Soft Comput (2007) 11: 269–279
DOI 10.1007/s00500-006-0068-4

FOCUS

Oscar Castillo · Leonardo Trujillo · Patricia Melin

Multiple objective genetic algorithms for path-planning optimization in autonomous mobile robots

Published online: 3 March 2006

Abstract This paper describes the use of a genetic algorithm (GA) for the problem of offline point-to-point autonomous mobile robot path planning. The problem consists of generating "valid" paths or trajectories, for an Holonomic Robot to use to move from a starting position to a destination across a flat map of a terrain, represented by a two-dimensional grid, with obstacles and dangerous ground that the Robot must evade. This means that the GA optimizes possible paths based on two criteria: length and difficulty. First, we decided to use a conventional GA to evaluate its ability to solve this problem (using only one criteria for optimization). Due to the fact that we also wanted to optimize paths under two criteria or objectives, then we extended the conventional GA to implement the ideas of Pareto optimality, making it a multi-objective genetic algorithm (MOGA). We describe useful performance measures and simulation results of the conventional GA and of the MOGA that show that both types of GAs are effective tools for solving the point-to-point path-planning problem.

Keywords Multiple objective optimization · Genetic algorithms · Autonomous robots · Path planning

1 Introduction

The problem of mobile robot path planning is one that has intrigued and has received much attention throughout the history of Robotics, since it is at the essence of what a mobile robot needs to be considered truly "autonomous". A mobile robot must be able to generate collision-free paths to move from one location to another, and in order to truly show a level of intelligence these paths must be optimized under some criteria most important to the robot, the terrain and the problem given. GAs and evolutionary methods have extensively been used to solve the path-planning problem, such as in [18] where a *Co-evolutionary* method is used to solve the path planning problem for two articulated robot arms, and in [1] where they use a GA to solve the path-planning problem in non-structured terrains for the particular application of planet exploration. In [4], an *evolutionary algorithm* is used for both off-line and on-line path planning using a linked list representation of paths, and [11] uses a *particle swarm optimization* (PSO) method based on *ant colony optimization* (ACO). However, the research work presented in this paper used the work done in [17] as a basis for comparison and development. In this work, a grid representation of the terrain is used and different values are assigned to the cells in a grid, to represent different levels of difficulty that a robot would have to traverse a particular cell. Also they present a codification of all monotone paths for the solution of the path-planning problem. The exact way in which the GAs are developed in this paper is presented in Sect. 3. Section 2 gives some basic theory needed to further understand the purpose of the problem we are trying to solve and explain the methods used in order to reach the goal expressed by the path- planning problem. Section 4 describes the details of the GAs used in this paper defining each aspect of its architecture and flow of the GAs used as well as the problem-specific path repair mechanism employed. Section 5 gives a more detailed look at the implementation of the GAs discussed in Sect. 4. Section 6 discusses the simulation results and performance of the GAs, and Sect. 7 concludes this paper with a discussion on future work.

O. Castillo (✉)· L. Trujillo · P. Melin
Deptartment of Computer Science,
Tijuana Institute of Technology Tijuana, Mexico
E-mail: ocastillo@tectijuana.mx

2 Basic theory

This section is intended to present some basic theory used to develop the GAs in this paper for use in the path-planning problem, covering topics like basic genetic algorithm theory, multi-objective optimization, triggered hypermutation and autonomous mobile robot point-to point path planning.

2.1 Genetic algorithms

A GA is an evolutionary optimization method used to solve, in theory "any" possible optimization problem. A GA [6] is

based on the idea that a solution to a particular optimization problem can be viewed as an *individual* and that these individual characteristics can be coded into a finite set of parameters. These parameters are the *genes* or the *genetic information* that makes up the *chromosome* that represents the real-world structure of the individual, which in this case is a solution to a particular optimization problem [8]. Because the GA is an evolutionary method, this means that a repetitive loop or a series of *generations* are used in order to evolve a *population S* of p individuals to find the *fittest* individual to solve a particular problem. The *fitness* of each *individual* is determined by a given *fitness function* that evaluates the level of aptitude that a particular *individual* has to solve the given optimization problem. Each *generation* in the genetic search process produces a new set of individuals through *genetic operations* or *genetic operators: crossover* and *mutation*, operations that are governed by the *crossover rate* γ and the *mutation rate* μ, respectively. These operators produce new *child chromosomes* with the intention of bettering the overall fitness of the population while maintaining a global search space. Individuals are selected for *genetic operations* using a *selection method* that is intended to select the fittest individuals for the role of *parent chromosomes* in the *crossover* and *mutation* operations. Finally these newly generated child chromosomes are reinserted into the population using a *replacement method*. This process is repeated a k number of *generations* . The simple GA [8] is known to have the next set of common characteristics:

- Constant number of p *individuals* in the genetic search *population*.
- Constant length binary string representation for the *chromosome*.
- One or two point *crossover* operator and single bit *mutation* operator, with constant values for μ and γ.
- Roulette wheel *selection or stochastic sampling with replacement (SSR) method*.
- Complete or *generational replacement method* or *generational* combined with an *elitist* strategy.

2.2 Multi-objective genetic algorithms

Real-world problem solving will commonly involve [9] the optimization of two or more objectives at once, a consequence of this is that it is not always possible to reach an optimal solution with respect to all of the objectives evaluated individually. Historically a common method used to solve multi-objective problems is by a linear combination of the objectives, in this way creating a single-objective function to optimize [15] or by converting the objectives into restrictions imposed on the optimization problem [16]. With regard to evolutionary computation, Schaffer [12] proposed the first implementation for a multi-objective evolutionary search. The proposed methods in [5,6,14] all center around the concept of *Pareto optimality* and the *Pareto optimal set*. Using these concepts of optimality of *individuals* evaluated under a multi-objective problem, they each propose a *fitness* assignment to each individual in a current population during an evolutionary search based upon the concepts of *dominance* and *non-dominance* of *Pareto optimality* [19], where the definition of *dominance* is stated as follows:

Definition 1 *For an optimization (minimization) problem with n objectives, solution u is said to be dominated by a solution v if:*

$$\forall i = 1, 2, \ldots, n, \quad f_i(u) \geq f_i(v) \tag{1}$$

$$\exists j = 1, 2, \ldots, n, \quad \therefore f_i(u) \geq f_i(v) \tag{2}$$

2.3 Triggered hypermutation

In order to improve on the convergence of a GA, there are several techniques available, such as [8], expanding the memory of the GA in order to create a repertoire to respond to unexpected changes in the environment.

