Proceedings of the 2011 IEEE
International Conference on Robotics and Biomimetics
December 7-11, 2011, Phuket, Thailand

Use of Genetic Algorithms for Target Distribution and Sequencing in Multiple Robot Operations

Alberto Valero-Gomez and Julio Valero-Gomez and Alvaro Castro-Gonzalez and Luis Moreno

***Abstract*— In this paper we propose a solution for the *Multiple Traveling Salesman Problem* applied to mobile robotics. A team of mobile robots (travelers) must visit a set of target locations, minimizing the time to visit all of them. Initially the targets are not distributed among the robots, but they are assigned to the team as a whole. A subset of points and its visiting order (sequence) must be assigned to each robot, minimizing the total length of the routes and the completion time of the global task (maximum path lenght). Using a genetic algorithm we compute a fast, robust, and efficient solution to this problem. The genetic algorithm is described in detail and a comprehensive experimental analysis of the proposed solution is presented, characterizing the genetic algorithm and supporting its applicability.**

I. Introduction

The Traveling Salesman Problem (TSP) is a classical problem in the field of combinatorial optimization. Given the position of n-cities, a traveler must find the shortest path to visit all of them [1]. This is an NP-hard problem widely addressed by operations research and theoretical computer science.

In this paper we address a generalized formulation of the TSP for mobile robots. We are considering that m robots must visit n points which are not initially assigned to any particular robot. This problem is known as the Multiple Traveling Salesmen Problem (MTSP). While the TSP involves only a decision of sequence, the MTSP involves decisions of sequence and selection [2]. The optimal solution consists of selecting which set of points each salesman must visit and finding the minimum-length route which includes exactly one point from each point-set. The optimal solution does not depend exclusively on the sequence of the points but also on their assignment to each of the travelers.

We are approaching the MTSP using Genetic Algorithms (GA). A GA is a meta-heuristic method, concerned with the minimization of a function with many independent variables, searching for a good solution in the solutions space, mimicking the process of natural selection. Their major weaknesses are 1) they cannot prove optimality, and 2) the dispersion of the solutions found on different runs can be very high. Their performance in terms of problem solving and computation time depends on the definition of the mutation, selection, and crossover operators, parameter tunning, and in the desired optimality of the solution [7]. A study on the application of evolutionary algorithms to solve MTSP can be seen in [11]. Example of GAs addressing the MTSP are [13], [14], [12]. Other bio-inspired solutions can be found in [8], [6]. In this paper we are proposing a GA capable of finding fast and robust solutions to the MTSP for robotic applications. Robotic applications requirements may differ from those strictly algorithmic. 1) *Computing time*: a fast and good solution can be satisfactory even if it is suboptimal; 2) *Robustness*: the algorithm should not reach outlier solutions far from the optimal solution.

Once a solution is found, the targets are distributed among the robots and the routes assigned. During the task execution this distribution may be re-computed, in order to deal with unexpected situations. This can be applied to multiple problems: a team of service robots may be required at different points in a fair; robots can be used in a hospital for delivering medicines or meals; in exploration robotics frontier points (limits between the explored known areas and the unexplored areas) must be visited; in scheduled operations different places of a plant must be reached for placing a specific sensor, etc. Similar problems have been addressed in [10], [5], [3], [4]. In all these examples a precalculated plan is not applicable as the number and locations of the robots or the targets to visit may change from time to time.

In the following section we will describe in detail the proposed GA. On Section III we will test our algorithm in two environments: the first environment will be a free space, equivalent scenario to the ones used in the euclidean MTSP studies. The second scenario will consist of a hospital, which rooms must be visited by a team of robots (the cost of each arc will be the length of the computed path among points). These experiments will characterize our proposal for real scenarios. The experimental data will support that the proposed algorithm is adequate for real applications, providing fast and robust solutions.

II. GA Description

In a GA, a population of strings (called chromosomes) which encode candidate solutions (called individuals) to an optimization problem, evolves toward better solutions. In Evolutionary Algorithms individuals, or solutions, can be defined by one ore more chromosomes. In our algorithm each individual is defined by one chormosome, and thus, the terms are exchangable. In the following we will describe the mechanisms guiding the evolution of our proposed GA.

