TASK QUARTERLY 7 No 1 (2003), 73–91

EFFICIENCY OF SELECTED META-HEURISTICS APPLIED TO THE TSP PROBLEM: A SIMULATION STUDY

HALINA KWAŚNICKA

Department of Computer Science, Wroclaw University of Technology,
Wybrzeze Wyspianskiego 27, 50-370 Wroclaw, Poland
kwasnicka@ci.pwr.wroc.pl

(Received 19 November 2001; revised manuscript received 1 September 2002)

Abstract: The paper presents a simulation study of the usefulness of a number of meta-heuristics used as optimisation methods for **TSP** problems. The five considered approaches are outlined: Genetic Algorithm, Simulated Annealing, Ant Colony System, Tabu Search and Hopfield Neural Network. Using a purpose-developed computer program, efficiency of the meta-heuritics has been studied and compared. Results obtained from about 40 000 simulation runs are briefly presented and discussed.

Keywords: ant colony, genetic algorithm, simulated annealing, tabu search, neural network

1. Introduction

Combinatorial optimisation is an example of a difficult optimisation problem. Numerous approaches have been proposed for such problems, some of which imitate natural processes, *e.g.* biological evolution or ant colonies. Most of them are still in their development phase, and it is difficult to assess which method is appropriate for a particular problem. The paper presents simulation studies of the efficiency of selected methods inspired by nature on the basis of the Travelling Salesman Problem (**TSP**). **TSP** was selected because it is one of the most studied NP-hard problems. In **TSP**, we have a set of N cities $C = \{C_1, C_2, \ldots, C_N\}$ and a set, E, of routes connecting the cities with one another. In other words, we have a full connected graph $G(C,E)$, where C is a set of nodes and $E = \{E_{ij}\}$, $i = 1,2,\ldots,N$, $j = 1,2,\ldots,N$, $i \neq j$, is a set of edges. Each edge E_{ij} (connecting nodes C_i and C_j) has a measure d_{ij} – the distance between cities C_i and C_j. The task is to find the shortest route between all the cities, assuming that each city has to be visited exactly once. In mathematical terms, **TSP** is defined as a problem of finding the minimal-length Hamiltonian circuit on the graph $G(C,E)$ – a closed tour between all N nodes. The length of a Hamiltonian circuit is the sum of distances $d_{i,j}$ of all edges $E_{i,j}$ belonging to the tour. In our experiments, cities are given by their co-ordinates (x_i, y_i) and distance d_{ij} is the Euclidean distance between cities C_i and C_j, so that we have a symmetric Travelling Salesman Problem:

the tours $C_1 \to C_3 \to C_5 \to C_2 \to C_4 \to C_1$ and $C_1 \to C_4 \to C_2 \to C_5 \to C_3 \to C_1$ are the same.

A number of meta-heuristics can be found for such a problem. In the paper, we consider some of them, namely Genetic Algorithm (**GA**), Simulated Annealing (**SA**), Ant Colony System (**ACS**), Tabu Search (**TS**) and a simple Hopfield Neural Network (**NN**).

The paper is organised as follows. The first five sections present very briefly the meta-heuristics used in our simulation studies. Section 2 describes Genetic Algorithm with special operators adjusted to the problem being solved. The technique called Simulated Annealing is presented in Section 3. Section 4 introduces the Ant Colony System. Section 5 is dedicated to the Tabu Search method. Section 6 contains a short presentation of Hopfield Neural Networks. Section 7 provides an overview of the obtained results. Discussion of the main characteristics of the considered methods on the basis of our simulation study is presented in the concluding section.

2. Genetic Algorithm in the TSP problem

Genetic Algorithm (**GA**) is a technique from the broadly-understood area of artificial intelligence, based on the natural process of biological evolution [1–4]. The main differences between conventional optimization methods and the Genetic Algorithm are:

- **GA** works with coded parameters in the form of chromosomes. Usually chromosomes are strings of bits. One chromosome (individual) codes a single point in the solutions space.
- **GA** searches solutions working simultaneously with a population of individuals.
- **GA** does not use derivatives or any other information about the function being optimized.
- **GA** uses probabilistic rules in the search process, exploiting areas with high fitness.

Genetic Algorithms use a vocabulary borrowed from genetics and imitate biological evolution according to the Neo-Darwinian paradigm. It assumes, that four statistical processes operating within populations and species can explain the history of life: reproduction, mutation, competition and selection (Figure 1). *Reproduction* is the process necessary for species to survive, but the potential abilities of species to reproduce are so large, that the size of a population would increase exponentially if all individuals could reproduce with success. *Mutation* guarantees diversity in biological systems. Because the environment of an evolving population is limited, there is *competition* between individuals. The outcome of *selection* is elimination of some individuals due to competition: only some individuals can survive and have offspring. Phenotype diversity is a consequence of recombination and errors in genome transcription.

Usually, binary coding is used, which means that a potential solution (called a *chromosome*) is coded as a string of bits. *Mutation* is implemented as random changes of bits: from 0 to 1 or opposite. *Crossover* means exchange of parts of bits' strings between two randomly selected individuals. Better individuals (solutions) are preferred for reproduction.

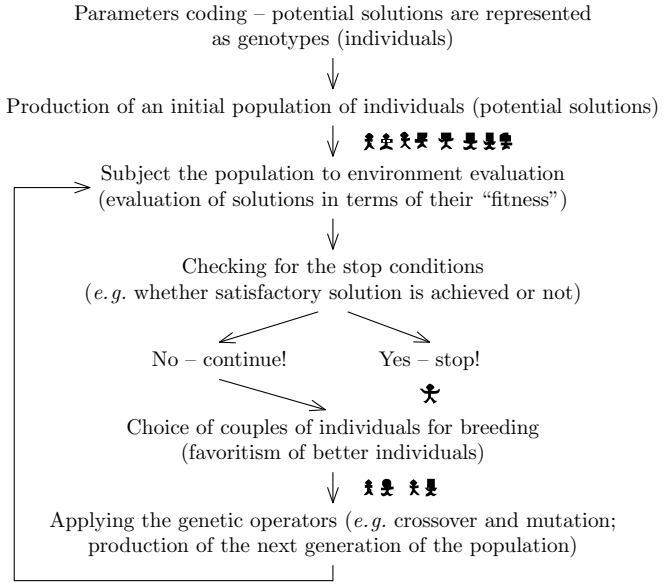


Figure 1. A general schema of **GA** performance

Using Genetic Algorithm for **TSP** we define a chromosome (an individual) as a sequence of city numbers (a list of cities), *e.g.* i^{th} individual $I_i = [13524]$ encodes the tour $C_1 \to C_3 \to C_5 \to C_2 \to C_4 \to C_1$. The distance F of an encoded tour is our goal function, we must find the tour with minimal distance:

$$F_g = \min_i(F(i)), \quad i = 1, \ldots, N, \tag{1}$$

where $F(i)$ is the distance of the route encoded by the i^{th} individual in the current population, N – the size of evolved population (number of individuals).

