals and the number of generations has been set to ten thousand, only 463 different combinations were evaluated on average. This means that there is approximately 50 generations before the algorithm was ended.
- *Probability of cross-over*: certainly more than 0.6, but rather in the range of 0.9 - 1 for 25 cities and in the range of 0.8 - 0.9 for 49 cities. If the probability of cross-over was smaller, algorithm quickly converges to a single solution. Although it can be tempered by mutation, but GA always showed worse results at low values of this parameter.
- *Probability of mutation*: strongly depends on the complexity of the problem (i.e. the number of cities). What is the number of cities bigger, the smaller value probability of mutation must be. Its experimental values are the following: 0.01 for 25 cities and 0.004 for 49 cities. If a mutation is too high, the algorithm starts to behave too random, and it is not able to find the minimum. On the other hand, if a mutation is too small and algorithm quickly converges to a single solution.
- *Threshold of termination* is a very sensitive parameter and depends on all other parameters. If its value was too small, the found route was not usually optimal. If its value was too big, algorithm ran through all generations because it was unable the condition fulfil.
- *Number of generations* should be more than 200 for 25 cities and more than 2000 for 49 cities. If a number of generations were smaller, GA had not a sufficient number of cycles for an evolution of its population, because they were still found unsuitable individuals in the population, who were involved in reproduction.
- Each calculation of GA should be run several times to use the average results. GA was often able to find optimal solutions even if parameters have been set worse. On the contrary, even if parameters have been set ideally, GA was not always able to find the optimal solution.

Simulated Annealing

To test SA should be set the following parameters:
Initial temperature is an integer greater than one, which sets the temperature of a beginning of an annealing process. *Used values:* usually 0.01 - 1000 (e.g. 1000×final temperature).
Final temperature is a number greater than zero, which represents the final temperature of a termination of a calculation. *Used values:* 0.00001 – 1.
Temperature reduction factor is a number from the interval <0; 1> that indicates about how many percent is the actual temperature cooled at each cycle. *Used values:* 0.01 – 0.999.
Number of iterations at a given temperature is an integer greater than zero that represents how many times is algorithms repeated at a given temperature. *Used values:* 1 – 17200.

Summary:
We could make the following conclusions regarding our experimental study of Simulated Annealing:

- *Temperature reduction factor*: > 0.6. If the value is smaller, the temperature cools too quickly. Consequently, the big leaps occur in the probability of acceptance of worse solutions. Recommended values are about 0.99 - 0.999. It is appropriate to the probability of acceptance of worse solutions has decreased slowly.
- *Number of iterations at a given temperature*: > 1. If a large number of iterations is set to be tested a large number of routes before the temperature cools. If a temperature reduction factor is set near one then temperature cools slowly leading to test a larger number of possibilities. Therefore, these parameters were tested together, and they were adjusted so that the number of tested routes was similar.

- *Initial temperature* should be from the interval <0.1; 50>. For higher temperatures, the probability of acceptance of worse solutions is too high. In contrast, the algorithm often gets stuck in a local minimum for smaller temperature.
- Each calculation of SA should be run several times (see GA).

Differential Evolution

To test DE should be set the following parameters:
Threshold of cross-over is a real number from interval <0, 1>, indicating how likely it will be placed at the position an element of noise or target vector. *Used values:* 0.05 – 0.9.
Size of population is the number of individuals in a population (see GA). *Used values:* 10 – 800.
Mutation constant is a real number and indicates how much will be multiplied the vector during a mutation. *Used values:* 0 – 2.
Number of generations see GA. *Used values:* 125 – 10000.

Summary:
We could make the following conclusions regarding our experimental study of Differential Evolution:

- The greater the complexity of the problem is, the smaller have to be set *threshold of cross-over*: range of 0.2 – 0.3 for 25 cities and in the range of 0.1 - 0.1 for 49 cities. The real values are usually set higher than the above recommended values in order to DE not quickly converge to a single solution. It could be used especially when it is set greater number of generations.
- Regarding setting a *mutation constant*, there were not recognized achieved significant differences in results depending on the setting of this parameter. In this case, it is possible to say, that the parameter is not significant in the course of DE.
- To prevent a stagnation process, it is necessary to do the following: either to set up a large number of individuals in a population, or threshold of cross-over should be higher.
- *Number of generations* should be higher then a complexity of the problem is. Simultaneously, it is suitable to set a size of the population size at higher value, because the parameter has proved as decisive regarding stagnation.
- Each calculation of DE should be run several times (see GA).

