Rescue Operation Planning by Soft Computing Techniques

Miroslav Kulich
Certicon a.s.
Applied Research Department
Václavská 12
Prague 2, 120 00
Czech Republic
Email: kulich@certicon.cz

Jan Faigl, Jiří Kléma, and Jiří Kubalík
The Gerstner Laboratory for Intelligent Decision Making and Control
Faculty of Electrical Engineering
Department of Cybernetics
Technická 2, Prague6, 166 27
Czech Republic
Email: {klema,kubalik,faigl}@labe.felk.cvut.cz

Abstract— **This paper presents an application of three soft computing techniques – ant colony optimisation, genetic algorithm, and neural networks to rescue operation planning. It considers the task as the multiple travelling salesmen problem and proposes suitable heuristics in order to improve the performance of the selected techniques. Then it applies the implemented solutions to a real data. The paper concludes with comparison of the implementations and discussion on the aspects of the utilisation of the proposed heuristics.**

I. Introduction

Recently, significant research initiatives have been undertaken, that specifically focus on the problem of developing systems to support search and rescue operations mainly in urban environments. The aim of these operations is activity coordination and planning for a rescue squad in case of emergencies or catastrophes in order to search for injured persons and to review of all specified places.

The main task (searching for survived persons or objects with an unknown position) is in the literature mentioned as the exploration problem: plan a tour for each squad member so that every point in the environment can be visible by at least one member. This problem can be solved in two steps. Locations for sensing (the sites, from which all points of the environment are visible) are found in the first step, followed by finding an optimal strategy how to connect these cities by m squad members. Such a problem can be restated as the Multiple Travelling Salesmen Problem (MTSP): given N cities and A agents, find an optimal tour for each agent so that every city is visited exactly once. A typical criterion to be optimized is the overall time spent by the squad (i.e., the slowest team member) during the task execution. The ability to solve the MTSP in a fast and optimal way therefore plays an important role in the rescue scenario.

During the last decade, a large number of applications of *ant colony optimisation* (ACO), *genetic algorithms* (GA), and *neural networks* (NN) to the combinatorial problems,and in particular the TSP, were published, showing that the techniques are capable of solving those problems, see [6], [2].

Next section briefly introduces ACO and GA and describes implementations with heuristics designed for solving MTSP. Moreover, our solution based on self-organising NN is presented. The experiments on real data are presented in Section III. The paper concludes with discussion on the performance of the used techniques.

II. Applied Soft Computing Techniques

A. Hybrid Genetic Algorithm

Genetic algorithms (GAs) are probabilistic search and optimisation techniques, which operate on a population of chromosomes, each representing a potential solution to the given problem [5], [6]. Each chromosome is assigned a fitness value expressing its quality. Such a population is evolved by means of reproduction and recombination operators in order to breed near optimal solutions. The evolution is running until some termination-condition is fulfilled and the fittest chromosome encountered during the evolution is returned as the found solution.

Representation. In our work we chose the natural path representation [6] for encoding the agents tours in the form of the linear string. This means that the tour is represented by a sequence of integer numbers, where each number represents the name of the city and the order of numbers is the order of cities in the tour. Here we consider the number of agents A to be greater than one, so the complete plan should represent A tours. This is simply ensured so that the starting city receives A copies in each legally structured chromosome.

Crossover and Mutation. We used the edge recombination crossover operator introduced in [7]. This operator is based on the fact that the links between cities represent an important information that should be transferred from parental solutions to their offspring. Thus the operator builds the offspring so that it tries to use as much of the links present in the parents as possible.

Besides the mutation involved in the crossover operation we used a mutation operator as well. The operator finds a sequence of cities within the whole plan by random and applies the inversion operation on it.