Classic mutation and recombination produce wrong solutions: some numbers of cities may be doubled while others may be absent. Therefore, specialized genetic operators for permutation problems are developed, which combine crossover and mutation effects. They assure production of only correct solutions (satisfying the constraints). Literature shows that the so-called Cycle Crossover (CX) and Asexual Crossover (AX) give relatively good results and operate quite quickly [5]. In the presented study we assume the following operators (with assumed probabilities).

Cycle Crossover (CX)

CX uses two parents, I1 and I2. It goes as follows:

Step 1: select randomly one gene in I1 (*e.g.* the first in Figure 2).

Step 2: offspring (I1') receives the selected gene from the first parent (offspring I1' takes his first gene – city number 1 – from parent I1),
repeat:

Step 3: find in the second parent (I2) the value of a gene placed in the same position as the redrawn gene (in the first position in I2 there is a value equal to 4 – a city number 4),

Step 4: redraw from the first parent (I1) the gene with the found value to the offspring – put value 4 on the fourth position of I1',

```
I1 : 1 2 3 4 5 6 7 8
I2 : 4 3 2 7 6 5 8 1

I1': 1 - - - - - - -
I1': 1 - - 4 - - - -
I1': 1 - - 4 - - 7 -
I1': 1 - - 4 - - 7 8
I1': 1 3 2 4 6 5 7 8

I2': 4 - - - - - - -
I2': 4 - - - - - - 1
I2': 4 - - - - - 8 1
I2': 4 - - 7 - - 8 1
I2': 4 2 3 7 5 6 8 1
```

Figure 2. An example of CX crossover

```
I : 1 2 3 4 5 6 7 8
I : 1 2 | 3 4 5 6 | 7 8
I1: 3 4 5 6
I2: 1 2 7 8
I2: 1 2 7 | 8
I': 1 2 7 3 4 5 6 8
```

Figure 3. An example of an AX operator

```
I : 1 2 3 4 5 6 7 8
I : 1 2 | 3 4 5 6 | 7 8
I': 1 2 | 6 5 4 3 | 7 8
I': 1 2 6 5 4 3 7 8
```

Figure 4. An example of Inversion

```
I : 1 2 3 4 5 6 7 8
I : 1 2 3 4 5 6 7 8
I': 1 2 3 4 5 6 7 8
```

Figure 5. An example of Mutation

until created offspring (I1') already contains the value found in step 3 (value 1 stops the loop during creation of I1', value 7 stops the loop when I2' is produced, see Figure 2).

Step 5: redraw the remaining genes from the second parent.

In a similar way the second offspring I2' is created.

Asexual Crossover (AX)

Asexual Crossover uses only one parent. An example of AX is shown in Figure 3:

Step 1: select randomly two points on the parent individual (*e.g.* points between 2 and 3, 6 and 7 in Figure 3),

Step 2: copy to the one temporary individual (I1 in the figure) the part between selected points, and to the second temporary individual (I2) – the rest of parent,

Step 3: select a random point on the second temporary individual (I2), *e.g.* between 7 and 8,

Step 4: create the offspring (I'): insert the sequence remembered on the first temporary individual (I1) to the position selected in *Step 3* on the second temporary individual (I2).

Inversion

Inversion acts on a single individual (parent). Figure 4 shows an example of inversion:

Step 1: select randomly two points on the parent individual (points between 2 and 3, 6 and 7 in Figure 4),

Step 2: invert the middle part of the chromosome.

Mutation

Mutation is similar to Inversion, but it concerns only two neighbour genes – number of cities (see Figure 5).

Step 1: select randomly two neighbour points (genes) on the parent individual (genes 3 and 4 in the figure),

Step 2: invert the selected numbers (replace 3 4 by 4 3).

Each individual is evaluated: distance of the encoded route is calculated (our goal function $F(i)$, which we should minimize). The goal function is transformed into a fitness function Fit:

$$Fit(i) = F_{\max} - F(i), \tag{2}$$

where $Fit(i)$ – fitness of the i^{th} individual, $F_{\max}$ is the maximal distance in the current population ($F_{\max} = \max_j F(j), j = 1,\ldots,N$).

So defined fitness function Fit can be used in a popular roulette wheel selection method.

3. Simulated Annealing as a tool for TSP solving

Simulated Annealing (**SA**) is a method based on cooling and freezing metal into its minimum energy crystalline structure, *i.e.* the annealing process. **SA** exploits the analogy between the annealing process and searching for a minimum in more general systems. The background of the method can be found in [6–8].

Before using **SA**, we must perform some initial steps (similarly as in **GA**). The analogy between the optimisation process and the physical concepts must be recognised. The energy function becomes the goal function, configurations of particles – configurations of questing parameter values, searching for minimum energy – searching for the near optimal solution, temperature – a parameter that controls the whole process (Figure 6). The way in which new configurations are obtained must also be established [9]. We assume (similarly as in **GA**) that we can calculate the value of the goal function $f(x)$ – distance of the coded tour – for each potential solution x.

```
Procedure SA
begin
  iteration t ← 0
  initial value of temperature T
  random choice of x_c
  evaluation of x_c
  repeat
    repeat
      choice x_n from neighbours of x_c using assumed way
      if f(x_n) < f(x_c) {the new tour is shortest} then x_c ← x_n
        else if random [0,1] < exp{−(f(x_n) − f(x_c))/T}
          then x_c ← x_n
    until (end condition)
    T ← g(T,t)
    t ← t+1
  until (stop criterion)
end
```

Figure 6. The pseudocode of **SA**

In the above pseudocode of **SA** '`end condition`' allows to stop the internal loop when temperature equilibrium is reached, which means that probability distribution of choosing a new solution is close to the Boltzman distribution. **SA** often assumes

that this part of the algorithm is repeated k times, where k is done as a parameter, or until no improvement of the solution is observed. $g(T,t)$ is a function decreasing temperature T in sequential iterations. '`Stop criterion`' acts when temperature T is very small; it means that the considered system is cooled (nothing worse than the current solution can be accepted).