Self-Organising Migrating Algorithms

To test SOMA should be set the following parameters:
Migration is a parameter, which equals to the parameter maximal number of generations in GA. *Used values:* 20 – 10000.
Minimal diversity means a terminal distance between the best and the worst individual in a given population. It is a terminating parameter (integer). *Used values:* 0 – 18 (30).
Path Length is a real number from the interval <1; 5>. It determines how far an individual goes on its way toward the best individual. *Used values:* 0.33 – 10.
Step is a real number from the interval <0.11 – *Path Length*>. It determines how many times individual stops during its path. *Used values:* 0.033 – 0.9.
Perturbation (PRT) is a parameter that replaces the mutation from GA. *Used values:* 0 – 1.
Size of population is the number of individuals in a population (see GA). *Used values:* 5 - 500.

Summary:
We could make the following conclusions regarding our experimental study of SOMA:

- *Pertubation* is set to small values ranging <0, 0.1>. The larger the dimension of the problem is, the smaller has to be the value of the parameter.
- *Migration* >1000. The larger dimension of the problem is, the greater has to be the value of the parameter. *Migration* is able to be replaced with parameters *step* and *path length*.
- It is not appropriate to set the parameter *path length* so that it would be a multiple of the parameter *step*. In such case, there is often a quick convergence caused by the fact that each individual is on its way stops exactly at the place where is situated the best individual.
- *Minimal diversity.* It is evident, if a parameter minimal diversity is set too high the algorithm terminates before an appropriate solution is found. On the other hand, if the parameter is set too low, the calculation will never be stopped by this parameter. During the experiments, it was shown that if the algorithm terminated by this parameter, the achieved solutions has worse values. It is appropriate to set the value of this parameter to zero.
- The biggest influence on the quality of the solution had *population*. A large population clearly reached the best results for 25 cities. Large population was not enough and proved to be the weakest parameter for 49 cities. This is a phenomenon that was observed in all tested algorithms.

OUTCOMES

Finally, all of these algorithms were tested in 10 × 10 grid, i.e. 100 cities. Due to the complexity of the problem, each algorithm was run only twice. There are parameters settings at single algorithms in Table 1.
Outputs of single algorithms are presented in the following graph (Fig. 6). From this graph it is clear that SA achieved the best results when it found a way which indicates the third best possible value. According to formula (5) the known best way is the following: $d_{e\text{-}min} = 10^2 = 100$. Fig. 7 shows that SA has found the ways, where $d = 104$.

Table 1: Parameters settings at single algorithms.

GA	SA
Size of population: 400 Probability of cross-over: 0.9 Probability of mutation: 0.001 Threshold of termination: 0.5 Max. number of generations: 60000	Initial temperature: 50 Final temperature: 0.0001 Temperature reduction factor: 0.999 Number of iterations at a given temperature: 1000
DE	**SOMA**
Size of population: 3000 Number of generations: 40000 Threshold of cross-over: 0.1 Mutation constant: 0.8	Perturbation: 0.01 Migration: 30000. Minimal diversity: 2 Path Length: 3 Step: 0.11 Size of population: 400