Another technique used to improve the overall speed of convergence for a GA is the use of a triggered hypermutation mechanism [3], which consists of using *mutation* as a control parameter in order to improve performance in a dynamic environment. The GA is modified by adding a mechanism by which the value of μ is changed as a result of a dip in the fitness produced by the best solution in each generation in the genetic search. This way μ is increased to a high *hypermutation* value each time the top fitness value of the population at generation k dips below some lower limit set beforehand; this causes the search space to be incremented at a higher rate, thanks to the higher mutation rate, and conversely μ is set back to a more conventional lower value once the search is closing in to an appropriate optimal solution.

2.4 Autonomous mobile robots

An autonomous mobile robot as defined in [18] can be seen as a vehicle that needs the capability of generating collision-free paths that take the robot from a starting position s to a final destination d, and needs to avoid obstacles present in the environment. The robot must be able to have enough relevant information of his current position relative to s and d, and of the state of the environment or terrain that surrounds it [13]. One advantage about generating paths or trajectories for these kinds of robots, compared to the more traditional robot arms, is that in general there are far less restrictions with regard to the precision with which the paths must be generated. The basic systems that operate in an autonomous mobile robot are:

(1) *Vehicle control*
(2) *Sensor and vision*
(3) *Navigation*
(4) *Path planning*.

2.5 Point-to-point path planning problem

The path-planning problem, when analyzed with the point-to-point technique [2], comes down to finding a path from one point to another (start and destination). Obviously, one of the most important reasons to generate an appropriate path for a robot to follow is to help it avoid possible danger or obstacles along the way; for this reason an appropriate representation of the terrain is needed generating a sufficiently complete map of the given surroundings that the robot will encounter along its route.

The general path-planning problem, that all autonomous mobile robots will face, has been solved (to some level of satisfaction) with various techniques, besides the evolutionary or genetic search, such as, using the *Voroni generalized graph* [2], or using a *fuzzy controller* [7], yet another is by the use of *artificial potential fields* [10]. These along with many other methods are alternatives to the evolutionary search mentioned earlier or the GA used in this paper.

3 Proposed method for path planning

The first step before we can continue and give the details of the GA implementation used to solve the path-planning problem is to explicitly define the problem and what is it that we are expecting out of the subsequent genetic search. To this end, we propose what will be the *input/output* pair that we are expecting from our GA as follows:

Input:

(1) An n x n grid, where the starting cell s for the robot is in one corner and the destination cell d is diagonally across from it.
(2) Each cell with a corresponding *difficulty weight* wd assigned to it ranging $[0, 1]$.

Output:

A path, defined as a sequence of adjacent cells joining s and d, and that complies with the following restrictions and optimization criteria:

(1) The path most not contain cells with wd $= 0$ (solid obstacles).
(2) The path must stay inside the grid boundaries.
(3) Minimize the path length (number of cells).
(4) Minimize the total difficulty for the path, that means, the combined values of wd for all the cells in a given path.

We must also establish a set of ground rules or assumptions that our GA will be operating under.

– The $n \times n$ grid is not limited to all cells in the grid having to represent a uniform or constant size in the terrain, each cell is merely a conceptual representation of spaces in a particular terrain.
– Each cell in a terrain has a given *difficulty weight* wd between the values of $[0, 1]$, that represents the level of difficulty that a robot would have to pass through it, where the lower bounds 0 represents a completely free space and the higher bounds 1 represents a solid impenetrable obstacle.
– The terrain is considered to be static in nature.
– It is assumed that there is a sufficiently complete knowledge with regard to the state of the terrain in which the robot will operate.
– The paths produced by the GA are all monotone paths.

4 GA Architecture

We now turn to the actual implementation of our GA, used to solve the path-planning problem for one and two optimization objectives. So we describe each of the parts of our GA and give a brief description of each, clearly stating any differences between the one and two optimization objectives' implementations, and Fig. 1 shows the flow chart for each.

4.1 Individual representation

Basically, the chromosome structure was taken from the work done in [17] where a binary string representation of monotone paths is used. The binary string chromosome is made up of $n - 1$ (where n is the number of columns and rows in the grid representing the map of a given terrain) pairs of *direction/distance* of length $3 + \log[2]n$, and an extra bit a which determines if the path is x-monotone ($a = 0$) or y-monotone ($a = 1$). And each pair of *direction/distance* codes the direction in which a robot moves inside the grid and the number of cells it moves through in that direction. The meaning of the bits in each pair of *direction/distance* is given in Tables 1 and 2. The coding used greatly facilitates its use in a GA, because of its constant length no special or revamped genetic operators are needed, a problem that would be very cumbersome to solve if using a linked-list chromosome representation of the path as done in [18].

4.2 Initial population

The population S used in the genetic search is initialized with p total individuals. Of the p individuals in S, $p - 2$ of them are generated randomly while the remaining two represent straight line paths from s to d, one of this paths is x-monotone and the other is y-monotone.

So we can clearly define the population S as being made up by:

$$S = \{x_0, x_1, x_2, \ldots, x_{p-2}, a, b\} \tag{3}$$

where x_i are randomly generated individuals by a and b that are x-monotone and y-monotone paths, respectively, that take a straight-line route from s to d.