The probability p of acceptation of a new solution x_n worse than the current solution x_c depends on the values of the goal function of these solutions and temperature T:

$$p = e^{-(f(x_n)-f(x_c))/T}. \tag{3}$$

4. Ant Colony System solving TSP

The ant algorithm was proposed by Dorigo and others [10, 11]. It is a relatively new approach to difficult combinatorial optimisation problems like the Travelling Salesman Problem (**TSP**) or the Quadratic Assignment Problem (**QAP**) [10]. The ant-based algorithm is becoming a popular subject in the scientific community and it is extended to various discrete optimisation problems [12]. Observations of real ant colonies were the inspiration for artificial ant algorithms. Ants are social insects: they live in colonies. The behaviour of a colony is aimed at the survival the colony as a whole. A particularly interesting pattern of behaviour among ant colonies is seen in how they can find the shortest route between food sources and their nest. Ants, when walking from food sources to the nest and back, deposit *pheromone* on the ground, thus forming the so-called pheromone trail. Choosing their way, ants prefer paths marked by strong concentrations of pheromone. If an ant colony has a number of possible paths to the food source, it is able to find the shortest path by exploiting the pheromone trails left by the individual ants. Ant colony behaviour in controlled conditions was studied by Deneubourg *et al.* [13]. Qualitative results are shown in Figure 7.

Without any barrier, ants' path to the food source is a straight line. When we put an obstacle in their way, both possible paths, the upper and the lower, are without pheromone, therefore ants select them with the same probabilities. As time goes on, random fluctuations cause that more ants randomly select one path, for example the upper one, as shown in Figures 7b and 7c. Walking ants deposit pheromone: the more ants on the path the greater amount of pheromone is deposited on it, and in turn, it is a stimulus for ants to choose this way, and so on. This experiment can be easily extended to the case with two different length paths. Based on the above observations, the following probabilistic model of artificial ant colonies has been developed. The ants which took the shortest way, come back to the nest from the food source first. When they start another trip, the shortest path has more pheromone and is selected with higher probability. So, we have a kind of distributed optimisation mechanism, to which each single ant gives only a very small contribution. Ants perform a difficult optimisation task using indirect communication mediated by pheromone laying: such communication is called *stigmergy*. Stigmergy has a physical nature of information distributed by communicating ants and it has a local nature of distributed information. We can find different realisations of the Ant Colony System depending on assumptions about forming a pheromone trail, evaporation of pheromone, and

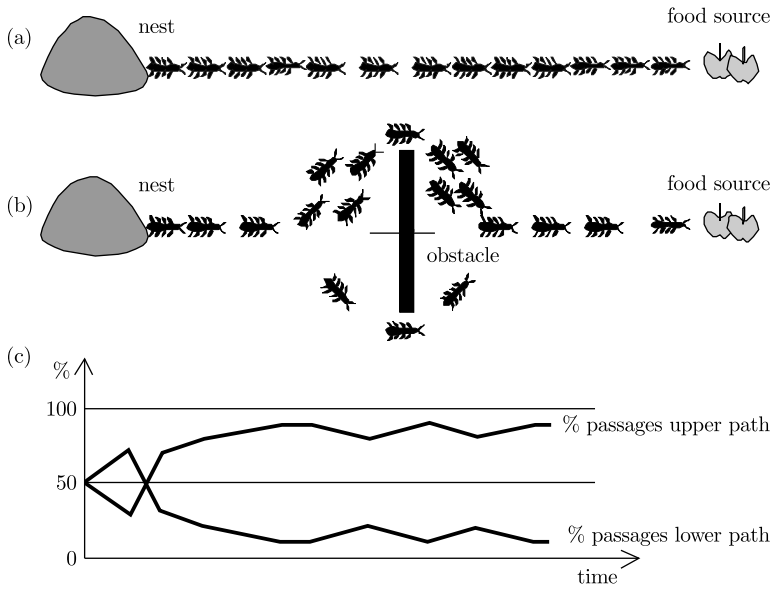


Figure 7. A general schema of Ant Colony behaviour [13]

so on. In the presented study we have implemented the algorithm yielding the best result [10].

4.1. Ant Colony System meta-heuristic applied to the Travelling Salesman Problem

An ant colony consists of m ants and solves the **TSP** problem in $t_{\max}$ iterations. We assume a cyclic algorithm, *i.e.* each ant deposits pheromone on the path after construction of the whole tour.[1]

The amount of pheromone $\tau_{ij}(t)$ on the arc (i,j) represents attractiveness of choosing this edge. All ants have memory: they remember cities visited and remaining for closing the Hamiltonian circuit.

Ant number k chooses the route between cities i and j (the j^{th} city is the neighbour of the i^{th}) during iteration t of tour construction with probability $p_{ij}^k(t)$ given by the following equations:

$$a_{ij}(t) = \frac{[\tau_{ij}(t)]^{\alpha}\left[\frac{1}{d_{ij}}\right]^{\beta}}{\sum\limits_{l\in N_i}[\tau_{il}(t)]^{\alpha}\left[\frac{1}{d_{il}}\right]^{\beta}}, \tag{4}$$

$$p_{ij}^k(t) = \frac{a_{ij}(t)}{\sum\limits_{l\in N_i^k} a_{il}(t)}, \tag{5}$$

1. There are other possible algorithms, called *ant-density* and *ant-quantity*. In an ant-cycle algorithm ants deposit pheromone after they have completed the whole tour, but in *ant-density* and *ant-quantity* algorithms ants deposit pheromone during construction of the tour: *ant-density* means that ants deposit a constant amount of pheromone, *ant-quantity* – that the amount of pheromone on the chosen arc is inversely proportional to its length.

where $\tau_{ij}(t)$ – the amount of pheromone on the edge (i,j) deposited in the t^{th} algorithm iteration; d_{ij} – the length of edge (i,j); N_i – the set of neighbour cities of the i^{th} city; α, β – parameters that control the relative weight of the pheromone trail and the length of the edge; $N_i^k \subseteq N_i$ – the set of neighbour cities of the i^{th} city not yet visited by ant k.

When all ants belonging to the colony have completed their tours, the pheromone on all edges evaporates. The role of evaporation is to prevent stagnation: it allows the ants to escape from a local optimum. Next, each ant k deposits $\Delta\tau_{ij}^k(t)$ pheromone on each used edge, according to the equation:

$$\Delta\tau_{ij}^k(t) = \begin{cases} \frac{1}{L^k(t)} & \text{if } (i,j) \in T^k(t), \\ 0 & \text{if } (i,j) \notin T^k(t), \end{cases} \tag{6}$$

where $T^k(t)$ – the tour done by the k^{th} ant at the t^{th} algorithm iteration; $L^k(t)$ – the length of the route $T^k(t)$.

The amount of pheromone deposit depends on how well the ant has performed, *i.e.* the shorter the tour, the more pheromone on it. In practice, the following rule is applied to all edges to calculate a new pheromone trail:

$$\tau_{ij}(t) = (1-\gamma)\tau_{ij}(t-1) + \sum_{k=1}^{m} \Delta\tau_{ij}^k(t-1), \tag{7}$$

where $\gamma \in (0,1]$ is a coefficient responsible for pheromone evaporation.