Alberto Valero is with the Robotics Lab, Universidad Carlos III de Madrid, Spain alberto.valero@uc3m.es

Julio Valero-Gomez is with the Universidad Politecnica de Madrid

Alvaro Castro is with the Robotics Lab, Universidad Carlos III de Madrid, Spain

Luis Moreno is with the Robotics Lab, Universidad Carlos III de Madrid, Spain

978-1-4577-2138-0/11/$26.00 © 2011 IEEE

A. Chromosome Coding

One of the key aspects of a GA is how to code the space of solutions into chromosomes. The genetic coding is the way of representing solutions/individuals in evolutionary computation methods. Our problem is characterized by m robots and n target points. Each robot has an assigned index in the interval $[0, m-1]$. The target points have assigned indexes in the interval $[m, n+m-1]$. Our chromosome contains thus $m+n$ elements. A chromosome encodes the solution in the following way.

1) Each chromosome begins with a robot index $\in [0, m-1]$.
2) The rest of the elements may be either an index of a robot or a target.
3) Elements cannot be repeated in the chromosome.
4) The sequence of points assigned to each robot consist of the targets beginning after their index and finishing when the next robot index is found.

Consequently, if we have an MTSP of 2 robots and 10 targets, the chromosome $< 1\ 3\ 4\ 5\ 9\ 2\ 0\ 6\ 8\ 7\ 11\ 10 >$ encodes the following solution:

- *Robot 0* route is formed by targets: 6 8 7 11 10, whilst
- *Robot 1* route is formed by targets 3 4 5 9 2.

B. Genetic Operators

1) Initial Population: In the first step of a GA, a specified number of individuals are generated to form an initial population. Traditionally, the initial population is generated randomly covering the entire range of possible solutions. This was the first implementation of our GA. In the initial experiments we observed that MTSP presented strong local optimal solutions (local minima) from which it was difficult to exit. An initial population that falls rapidly into a local optimum may lead to an outlier solution, which compromises the robustness of the GA. To avoid this problem, our proposed GA initially generates 5 independent random populations, which evolve independently for 1000 generations. The best $population\_size/5$ individuals are then taken from each population, constituting a new population that will continue the GA evolution. Doing this we increase initial diversity, and if any of the initially random generated populations has fallen into a local minimum, it will be overcome by the rest of individuals during the evolution of the joint population. As we will see below we also used *Immigration* to avoid outlier solutions.

2) Crossover: The crossover operator generates a new individual from n individuals/chromosomes. We are crossing two individuals (parents) to produce a new one (child). The employed crossover, traditionally known as Order Crossover (OX), works as follows:

1) A random substring is chosen from one of the parents.
2) A child is produced by copying the substring into the same position it was located in the parent.
3) The missing elements are placed at the free spaces, starting from the beginning, in the same relative order as found in the second parent.

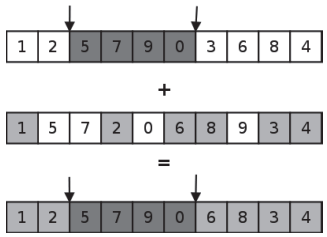


Fig. 1. Crossover Operator

A crossing operation is depicted in Fig. 1.

We make $population\_size$ crossings to generate a $population\_size$ offspring. Part of this offspring will be afterwards mutated.

3) Selection: A *probabilistic selection* procedure is applied to choose the pair of individuals that will participate in the reproduction process to give birth to each individual of the next generation. In order to make the whole population evolve towards good solutions, selection is executed probabilistically based on a fitness function (Sec. II-C). For each individual, the probability of being chosen is given by the following Equation:

$$Pc(chromosome_i) = \frac{1}{pf^{i+1}} \times \frac{1}{\sum_{j=0}^{genSize} \frac{1}{pf^{i+1}}} \qquad (1)$$

where pf is the *probability factor* and i the position of the chromosome in the sorted population according to the fitness function. Considering that the individuals are sorted by non decreasing cost, lower cost individuals will have a higher chance of being selected. The pf parameter determines the slope of the probability function $Pc(chromosome_i)$ (Fig. 2).

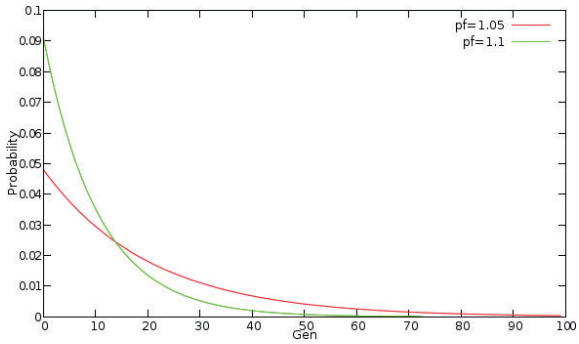


Fig. 2. Selection function for Crossing

4) Clonation and Mutation: After crossover, generated children are subject to a mutation process, in which both the original and mutated chromosomes are kept in the resulting population, thus avoiding losing a good solution because it mutates into a worse one. Children to undergo mutation are chosen randomly among the whole offspring with a uniform probability pm.