4.3 Path repair mechanism

Each path inside the population S is said to be either *valid* or *non-valid*, where the criteria for *non-validity* are:

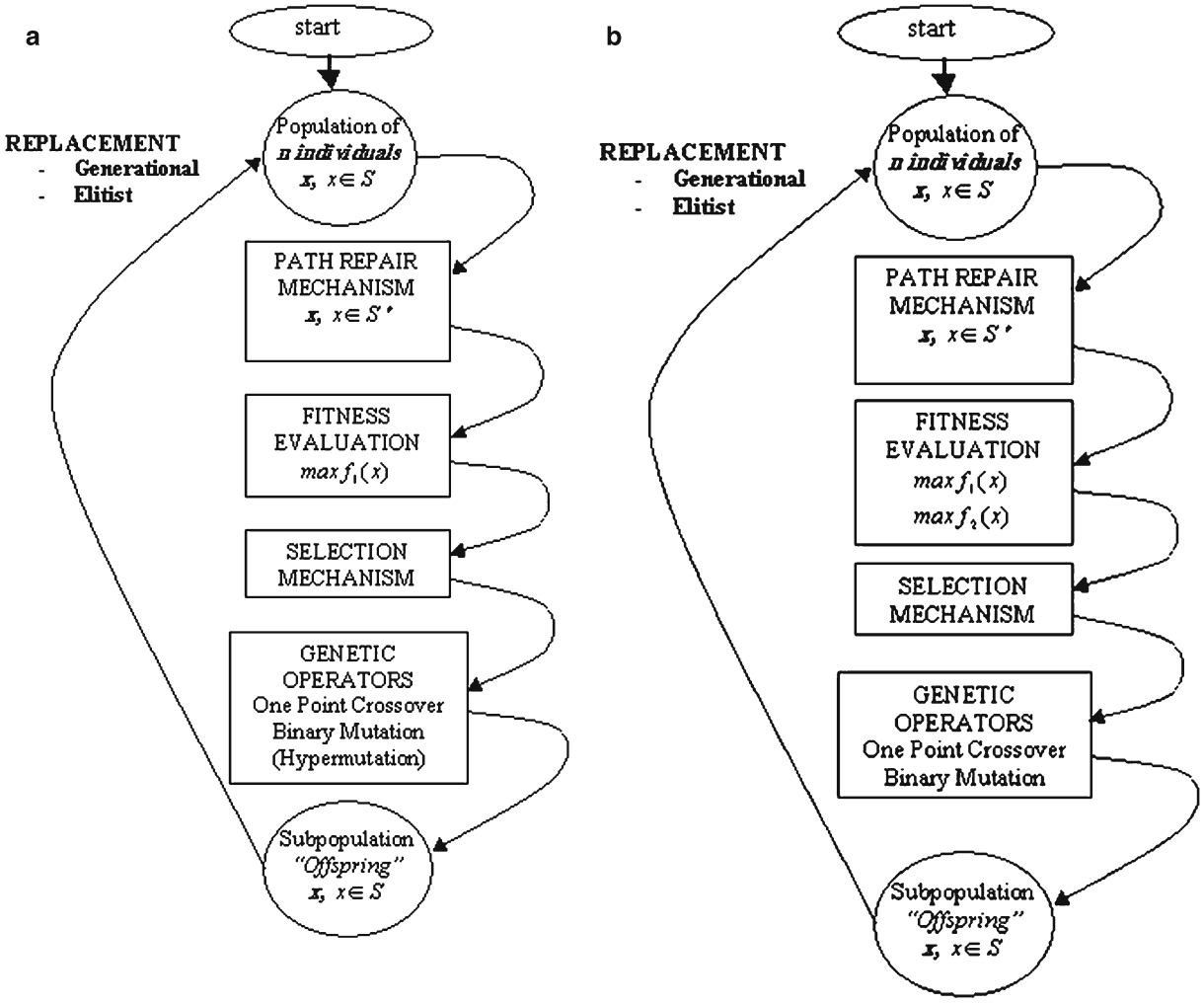

Fig. 1 Genetic algorithm (GA) flowchart (**a**) conventional GA with one optimization objective, (**b**) Multi-objective genetic algorithm (MOGA) with two optimization objectives

Table 1 Coding of each *direction/distance* pair when $a = 0$

First two bits	Remanding bits	Movement
00	Number of cells (direction given by sign)	Vertical
01	Ignored	Diagonal – Up
10	Ignored	Horizontal
11	Ignored	Diagonal – Down

Table 2 Coding of each *direction/distance* pair when $a = 1$

First two bits	Remanding bits	Movement
00	Number of cells (direction given bye sign).	Horizontal
01	Ignored	Diagonal – Left
10	Ignored	Vertical
11	Ignored	Diagonal – Right

(1) Path contains a cell with a solid obstacle (wd = 1).
(2) Path contains cells out of bounds
(3) The paths final cell is not *d*.

Using this set of three rules to determine the state of validity of a given path for a particular genetic search, we can define a *subpopulation* S' which is made up of entirely *non-valid* paths in S.