Initial values of parameters may be the following: $\tau_{ij}(0)$ is set to a small positive value on all edges, $\alpha = 1$, $\beta = 5$, $\gamma = 0.5$, and the number of ants m is equal to the number of cities N [10]. Too high α can cause stagnation – all ants select the same route; with too high β ants always choose the nearest city.

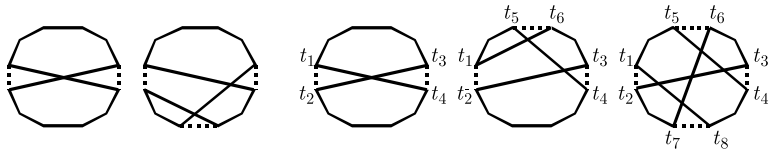
5. Tabu Search meta-heuristic

Tabu Search (**TS**) is a meta-heuristic with possibilities of escaping from local optima. It uses short-time memory to remember the latest solutions. In **TS**, discovered solutions are written to the so-called *tabu list* the search process cannot use the solutions present on this list. The tabu list has a finite length. At the beginning, a tabu list is empty. As an optimisation process goes on, successive solutions are written to this list. If it is full, the oldest solution is removed and a new one is added. The larger the problem, the longer the tabu list, but a long tabu list causes that the optimisation process consumes more time. Therefore, a tabu list should be as small as possible, enough to a search large solution space, but long enough to allow the escape from local optima. Another component of **TS** is the aspiration level, which allows for using a tabu (*i.e.* forbidden) solution to escape from a local optimum.

A **TS** algorithm starts with an initially generated solution (tour). It can be done using any method, for example, nearest neighbour heuristics. An initial tour becomes *current solution* and *best solution*. Using local heuristic search, based on a current solution, **TS** generates new, neighbourhood solutions. Potential solutions are evaluated and the best one is selected as a candidate for *current*. The tabu list is checked and the aspiration level is taken into account. The process is repeated until

the stop criterion is reached. The **TS** method does not guarantee reaching a global optimum.

TSP is a good test problem for Tabu Search. After Tsubakitani and Ewans [14], for **TSP** with N cities, the size of a tabu list may be equal to $N/4$. An initial solution can be stated using the nearest neighbour method. The question concerns local heuristic, *viz.* how we should select neighbourhood solutions. For **TSP**, such heuristics are based on k-Opt moves. Neighbourhood solutions are obtained by removing k edges from the current tour and adding such k new edges that constitute a legal tour. k-Opt moves are the basis of three frequently used heuristics: 2-Opt (Figure 8a), 3-Opt (Figure 8b) and Lin-Kernigan (Figure 8c). Exchanging four or more edges can cause the loss of tour consistency, therefore a sequential method of edge removal is proposed in [15]. The set of removed edges is enlarged until it is promising. Figure 8c shows the three first steps of the Lin-Kernigan method. The shortest route found becomes the current solution.



(a) 2-Opt (b) 3-Opt (c) Lin-Kernighan heuristic

Figure 8. Neighbour solutions searching heuristics

In the presented paper, the first two heuristics are taken into account in searching for neighbourhood solutions. Additionally, a roulette wheel method for edge selection in the 2-Opt and 3-Opt methods is implemented. A roulette wheel is built from the distances between two subsequent cities on the current route which are greater than the mean distance. Applied heuristics do not work well in situations where only one city in the route should be changed to improve solution. Therefore additional (optimisation) process is proposed. Solutions given by the 2-Opt or 3-Opt heuristics are tested sequentially: another city is inserted between two neighbour cities and the shortest route is selected as the current solution. The current tabu list is always taken into account.

6. Hopfield Neural Network in TSP problem solving

Neural networks (**NN**) are a strongly developed field, starting from the early forties, when McCulloch and Pitts proposed their architecture [16, 3, 17, 18]. Similarities of artificial and biological neural networks lie in the possibilities of learning (and generalizing) on the basis of a training set. In the last decades, new methods of training have been developed, as well as their mathematical basis. **NN**'s are useful in a variety of practical problems.

An artificial neural network is a collection of connected units, called neurons. All connections are weighed (weights are usually real values). A neural network can perform a desired task without knowing any algorithm of the task solution. Instead of this, a network has to learn (be trained) to do a specific task. The learning of neural networks means the adjustment of the weights of all connections in the network [17, 18].

For **TSP**, we can use a Hopfield Neural Network. In a recurrent Hopfield Neural Network, all connections between neurons are allowed: outputs of neurons can be connected as inputs to neurons of the same or earlier layer. A signal processed in such a network oscillates between neurons until a convergence criterion is fulfilled, then the output signal is produced. In analogue Hopfield networks, continuous activation functions are assumed. An analogue network consists of M working elements (neurons) with the activation function:

$$x_i = f_\beta(u_i) = \frac{1}{2}(1+\tanh(\beta u_i)), \tag{8}$$

where u_i – an input signal of neuron i, x_i – the output signal produced by neuron i, β – an assumed parameter ($\beta > 0$).

For **TSP** with N cities, a Hopfield network has N^2 neurons divided into N groups. Each group contains N neurons and is responsible for a single city. A value equal to 1 on the output of the k^{th} neuron in the m^{th} group denotes that the m^{th} city is in the k^{th} position in the tour (solution). Because each city must be present in the tour only once, only a single neuron in each group can produce the output signal equal to 1. Therefore, some constraints must be defined.

Let us assume that x_{pi} denotes the value of the output signal of the i^{th} neuron in the p^{th} group. We can formulate the required constraints as follows:

$$x_{pi} \in \{0,1\}, \quad p = 1,2,\ldots,N; \quad i = 1,2,\ldots,N, \tag{9}$$

$$h_1(x) = \sum_p \sum_i \sum_{j \neq i} x_{pi} x_{pj} = 0, \tag{10}$$

$$h_2(x) = \sum_p \sum_i \sum_{p \neq q} x_{pi} x_{qi} = 0, \tag{11}$$

$$h_3(x) = \left(\sum_p \sum_i x_{pi} - N \right)^2 = 0, \tag{12}$$

where Equation (9) denotes that outputs of all neurons must be equal to 0 or to 1; Equation (10) – that each city can be visited only once; Equation (11) – that two different cities cannot be placed on the same position in the route; Equation (12) – that all cities have to be present in the tour.

With above constraints defined, the goal function is:

$$J(x) = \sum_p \sum_{p \neq q} \sum_i d_{pq} x_{pi} (x_{q,i+1} + x_{q,i-1}), \tag{13}$$

where d_{pq} denotes the distance between cities p and q, x_{q0} means x_{qN}, and $x_{q,N+1}$ means x_{q1} (the tour is closed, a salesman must go back to the initial city).

