Journal of Global Optimization **23:** 155–170, 2002.

Heuristic optimization methods for motion planning of autonomous agricultural vehicles

K.P. FERENTINOS[1], K.G. ARVANITIS[2] and N. SIGRIMIS[2]
[1]*Department of Agricultural and Biological Engineering Cornell University, Ithaca, NY 14853, USA (e-mail: kpf3@cornell.edu)*
[3]*Department of Agricultural Engineering, Agricultural University of Athens, Iera Odos 75, Botanikos 11855, Athens, Greece (e-mail: karvan@aua.gr, n.sigrimis@computer.org)*

Abstract. In this paper, two heuristic optimization techniques are tested and compared in the application of motion planning for autonomous agricultural vehicles: Simulated Annealing and Genetic Algorithms. Several preliminary experimentations are performed for both algorithms, so that the best neighborhood definitions and algorithm parameters are found. Then, the two tuned algorithms are run extensively, but for no more than 2000 cost function evaluations, as run-time is the critical factor for this application. The comparison of the two algorithms showed that the Simulated Annealing algorithm achieves the better performance and outperforms the Genetic Algorithm. The final optimum found by the Simulated Annealing algorithm is considered to be satisfactory for the specific motion planning application.

Key words: Autonomous agricultural vehicles; Genetic algorithms; Heuristic optimization; Motion planning; Simulated annealing

1. Introduction

Motion planning in dynamic environments consists mainly of solving two problems: the *path planning* and the *velocity planning*. It has been shown (Canny and Reif, 1987) that this task is *NP-Complete* and thus, intractable to solve with conventional methods. The path planning, which is the kinematic part of the problem, involves the computation of a collision-free path from a start point to a goal point. There are basically two approaches to this problem: obstacle avoidance (Borenstein and Koren, 1989) and hybrid force/position control (Asada and Slotine, 1981). Here, the obstacle avoidance approach is used.

Several applications of motion planning in robotic vehicles can be found in the literature. Some of them have used deterministic approaches, like Pontryagin's maximum principle (Galicki, 1998) or simplified physical models (Cherif, 1999) and some others exhaustive search techniques or simple heuristics (Fiorini and Shiller, 1998). In the literature of autonomous agricultural vehicles however, very limited research can be found in the motion planning in the notion considered here. Most research deals with the path planning of agricultural vehicles in the cultivated part of the field, where in most of the cases, straight-line paths have to be followed (for example, Hague et al., 2000; Debain et al., 2000; Tillett et al., 1998; Toda et

al., 1999). The approach considered here involves the motion planning of these vehicles when they are situated in fields with several obstacles or complicated field borders and have to come up with the optimal path from their current point to a goal point.

In this paper, two sophisticated heuristic optimization algorithms are used and compared to solve the motion planning task: simulated annealing and genetic algorithms.

2. Materials and methods

Heuristic optimization techniques are widely used in high-dimensional engineering optimization problems that have search spaces with numerous sub-optimal solutions (local optima). The most common heuristic methods are simulated annealing algorithm, genetic algorithms, taboo search algorithm and several versions of the evolutionary strategies. In the application examined here, i.e., the planning of an autonomous agricultural vehicle's motion, the acquisition of a good solution in a high-dimension search space and in a very restricted time period, is the goal. Time restriction is a critical matter because navigation decisions have to be made rapidly in a dynamically changing environment of a moving vehicle navigated in a field of agricultural use. Thus, for the reasons of high dimensionality and time restriction, the heuristic approach seems the most reasonable and promising one.

2.1. DESCRIPTION OF THE PROBLEM

Two basic schemes for mobile vehicle navigation can be distinguished: the Cartesian map-based method and the relative sensor-based method (Freyberger and Jahns, 2000). In this work, the first method is used. We have a set of initial and final position for an autonomous agricultural vehicle, its velocity values in these positions, as well as some obstacles with known locations and sizes. More specifically, there are nine small obstacles on a part of the field with area of dimension 10×20 m. The goal is to find an optimal path through this limited part of the field, from the initial position to the final position.

The vehicle's path $(x(t), y(t))$, consists of the position variables $x(t)$ and $y(t)$, where t is a time-related parameter that ranges from 0 to 1 as the vehicle goes from the initial position to the final position. The variables $x(t)$ and $y(t)$ are each represented by a polynomial of degree 21. With a polynomial of such a large degree, even extremely complicated, multi-curve paths can be accurately represented. In addition, there are eight known boundary conditions: $b = [x(0)y(0)x'(0)y'(0)x(1)y(1)x'(1)y'(1)]^T$. These linear boundary conditions can be expressed by an equation of the general form: $\mathbf{A}\ z = b$, where z is the vector containing the 44 coefficients of the two 21-degree polynomials of the position variables $x(t)$ and $y(t)$ and $\mathbf{A}$ is the matrix that combines these coefficients with the boundary conditions. By

incorporating the eight linear boundary conditions into a representation of an input variable, we come up with an input vector s of size 36. This vector s forms the input to the cost function.