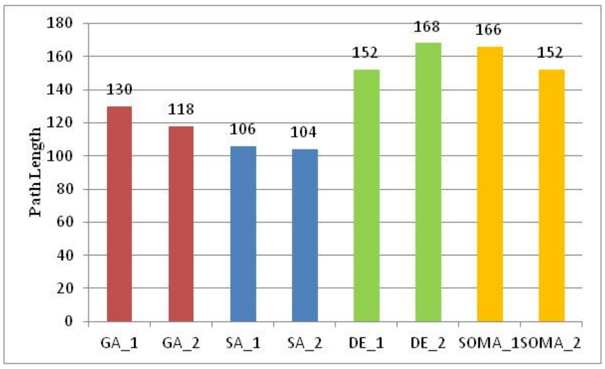

Figure 6: Comparative results of single algorithms with 100 cities

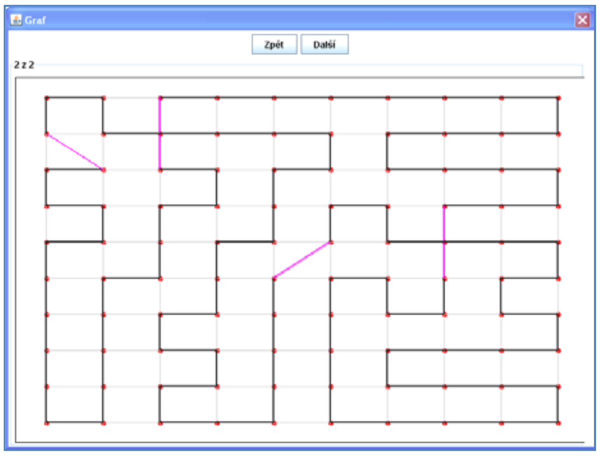

Figure 7: Simulated Annealing – 100 cities

CONCLUSION

In this paper, an experimental comparative study of four popular approximation algorithms (GA, SA, DE, and SOMA) is presented for travelling salesman problem. All four heuristics assume and exploit regularities present within the search space, i.e., search spaces where good solutions have higher probabilities of leading to better solutions. All four tested algorithms have been found to be effective and robust on problems where some measure of progress can be shown. These algorithms discussed incorporate domain specific knowledge to dictate the search strategy. The principle deference among them is how and where domain-specific knowledge is used. In this work our intention has been to study the behaviour of the four heuristics in solving a hard combinatorial problem, and not to demonstrate the superiority of one algorithm over the other over all problem domains. Each one of them has its merits. Actually it would be unwise to generalize the results reported here over all classes of problems. To solve such question would require at least that similar experiments on other category of problems be performed. Such experiments are the subject of our future work.

ACKNOWLEDGMENT

The research described here has been financially supported by University of Ostrava grant SGS/PřF/2014.

REFERENCES

Back, T., Hammel U. and Schwefel H.-P. 1997. Evolutionary Computation: Comments on the History and Current State. *IEEE Trans. on Evolutionary Computation*, pp. 3-17.

Garey, M. and Johnson, D.S. 1979. Computers and Intractability: *A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company.

Hrabal, R. 2010. NP-problems solving via evolutionary algorithms. Master thesis (in Czech). University of Ostrava. 66 p.

Kirkpatrick S., Gelatt C. D., Vecchi M. P. 1983. Optimization by Simulated Annealing, Science, 13, Volume 220, Number 4598, pp. 671 – 680

Price, K. 1999. An Introduction to Differential Evolution. In: D. Corne, M. Dorigo and F. Glover (eds.) *New Ideas in Optimization*. London: McGraw-Hill, pp. 79–108.

Volná, E. 2013. *Introduction to Soft Computing*. bookboon.com Ltd. ISBN: 978-87-403-0391-9 (available from http://bookboon.com/en/introduction-to-soft-computing-ebook)

Zelinka I. 2004. SOMA – Self Organizing Migrating Algorithm, In: Babu, B.V. Onwubolu G. (eds), *New Optimization Techniques in Engineering*. Springer-Verlag, ISBN 3-540-20167X

EVA VOLNA is an associate professor at the Department of Computer Science at University of Ostrava, Czech Republic. Her interests include artificial intelligence, artificial neural networks, evolutionary algorithms, and cognitive science. She is an author of more than 50 papers in technical journals and proceedings of conferences.

MARTIN KOTYRBA is an assistant professor at the Department of Computer Science at University of Ostrava, Czech Republic. His interests include artificial intelligence, formal logic, soft computing methods and fractals. He is an author of more than 30 papers in proceedings of conferences.