Having formulated the problem, we can use a Hopfield Neural Network with the energy function given by the equation:

$$E(x) = \frac{A}{2} h_1(x) + \frac{B}{2} h_2(x) + \frac{C}{2} h_3(x) + \frac{D}{2} J(x) + \sum_p \sum_i \frac{1}{\tau} \int_{0.5}^{x_{pi}} f_\beta^{-1}(\xi) d\xi, \tag{14}$$

where A, B, C, D, and τ are positive coefficients, f_β is given by Equation (8).

By minimising $E(x)$, we minimise all its components. The first three components have minimum values equal to zero, by minimising the fourth component we

minimise the length of a route. The last component ensures that constraint (8) is fulfilled. When a network reaches a stable state, the process stops and the route has been found. In our computer implementation, the algorithm proposed by Muller, Reinhardt and Strickland [17] has been used.

7. Simulation study of particular meta-heuristics

A computer program[2] that has been used for this simulation study works in the Microsoft Windows environment [19]. Special emphasis has been put on efficiency of the methods and on possibilities of assuring comparable conditions for all the methods (*e.g.* screen refreshing is blocked when time is measured).

7.1. Initial experiments (configurations of the studied methods)

The main aim of the initial experiments is to select appropriate values of parameters of the tested methods. We must take into account the quality of the obtained solution (*i.e.* the length of the found tour) and the computation time.

Genetic Algorithm (**GA**) is sensitive to probabilities of genetic operators and population size. A list of parameters which need to be adjusted is as follows: probabilities of *Asexual Crossover* (p_{AX}), *Mutation* (p_{mut}), *Cycle Crossover* (p_{CX}), *Inversion* (p_{inv}), *Size* of the evolving population (N_{pop}) and *Fmultiple* – a parameter used for fitness scaling. Option Use *SN* allows to use the nearest neighbour method in the initial phase.

Experiments (50 cities, 100 individuals, $p_{\mathrm{AX}} = 0$, $p_{\mathrm{CX}} = 0$, *Fmultiple* $= 2$, 10 runs with the same conditions) show that **GA** works well with $p_{\mathrm{mut}} = 0.2$, $p_{\mathrm{inv}} = 0.8$, but with crossover used somewhat lower values of p_{mut} and p_{inv} are suitable (0.15 and 0.6 respectively). Comparing the results presented above with those obtained with p_{mut}, p_{inv}, and p_{CX} working together, we can see that **GA** works well without cycle crossover. A series of experiments was made to study the influence of p_{AX} on the algorithm's efficiency. It seems that $p_{\mathrm{AX}} = 0.4$ is near the optimal value. In further experiments, we have included the nearest neighbour heuristic to create the initial population. In such a case, the best results are given by $p_{\mathrm{AX}} = 0.6$ and $p_{\mathrm{CX}} = 0.3$. An algorithm with all four operators switched on (mutation, inversion, cycle crossover and asexual crossover) and with randomly generated initial population gives better results than those obtained using only one, two or three operators. When **GA** works with four operators switched on and an initial population is created by a nearest neighbour heuristic, the appropriate parameter values are $p_{\mathrm{AX}} = 0.1$ and $p_{\mathrm{CX}} = 0.2$. The most suitable value of *Fmultiple* has also been tested; the results show that *Fmultiple* $= 2$ is relevant (with option *NS*, also the value of 3 works well).

The above described experiments indicate the optimal values of **GA** parameters. A number of experiments have been performed to check these values, with random initial populations ($p_{\mathrm{CX}} = 0.3$, $p_{\mathrm{AX}} = 0.6$, $p_{\mathrm{mut}} = 0.15$, $p_{\mathrm{inv}} = 0.6$, *Fmultiple* $= 2$, *Population* $= 100$) and with the nearest neighbour heuristic ($p_{\mathrm{CX}} = 0.2$, $p_{\mathrm{AX}} = 0.1$,

2. The computer program used in all the experiments was developed by Adrian Stefaniak during work on his master thesis [19] at the Department of Computer Science, Wroclaw University of Technology. Also some results presented in the paper were obtained by A. Stefaniak during his work on the master thesis under supervision of the author of the paper.

$p_{\text{mut}} = 0.15$, $p_{\text{inv}} = 0.6$, $Fmultiple = 3$, $Population = 100$). Experiments with different sizes of the problem show that optimisation of individuals in the initial populatin by the nearest neighbour heuristicis very significant: the best solution is found quicker, for 100 cities the computation time is about 30% shorter. Additionally, for **TSP** with more than 100 cities, *e.g.* 200 and more, the quality of solutions obtained is much better.

Simulated Annealing (**SA**) requires that the user settles a number of parameters influencing the efficiency of **SA**. Parameters specific for **SA** are: *Annealing_factor*, responsible for the rate of temperature decrease, number of iterations (*How_many_steps*), initial (*Begin_temp.*) and final temperature (*End_temp.*), maximum number of trials of route searching during one iteration (*Max_path_PS*), and maximum number of changes of route for a single iteration (*Max_changes_PS*). It is possible to start with a route given by the nearest neighbour heuristics.

By means of simulations, we have found, that for 100 cities, the initial value of temperature 5 is optimal. The result confirms recommendation found in [20]:

$$T_{\text{initial}} = \frac{Length\_of\_initial\_route}{Number\_of\_cities^{1.5}}. \tag{15}$$

In our experiments, such theoretical value is equal to 4.9. While searching for an optimal number of modifications in a single iteration (*Max_changes_PS*), other experiments have been made. According to simulation results, *Max_changes_PS* can be assumed as follows:

$$Max\_changes\_PS = 10\, Number\_of\_cities. \tag{16}$$

Computation time does not depend significantly on this parameter.

The parameter *Max_path_PS* strongly influences computation time, as well as solution quality. For 100 cities, values from the range [6400, 12800] seem to be adequate. Because the quality of solution (the length of the obtained route) is a significant issue in **TSP**, we assume that:

$$Max\_path\_PS = 100\, Number\_of\_cities \tag{17}$$

Another tested parameter is the annealing factor. Selection of this parameter together with the number of iterations is very significant for the quality of results; **SA** needs time to anneal temperature to zero. A higher value of the annealing factor causes that temperature is reduced slowly. For 200 cities, the value of 0.99 gives good results if **SA** is not stopped too early. Further tests (with different numbers of cities) confirm that higher annealing factors give better results, but lower values (*e.g.* 0.1) can produce good solutions only if **SA** works for a longer time. Using the nearest neighbour heuristic to produce the initial tour does not significantly influence the efficiency of **SA**. The quality of the obtained route and computation time are not better than the previous ones.

An **Ant Colony System** (**ACS**) is very sensitive to the initial configuration of cities, therefore many different configurations were used to select the suitable values of parameters.

The **ACS** parameters are as follows: *Number_of_ants* is the size of ants' population, *Beta* controls ants' decisions, *Gamma* controls the rate of pheromone

evaporation, *Q3* modifies the amount of pheromone deposited, *Tau* is the initial value of pheromone, and *Steps_to_do* is the number of iterations.