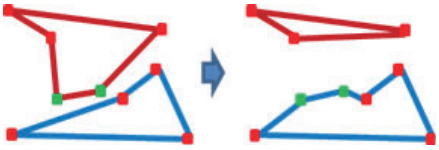
Mutation probability is changed dynamically during the evolution process depending on the number of continued generations without evolution and may reach up to 0.5. When

the GA does not evolve for many generations it means that it has found a minimum. By increasing *pm* we increase the probability to escape (if it is not the optimum) by mutation. Note that there is no risk in losing that solution as original solutions are also kept in the mutation process.

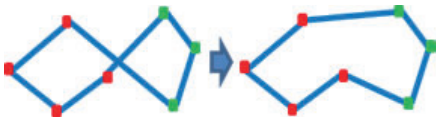
After the mutation process the resulting population will exceed the size of the original population. The GA sorts the resulting population by increasing cost to afterwards crop the population to its original size, which is fixed throughout the search.

There are three mutation operators. They are domain specific, improving the performance of the GA for this specific problem. Only one of them acts each time, with different probabilities. These are:

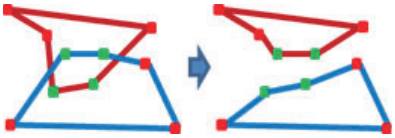
1) Path reallocation: A random portion of a robot path is extracted and placed at a random position of the same or another robot path, in the same or reverse order. This introduces diversity in the population generating new unexplored solutions (Fig. 3(a))
2) Path inversion: A random portion of a robot path is placed in the same position but in reverse order. This mutation helps minimizing one robot route length avoiding closed loops within individual paths (Fig. 3(b))
3) Path crossover: Random portions of two different robots paths are exchanged, in the same or reverse order. This mutation helps avoiding unoptimal crossings between different robots routes (Fig. 3(c))



(a) Path Reallocation



(b) Path Inversion



(c) Path Crossover

Fig. 3. Mutation Operators

The crossover and mutation procedure works as sketched in Alg. 1.

5) Immigration: Mutation may not be enough to avoid falling into a local minima. In the proposed algorithm, we have also introduced an immigration operator. Immigration consists of creating a full new population which has evolved for a number of generations (10000 in this case) and introducing the best individuals (25 for our algorithm) in the main population (deleting the worst 25 individuals of it). Our GA performs one immigration after a preset number of generations without improvement.

Algorithm 1: GA reproduction

```
//Crossover;
for i = 0; i < population_size; i++ do
    ind1 = randInd(); //with prob pc;
    ind2 = randInd(); //with prob pc;
    newInd = cross(parents[ind1], parents[ind2]);
    children.push_back(newInd);
//Clonation and Mutation;
for i = 0; i < population_size; i++ do
    if rand()%100 < pm then
        children.push_back(children[i].clone()) ;
        children[i].mutate();
children.sort();
children.resize(population_size);
parents = children;
```

C. Fitness Function

Cost or fitness functions are associated to optimization problems, consisting in finding the "best available" values of some objective function given a defined domain. In our problem we seek to minimize a real function by systematically choosing the best individuals from within a population.

An important difference between the classical MTSP treated as a theoretical problem and real applications lies in the fitness function. The classical solution for the MTSP minimizes the total length of the routes (sum of the single paths length). This may lead to a solution in which the minimal path could be achieved by using just one robot or a small subset of the robots visiting all the points while the rest remain still. On the contrary, if we want to minimize the task completion time, the solution will try to minimize the longest route length, which will set the task total time. Depending on the particular applications the fitness function will change. For our problem we want to minimize the task completion time. The GA seeks to minimize two values:

- Total route length (1^{st} term in Eq. 2).
- Maximum individual length (2^{nd} term Eq. in 2).

At the same time it is preferred that all robots are assigned similar length routes to avoid different utilization of the resources. This is achieved introducing the 3^{rd} term in Eq. 2. Doing so the GA will also evolve to maximize the shortest individual path length whilst minimizing the longest. The implemented fitness function is the following:

$$Fitness(ind_i) = \Sigma_{i=0}^{m-1}(R_i) + 0.9 \times \max(R_i) - 0.9 \times \min(R_i) \quad (2)$$

where m is the number of robots, and R_i the total length of the route assigned to robot i comprising the way back to the starting point.