The path repair mechanism used with the GA is a *Lamarckian* process designed to take *non-valid* x', where $x' \varepsilon S'$, and determine if they can be salvaged and return to a *valid* state, so as to be productive in the genetic search. As just because a particular path is determined to be *non-valid*, this does not preclude it from having possible information coded in its chromosome that could prove to be crucial and effective in the genetic search process. This is how *non-valid* paths are given low fitness values with the penalty scheme used in the fitness evaluation, only after it has been determined that its *non-valid* state cannot be reversed. The path repair mechanism is therefore equipped with three process designed to eliminate each point of *non-validity*. They are:

– *Solid obstacle intersection repair process:* this process is intended to eliminate single non-adjacent cells in a path that contain solid obstacles in theme. It eliminates intersections

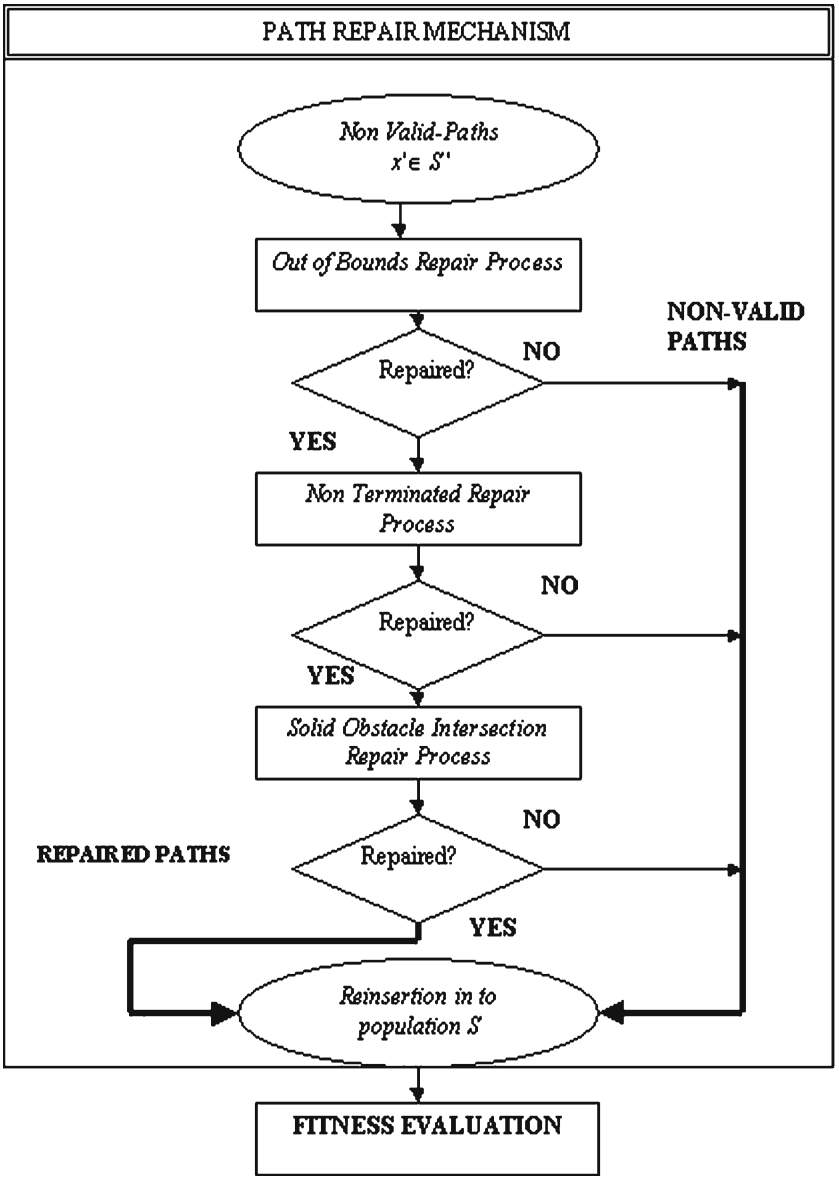

Fig. 2 Path repair mechanism

with solid obstacles in horizontal, vertical and diagonal segments in a given *non-valid* path.

– *Out of bounds repair process:* designed to repair paths that go beyond the bounds of a given grid representation of a terrain, and comes in back. If a given path lives the boundaries of the grid and comes back at another cell, the process fills out the missing links between the two points in the grid with straight line segments along the x and y axis.
– *Non-terminated path repair process:* these repair process takes paths in which the final cell is not the destination cell d, but instead a cell in the final row or column in the grid, the process is then designed to fill out the remanding cells in the final row or column (which ever is appropriate for the given path) to close the path and bring it to end at the correct position in the grid.

Figure 2 shows the functional flow of the path repair mechanism. The mechanism takes every path in S' and determines the cause or causes of the *non-valid* state, it then applies each of the process in the shown sequence. First it repairs paths that go out of bounds; this is done because paths with this characteristic, once they have been repaired, could still be in violation of the other two rules of *validity*. Once a process has been applied to a path, it is determined if the path is no longer a *non-valid* path; if it is not, it is sent back to the population pool S with the rest of the *valid* paths; but if the path is still found to be *non-valid*, then it must first be determined if the original cause of *non-validity* has been corrected and then the next process is applied and so on; but if it has not been corrected then this means that the path is irreparable and is sent back to S and destined to be penalized once it has its fitness evaluated. The path repair mechanism

repeats this sequence of steps for each repair process, ending with the solid obstacle intersection repair process, which is applied last because of the probability that the two other repair (non-terminated repair process and out of bounds repair process) processes can unwittingly cause one or more intersections with solid obstacles during the repair.

4.4 Fitness evaluation

As was mentioned earlier, we introduce here both single- and two-objective optimization of the path-planning problem, taking into account the length of a given path and the difficulty of the same as the two criteria for optimization for paths in the population. Hence, the way in which each implementation of the GA assigns fitness values differs for obvious reasons. That is why we must give two different fitness evaluation procedures. There is still one similarity between the conventional GA and the MOGA when it comes to fitness evaluation, that in both we implement a penalty scheme for *non-repaired non-valid* paths.

4.4.1 Single-objective optimization

First, we consider the simplified approach to the path-planning problem, and probably the most common, the optimization of the path length objective, of minimizing the possible time and energy consumption of a mobile robot by minimizing the length of the path that it must take to move from one place to another.