The path obtained must be relatively short, with the fewer sharp turns possible (low curvature) and such that the obstacles are avoided. Consequently, the cost function for each possible path is a weighted summation of three component costs:

(i) the total length of the path obtained, L:

$$L = \int_0^1 \sqrt{\left(\frac{\mathrm{d}x}{\mathrm{d}t}\right)^2 + \left(\frac{\mathrm{d}y}{\mathrm{d}t}\right)^2}\,\mathrm{d}t$$

(ii) a measure of the curvature of the path, C:

$$C = \int_0^1 \left(\frac{\frac{\mathrm{d}x}{\mathrm{d}t}\frac{\mathrm{d}^2y}{\mathrm{d}t^2} - \frac{\mathrm{d}y}{\mathrm{d}t}\frac{\mathrm{d}^2x}{\mathrm{d}t^2}}{\left(\left(\frac{\mathrm{d}x}{\mathrm{d}t}\right)^2 + \left(\frac{\mathrm{d}y}{\mathrm{d}t}\right)^2\right)^{\frac{3}{2}}} \right)^2 \mathrm{d}t$$

(iii) a measure of the minimum distance from the centers of the obstacles to the path (for given obstacle locations), O:

$$O = \sum_j \exp\left(\frac{D_j^2}{18}\right) \quad \text{where: } D_j^2 = (x_j - x(t_j*))^2 + (y_j - y(t_j*))^2,$$

for t_j* such that:

$$(x_j - x(t_j^*))\frac{\mathrm{d}x}{\mathrm{d}t} + (y_j - y(t_j^*))\frac{\mathrm{d}y}{\mathrm{d}t} = 0$$

Thus, the final cost function is given by the equation:

$$J(s) = w_1 * L + w_2 * C + w_3 * O$$

subject to given boundary conditions (b) and obstacle locations. The weights w_1, w_2 and w_3 are arbitrary selected to incorporate the common sense of significance of each of the three factors. After some rudimental exploration, w_1 was set equal to 3, w_2 equal to 10 and w_3 equal to 10^9. Thus, paths that led to collisions with obstacles were considered unacceptable (enormous weight factor), while the curvature factor was considered more than three times more significant than the length factor.

```
initialize (temperature T, random starting point)
while cool_iteration <= max_iterations
    cool_iteration = cool_iteration + 1
    temp_iteration = 0
    while temp_iteration <= nrep
        temp_iteration = temp_iteration + 1
        select a new point from the neighborhood
        compute current_cost (of this new point)
        δ = current_cost - previous_cost
        if δ<0, accept neighbor
        else, accept with probability exp(-δ/T)
    end while
    T = α * T      (0<α<1)
end while
```

Figure 1. Simulated Annealing Algorithm Pseudocode

2.2. HEURISTIC ALGORITHMS

In this section, the main features of the two heuristic optimization algorithms that were used in this work, Simulated Annealing and Genetic Algorithms, are presented.

2.2.1. *Simulated Annealing Algorithm*

Simulated Annealing algorithm (SA) is based on ideas first presented by Metropolis et al. in 1953. Metropolis's algorithm simulates the change in energy of a system when subjected to a cooling process, until it converges to a steady 'frozen' state. Thirty years later, Kirkpatrik et al. (1983) suggested that this type of simulation could be used for solving hard combinatorial optimization problems.

Its name comes from the physical process of annealing, which is the physical correspondence of the Metropolis algorithm. In this process, a solid is heated until all particles randomly arrange themselves in the liquid state. Then, a slow cooling follows, which spends relatively long time near the freezing point. At each temperature, the solid is allowed to reach thermal equilibrium, where energy levels follow Boltzmann distribution. As temperature decreases, the probability tends to concentrate on low energy level states (Rodrigues and Anjo, 1993). The cooling process must be careful so that the temperature is not lowered before thermal equilibrium is reached. Here, a real-valued version of the algorithm was used, as it has been shown that generally, in continuous optimization problems with real variables, the binary SA algorithm performs poorly. The algorithm can be described in the following five steps (its pseudocode is shown in Fig. 1):

(i) The parameters of the algorithm (initial random set of variables, initial temperature, cooling rate) are initialized and the cost of the initial set of variables is calculated.

(ii) A random new set of variables is chosen and its cost is evaluated.

(iii) If the new set of variables is an improvement, then it is accepted. If it is not an improvement, then it is accepted with probability *exp(-ä / T)*, where *ä* is the cost difference and *T* is the current temperature. iv) If the algorithm is *homogeneous*, the process goes to step (ii); when the maximum number of 'constant-temperature repetitions' is reached, the temperature is decreased according to the cooling schedule and the process continues to the next step. If the algorithm is *inhomogeneous*, the temperature is decreased according to the cooling schedule.

(iv) The process repeats from step (ii) until a sufficient solution is found or the maximum number of iterations is reached.