D. Termination Rule

The initial population will evolve until some termination conditions are reached. These conditions depend on the problem. Our GA is designed to evolve until a steady solution is found. This condition relies on the convergence of the GA: if it does not converge it will never terminate. Our proposed GA will terminate after j generations without improvement (the fitness function of the best individual of the population provides the same value). In the *Experimental Results Section* we will present the resulting data for different values of j. The evolution and termination is thus sketched in Alg. 2.

Algorithm 2: GA evolution and termination

```
generation.generateInitPopulation();
prevCost = cost = F(generation.bestIndividual);
steadyNumber = 0;
while steadyNumber < j do
    pm = steadyNumber/100; //mutation probability;
    if pm > 50 then
        pm = 50;
    generation.reproduce(); //crossover and mutation ;
    if steadyNumber % 1000 = 0 then
        generation.immigration();
    cost = F(generation.bestIndividual);
    if cost = prevCost then
        steadyNumber++;
    else
        steadyNumber = 0;
    prevCost = cost;
solution = generation.bestIndividual;
```

III. Experimental Analysis

In order to test and characterize the prosed GA we run a set of experiments with different parameter values and different task configurations. The **independent variables** were: *number of robots* (from 2 to 5), *number of target locations* (30, 40, 50, 60, 70), *termination condition* (number of steady iterations until the GA terminates: 10,000; 50,000; 100,000), and *population size* (50, 100). The **measured variables** were: *Cost of the solution* (calculated with Eq. 2), *route length of each robot*, *number of generations to find the solution*, and *CPU time*. 100 tests were run for each combination of conditions in order to get meaningful statistical data. Experiments were run on a Toshiba Laptop equipped with a Intel(R) Core(TM) i3 CPU M330@2.13 GHz Processor and 4Gb RAM running Debian Squeeze. The GA was implemented using c++. The environment, the target locations, and the robots starting positions did not change over the tests. Figure 4 shows the solution of one of the tests in the hospital envrionment.

We cannot present all collected data in this paper. The full set of logs can be found on http://iearobotics.com/alberto/doku.php?id=sofware:gamtsp alongside with the GA code in C++ (under GPL license). This page also includes multimedia content demonstrating the GA functioning: several videos have been uploaded showing the solution evolution for several experimental conditions on a free space. The videos show several tests. For recording the videos the GA were slowed down as otherwise the evolution of the solution could not be appreciated.

A. Results

1) Variations of the proposed GA: The proposed GA has underwent two major modifications since its initial version. The initial version is equivalent to the one presented in [13]. There, the solutions found and CPU time were good enough but the variability of the solution was too high, presenting outlier solutions which differed around 15% from the average solution. In the second version, our GA included Initial Population Generation (Sec. II-B.1). In the third version Clonation (Sec. II-B.4) and Immigration (Sec. II-B.5) were added. Figure 5 shows the dispersion of the solution for each GA version (5 robots, 50 targets, 100,000 steady condition). It shows that the solution cost remained the same, but the variability of the solution decreased, thus improving robustness of the proposed GA.

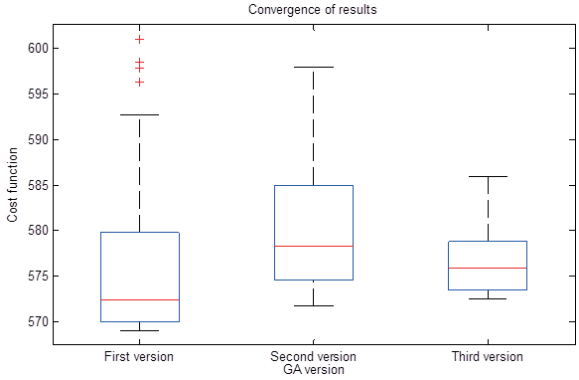


Fig. 5. *Structured Environment. 5 robots; 50 points; 100,000 steady number*. **Version 1**: Crossover and Mutation. **V.2**: Initial Population Generation Added. **V.3**: Cloning and Immigration Added.

2) Convergence: The convergence of a GA measures whether the GA evolves towards an optimal solution or not. GAs can be considered as directed random search algorithms, random solutions are generated and the GA mechanism make the solutions evolve towards a local optimum. To check this we have generated 1,000,000 random solutions for each experimental combination, taking afterwards the best of the solutions (generating 1,000,000 random solutions requires aproximately twice the CPU time that the GA is requiring to terminate). Table I shows the results of the best random generated solution and the result of running the proposed GA under the same conditions and for the same problem.