Single Tour Optimisation – STO. GAs have been shown to be powerful optimisation techniques. However when using GAs for any problem one should try to incorporate the

knowledge of the problem into the GA as much as possible. In this work we use techniques for local optimisation of the plan. The single tour optimisation takes as an input the set of N cities assigned to one agent and generates the optimised tour through the cities as follows

1) Sort the cities according to their distance from the *depot* in ascending order and store them in array $C_{i=1}^{N}$. The *depot* is C_1.
2) Take the first two cities (the closest to the *depot*) and make the tour $C_1C_2C_3$.
   Set the counter of used cities to $k = 3$.
   Calculate the length of the partial tour.
3) If $k == N$ then end.
4) Take the next unused city $next = C_{k+1}$.
5) For all links C_iC_j present in the partial tour calculate the added value

$$AV = dist(C_i, next) + dist(C_j, next) - dist(C_i, C_j),$$

   where $dist(.)$ is a distance between two points.
6) Insert the city $next$ into the tour in between cities C_i and C_j, for which the added value is minimal.
7) Increment the counter k.
8) Goto step 3.

Note, that this algorithm does not guarantee finding an optimal tour through the given set of cities so the tour generated by the algorithm is accepted only if it is shorter than the original tour.

Longest Tour Shortening – LTS. The second optimisation routine used in our implementation tries to shorten the longest tour of the plan. First, it selects some city C of the longest tour by random. Then it adds the city C into all other tours using the strategy described in the algorithm above - steps 5.-6. Finally if there exist one or more tours with the node C such that they are shorter than the original longest tour, then the city C is removed from the current longest tour and is placed to the tour, which has the shortest length with the city C. Put simply, this routine tries to delegate a part of the load of the most busy agent to some other one.

Note that the changes made by the LTS routine are accepted only if they improve the whole plan. Contrary to that the *single tour optimisation* routine does not necessarily have to improve the whole plan to be accepted. Its role is to make all the individual tours shorter and possibly without crossings.

Evaluation Function. Generally the evaluation function is the only information the GA has to direct the search for promising solutions. So it has to cover all the aspects of the sought solution. In our case we are seeking the plan such that (1) the total time needed to complete all the tours is minimal and (2) all agents must be involved. For evaluation of potential solution i we used the fitness function defined as follows

$$fitness(i) = max\_tour(i) * \frac{All\_Agents}{Involved\_Agents},$$

where $max\_tour(i)$ is the length of the longest tour of the whole plan i, All_Agents is the number of agents we want to use, and $Involved\_Agents$ is the number of agents with non-zero length tour in the plan. The goal is to minimise this function. The function is designed so that if two solutions have the longest tour of the same length then the one with more agents involved have smaller fitness value. In other words, the term $All\_Agents/Involved\_Agents$ represents a penalty for each agent with zero-length tour in the plan. The term $All\_Agents/Involved\_Agents$ pushes the GA towards solutions with all agents involved.

Evolutionary Model. We used a genetic algorithm with the steady-state evolutionary model. This means that is operates on a single population. The whole population is randomly initialised first. Then the population is modified through the steps – selection of parents,cross the parents over, and applying the mutation, STO and LTS to the offspring – until the stopping criterion is fulfilled. The currently worst individual of the population is replaced by the newly generated solution in each generational cycle.

B. Ant Colony Optimisation

Ant algorithms are multi-agent systems in which the behavior of each single agent called ant is inspired by the behavior of real ants. The basic ant colony optimization (ACO) idea is that a large number of simple artificial agents are able to build good solutions to hard optimization problems via low-level based communications. Real ants cooperate in their search for food by depositing chemical traces (pheromones) on the floor while artificial ants cooperate by using a common memory that corresponds to the pheromone of real ants [3].

Basic implementation. The basic implementation (denoted as ACO) is based on proposals and recommendations published in [1], [3]. The problem is approached to the traditional TSP as follows: the depot is duplicated a number of times equal to the number of agents, $k = 1, ..., N + A - 1$ ants has to visit all the cities exactly once constructing a plan consisting of A tours. The next city to be visited is selected by a random propositional rule [2] based on the pheromone trails and visibility (reciprocal of the true distance) connected to the individual arcs. The pheromone trails are updated as proposed in [4]. The final plan length used to select the best ant and to update the pheromone trails has to be adjusted to MTSP task. In our implementation it is calculated as follows:

$$length(k) = max\_tour(k) - min\_tour(k) + \frac{tour\_sum(k)}{2},$$

where $max\_tour(k)$ and $min\_tour(k)$ are the true lengths of the longest and shortest tours of the plan, $tour\_sum$ is the total length of all the tours. The ant has not only to minimise the longest tour but also optimise distribution of tours among agents and also the rest of the plan in order to update the pheromone trails correctly.