Generally, the SA algorithm can be described by a sequence of Markov Chains. It has been shown that these sequences converge asymptotically to a global optimum provided that the cooling is sufficient slow. Unfortunately, there is no way of defining this 'sufficient slow' cooling schedule. Thus, the parameters of the cooling schedule are defined experimentally. These parameters are:

– Initial value of temperature
– Length of Markov Chains
– Rule for decreasing temperature
– Final value of temperature.

The rule of decreasing temperature is a very important parameter. In the inhomogeneous algorithm, the temperature is decreased after each iteration, while in the homogeneous algorithm, it is decreased after a fixed or adaptive number of iterations (larger than one). The way of decreasing the temperature is usually geometric (with a specific cooling rate), although several other methods are often used successfully. Also, self-adaptive schedules that adjust themselves to a given problem instance are widely used. A good example is presented in Huang et al. (1986). In the SA algorithm used here, a geometric reduction function was used for the cooling schedule ($T(t+1) = \alpha T(t), 0 < \alpha < 1$), with several initial temperature values ($T(0)$) and reduction rates (α).

The other major parameter of SA is the neighborhood* definition. A neighborhood is basically defined by its size (i.e., the value of the *search range* parameter) and its sampling method. If the optimization problem is real-valued, then the specific neighborhood size can be replaced by adding a normally distributed quantity to a variable of the problem. The sampling of the neighborhood is usually random. The use of *cycling* can sometimes improve the algorithm's performance, as

* Neighborhood is the set of solutions that can be reached from the current solution in a single iteration of the algorithm.

```
generate an initial random population
while iteration <= maxiteration
  iteration = iteration + 1
  calculate the fitness of each individual
  select the individuals according to their fitness
  perform crossover with probability pc
  perform mutation with probability pm
  population = selected individuals after
               crossover and mutation
end while
```

Figure 2. Genetic Algorithm pseudocode

every element of the neighborhood is tried once before any are considered for a second time. In addition, steepest-descent is sometimes used for the neighborhood sampling, but the time consumption of this approach is often prohibited. Here, a constant neighborhood size was used, even though several values of search range were tested in the preliminary experimentation of the algorithm in order to find the best neighborhood size. The sampling method was random. A steepest descent sampling was not used, mainly for its large time consumption in our specific problem.

2.2.2. *Genetic Algorithm*

Genetic Algorithms (GAs) (Holland, 1975) are optimization techniques inspired by the genetic evolution process. In GAs, a number of individuals, namely the *population*, evolve through several *generations*, with the application of some *genetic operators*. Each individual is identified by its *chromosome*. Thus, each chromosome, which contains a number of *genes*, represents a set of variables, i.e. a solution. All possible values of the genes are called the *alleles*. The *genotype* is the complete gene encoding of an individual, while the *phenotype* is the set of characteristics of the individual in reality. The genetic operators are *selection*, *crossover* and *mutation* and they will be defined later. GAs are typically implemented as follows (their pseudocode is show in Fig. 2):

(i) A *fitness function* of the problem considered is defined, in a way that it indicates the quality of any potential solution (chromosome).

(ii) A population of chromosomes is initialized (possibly subject to certain constraints). Each chromosome is coded as a vector of fixed length (*string*), with values from the alleles. If the binary encoding is followed, then this length is determined by the desired degree of precision (Fogel, 1995).

(iii) *Selection* takes place. Each string is assigned a probability of reproduction (Fogel, 1995) and is selected according to this probability. The probability

is usually proportional to the fitness of each string. In this way, a new population is formulated.

(iv) The newly formed population of strings is now passed though the genetic operators of *crossover* and *mutation*. In crossover, two strings exchange one or more parts in randomly chosen positions. For example, for one-point crossover, the section that appears before the selected position in the first string is spliced with the section that appears after the selected position in the second string, and vice versa. In mutation, a random gene in a string is flipped. These genetic operators happen with certain probabilities. Typical values for the probability of crossover range from 0.6 to 0.95 and for the probability of mutation from 0.001 to 0.01 (Fogel, 1994). However, certain optimization problems seem to 'prefer' values far outside these ranges.

(v) The process halts if a satisfactory solution is reached or when the predefined number of maximum generations is reached. Otherwise, it goes to step (iii) and the cycle repeats.