Figure 7 shows the convergence of the GA for a run with 5 robots and 100 target locations. The termination rule was set to 5000 in order to make clearer the non steady region of the graph.

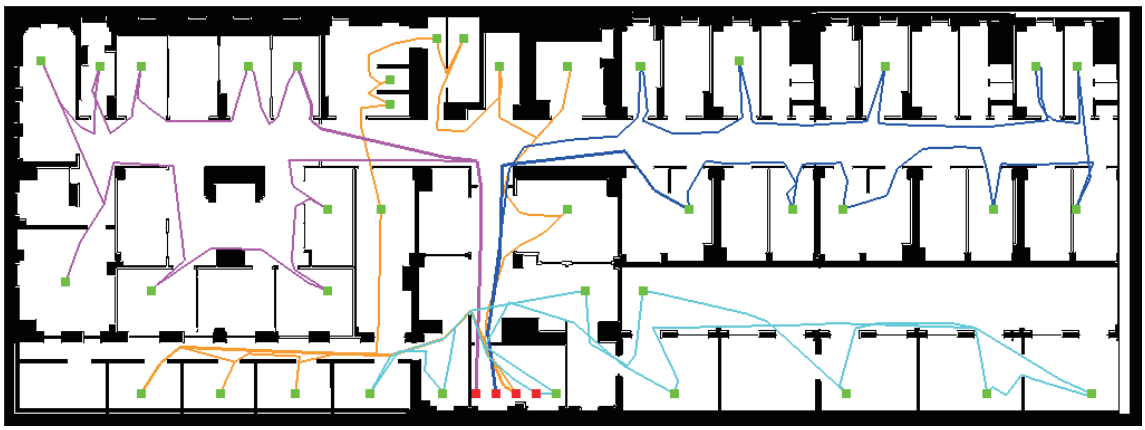


Fig. 4. Hospital Environment. 4 robots (red); 40 points (green); 10.000 steady number

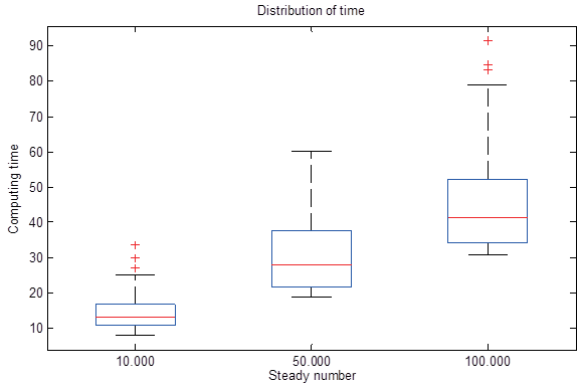


Fig. 6. CPU Time in sec. (Intel(R) Core(TM) i3 CPU M330@2.13 GHz) wrt. Number of Steady Iterations (termination condition). Hospital Environment. 5 robots; 50 points.

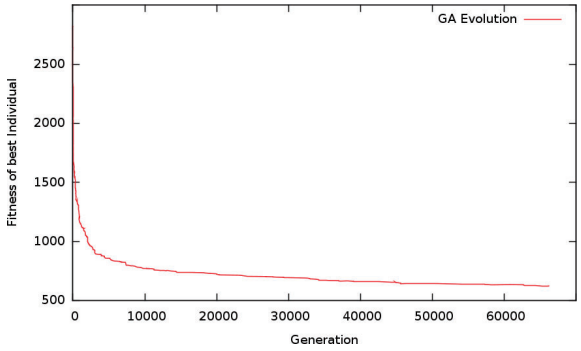


Fig. 7. Evolution of the GA for 5 robots and 100 travelers.