Heuristic Plan Optimisation. Hybrid ant colony optimization (hACO) uses *the 2-opt-heuristic*, which is often applied in TSP like tasks [1]. The heuristics generates a so called 2-optimal tour, i.e., the tour in which there is no possibility to shorten the tour by exchanging two arcs.

The second heuristic applied in hACO is *the longest tour shortening* described in Section II-A. The resulting hybrid algorithm can be described as follows:

1) For each iteration do:
   a) For each ant $k = 1, ..., N + A - 1$ placed at different starting city generate a new plan using ACO outlined above.
   b) For each ant improve all tours using the 2-opt-heuristic.
   c) For each ant try the random LTS routine once.
   d) Select the best ant and use it to update the pheromone trails (the iteration best strategy).
2) Take the best ant over all the iterations and try the random LTS routine repeatedly until no change appears for $2 * N/A$ trials.

C. Neural networks

Neural networks and competition-based structures especially are widely used for solving optimisation problems. Our approach extends Somhom's algorithm described in [8]. The idea of the algorithm is to represent a path of each particular agent (salesman) by a ring of neurons, where neighbouring neurons are connected. At each iteration, a nearest neuron to a randomly selected city is determined and together with its neighbours moved closer to the guard. The algorithm can be described as follows:

1) Denote N as the number of cities, $\alpha = 0.1$, and A the number of agents. A rings are initially created so that each ring consists of $M = \frac{2N}{A}$ neurons. The neurons are positioned on a small ring equidistantly and one neuron is placed in the depot.
2) Choose a permutation of cities randomly, i.e. C_{p_i} is i-th city in the permutation.
3) For each city C_{p_i} in the permutation do:
   a) Determine the nearest neuron to the C_{p_i} according to the following rule:

$$N_k^r = \arg\min_{r,k} \left(dist(N_k^r, C_{p_i}) * weight(r)^4 \right),$$

   where

$$weight(r) = \frac{length(r) - AVG}{AVG}$$

   and N_k^r is k-th neuron on the r-th ring, $dist(.)$ is a distance between two points, $length(.)$ is the length of a ring and AVG is the average length of a ring.
   b) Adapt coordinates of the winner and its neighbors on the ring r:

$$N_b^r = N_b^r + \mu f(b,k) dist(N_b^r, C_{p_i}),$$

   where $\mu = 0.6$ and $f(b,k)$ stands for the neighborhood function:

$$f(b,k) = \begin{cases} e^{\frac{d^2}{G^2}} & d < 0.2M, \\ 0 & \text{otherwise} \end{cases}$$