The basic theoretical notion of Holland's GAs is that of *schemata* (Reeves, 1995). A *schema* is a subset of similar chromosomes. Their similarity is based on the fact that they have the same fixed genes at specific positions of the string. Thus, all these similar chromosomes can be represented by a general template made by these fixed genes and a wild card symbol ($*$) that, in each position that it is placed, matches any possible gene (from a predefined alphabet of course). This template is the schema. The *order* of a schema is the number of fixed genes and its *length* is the distance between its outermost fixed genes. Holland recognized that when the fitness value of a single string is evaluated, partial information about the expected fitness of all possible schemata in which that string belongs is gained. This characteristic is called *implicit parallelism*. Holland also speculated that the more implicit parallelism an optimization process has, the better the results are and he proved that this is achieved for alphabet of size 2, i.e., binary representation. However, most real-valued optimization problems seem to prefer real-valued versions of GAs. The fitness of a schema is the average fitness of all the strings that it contains. It can be proved that if the average fitness of a schema is high, then the expected number of strings belonging to that schema will be high, otherwise it will be low. The *Schema Theorem* gives the estimate of the strings belonging to a particular schema H:

$$E(m_H(t+1)) \geqslant \left(\frac{\Phi(H(t))}{\bar{\Phi}(t)} m_H(t)\right)\left(1 - p_c \frac{\mathrm{d}(H)}{l-1}\right)[(1-p_m)^{o(H)}]$$

where: $o(H)$ and $d(H)$ are the order and the length of the schema H respectively, $m_H(t)$ represents the strings in the population that are instances of H, $\Phi(H(t))$ is the average schema fitness at time t, $\bar{\Phi}(t)$ is the average fitness of the population at time t, l is the length of the strings and p_c and p_m are the crossover and mutation probabilities respectively.

This schema theorem takes into account the proportional selection (first parenthesis of the second part of the inequality) as well as the effects of crossover

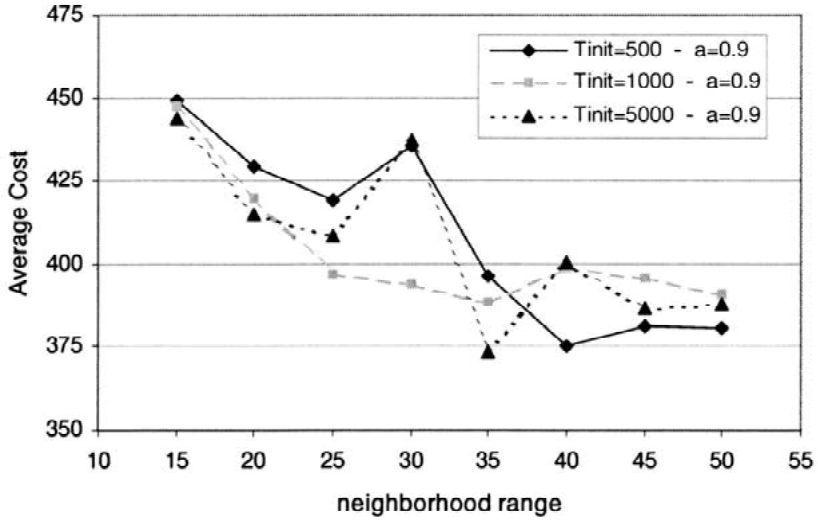

Figure 3. Average performance of SA for several neighborhood sizes.

and mutation. When evaluating a population of strings, GAs estimate the average fitness of all schemata that are present in that population and increase or decrease their representation according to the Schema Theorem.

GAs have been successfully applied in several famous optimization problems, like the Traveling Salesman Problem (TSP), sequencing and scheduling applications, graph coloring, neural network learning, etc. In our case, a real-valued version of GAs was developed and tested.

3. Preliminary experimentations

In this section, the preliminary experiments conducted with the two algorithms described before are presented. Their goal is to find the best parameters for each algorithm, so that the best-tuned versions of each one are tested and compared in the next section. All preliminary experiments were run for up to 500 cost function evaluations. Several runs (50, in most cases) were performed with different random initial points for each set of algorithm parameters explored and the average values of the performance over these runs were considered.

3.1. SA EXPERIMENTATION

The first experimentation with the SA algorithm was to determine the most appropriate combination of initial temperature, temperature reduction rate (α) and neighborhood size (*NS*) (search range value). Four different values of initial temperature (500, 1000, 5000 and 10 000) were tested with several different values of α (from 0.5 to 0.9 with step 0.1) and several values of the search range para-

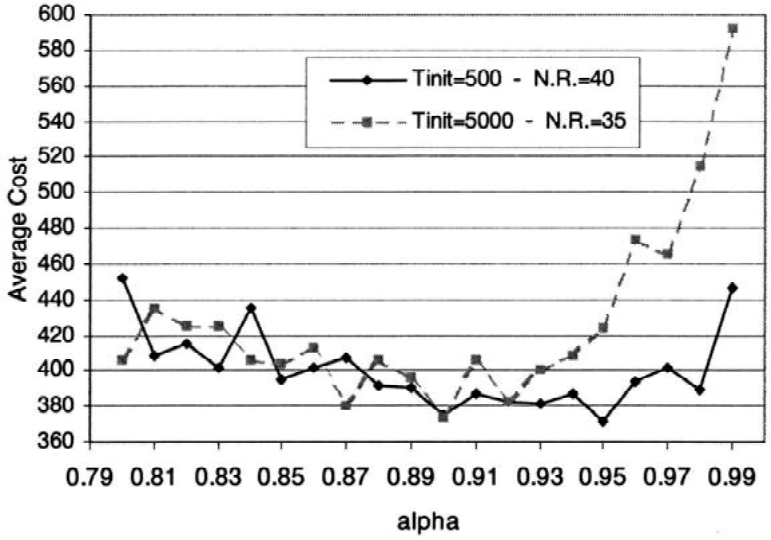

Figure 4. Average performance of SA for several values of α for the two best combinations of T_{init} and neighborhood range.