Best of 1,000,000 Random Solutions			
	Best Solution		
	30 targets	40	50
Free Space			
2	596.047	815.863	956.407
3	719.478	954.969	1110.76
4	670.347	972.464	1197.4
5	714.729	943.873	1117.63
Hospital			
2	828.679	1121.66	1399.62
3	946.522	1282.91	1641.55
4	970.171	1229.08	1578.9
5	1037.55	1439.81	1735.15
Evolution of GA with termination condition = 10,000			
	Total Cost Average (stdev), CPU Time (in secs.)		
	30 targets	40	50
Free Space			
2	188.3 (0.3%), 8.94	209.8 (1.0%), 10.94	223.0 (1.0%), 12.70
3	224.7 (0.4%), 9.11	236.1 (0.6%), 11.62	250.2 (1.0%), 13.92
4	242.0 (0.3%), 9.49	269.8 (0.4%), 11.97	275.5 (0.8%), 13.92
5	273.5 (3.1%), 11.10	291.0 (0.8%), 12.68	308.7 (0.7%), 14.46
Hospital			
2	360.7 (1.2%),10.67	419.9 (1.8%), 11.51	466.5 (1.8%), 12.67
3	380.7 (0.0%), 7.71	431.0 (0.9%), 12.38	474.6 (2.8%), 13.56
4	430.0 (0.3%), 10.15	499.4 (0.8%),12.92	550.0 (0.8%), 14.61
5	477.4 (0.2%),10.56	520.1 (0.3%), 12.04	576.4 (0.5%), 14.52

TABLE I

CONVERGENCE OF THE GA. FREE SPACE / HOSPITAL EXPERIMENT. RESULTS OBTAINED FROM 100 TESTS (FOR THE GA). TOTAL COST AVERAGE AND STANDARD DEVIATION. CPU TIME AVERAGE (IN SECONDS)

B. Scalability

The variation of the CPU time with the number of target locations was used to measure the scalability of the GA. For this experment 100 tests were ran using 5 robots and 30, 40, 50, 60, and 70 target locations. The environment was the free space used previously. The location of the robots and the targets was randomnly generated at every single test. This gives us an idea of the CPU time, independenly on the robots and targets locations. Collected data is shown on Table II

C. Performance and Robustness

Table III shows the statistical analysis of the collected data for 5 robots and 50 target locations. We present only this experimental combination as it represents the most complex condition for the hospital scenario. The size of the population is of 100 individuals (for 50 the variability of the solution was greater). We analyze the combination of the rest of the controlled variables. The analyzed data is:

- Average and standard deviation of the cost. The standard

	Number of Target Locations (5 Robots)				
	30	**40**	**50**	**60**	**70**
Avg. Time	8.87	10.27	13.63	18.05	22.70
Std. dev.	2.66	3.27	4.47	5.47	8.20

TABLE II

SCALABILITY OF THE GA. FREE SPACE. RESULTS OBTAINED FROM 100 TESTS. CPU TIME AVERAGE AND STANDARD DEVIATION.

deviation is presented in percentage, to give a clearer idea of the variability of the solution.

- Minimum and Maximum values of the cost over the 100 tests.
- Number of required generations to find the final solution and the CPU Time.

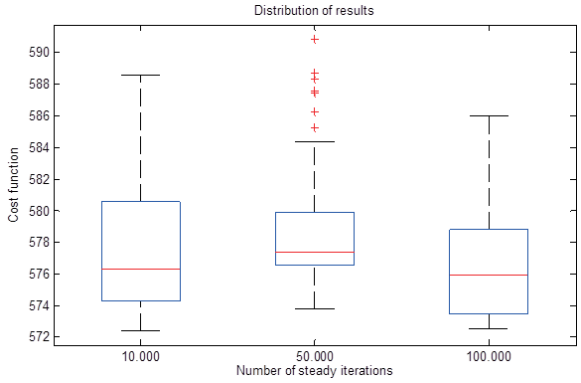


Fig. 8. Cost (Fitness) wrt. Number of Steady Iterations (termination condition). Hospital Environment. 5 robots; 50 points.

IV. DISCUSSION

We are characterizing the performance of the proposed GA evaluated in terms of computation time and variability of the solution (robustness). The results presented at Table III analyze the most complex condition of 5 robots and 50 target locations. The required CPU time is smaller than 15 seconds for 10,000 iterations (termination condition), 30 seconds for 50,000, and 50 seconds for 100,000. In the worst case the variability of the solution is under 3% and in most cases remains under 1%.

Analyzing the quality of the solution with respect to the number of iterations (Fig. 8) and the required CPU time (Fig. 6) we can conclude that 10,000 iterations is enough to get a good solution, as the quality of the solution does no improve with the iterations, while the computation time is smaller.