$$d = \min(|b-k|, M - |b-k|),$$

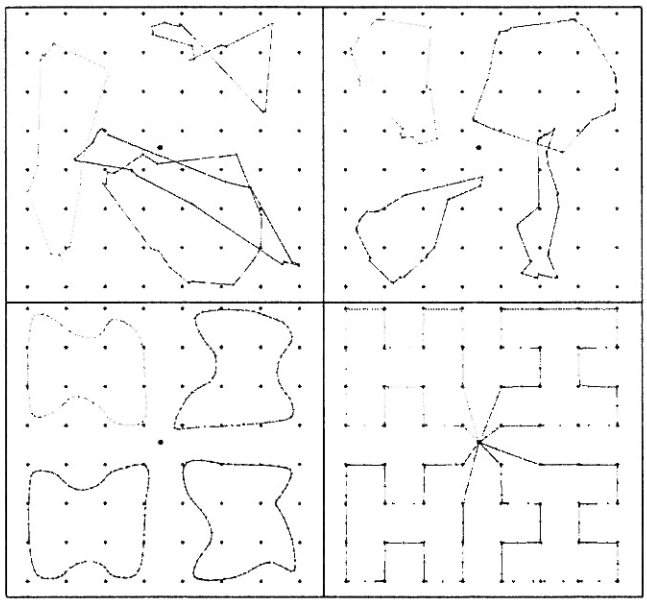

Fig. 1. Evolution of neural networks. Figures show a state of NN after 10, 30, 50, and 60 iterations.

   where d stands for the cardinal distance measured along the ring between neurons b and k and G is a constant initially set to 10.
4) Set $G = (1 - \alpha)G$.
5) If the convergence criterion is fulfilled then stop, otherwise go to step 2. The criterion can be a maximum distance between city and a nearest neuron to this city. The criterion is satisfied if the distance is smaller than a defined threshold.

Generalization for environments with obstacles. The above-described algorithm gives results comparable to genetic algorithms and ant colonies optimisation. Unfortunately, the algorithm works only in the case that distances between a neuron and a city can be determined effectively. It holds only for environments without obstacles, where the distance is clearly an Euclidean distance.

The situation is more difficult for environments with obstacles (represented by a polygon with holes), because the distance between two points has to be determined with respect to these obstacles. The simplest approach is based on visibility graphs:

1) Let C is a city and $\{V_i\}$ for $i = 1 \ldots N_v$ is a set of vertices of obstacles.
2) Construct a visibility graph of $\{C\} \cup \{V_i\}$.
3) Use Dijkstra algorithm to compute a shortest path and its length from C to each vertex V_i. Denote the i-th path P_i and its length $d(CV_i)$.

Using this structure queries "what is the distance of a point p to C" (so-called one-point queries) can be answered in $O(n \log n)$ time:

1) Determine vertices that are visible from point p and denote them $V^p = \{V_i^p\}$, where $i = 1 \ldots N_p$. Furthermore, $|v_i, p|$ stands for the Euclidean distance of

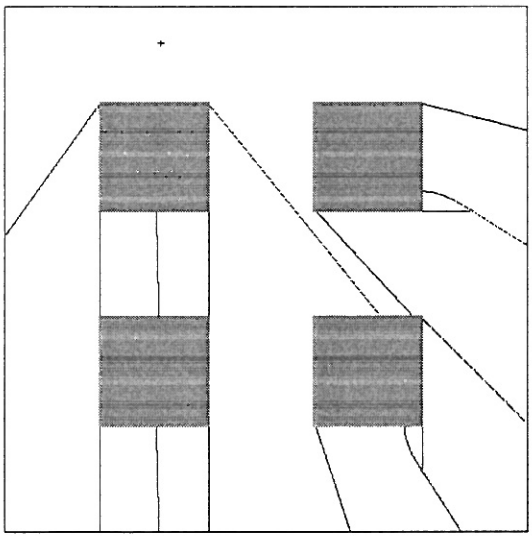

Fig. 2. Shortes Path Map. The city is represented by the cross, obstacles are in grey.

v_i and p.

2) Find a vertex $v \in V^p$ so that the length from C to v plus the length from v to p is minimal over V^p:

$$v = \arg \min_{v_i in V^p} d(C, v_i) + |v_i p|.$$

3) The desired minimal path from C to p is then the union of the path from C to v and the line vp while the distance is $d(C, v) + |vp|$. The distance of two points with respect to obstacles is called Euclidean geodesic.

$O(nlogn)$ time of this naive algorithm is too high what causes that the neural net optimisation runs several minutes even for a scene with 250 vertices. Therefore more sophisticated structure has to be used that provides answering one-point queries more effectively.

The structure we use is the Shortest Path Map (SPM). $SPM(C, map)$ with respect to a point (city) C and polygon with holes (map) is a partition of a free space defined by map into maximal regions (called cells) that correspond to sets of points with the same root (first vertex on the shortest path from a point to C, i.e. v in the naive algorithm) or a set of roots with respect to C. Example of SMP is shown in Figure 2.