Considering our conventional GA, we can say that for paths inside S we optimize for only one objective which is the path length; therefore we define fitness $f_1(x)$ as given by:

$$f_1(x) = n^2 - c \tag{4}$$

where c is the number of cells in a given path x. As was mentioned in Sect. 4.3 fitness assignment also takes into account the *non-validity* criteria in Sect. 4.3, and the fitness value of a *non-valid* path is thus assigned as follows; once $f_1(x)$ has been calculated by Eq. (4),

(1) $f_1(x) = 1$ if a path is out of bounds.
(2) $f_1(x) = (n^2 - c)/20xI$ where I is the number of intersections that a path has with solid obstacles, if a given path x intersects with solid obstacles.

4.4.2 Two-objective optimization

Besides the fitness $f_1(x)$ used in Sect. 4.4.1 given for path length, a second fitness assignment $f_2(x)$ for path difficulty is given, and is calculated by,

$$f_2(x) = n^2 - \sum \mathrm{wd}_i \tag{5}$$

where the second term in (5) is the sum of wd for each cell in a given path x. With this we are forced to use Pareto optimality for a rank based system for individuals in population S. So for a path x where $x \varepsilon S$ its final fitness values is given by their rank value inside of S determined by,

$$\mathrm{rank}(x) = p - t \tag{6}$$

where p is the size of population S and t is the number of individuals that dominate x in S.

4.5 Genetic operators

As in Sect. 4.4, we must make a distinction between the conventional GA and the MOGA used to solve the path-planning problem; we summarize the genetic operators used as follows:

- One optimization objective:
  - Crossover: One point crossover (variable location).
  - Mutation: One bit binary mutation and Triggered Hypermutation.
- Two optimization objectives:
  - Crossover: One point crossover (variable location).
  - Mutation: One bit binary mutation. (due to the fact that triggered hypermutation is based on the ability to modify μ in a genetic search, when the population's *best* individual drops its fitness value below a predefined threshold, its use in our MOGA cannot be clearly defined because of its reliance on Pareto ranking for fitness evaluation).

4.6 Replacement, selection and termination criteria

The selection method used for both the conventional and MOGAs was Roulette wheel selection or SSR as it is the most common method used in conventional GAs. As for the replacement scheme both generational and elitist strategies where employed. For the termination criteria for the GA a fixed upper limit k gave the maximum number of generations per search.

5 Experimental results

In this section, we present some experimental results. These are divided in two sets: one for each implementation of our GA.

5.1 Conventional GA with a single optimization objective

As mentioned before, the GA used to solve the path-planning problem in a binary representation of the terrain, was a conventional GA, but due to experimental results some modifications where made, were the most important modification made to the algorithm was the inclusion of a triggered hypermutation mechanism [3]. Usually this mechanism is used for problem-solving in dynamic environments, but used here it drastically improved the overall performance of the our GA. Table 3 summarizes the simulation results of the best

Table 3 Simulation results for the conventional genetic algoright (GA)

Replacement	Population (p)	n	Mutation	Percent Ideal fitness	Solutions
Elitist	30	16	Binary mutation $\mu = 0.08$	97.96	95
Elitist	100	16	Binary mutation $\mu = 0.08$	97.8	96.4
Elitist	30	16	Hypermutation	98.62	97
Elitist	100	16	Hypermutation	98	98
Generational	30	16	Binary mutation $\mu = 0.08$	97	91
Generational	100	16	Binary mutation $\mu = 0.08$	96.9	92
Generational	30	16	Hypermutation	97.5	95
Generational	100	16	Hypermutation	98	96
Elitist	30	24	Binary mutation $\mu = 0.08$	98.5	69
Elitist	100	24	Binary mutation $\mu = 0.08$	98.7	75
Elitist	30	24	Hypermutation	98.7	76
Elitist	100	24	Hypermutation	98.8	94
Generational	30	24	Binary mutation $\mu = 0.08$	97.8	83
Generational	100	24	Binary mutation $\mu = 0.08$	98.1	92
Generational	30	24	Hypermutation	97.7	94
Generational	100	24	Hypermutation	98.3	96
Elitist	30	32	Binary mutation $\mu = 0.08$	99	31
Elitist	100	32	Binary mutation $\mu = 0.08$	98.2	39
Elitist	30	32	Hypermutation	98.8	48
Elitist	100	32	Hypermutation	98	56
Generational	30	32	Binary mutation $\mu = 0.08$	98	48
Generational	100	32	Binary mutation $\mu = 0.08$	98.56	83
Generational	30	32	Hypermutation	97.7	60
Generational	100	32	Hypermutation	97.2	88

configuration of the GA with terrains represented by $n \times n$ grids. We used $n = 16, 24, 32$, each with 100 runs, 500 generations per search, and randomly generated maps, with a 35% probability for placing an obstacle in any given cell (35% probability was used due to the fact that experimentally, a lower number usually produce *easy* terrains, and a higher number produced *unsolvable* terrains without a clear path between s and d). The results gathered from our test runs provide us with a couple of interesting and promising trends. First we can clearly state that Hypermutation slightly increases the performance of our conventional GA, anywhere from 2 to 5% in terms of problems solved with a valid solution where all else is equal. Also we determine that the most decisive factor for finding a valid solution as the value of n increases is the use of a generational replacement scheme for the GA, increasing the percentage of valid solutions found by as much as 44% with all other things equal. We can also see that the small percentage gained in terms of fitness value, when using an elitist strategy does not justify the poor overall problem solving performance that the GA produces with this replacement scheme.

5.2 MOGA with two optimization objectives

The complete solution we want for the path-planning problem includes a terrain with not only free spaces and solid obstacles but also difficult terrain that a robot should avoid when possible, making it a multiple objective optimization problem. A direct comparison is made in Table 4 between the MOGA proposed here and the GA proposed by Sugihara [17].