Experiments were made with 100 and 50 cities (about 9000 runs). The results show that parameter *Beta* plays a crucial role. The best results are obtained with $Beta = 6$. Enlarging *Q3* causes a worsening of the performance.

For 100 cities, with $Beta = 6$, $Gamma = 0.5$, $Q3 = 100$, $Tau = 0.1$, we have searched for an optimal number of ants. A small number of ants (1, 5, 10, 16) is able to find worse (longer) routes than larger populations. It seems that a population of 32 ants repeatedly gives satisfactory results. Results obtained for other values of parameters ($Beta = 6$, $Gamma = 0.6$, $Q3 = 10$, $Tau = 0.1$) are similar to the previous ones. Experiments searching for an optimal value of *Tau* (with $Beta = 6$, $Gamma = 0.5$, $Q3 = 100$, 32 ants) show that $Tau = 0.1$ yields the best results.

Tabu Search (**TS**) has been initially tested taking into account particular heuristics. The required parameters are: length of tabu list (*TL_length*), number of iterations (*Steps_to_do*), choice of heuristic (*2-Opt* or *3-Opt*), including or excluding the optimisation process when **TS** reaches a local optimum (*Optimize_TS*), switching on the roulette wheel (*Use_roulette*) in the edge selection process, and the nearest neighbour (*Use_it*) or the random (*Randomize*) method for initial route construction.

The nearest neighbour heuristic applied with the 2-Opt method searching for a new neighbour solution gives better results: shorter routes and lower dispersion of results. Similarly, using the nearest neighbour with 3-Opt improves performance of Tabu Search. Including the optimisation method makes Tabu Search much more efficient: obtained results are better.

The tabu list is a very significant parameter of the Tabu Search method. Its length is responsible for the balance between efficiency and capacity to escape from local optima. Experiments (for 200 cities) show that, in our program, the optimal length of the tabu list is about 40, which constitutes 20% of the number of cities. For 100 cities the length of the tabu list should be between 10 and 20. A long tabu list requires a lot of computation time, too short a tabu list can cause stagnation. As we remember, in [14] we can find that the length of tabu lists should be 25% of the number of cities. The roulette wheel method as a way of edge removal brings slightly shorter routes and saves computational time. Combining the roulette wheel and the optimisation process, the 3-Opt heuristic gives better results, taking into account the required computational time. For large **TSP** (more than 100 cities), the 2-Opt heuristic seems to be more convenient: it is 100 times quicker than 3-Opt. But the results given by 3-Opt have smaller dispersion. This means that if we require a quick solution, we should use 2-Opt, but if we need a good solution (short route), the 3-Opt heuristic is better.

A **Hopfield Analog Neural Network** (**NN**) requires determination of its specific parameters, *i.e.* for modification of energy – *Time_increment*, *Tau*, three parameters called *Lambda[1]*, *Lambda[2]*, *Lambda[3]* and coefficient *Inverse_slope* are required during initialisation of the network. Additionally, we can switch on *Adjust_Lagrange* parameter, which causes dynamical modification of the *Lambda* parameters.

The Neural Network used in this simulation study is able to find solutions only for a limited number of cities. Initial tests have been made for 50 cities.

The results show that the three *Lambda* parameters equal to 1 work well, however values 1.5, 1, 0.5 respectively, also give good results (experiments were made with $Time\_increment = 0.0005$, $Tau = 1$, coefficient $Inverse\_slope = 0.03$, *Adjust_Lagrange* switched off). With *Adjust_Lagrange* switched on, the optimal values of all *Lambda* parameters are equal to 0.1. *Time_increment* equal to 0.00075 gives good results and is acceptable taking into account time requirements. *Inverse* equal to 0.03 assures good solution and acceptable computation time. The optimal value of *Tau* is 0.06, no matter if dynamic modification is switched on or off. The higher values give worse results, but the required computation time is lower.

7.2. Comparison of efficiency of selected meta-heuristics

Because the main objective of our study is to compare the efficiency of selected methods, the presented results have been obtained using each method with the most suitable parameters. They have been set as below:

- **GA** without nearest neighbour heuristic: $p_{\mathrm{CX}} = 0.6$, $p_{\mathrm{AX}} = 0.3$, $p_{\mathrm{mut}} = 0.15$, $p_{\mathrm{inv}} = 0.6$, $Fmultiple = 3$, $Population = 100$ individuals;
- **GA** with nearest neighbour heuristic: $p_{\mathrm{CX}} = 0.2$, $p_{\mathrm{AX}} = 0.1$, $p_{\mathrm{mut}} = 0.15$, $p_{\mathrm{inv}} = 0.6$, $Fmultiple = 3$, $Population = 100$ individuals;
- **SA**: $Annealing\_factor = 0.99$, $Begin\_temp.$ – done automatically (Equation (15)), $End\_temp. = 0$, $Max\_path\_PS = 100\,Number\_of\_cities$, $Max\_changes\_PS = 10\,Number\_of\_cities$;
- **ACS**: $Beta = 6$, $Gamma = 0.5$, $Q = 100$, $Tau = 0.1$;
- **TS**: with roulette wheel and additional optimisation;
- **NN**: $Lambda[1] = Lambda[2] = Lambda[3] = 1$, $Inverse\_slope = 0.03$, $Time\_increment = 0.00075$, $Tau = 0.6$, *Adjust_Lagrange* parameters switched off.

For all conditions, each algorithm has been run 10 or 20 times.

Experiment 1. *Sensitivity of the methods to the number of cities*

Simulations for **TSP** consisting of different number of cities, varying from 20 to 1000, have been made. Results obtained are collected in Table 1.

All the used methods are able to find an optimal route for a small **TSP**, *e.g.* one containing 20 cities. Only the Hopfield Neural Network has problems with tuning its solution. Genetic algorithm, when it uses the nearest neighbour heuristic in the step of creation of an initial population, works better. Sometimes for a small number of cities it has the shortest route in the initial population. The Ant Colony System and Tabu Search return solutions with the shortest processing time. For greater number of cities, Simulated Annealing and Tabu Search are the most efficient methods. **TS** with the 2-Opt heuristic is much quicker than **SA**, albeit for large numbers of cities (between 500 and 1000) it produces slightly worse solutions. The advantage of **SA** is an exactly defined stop time: when the temperature is near to zero. **GA** is able to find a near optimal solution for more than 50 cities, but it consumes a lot of computation time (compared with other methods). The Ant Colony System is a very fast method, but the results become worse as we increase the number of cities. **ACS** is sensitive to the initial configuration of cities, but not so strongly as the Hopfield Neural Network.