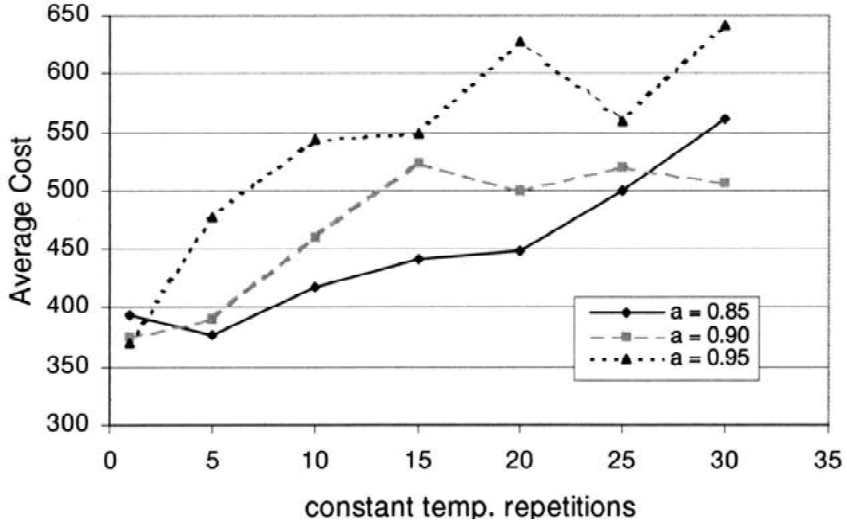

Figure 5. Average performance of SA for different numbers of constant temperature repetitions.

meter *NS* (from 15 to 50). The algorithm was run 50 times for each combination of parameters in all experimentations. The best results were achieved with values of α equal to 0.9. Figure 3 shows the average performance of the three best initial temperatures for this value of α (0.9), for all different neighborhood sizes that were tested. From this graph, we can see that the best average performance is achieved with initial temperature $T_{\text{init}} = 500$ and search range of 40 and with $T_{\text{init}} = 5000$

and search range of 35. Thus, these two combinations were further analyzed with all possible α values from 0.8 to 0.99 with step 0.01 (Fig. 4). The best performance was achieved with the '$T_{\text{init}} = 500/NS = 40$' combination and α=0.95. In addition, several different schemes of temperature reduction were tested, with values of steady temperature repetitions from 5 to 30 (all previously mentioned experiments were conducted with just one repetition of each temperature value), and the results are shown in Fig. 5. As one can see, no other schema can give better results than the originally tested one (one repetition per temperature value).

From all these experimentations we can conclude that the final SA algorithm that is going to be compared with the Genetic Algorithm, is going to have an initial temperature equal to 500 and a neighborhood search range of 40. The experimentations were run for up to 500 iterations. The final testing of the SA algorithm will have 2000 iterations, i.e. 2000 cost function evaluations. This limit on the maximum number of cost-function evaluations is forced by the limited time period in which a decision on the motion path has to be taken. Thus, the α value of 0.95 that was proved to be the best for the 500 iterations must be adjusted so that the final temperature is the same for the 2000 iterations case. Such an adjustment gives a value of α equal to 0.9872.

3.2. GA EXPERIMENTATION

As previously outlined, there are many parameters involved with the genetic algorithm. These parameters are the population size, the probability of crossover (P_c), the parameter for crossover (arithmetic), the probability of mutation (P_m) and the number of generations. In our case, as explained before, the maximum number of cost evaluations was specified as 2000 so the number of generations allowed was completely determined by the population size. If, for example, the population size were 100, then only 20 generations would be allowed. For the other parameters, it is reasonable to assume that the optimal values would be different for each crossover type. Hence, a number of experiments had to be carried out to try to determine the best possible parameters for each crossover type. After that, the various crossover types could be compared resulting in what would be the best parameters for a genetic algorithm for the specific problem under consideration.

The GA used was a real-valued version of genetic algorithms. Each chromosome consisted of 36 real numbers, which represented the input vector to the cost function, as explained in Section 2.1. The operation of crossover did not have any differences from that of binary GA crossover. Mutation however, was quite different. Instead of flipping a binary value, Gaussian noise with mean zero and some specific variance was added to some random real-valued gene. Thus, another parameter of the algorithm exists: that of the variance of the mutation operation.