We use a subquadratic algorithm for computing the SPM presented in [11]. This approach is based on continuous Dijsktra paradigm [10], which simulates the effect of a "wavefront" propagating out from a city. The wavefront at a distance d is the set of all points of free space that have an Euclidean geodesic distance equal to d. The structure of the wavefront changes while d is growing only at some specific distances (events) due to the following three possibilities:

- a part of the wavelet disappears
- the wavelet collides with an obstacle vertex
- a part of the wavelet collides with other part.

The events correspond to changes in the SMP. The basic idea of the algorithm is therefore to detect these events and to process them.

The SMP is a subdivision of a plane, where boundaries between two cells can be either a portion of straight line or a hyperbolic arc. If we approximate each hyperbolic arc by a polyline (with a defined precision), then a cell can be represented by a polygon. Having a SMP for each city, we can compute the overlay of these subdivisions using standard plane sweep algorithm [9] sequentially:

1) Let SMP_i is a shortest path map with respect to the city C_i, $i = 1 \ldots N$ and S stands for a resulting subdivision.
2) $S = C_1$.
3) For $i = 2 \ldots N$ do
   $S = S \biguplus C_i$, where $\biguplus$ stands for the overlay.

After applying the previous algorithm, we get a subdivision with the following properties:

- Each cell is represented by a polygon.
- For each city and cell there exists one root that is same for all points of the cell. In other words, all points of the cell have the same path topology to a city.

These properties guarantee that the subdivision allows to find the length of the shortest path between an arbitrary point of the plane (neuron) and a city in time $O(logk)$ (by point location), where k is the number of points of the subdivision. Moreover, a shortest path can be produced in time $O(logk+b)$, where b is the number of bends (vertices) in the path.

Additional improvement of computation time of finding a cell in which a neuron lies during a self-organization process of a neural net is based on fact, that a neuron is not placed randomly in the place but moves on the shortest path to some city only. The cell, where a neuron lies ca be then effectively determined by the following process:

1) Let $Cell_a$ is a cell where the neuron p is placed and its position is adapted so that it is tightened to the city C. Denote obstacle vertices lying on the shortest path from p to C as $v_1, v_2, \ldots, v_d$.
2) Compute the distance at which the neuron will travel during the adaptation step similarly to step 3b) in the original algorithm:

$$L = \mu f(b, k) d(C, p),$$

   where $d(C, p)$ is the length of the shortest path between C and p (i.e. their geodetic distance) and m and f have the same meaning as in the original algorithm.
3) Determine the point q that lies on the shortest path between C and p and its geodetic distance is equal to L. Denote the portion of the path on which q lies $v_x v_{x+1}$.
4) Determine the cell in which the point v_x lies and denote it $Cell_x$. This information can be obtained during SMP-building process and stored for each obstacle vertex.
5) Detect whether q lies in $Cell_x$. If yes then stop - $Cell_x$ is the desired cell. Otherwise determine intersection $Cell_x \cap v_x q$. Set v_x as the intersection point. The new point v_x lies on the boundary of two cells – $Cell_x$ and one of its neighbours. Denote this neighbour $Cell_x$ and go back to step 5.

Time complexity of steps 4 and 5 is $O(m)$, where m is the number of cell vertices. Moreover, step 5 is repeated as many times as many cells is visited on the path between p and q – denote this number r. Complexity of step 2 is constant and complexity of step 3 is r and therefore overall complexity of the algorithm is $O(mr)$. On the assumption that the distance between p and q is small (which is a frequent case in the self-organizing process) r is constant and therefore the complexity of the algorithm is $O(m)$.