Table 1. Results given by different methods for **TSP** – dependency on the number of cities

Algorithm	Number of cities	The best route	Average route	The worst route	Dispersion	Time [ms]
GA*	20	380.06	386.07	408.34	8.85	774
GA**	20	380.06	380.06	380.06	0	0
SA	20	380.06	380.06	380.06	0	448
ACS	20	380.06	382.45	427.96	10.7112	1
TS	20	380.06	382.84	393.95	5.70	17
NN	20	401.90	428.78	461.76	14.5260	1800
GA	50	673.30	679.15	683.60	4.5479	8121
SA	50	645.14	646.91	652.41	2.5117	3493
ACS	50	678.51	678.51	678.51	0.0000	27
TS	50	645.14	652.35	681.95	9.52	456
NN	50	823.49	1027.58	1242.65	96.4085	109934
GA	100	804.92	811.87	814.92	4.0932	29478
SA	100	782.80	790.95	801.66	5.4153	10060
ACS	100	842.55	878.49	903.89	18.3044	1877
TS	100	788.15	798.30	809.42	6.39	3179
NN	100	2203.37	2412.43	2665.69	193.7298	364263
GA	150	961.41	983.06	995.77	15.0987	173148
SA	150	936.47	952.83	975.14	12.3128	20765
ACS	150	1096.55	1102.36	1110.21	4.1938	1096
TS	150	978.72	993.64	1002.02	11.84	3
NN	150	6882.44	7026.00	7198.01	159.6962	29700
GA	200	1166.77	1190.56	1200.75	13.5490	372922
SA	200	1111.91	1135.97	1152.79	9.8380	28636
ACS	200	1259.99	1276.63	1294.96	11.5443	2752
TS	200	1116.29	1143.07	1155.65	13.77	6589
NN	200	9227.68	9489.80	9626.14	155.3831	222041
SA	500	1659.67	1676.14	1688.21	8.90	99111
TS	500	1689.48	1706.88	1725.69	11.07	42653
SA	1000	2432.11	2445.76	2461.97	10.02	459206
TS	1000	2433.21	2448.95	2464.55	10.35	115420

* – without optimisation of initial population by the nearest neighbour heuristic

** – with optimisation initial population by the nearest neighbour heuristic

Experiment 2. *Sensitivity of the methods to the number of removed edges – optimisation of an incomplete* **TSP**

In this experiment the methods are used for an incomplete **TSP**, which means that not all the cities are directly connected. The number of removed routes (edges) is equal to 100 or 1000. In all simulation runs, **TSP** consisting of 50 cities has been solved. The results are collected in Table 2. Parameters of the methods are the same as in Experiment 1, only in Tabu Search the 3-Opt heuristic has been used.

All simulations show that only Simulated Annealing and Tabu Search with the 3-Opt heuristic are able to solve **TSP** (find an optimal route) with 50 cities and 100 removed edges. As the number of removed edges increases, the computation time of **SA** and **TS** decreases. The result occured in 20 simulation runs, so it is hardly incidental. It seems to be connected with the way in which these methods search for neighbour solutions. A smaller number of edges in the **TSP** problem leads to a smaller number of tours to check. **SA** works quicker than **TS**, which is an effect of including the 3-Opt heuristic into **TS**. The Ant Colony System relatively quickly finds a solution of **TSP** with 100 removed edges, but – similarly to the Hopfield Neural Network – produces

Table 2. Results given by different methods for **TSP** – dependency on the number of removed edges

Algorithm	A number of removed edges	The best tour	Average tour	The worse tour	Dispersion	Time [ms]
GA	100	745.97	805.41	859.85	32.5961	19529
SA	100	667.55	669.74	681.07	3.9647	3385
ACS	100	746.84	773.61	809.88	17.1711	269
TS	100	667.55	674.29	684.15	5.5533	10369
NN	100	1106.50	1299.70	1444.73	80.8202	54505
GA	1000	986.44	1019.35	1051.73	23.2975	27048
SA	1000	866.99	872.10	879.48	4.6392	2132
ACS	1000	911.91	944.06	979.10	18.8485	374
TS	1000	884.72	899.16	912.90	9.5848	9876
NN	1000	2568.79	2568.79	2568.79	0	0

a solution far from the optimum. However, for 1000 removed edges the error is not very significant. **NN** cannot find any solution of **TSP** with 1000 removed edges. Such results are probably caused by the implemented way of edge removal: the distance on a removed edge is assumed to be very large (compared with real distances it can be treated as a non-existing connection).

Experiment 3. *Efficiency of the methods for a* **TSP** *with uniformly distributed cities*

The study presented below concerns efficiency of the methods tested on a **TSP** problen in which cities are distributed uniformly. The number of cities varies between 16 and 100. The parameters of the methods are the same as in Experiment 1. Results obtained are summarised in Table 3.

The Ant Colony System and Tabu Search emerge as the quickest methods for 16, 25 and 36 cities. All methods solve this task easily, only **NN** has a problem with accuracy. For 25 cities **ACS** is the fastest method, however, for 36 cities **TS** is a bit faster. **GA** is able to find a solution, but works slowly and dispersion of the produced solutions is high. **SA** and **TS** work well on **TSP** with 49 cities, **GA** has a little problem with the optimum, produces an acceptable route but requires relatively long simulation time. **ACS** is able to find an acceptable solution relatively quickly, but evidently has a problem with tuning. **NN** seems to produce solutions accidentally: the dispersion is relatively high. **TS** has found an optimal tour for 64 cities in all runs and in relatively short time. **SA** has also found solutions in all runs, while **ACS** sometimes produces wrong answers. **SA** is able to find the shortest route along 81 cities, **ACS** works very fast but imprecisely, **TS** returns better results than **ACS** but imprecise, and the working time of **TS** is long. In **TS**, it is possible to improve the obtained solution by increasing the number of iterations, but processing time also increases. One hundred cities **TSP** is too big a task for **NN**, for which it is quite unhelpful. **TS** has found an optimum in all runs, **SA** and **ACS** also have found optimal tours, but not in all runs. In acceptable time, **GA** has problems with the quality of its solutions: the produced tour is far from the optimum.