We use the benchmark test presented in Fig. 3, which was used in [16] due to its capability of projecting an accurate general performance score for the GA, and the performance measure of *probability optimality* $L_{\text{opt}}(k)$, which is a representation of the probability that a GA has of finding an optimal solution to a given problem. In this case, is the probability of finding a solution on the Pareto optimal front. Using $L_{\text{opt}}(k)$ as the performance measure we present a set of optimal operating parameters for our MOGA using both a generational and elitist replacement schemes, Figs. 4, 5, 6, 7, 8, 9, 10 and 11 show the simulation results that support these values. We also compare the two methods along with the GA proposed in [17] and we show the results in

Table 4 Sugihara and mukti-objective genetic alogorithm (MOGA) methods

	Sugihara [17]	MOGA
Paths	Monotone	Monotone
Fitness	Linear combination	Pareto optimality
Repair mechanism	Out of bounds	Out of bounds, collisions and incomplete paths
Genetic Operators	One point crossover and binary mutation	One point crossover and single bit binary mutation
Selection method	Roulette wheel with tournament	Roulette wheel
Replacement Method	Generational	Generational, and elitist strategy
Termination	Max. generations	Max. generations

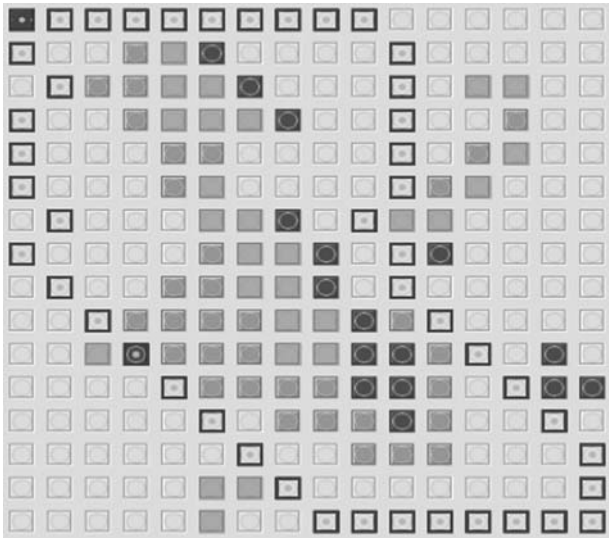

Fig. 3 First benchmark test, with two paths on the pareto optimal front

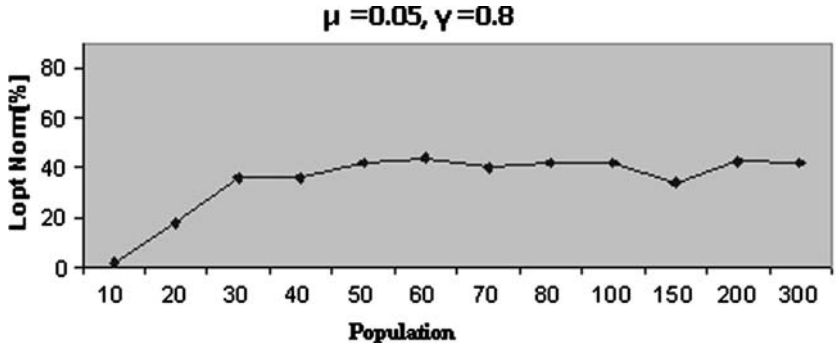

Fig. 4 Normalized $L_{\text{opt}}(k)$ and population size with generational replacement

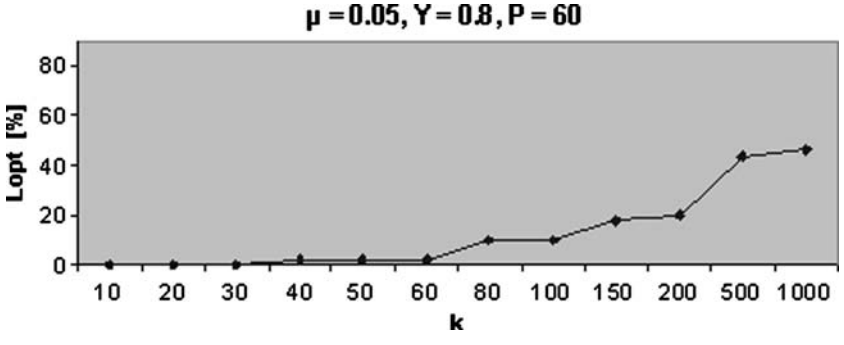

Fig. 5 $L_{\text{opt}}(k)$ and number of generations with generational replacement