The second aspect to be taken into account is computation of the equation in step 3a) of the original algorithm. In order to evaluate this equation, lengths of rings have to be determined, which leads to enumeration of distances between neurons. Unfortunately, determination of neuron-neuron distances is very time consuming, because no structure like SMP for neuron-city distances can be used. Therefore, instead of computing lengths of rings directly, we compute the length of a path travelled by each salesman in the following procedure:

1) For each city C_i determine the nearest neuron n_i.
2) For each ring r do
   a) Gradually examine all neurons in the order they appear in the ring. For each neuron which is nearest to some city, remember this city. By this process we get an ordered list of cities $C_{r_1}, C_{r_2}, \ldots C_{r_n}$.
   b) Compute the length of the ring as a sum of geodesic distances between connected cities in the list:

$$length(r) = \sum_{i=1}^{n-1} d(C_{r_i}, C_{r_{i+1}}).$$

III. Experiments and Results

This section presents an empirical analysis of all previously described soft computing techniques. The comparisons are based on the quality of the solutions achieved by the algorithms for three different environments (Map1 with 37 cities, Map2 with 32 cities, and Map3 with 100 cities) and for the number of agents changing from 2 to 5. The tests were performed for genetics algorithms and ant colony optimization with (hGA, hACO) and without applying heuristics (GA, ACO) and for neural networks with city-to-city (NNCity) and neuron-to-neuron (NNneuron) distances. Each experiment was replicated 100 times and the best, worst and average solution and standard deviation of the solutions were studied.

The configuration of ACO:

- initial learning step $\alpha_i = 0.05$, linearly increased during learning up to $2\alpha_i$
- initial explore/exploit parameter $q_0 = 0.9$, linearly increased up to 1
- discount factor $\gamma = 0.3$
- stopping criterion: plan evaluations = $(N + A - 1) * 1000$

The configuration of hGA:

- population size: 1000
- selection: tournament selection with $N = 3$
- crossover operator: edge recombination operator, probability of crossover 1.0

TABLE I
Comparison of GA, ACO and NN on the real data

Map	Number of agents	Method	Path Length			
			best	avg	worst	stdev
1	2	GA	1797.61	1909.10	2130.91	82.3
		hGA	1736.52	1800.00	1886.20	35.8
		ACO	1858.12	1977.22	2088.07	50.9
		hACO	1722.47	1786.90	1819.41	25.94
		NNcity	1772.16	1990.78	2430.72	139.4
		NNneuron	1767.49	1962.06	2299.80	98.5
	3	GA	1372.38	1471.90	1689.39	62.90
		hGA	1310.11	1389.70	1601.90	51.10
		ACO	1416.67	1516.38	1656.79	51.5
		hACO	1307.96	1374.90	1419.18	22.48
		NNcity	1362.86	1583.36	1895.94	134.3
		NNneuron	1310.17	1541.11	1935.61	132.8
	4	GA	1211.44	1339.30	15.88.52	71.5
		hGA	1173.17	1203.90	1274.26	24.50
		ACO	1238.52	1312.11	1395.37	35.2
		hACO	1173.77	1236.97	1308.36	32.47
		NNcity	1188.40	1445.29	2018.22	166.9
		NNneuron	1179.86	1481.43	2250.64	202.1
2	2	GA	9425.37	10139.80	10704.03	316.0
		hGA	8934.99	8964.90	9090.57	42.3
		ACO	9180.68	9361.48	9648.84	146.9
		hACO	8934.69	8955.22	9036.07	33.23
		NNcity	8997.18	9645.00	10887.93	327.3
		NNneuron	8997.17	9696.16	14826.37	852.2
	3	GA	7192.44	7803.10	8797.46	301.4
		hGA	6944.45	6975.60	7295.66	87.3
		ACO	7242.22	7484.37	7793.24	101.2
		hACO	6944.45	6946.37	6958.41	4.6
		NNcity	6958.39	7722.42	9270.25	398.7
		NNneuron	7059.10	8037.12	10304.32	586.1
	4	GA	6542.09	7077.1	1856.98	204.4
		hGA	6338.76	6542.90	6598.56	49.4
		ACO	6756.36	6934.08	7107.97	75.8
		hACO	6330.48	6538.00	6743.56	96.9
		NNcity	6471.53	6959.99	8553.00	365.9
		NNneuron	6549.30	7702.42	9528.09	633.8
	5	GA	6268.83	6626.70	6966.60	203.1
		hGA	5918.51	6111.17	6142.76	36.4
		ACO	6257.11	6636.31	6906.00	123.4
		hACO	5903.48	6538.00	6314.50	73.9
		NNcity	5900.02	6630.40	9047.32	491.7
		NNneuron	6357.27	7553.81	9408.23	600.6
3	2	GA	2334.11	2530.90	2723.07	97.5
		hGA	2210.45	2322.90	2435.30	49.1
		ACO	2385.22	2497.32	2661.22	58.9
		hACO	2165.30	2242.62	2297.16	22.8
		NNcity	2234.66	2312.40	2439.19	45.9
		NNneuron	2234.32	2349.60	2489.04	49.7
	3	GA	1656.17	1834.90	2065.00	88.6
		hGA	1526.43	1647.50	1733.30	45.5
		ACO	1669.63	1821.21	1967.88	57.9
		hACO	1533.01	1593.43	1643.95	25.8
		NNcity	1539.41	1618.24	1838.37	46.1
		NNneuron	1523.91	1662.37	1825.88	64.6
	4	GA	1359.66	1558.70	1847.17	115.9
		hGA	1237.22	1314.40	1441.70	41.9
		ACO	1292.70	1434.39	1523.32	41.6
		hACO	1206.29	1287.48	1361.37	29.9
		NNcity	1193.73	1281.23	1389.14	43.4
		NNneuron	1192.24	1336.44	1930.34	92.5
	5	GA	1129.46	1304.10	1513.80	84.8
		hGA	1037.96	1136.90	1294.50	69.2
		ACO	1101.78	1249.34	1346.43	48.5
		hACO	1034.95	1122.30	1222.76	30.8
		NNcity	1024.28	1094.14	1242.18	36.7
		NNneuron	1038.18	1143.08	1464.25	60.0