Experiments show that for odd numbers of cities (49 and 81), distributed uniformly on a rectangle, the tested algorithms have some problems with finding the optimal tour. **SA** is usually able to find an optimum, but needs more time than

Table 3. Efficiency of methods for uniformly placed cities in the **TSP** problem

Algorithm	Number of cities	The best route	Average route	The worst route	Dispersion	Time [ms]
GA	16	320.00	320.00	320.00	0	1016.0
SA	16	320.00	320.00	320.00	0	9.5
ACS	16	320.00	320.00	320.00	0	0.5
TS	16	320.00	320.00	320.00	0	0.5
NN	16	336.56	351.48	353.13	5.24	2951.6
GA	25	422.35	423.82	426.04	1.46	5152.6
SA	25	422.35	422.37	422.63	0.09	113.4
ACS	25	422.35	422.35	422.35	0	2.0
TS	25	422.35	422.35	422.35	0	4.6
NN	25	462.10	473.27	489.40	9.31	8247
GA	36	508.00	517.98	523.32	6.10	10443.2
SA	36	508.00	510.00	512.00	1.63	144.3
ACS	36	508.00	509.72	525.20	5.44	17.1
TS	36	508.00	509.80	512.00	1.48	12.1
NN	36	627.82	649.46	673.84	16.48	39625.7
GA	49	621.46	633.21	650.23	9.37	17669.5
SA	49	610.69	613.43	616.70	1.89	442
ACS	49	632.69	636.35	640.00	3.85	10.0
TS	49	612.69	619.98	626.08	4.57	27.0
NN	49	852.63	916.84	961.83	42.10	98691.5
GA	64	731.34	743.18	758.68	9.65	54302.9
SA	64	704.00	704.00	704.00	0	1100.9
ACS	64	704.00	706.72	717.60	5.73	11.9
TS	64	704.00	704.00	704.00	0	18.6
NN	64	1128.77	1193.66	1269.37	47.06	197626.2
GA	81	851.36	871.43	896.85	14.05	86844.0
SA	81	814.14	815.80	822.43	3.49	681.3
ACS	81	856.57	862.17	870.99	6.52	19.3
TS	81	822.43	831.95	838.99	4.80	72.4
NN	81	1454.51	1473.70	1478.12	9.35	51506.5
GA	100	979.83	996.92	1025.93	16.54	128738.9
SA	100	900.00	903.73	907.46	3.93	2378.4
ACS	100	900.00	904.45	911.13	5.74	29.1
TS	100	900.00	900.00	900.00	0	57.5
NN	100	1658.04	1658.04	1658.04	0	0

ACS or **TS**. It seems that **TS** gives a little better results than **ACS**, because imprecise solutions produced by **TS** are closer to the optimum than those produced by **ACS**. For 49 and more cities **GA** has a problem with the precision of its solutions and with computation time. A neural network cannot give precise solutions even for small problems.

8. Summary

All meta-heuristics tested in the paper are very popular optimisation techniques. I am frequently asked: what meta-heuristic should be used for this or that problem? There seems to be no generally good answer for this question. Therefore, I have attempted to find a partial answer to the question. Five approaches have been selected and simulation studies of their efficiency on the basis on one task, the **TSP** problem, have been performed. Using a purpose-developed computer program, about 40000 runs have been made. Part of them have been used to recognise the sensitivity

of the methods to their parameters and to discover the appropriate combinations of parameters. To compare efficiency of the methods, suitable sets of parameters were settled for each method in each experiment.

All experiments show that the best results, independently of the complexity of the problem, are produced by Simulated Annealing and Tabu Search. These two meta-heuristics give similar results, are flexible and work well also with incomplete graphs. Simulated Annealing has some advantages in that some of its parameters can be determined automatically, ensuring adequate values for solving **TSP**, and in the stop criterion being clear. Including additional optimisation into Tabu Search improves its performance. Application of the 2-Opt heuristic with the roulette wheel method of edge selection makes **TS** work relatively quickly. But the problem is to define a clear stop criterion adequate for the given **TSP**. Genetic Algorithm needs reconfiguration of parameters for particular **TSP**'s and simulation time for satisfactory results is relatively long. The Ant Colony System is able to find an imprecise solution relatively quickly, but it has problems with tuning the route. Another feature of **ACS** is its sensitivity to the initial configuration. Experiments with the Hopfield Neural Network do not allow us to say that it is a satisfactory method of solving **TSP**. Possibly, changes in network configuration may bring improvement of its performance.

Recently, hybrid methods have become more popular in solving difficult problems. It seems that for **TSP** it is worth developing a hybrid method, for example an Ant Colony System starting as the first method, the obtained approximate solution being fed as an input tour for the Tabu Search method. Other combined techniques, *e.g.* Genetic Algorithm and Tabu Search, could produce good results. Such studies are envisaged to be done in the nearest future.

References

[1] Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley
[2] Holland J H 1975 *Adaptation in Natural and Artificial Systems*, The University of Michigan
[3] Kwaśnicka H 1999 *Evolutionary Computations in Artificial Intelligence*, Oficyna Wydawnicza, Wroclaw University of Technology, Wroclaw, Poland (in Polish)
[4] Michalewicz Z 1992 *Genetic Algorithms + Data Structures = Evolutionary Programs*, Springer-Verlag
[5] Schmit L and Amini M M 1998 *European J. Operational Research* **108** 551
[6] Cerny V 1985 *J. Optim. Theory Appl.* **45** 41
[7] Kirkpatrick S, Gellat C D and Vecchi M P 1983 *Science* **220** 67
[8] Metropolis N, Rosenbluth A, Rosenbluth M, Teller A and Teller E 1953 *J. Chem. Phys.* **21** 1087
[9] Davis L 1987 *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publishers, Inc., San Mateo, California
[10] Dorigo M, Di Caro G and Gambardella L M 1998 *Tech. Rep. IRIDIA/98-10*, Universit'e Libre de Bruxelles, Belgium, http://iridia.ulb.ac.be/∼mdorigo/ACO/ACO
[11] Dorigo M 1992 *Optimization, Learning and Natural Algorithms*, PhD Thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, http://iridia.ulb.ac.be/∼mdorigo/ACO/ACO.html
[12] Bonabeau E, Dorigo M and Theraulaz G 1999 *From Natural to Artificial Swarm Intelligence*, Oxford University Press
[13] Deneubourg J L, Aron S, Goss S and Pasteels J M 1990 *J. Insect Behaviour* **3** 159
[14] Tsubakitani S and Evans J R 1998 *Computers Ops. Res. Elsevier Science Ltd.* **2** (25) 91

[15] Helsgaun K 2000 *European J. Operational Research* **126** 106
[16] Korbicz J and Obuchowicz A 1994 *Artificial Neural Networks. Basis and Applications*, Akademicka Oficyna Wydawnicza PLJ, Warsaw, Poland (in Polish)
[17] Mueller B, Reinhardt J and Strickland M 1995 *Neural Networks*, Springer Verlag
[18] Tadeusiewicz R 1993 *Neural Networks*, Akademicka Oficyna Wydawnicza, Warsaw, Poland (in Polish)
[19] Stefaniak A 2001 *Comparison of Selected Methods Inspired by Nature Using **TSP** Problem as an Example*, MSc Thesis, Department of Computer Science, Wroclaw University of Technology, Wroclaw, Poland (in Polish)
[20] Podleśny A 1999 *About Simulated Annealing and its Application to Optimal Block Generation*, http://chimera.ae.krakow.pl/ketrii/sssg/sssgdaw/a99/podlesny.htm