For the first parameter that was explored, the population size, the GA was executed with different population sizes but with all other parameters kept constant. This approach of keeping all other parameters constant except for the parameter

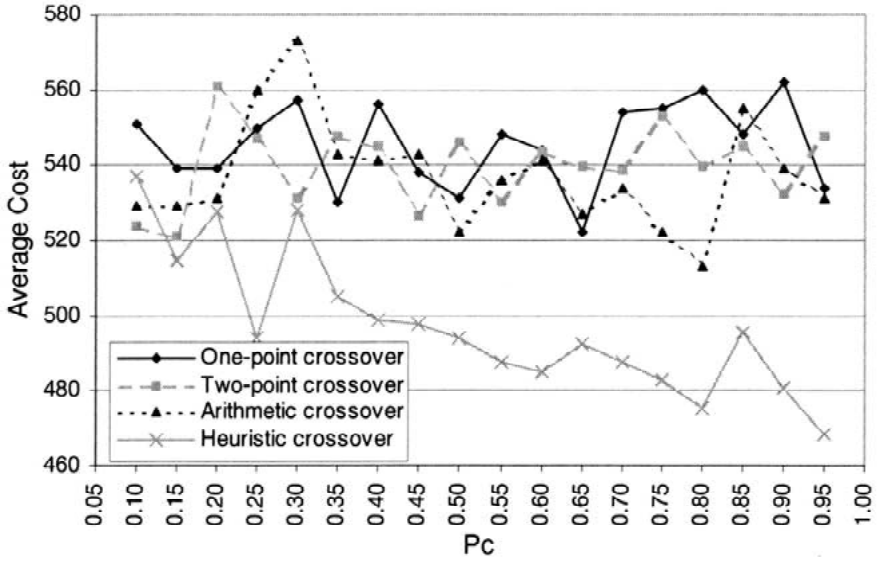

Figure 6. Average performance of GA versus several probabilities of crossover (P_c).

in question was repeated for every experiment. The number of cost-function evaluations (i.e., the size of the population multiplied by the number of generations) was kept at around 500 for all the experimentations. For this reason, an initial experiment was conducted in order to find the best population size / number of generations combination that performs around 500 cost-function evaluations. That combination (population of 60 individuals evolving over nine generations) was used throughout the experiments. After the best parameters were found, a last experiment found the optimal combination of population size/generations for the 2000 cost-function evaluations of the final algorithm, as it will be shown later.

Figures 6 and 7 show the results of the experimentations for several probabilities of crossover and probabilities of mutation, for the four different types of crossover examined (one-point, two-point, arithmetic and heuristic). In Fig. 6 the mutation probability was kept constant at 0.2, while in Fig. 7, the crossover probability of each crossover type experimentation was the best one found from the previous experiment (see Fig. 6). The next two experimentations were to decide the optimal variance of mutation for each crossover type and the best parameter of the arithmetic crossover. Finally, as mentioned before, the best population size and number of generations were found for the optimal parameters of each crossover type for a maximum of 2,000 cost-function evaluations (Fig. 8).

The final optimal parameters for each crossover type are summarized in Table 1. The first two columns correspond to probabilities of crossover and mutation respectively, the third to the variance of the mutation and finally the last two columns correspond to the size of population and number of generations of the algorithm.

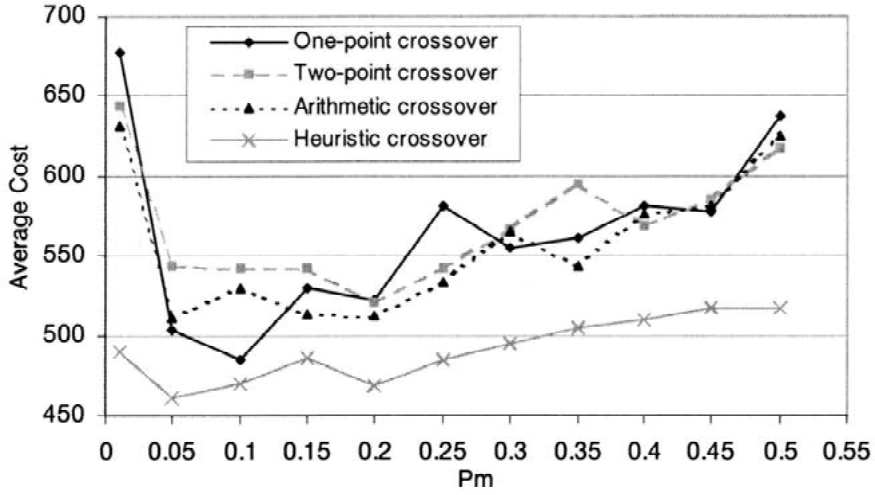

Figure 7. Average performance of GA versus several probabilities of mutation (P_m).