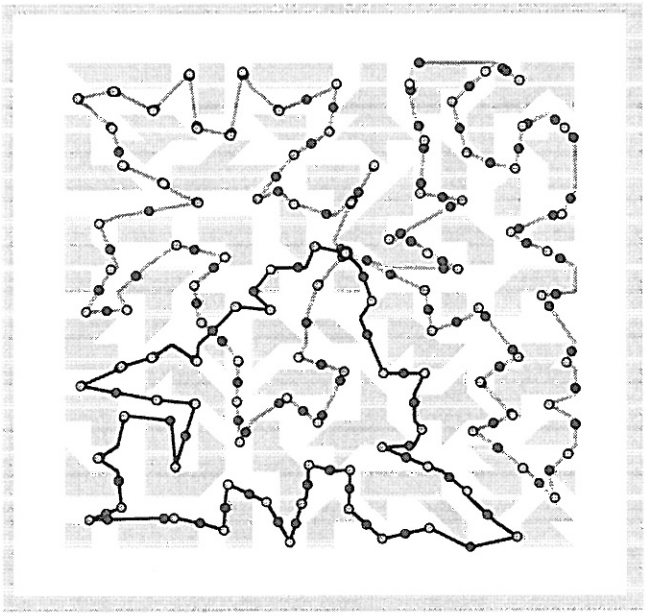

Fig. 3. The best solution found for three agents in Map3.

- mutation: simple city-pair swapping mutation operator, $P_m = 10\%$
- heuristic rates: $P_{STO} = 10\%$, $P_{LTS} = 10\%$
- stopping criterion: plan evaluations = $(N + A - 1) * 1000$

The configuration of NN:

- initial learning rate $\mu = 0.6$
- initial gain parameter $G = 0.1$
- decreasing factor of gain parameter $\alpha = 0.1$
- stopping criterion: the position of each city and its nearest neuron is smaller than 0.01

IV. Conclusions

The paper presents three principally different soft computing methods to rescue operation planning. In our approach the planning is decomposed into two phases. Within the first phase, locations for sensing are found and distances among them are computed. The second phase rephrases and solves the planning as MTSP. By its nature, ACO and GAs can rely solely on the precomputed location distances, i.e., possible obstacles are sufficiently represented in the distance matrices. On the contrary, optimization based on self-organizing NN has to regard geometric representation of the obstacles also during its run. That is why the emphasis is put on effective generalization of usual NN approach for environments with obstacles.