The first version of the presented algorithm implements a strategy similar to the GA presented in [13]. Crossover operator is equivalent. Mutations are also similar, but we introduce a third type of mutation, specifically adapted for avoiding crossing of different robot routes (Fig. 3(c)). In addition we introduce a new way of generating the initial population, immigration, a new mutation operator as well as the clonation prior to the mutation. These are not present in [13]. As shown in Fig. 5 these operators highly contribute to the robustness of the algorithm, decreasing the variability of the solutions. Unfortunately no statistical analysis or robustness proof is presented in the referenced paper, nor the experimental conditions they used. Consequently we have no statistical basis to compare our results to theirs. Based on the contribution of the selection of the initial population, immigration, mutation, and clonation, we presume that both GAs drive to similar optimality in terms of the fitness function, but our solution should be more robust than theirs (in terms of dispersion). In [6] the MTSP is solved using neural networks. In their experimental results the dispersion of the solutions varies from 6,23% to 38,60%, but they are addessing environments with more target locations (from 225 to 1024). Regarding the scalability in terms of computation

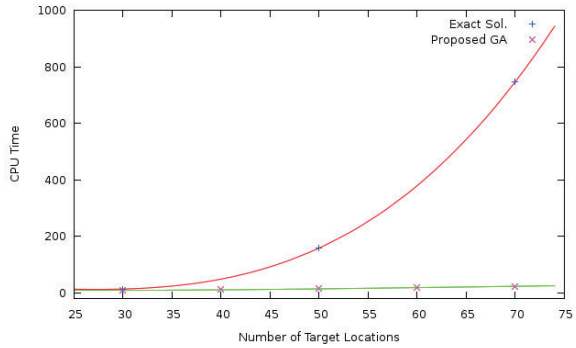


Fig. 9. CPU Time of the Proposed GA and of the Exact Solution Presented in [9]

time, we are comparing our propsed GA with the exact solver presented in [9]. Their CPU times are: for problems with 30 target locations over a total of 18 instances the average solution time was 13.06 seconds (on a Pentium III 1400 MHz PC, ours is a i3 2300 MHz); for 50 locations 158.33 seconds; and for 70 target locations 747.17 seconds. Fig. 9 shows a comparison of the scalability of our GA and the referenced work. Regarding the algorithm speed we cannot stablish a exact time comparison as processors are not the same, but it is clear that theirs does not scale well. They reach optimallity but sacrifice scalability (time). Depending on the kind of application, one of the two, optimality or fastness, may be required.

V. CONCLUSIONS AND FURTHER WORK

In this paper we have presented a genetic algorithm capable of solving *on the fly* a multiple traveler salesmen problem. The proposed GA introduces innovation in mutation, immigration, and enhanced initial population regarding the last published article on MTSP for mobile robots [13] (to our knowledge). Thanks to this the GA provides robust solutions avoiding high dispersion and maintaining the overall performance. The variability of the solution stays below 1% (wrt. solution) in most cases, and the CPU time is around 15 seconds. Our proposed GA can be used in a variety of real applications, whether they involve mobile robots or not. With the presented algorithm a set of locations can be

N. Robots	Average (stdev) of Cost (Eq. 2)		Min, Max of Cost		Av. Generations, Av. Time (in sec.)	
	Free Space	Hospital	Free Space	Hospital	Free Space	Hospital
Number of Required Steady Iterations (Termination Condition)						
10000						
2	222.94 (2.05, 0.91%)	467.09 (8.11, 1.7%)	221.35, 233.96	455.45, 505.96	13254.66, 12.70	11720.56, 12.67
3	250.17 (2.52, 1.01%)	472.74 (13.45, 2.8%)	247.04, 256.85	465.28, 523.89	16082.51, 13.92	14268.42, 13.61
4	275.80 (2.59, 0.93%)	546.26 (3.92, 0.7%)	273.06, 284.35	539.67, 556.88	16664.42, 13.92	15615.9, 14.59
5	308.77 (1.91, 0,61%)	577.29 (3.76, 0.7%)	306.81, 316.42	572.41, 588.54	17019.41, 14.46	14875.91, 14.52
50000						
2	223.02 (2.05, 0.92%)	464.65 (6.95, 1.5%)	221.35, 233.01	453.24, 475.89	28650.78, 24.88	32035.15, 26.26
3	250.17 (2.65, 1.06%)	477.19 (14.53, 3.0%)	247.04, 260.10	469.15, 531.46	24129.99, 23.81	38449.15, 28.35
4	275.60 (2.01, 0.72%)	551.02 (4.82, 0.9%)	273.06, 282.55	543.49, 569.65	30683.38, 26.18	31458.93, 26.69
5	309.17 (2.41, 0.78%)	578.57 (3.50, 0.6%)	306.81, 316,39	573.77, 590.81	30493.25, 26.52	42151.54, 30.57
100000						
2	222.96 (2.12, 0.95%)	466.52 (8.48, 1.8%)	221.35, 230.08	454.25, 490.76	37864.81, 38.23	49805.15, 43.20
3	250.22 (2.61, 1.04%)	474.60 (13.40, 2.8%)	247.04, 261.00	468.48, 525.45	50085.65, 42.61	57925.58, 46.26
4	275.48 (2.21, 0.80%)	549.96 (4.34, 0.8%)	273.06, 284.92	544.06, 562.992	29908.92, 37.70	50338.32, 43.77
5	308.71 (2.20, 0.71%)	576.44 (3.11, 0.5%)	306.81, 320.44	572.54, 586.008	37328.51, 39.97	54906.43, 45.73