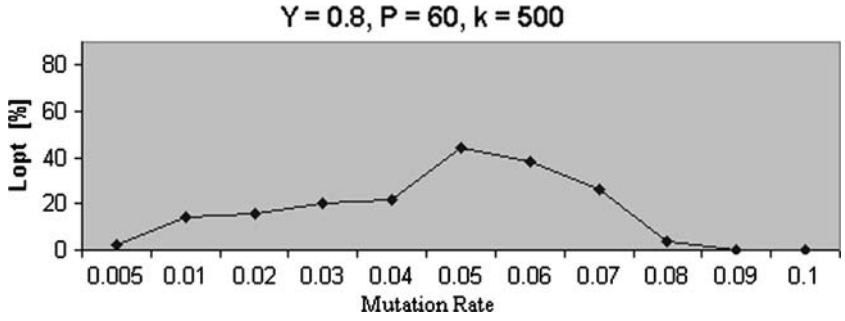

Fig. 6 $L_{\text{opt}}(k)$ and crossover rate with generational replacement

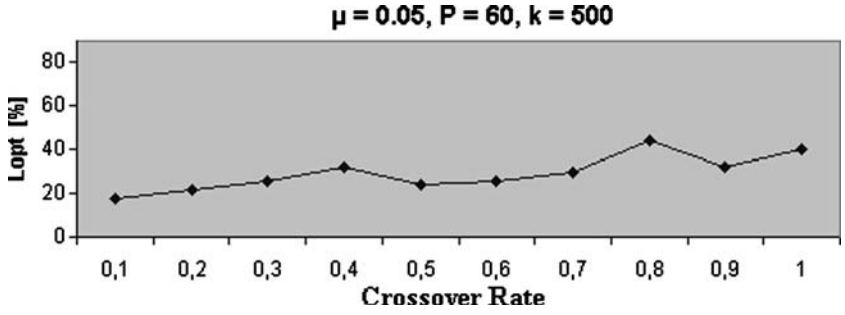

Fig. 7 $L_{\text{opt}}(k)$ and mutation rate with generational replacement

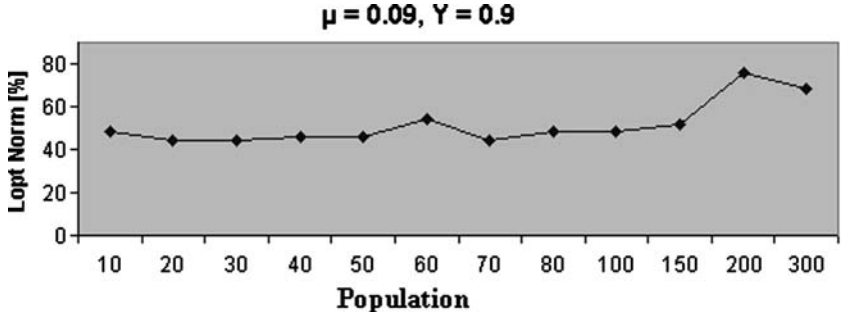

Fig. 8 Normalized $L_{\text{opt}}(k)$ and population size with elitist replacement

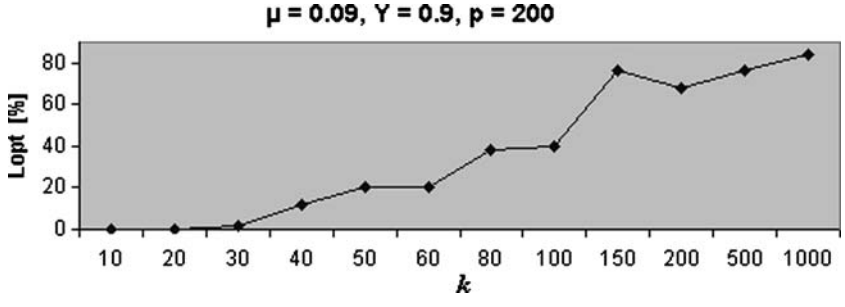

Fig. 9 $L_{\text{opt}}(k)$ and number of generations with elitist replacement

Table 5 Simulation results for MOGA

	Sugihara [17]	AGOM generational	AGOM elitism
Population	30	60	200
No. of generations (k)	1,000	500	150
Mutation rate (μ)	0.04	0.05	0.09
Crossover rate (γ)	0.8	0.8	0.9
Win probability (ω)	0.95	Not applicable	Not applicable
Probability of optimality $L_{\text{opt}}(k)$	45%	44%	76%

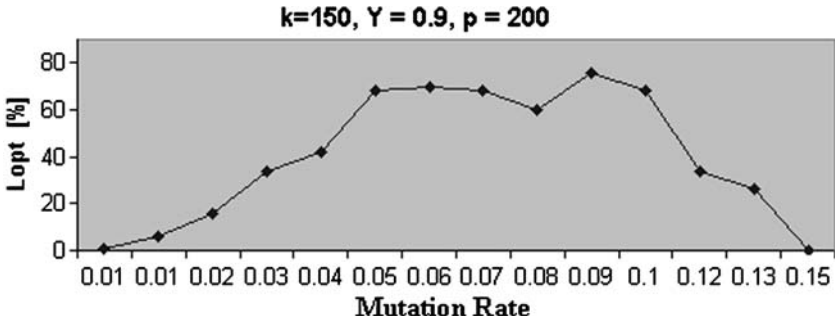

Fig. 10 $L_{\text{opt}}(k)$ and mutation rate with elitist replacement

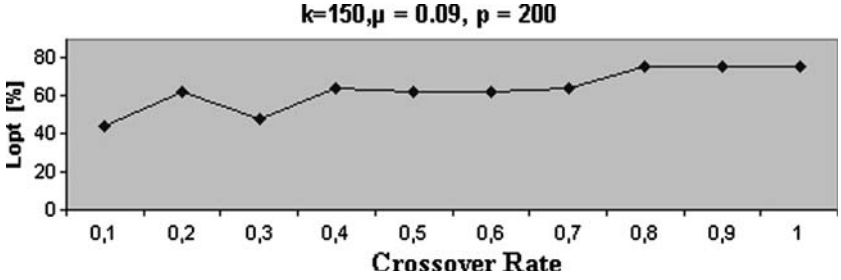

Fig. 11 $L_{\text{opt}}(k)$ and crossover rate with elitist replacement

Table 5, the comparison is made under a normalized value for $kp = 30,000$ keeping the overall computational cost equal for each GA.

From Table 5, we can see how decisively the use of an elitist replacement strategy along with a Pareto-based approach for the multi-objective problem gives a more than acceptable performance gain to that offered by Sugihara [17]. Along with the results obtained from benchmark test 1, we also include another test proposed in [15] as an accurate description of a GA performance for the path-planning problem, shown in Fig. 12. Due to the more difficult nature of this given terrain, we include a second performance measure $L'_{\text{opt}}(k)$ which we define as the *probability for optimality*, taking into account solutions that come within one cell in length of the optimum path, as optimum solutions themselves for the given problem. With these we present Table 6 that clearly demonstrates the high performance measure for the MOGA with an elitist replacement scheme. Three other terrains, with known Pareto optimal fronts were used and each presented a $L_{\text{opt}}(k)$ of 100%. And in 50 randomly generated terrains for different n values, the MOGA continually provided a Pareto front with two to five different solutions, a set of solutions from where to choose the most useful solution, given a particular situation. Thus, showing its ability to give a decision making process.