The section III and results it contains we interpret as follows. The heuristics applied in ACO and GAs substantially increase overall performance, which favors these hybrid approaches from this point of view. ACO shows slower convergence to a single highly-fit plan. This characteristic makes the plain ACO results worse when compared with the plain implementation of GA, while utilisation of heuristics brings more focus into the ant colony without early reduction of the search space, which probably results in the observed minimisation of the overall time spent by the squad (hACO often outputs the best plans found in the discussed tasks). On the other hand, utilization of the heuristics reasonably decreases scalability of hACO and hGA, i.e., ability to cope with larger areas to be covered by a rescue team. The self-organizing NNs have geometric properties that enable to avoid utilization of heuristics (rings of neurons cross rarely, similarity of the ring lengths can be relatively straightforwardly guaranteed). Having scalable implementation dealing with the obstacles, the NN method seems to be the best approach in higher dimensions.

Future work lies in further speed up of three presented methods and their testing on larger areas. In both hACO and hGA approaches we will try to minimize application of heuristics. Additionally, ACO approach can minimize its execution time by optimization of its local updating rule.

Acknowledgments

This research work was supported within the IST-2001-FET framework under project no. 38873 "PeLoTe - Building Presence through Localization for Hybrid Telematic Systems" and by the Grant Agency of the Czech Republic within the project No. 102/02/0132. The work of Jan Faigl has been also carried out with the support of FRVŠ grant No. G1 2042/04.

References

[1] Bullnheimer B., Hartl R., Strauss C.: *Applying the Ant System to the Vehicle Routing Problem.* 2nd Metaheuristic Int. Conference, Sophia-Antipolis, France, 1997.

[2] Dorigo M., Gambardella L. M.: *Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem.* IEEE Transactions on Evolutionary Computation Vol.1, No. 1, pp. 53-66, 1997.

[3] Gambardella L. M., Taillard E., Agazzi G.: *MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows.* In Corne, Dorigo, Glover (Eds.), New Ideas in Optimization. McGraw-Hill, London, UK, pp. 63-76, 1999.

[4] Gambardella L. M., Dorigo M.: *Ant-Q: A Reinforcement Learning Approach to The Traveling Salesman Problem.* Proceedings of 12th Int. Conference on Machine Learning, A. Prieditis and S. Russell (Eds.), Morgan Kaufmann, pp. 252-260, 1995.

[5] Goldberg D. E.: *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[6] Michalewicz Z.: *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, Berlin Heidelberg, Second edition, 1994.

[7] Whitley D., Starkweather T., Fuguay D.: *Scheduling Problems and Traveling Salesman.* The Genetic Edge Recombination Operator. In Proceedings of the Third ICGA, San Mateo, CA, Morgan Kaufmann Publishers, pp. 133-140, 1989.

[8] Somhom S., Modares A., Enkawa T.: *Competition-based neural network for multiple travelling salesmen problem with minmax objective*, Computers & Operations Research, 26, pp. 395-407, 1999.

[9] de Berg M., van Kreveld M., Overmars M., Schwarzkopf O.: *Computational Geometry: Algorithms and Applications*, Springer-Verlag, 2000.

[10] Mitchell J. S. B.: *A new algorithm for shortest paths among obstacles in the plane*, Annals of Mathematics and Artificial Intelligence 3, pp. 83-106, 1991.

[11] Mitchell J. S. B.: *Shortest paths among obstacles in the plane*, International Journal of Computational Geometry and Applications, 6(3), pp. 309-332, 1996.