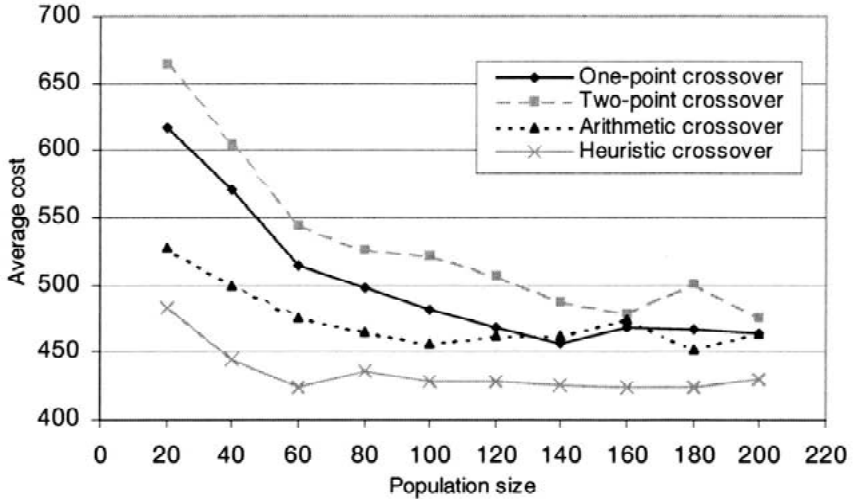

Figure 8. Average performance of GA for several population sizes.

Table 1. Optimal parameters of GA for all crossover types

	P_c	P_m	Var.	Pop.	Gen.
One-point	0.65	0.10	5	140	14
Two-point	0.15	0.20	5	200	10
Arithmetic	0.80	0.05	5	180	11
Heuristic	0.95	0.05	5	160	12

4. Comparison of the algorithms

The experimentations that were presented in the previous section led to the optimal parameter tuning of the two algorithms considered, for this specific optimization problem. The SA algorithm parameterization led to parameter values: $T_{\text{init}} = 500$, $\alpha = 0.9872$ and neighborhood range equal to 40. The genetic algorithm parameterization gave different optimal parameters for each crossover type (Table 1) and all crossover types were tested in this final comparison. These tuned algorithms are compared in this section, in order to chose the best one, that is, the one that gives the minimum cost-function value after a specific number of function evaluations. As explained earlier, the optimization problem under analysis usually occurs in situations where the time is very limited and therefore the most crucial factor. For this reason, the algorithms are compared for their performance in a maximum number of 2000 function evaluations.

Another point that must be mentioned here is that this comparison has as a goal the achievement of the best optimal solution to this specific case of motion planning and not the investigation of average performance of the examined algorithms. However, this average performance would be useful as an indication of the robustness of the algorithms. For these reasons, the decision parameter in the comparison of the algorithms is mainly the best optimal solution achieved, but in addition, the average performance of each algorithm over a number of runs was also considered.

The final well-tuned algorithms were ran for up to 2000 cost-function evaluations, with several different random initial parameters. Specifically, they were ran 100 times each. The main goal was to find the one that would give the minimum value of the cost function. From all variations of the GA, the one that gave the best results was that with the heuristic crossover. As described in the previous section, this algorithm gave best results with $P_m = 0.95$, $P_c = 0.05$, $V = 5$, population size equal to 160 and 12 generations. The best optimum found with SA was 299.2, while with the GA was 359.6 (Table 2). In addition, the average performance of the SA was much better than that of the GA, as shown in Table 2. This indicates that SA algorithm would give better performance in general if applied to a specific motion planning problem like the one examined. It also conveys a better robustness for this algorithm, if it is applied to an on-line motion planning mechanism. These can also be observed in Fig. 9, where the average best costs after specific numbers of cost-function evaluations are plotted for the two algorithms. SA obviously outperforms GA in any progress stage of the optimization process. The optimal path obtained after the SA optimization is shown in Fig. 10.

5. Conclusions

Heuristic optimization algorithms were successfully used to obtain optimal motion planning of autonomous agricultural vehicles. The Simulated Annealing algorithm was proved satisfactory for the specific problem examined in this work and it outperformed the Genetic Algorithm. However, other stochastic algorithms have

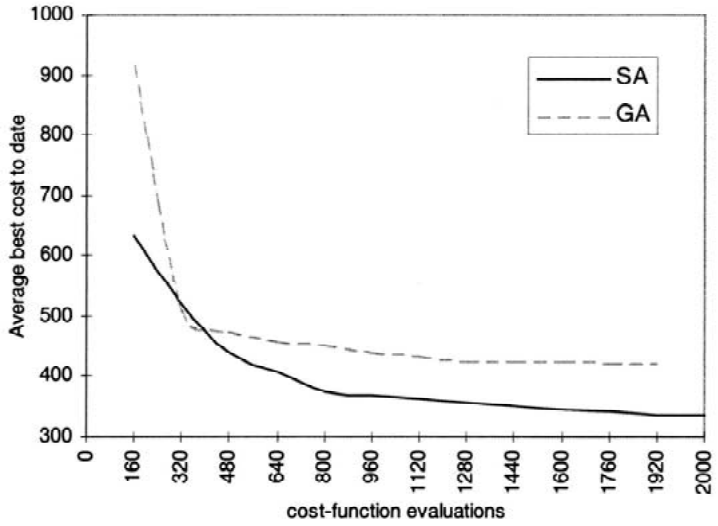

Figure 9. Average best costs versus cost-function evaluations, for both algorithms.