Table 6 Simulation results for MOGA

	MOGA generational(%)	MOGA elitist
$L_{\text{opt}}(k)$	32	34
$L'_{\text{opt}}(k)$	48	100

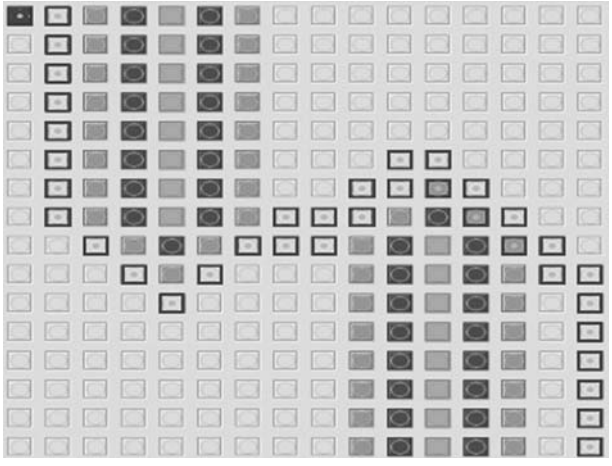

Fig. 12 Second benchmark test, with two paths on the Pareto optimal front

Note that the ideal fitness column expresses the percentage of the ideal solution for a grid configuration that is the value of a map with zero obstacles, which a particular best solution of a genetic search reaches.

6 Conclusions

This paper presented a GA designed to solve the mobile robot path-planning problem. We showed with simulation results that both a conventional GA and a MOGA – based on Pareto optimality, equipped with a basic repair mechanism for *non-valid* paths – can solve the point-to-point path-planning

problem when applied to grid representations of binary and continuous simulation of terrains, respectively. From the simulation results gathered from experimental testing the conventional GA with a generational replacement scheme and triggered hypermutation (which is commonly referred to as a conversion mechanism for dynamic environments) gave consistent performance to varying degrees of granularity in the representation of terrains without a significant increase in population size or number of generations needed in order to complete the search in a satisfactory manner, while the MOGA based on Pareto optimality combined with an elitist replacement scheme clearly improves upon previous [17] work done with multiple objective path-planning problem based on linear combination, with the added advantage of providing more than one equally usable solution. Some possible work that we suggest, that may provide us with more insight into the full effectiveness of the work done here is:

- Direct comparisons of the MOGA with conventional methods, not just other GA
- Using other multi-objective mechanisms, besides Pareto optimality
- Doing a more global search of possible GA operators, selection and replacement schemes
- Include the GAs presented here, into experiments using *real* autonomous mobile robots.

References

1. Ali AD, Babu MSN, Varghese K (2002) Offline path planning of cooperative manipulators using co-evolutionary genetic algorithm. In: Proceedings of the international symposium on automation and robotics in construction, 19th (ISARC), pp 415–124
2. Choset H, La Civita ML, Park JC (1999) Path planning between two points for a robot experiencing localization error in known and unknown environments. In: Proceedings of the conference on field and service robotics (FSR'99), Pittsburgh, PA
3. Cobb HG (1990) An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Technical Report AIC-90-001, Naval Research Laboratory, Washington DC
4. Farritor S, Dubowsky S (2002) A genetic planning method and its application to planetary exploration. ASME J Dyn Syst Meas Control 124(4):698–701
5. Fonseca CM, Fleming CJ (1993) Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. In: 5th international conference genetic algorithms, pp 416–423
6. Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning, Addison-Wesley, Reading, MA
7. Kim BN, Kwon OS, Kim KJ, Lee EH, Hong SH (1999) A study on path planning for mobile robot based on fuzzy logic controller. In: Proceedings of IEEE TENCON'99, pp 1–6
8. Man KF, Tang KS, Kwong S (1999) Genetic algorithms, 1st ed. London Springer
9. Oliveira GMB, Bortot JC, De Oliveira PPB (2002) Multiobjective evolutionary search for one-dimensional cellular automata in the density classification task. In: Proceedings of artificial life VIII. Cambridge, MIT Press, pp 202–206
10. Planas RM, Fuertes JM, Martinez AB (2002) Qualitative approach for mobile robot path planning based on potential field methods. In: 16th international workshop on qualitative reasoning (QR'02), pp 1–7
11. Sauter JA, Matthews R, Parunak, HVD, Brueckner S (2002) Evolving adaptive pheromone path planning mechanisms. In: 1st international conference on autonomous agents and multi-agent systems, Bologna, Italy, pp 434–440
12. Schaffer JD (1985) Multiple objective optimization with vector evaluated genetic algorithms. Genetic algorithms and their applications. In: Proceedings of the first international conference on genetic algorithms, pp 93–100
13. Spandl H (1992) Lernverfahren zur Unterstützung der Routenplanung fur eine Mobilen Roboter. Dissertation, Universität Karlsruhe, auch VDI-Forsch-Heft Reihe 10 Nr. 212, Düsseldorf, Germany, VDI-Verlag
14. Srinivas M, Deb K (1994) Multiobjective optimization using nondominated sorting in genetic algorithms. Evol Comput 2(3):221–248
15. Sugihara K (1997) A case study on tuning of genetic algorithms by using performance evaluation based on experimental design. Technical Report ICS-TR-97-01, Deptartment of Information and Computer Sciences, University of Hawaii at Manoa
16. Sugihara K (1997) Measures for performance evaluation of genetic algorithms. In: Proceedings of the 3rd joint conference on information sciences, vol I, Research Triangle Park, NC, pp 172–175
17. Sugihara K (1999) Genetic algorithms for adaptive planning of path and trajectory of a mobile robot in 2d terrains. IEICE Trans Inf Syst E82-D:309–313
18. Xiao J, Michalewicz Z (2000) An evolutionary computation approach to robot planning and navigation. In: Hirota K, Fukuda T (eds) Soft computing in mechatronics. Springer, Berlin Heidelberg, New york, pp 117–128
19. Zitzler E, Thiele L (1999) Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Trans Evol Comput 3(4):257–271