TABLE III

FREE SPACE / HOSPITAL EXPERIMENT. 50 TARGETS. RESULTS OBTAINED FROM 100 TESTS. TOTAL COST AVERAGE AND STANDARD DEVIATION. MINIMUM AND MAXIMUM OBTAINED COSTS. REQUIRED NUMBER OF GENERATIONS TO FIND THE SOLUTION AND CPU TIME

distributed among a team of agents and the routes assigned to each of the agents. The solution obtained minimizes mission completion time, while distributing the task homogeneously among available resources and minimizing the total length of the routes.

Further work should include a deeper analysis on the scalability of the GA. The GA works well up to 100 target locations, which is a reasonable limit for the kind of applications our GA addresses. Further research is required to study how the modification of the selection criteria for the mutation process affects the solution and the CPU time. Currently the GA applies only one mutation to each individual. We hypothesize that applying several mutations to the same chromosome may improve the searching for a greater number of target points (but also the overhead will increase). Parallel evolution of several populations and immigrations processes (crossing different populations) could also result in better solutions for a higher number of target points. Also an experimental comparison with other kinds of algorithms for addressing the MSTP is programmed.

REFERENCES

[1] D. Applegate. *The Traveling Salesman Problem*. Princeton University Press, 2006.

[2] T. Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209 – 219, 2006.

[3] B. Brumitt and A. Stentz. Dynamic mission planning for multiple mobile robots. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 3, pages 2396 –2401 vol.3, Apr. 1996.

[4] B. Brumitt and A. Stentz. Grammps: a generalized mission planner for multiple mobile robots in unstructured environments. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2, pages 1564 –1571 vol.2, May 1998.

[5] M. B. Dias, M. Zinck, R. Zlot, and A. Stentz. Robust multirobot coordination in dynamic environments. In *IEEE International Conference on Robotics and Automation, ICRA*, pages 3435–3442, 2004.

[6] J. Faigl, M. Kulich, and L. Peuil. Multiple traveling salesmen problem with hierarchy of cities in an inspection task with limited visibility. In *5th Workshop On Self-Organizing Maps*, September 2005.

[7] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[8] P. Junjie and W. Dingwei. An ant colony optimization algorithm for multiple travelling salesman problem. In *First International Conference on Innovative Computing, Information and Control (ICICIC 2006)*, pages 210–213, 2006.

[9] I. Kara and T. Bektas. Integer linear programming formulations of multiple salesman problems and its variations. *European Journal of Operational Research*, 174(3):1449 – 1458, 2006.

[10] C. Rossi, L. Aldama, A. Barrientos, A. Valero, and C. Cruz. Negotiation of target points for teams of heterogeneous robots: an application to exploration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pages 5868–5873, 2009.

[11] D. Sofge, A. Schultz, and K. De Jong. Evolutionary computational approaches to solving the multiple traveling salesman problem using a neighborhood attractor schema. In S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. Raidl, editors, *Applications of Evolutionary Computing*, volume 2279 of *Lecture Notes in Computer Science*, pages 153–162. Springer Berlin / Heidelberg, 2002.

[12] L. Tang, J. Liu, A. Rong, and Z. Yang. A multiple traveling salesman problem model for hot rolling scheduling in shanghai baoshan iron & steel complex. *European Journal of Operational Research*, 124(2):267–282, July 2000.

[13] Z. Yu, L. Jinhai, G. Guochang, Z. Rubo, and Y. Haiyan. An implementation of evolutionary computation for path planning of cooperative mobile robots. In *Intelligent Control and Automation, 2002. Proceedings of the 4th World Congress on*, volume 3, pages 1798 – 1802 vol.3, 2002.

[14] T. Zhang, W. Gruver, and M. Smith. Team scheduling by genetic search. In *Intelligent Processing and Manufacturing of Materials, 1999. IPMM '99. Proceedings of the Second International Conference on*, volume 2, pages 839 –844 vol.2, 1999.