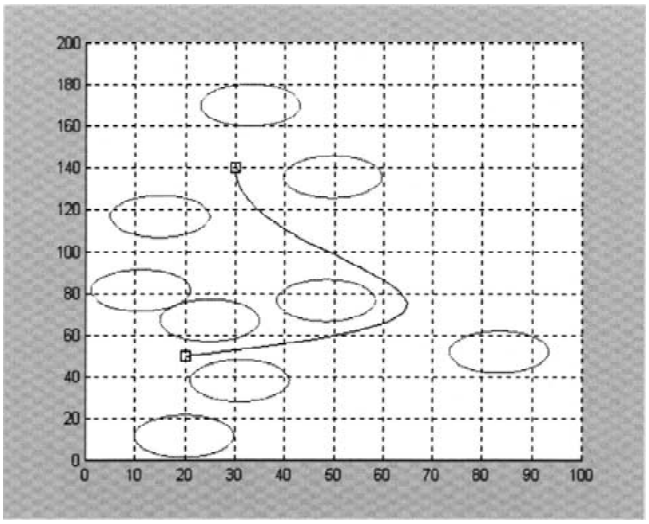

Figure 10. The optimal vehicle path found with the SA algorithm after 2000 cost-function evaluations.

also to be applied and tested in order to gain a better understanding of the specific search space and the actual level of sophistication required for finding a satisfactory optimal solution in reasonable time.

Furthermore, as a continuation of this work, an on-line heuristic-based navigation system that takes into account the dynamics of the agricultural vehicle should be constructed, that finds optimal paths from starting to ending points and also dynamically adjusts and improves the vehicle's motion.

Table 2. Best and Average performance and standard deviation for both algorithms after 2000 cost-function evaluations.

	SA	GA
Best	299.2	359.6
Average	334.3	421.1
std	12.47	9.52

References

Asada, H. and Slotine, J.-J. E. (1981), *Robot Analysis and Control*, Wiley, New York.

Borenstein, J. and Koren, Y. (1989), Real-time obstacle avoidance for fast mobile robots, *IEEE Transactions on Systems, Man and Cybernetics* 19, 1179–1187.

Canny, J. and Reif, J. (1987), New lower bound techniques for robot motion planning problems, *28th IEEE Symposium on Foundations of Computer Science*, Los Alamos, CA.

Cherif, M. (1999), Motion planning for all-terrain vehicles: a physical modeling approach for coping with dynamic and contact interaction constraints, *IEEE Transactions on Robotics and Automation* 15(2), 202–218.

Debain, C., Chateau, T., Berducat, M. and Martinet, P. (2000), A guidance-assistance system for agricultural vehicles, *Computers and Electronics in Agriculture* 25(1–2), 29-51.

Fiorini, P. and Shiller, Z. (1998), Motion planning in dynamic environments using velocity obstacles, *The International Journal of Robotics Research* 17(7), 760–772.

Fogel, D.B. (1994), An introduction to simulated evolutionary optimization, *IEEE Transactions on Neural Networks* 5(1), 3–14.

Fogel, D.B. (1995), *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE Press, New York, NY.

Freyberger, F. and Jahns, G. (2000), Symbolic course description for semiautonomous agricultural vehicles, *Computers and Electronics in Agriculture* 25(1–2), 121–132.

Galicki, M. (1998), The planning of robotic optimal motions in the presence of obstacles, *The International Journal of Robotics Research* 17(3), 248–259.

Hague, T., Marchant, J.A. and Tillett, N.D. (2000), Ground based sensing systems for autonomous agricultural vehicles, *Computers and Electronics in Agriculture* 25(1–2), 11–28.

Holland, J.H. (1975), *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press, Ann Arbor.

Huang, M.D., Romeo, F. and Sangiovanni-Vincentelli, A. (1986), An efficient general cooling schedule for simulated annealing, *IEEE International Conference on Computer Aided Design*, 381–384.

Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P. (1983), Optimization by Simulated Annealing, *Science* 220(1998), 671–680.

Reeves, C.R. (1995), *Modern Heuristic Techniques for Combinatorial Problems*, McGraw-Hill, UK.

Rodrigues, M. and Anjo, A. (1993), On simulating thermodynamics, in René V.V. Vidal (ed.), *Applied Simulated Annealing*, Springer, Berlin, Germany.

Tillett, N.D., Hague, T. and Marchant, J.A. (1998), A robotic system for plant-scale husbandry, *Journal of Agricultural Engineering Research* 69(2), 169–178.

Toda, M., Kitani, O. and Okamoto, T. (1999), Navigation method for a mobile robot via sonar-based crop row mapping and fuzzy logic control, *Journal of Agricultural Engineering Research* 72(4), 299–309.