Genetic Algorithms for Adaptive Motion Planning of an Autonomous Mobile Robot

Kazuo Sugihara* John Smith

Department of Information and Computer Sciences
University of Hawaii at Manoa, Honolulu, HI 96822
Email: sugihara@hawaii.edu and jksmith@hawaii.edu

Abstract

This paper proposes genetic algorithms (GAs) for path planning and trajectory planning of an autonomous mobile robot. Our GA-based approach has an advantage of adaptivity such that the GAs work even if an environment is time-varying or unknown. Therefore, it is suitable for both off-line and on-line motion planning. We first presents a GA for path planning in a 2D terrain. Simulation results on the performance and adaptivity of the GA on randomly generated terrains are presented. Then, we discuss extensions of the GA for solving both path planning and trajectory planning simultaneously.

1 Introduction

Motion planning [3] is one of the important tasks in intelligent control of an autonomous mobile robot. It is often decomposed into path planning and trajectory planning, although they are not independent of each other. *Path planning* is to generate a collision-free path in an environment with obstacles and optimize it with respect to some criterion. An algorithm for path planning is said to be *off-line* if the environment is a known static terrain and it generates a path in advance. It is said to be *on-line* if it is capable of producing a new path in response to environmental changes. *Trajectory planning* is to schedule the movement of a mobile robot along the planned path.

There have been a number of methods proposed for motion planning of a mobile robot [3]. However, few algorithms have been developed for on-line motion planning in a time-varying or unknown terrain. In this paper, we address *adaptive* motion planning which can modify the existing path whenever an environmental change occurs.

*This research was supported in part by NOAA / Sea Grant R/ES-4 of U. S. Department of Commerce.

Recently, applications of *genetic algorithms* (*GAs* for short) [1] to path planning or trajectory planning have been recognized [4, 8]. GA is a search strategy using a mechanism analogous to evolution of life in nature. It has widely been recognized that GA works even for complex problems such that traditional algorithms cannot find a satisfactory solution within a reasonable amount of time [2].

However, the previously proposed GAs have drawbacks and could not fully exploit benefits and ability of GAs. First, a coding which maps solutions to symbolic representations often employs a variable-length string such as a sequence of line segments in path planning [4]. Since the string length of an optimal solution is not known, the configuration (e.g., genetic operators) of a GA must be designed in an *ad hoc* manner to guarantee generation of strings of arbitrary length. Second, the GAs solve either path planning or trajectory planning, but not both.

This paper proposes GAs for motion planning. Our GA-based approach has an advantage of adaptivity such that the GAs work even if an environment is time-varying or unknown. Therefore, it is suitable for both off-line and on-line motion planning. We first presents a GA for path planning in a 2D terrain with obstacles of two kinds: A solid obstacle and a hazardous obstacle which allows a path to intersect it at the expense of some cost. A coding scheme used in the GA represents a path with a binary string of fixed-length. The performance and adaptivity of the GA are evaluated by simulation. Then, we discuss extensions of the GA for motion planning for solving both path planning and trajectory planning simultaneously.

2 Preliminaries

Assume that path planning is considered in a rectangular terrain and a path between two locations is approximated with a sequence of adjacent cells in the

0-8186-8138-1/97 $10.00

grid corresponding to the terrain. Suppose that the unit of distance is the size of a cell. The length $\ell(i,j)$ from cell i to its adjacent cell j is defined as the Euclid distance from the center ρ_i of one cell to the center ρ_j of another times $(w(\rho_i,t_i)+w(\rho_j,t_j))/2$, where $w(\rho,t)$ denote the weight of the location ρ at time t and a mobile robot passes ρ_i and ρ_j at time t_i and t_j, respectively.[1] Note that any way of storing map data such as *quadtrees* can be used as long as we can efficiently retrieve information for a given location (e.g., whether there exists an obstacle at the location).

Path Planning Problem

Input: (1) m by n grid, (2) start cell s in the grid, (3) destination cell d in the grid, and (4) obstacles (a collection of cells in the grid) of two types *solid* and *hazardous* such that a path cannot intersect solid obstacles at all while hazardous obstacles allow a path to intersect them at the expense of a longer path length for each cell in proportion to the obstacles' *weights* representing various cost factors

Output: A path from s to d such that *the total length of the weighted path is minimum*, subject to the constraint that *the path does not intersect any solid obstacle*

For simplicity, we assume that $m = n$ and s and d are corner cells located diagonally to each other.

In 2D space, a path is said to be *monotone with respect to* x-coordinate (*x-monotone* for short) if no lines parallel to y-axis cross the path at two distinct points, i.e., the projection of the path on x-axis is nondecreasing. When there is no confusion, we simply say *monotone*.

A *genetic algorithm* (*GA*) [1] for an optimization problem consists of two major components. First, GA maintains a *population* of *individuals*, where each individual corresponds to a candidate solution and the population is a collection of such potential solutions. In GA, an individual is commonly represented by a binary string. The mapping between solutions and binary strings is called a *coding*. The number of individuals in a population is called the *population size*. Second, GA repeatedly transforms the population by using a mechanism analogous to biological evolution. The mechanism includes the following steps: (1) *Fitness evaluation:* The fitness corresponding to an optimization criterion (i.e., objective function) is calculated for each individual. (2) *Selection:* Some individuals are chosen from the current population as *parents*, based on their fitness values. (3) *Recombination:* New individuals (called *offspring*) are produced from the parents by applying *genetic operators* such as *crossover* and *mutation*. (4)*Replacement:* Some individuals (not necessarily parents in general) are replaced by some offspring.

The population produced at each transformation is called a *generation*. By giving highly fit individuals more opportunities to reproduce, the population becomes likely to include "good" individuals throughout generations.

[1] Since the weight may be time-dependent (e.g., the weight strongly depends on the tide and current in a shallow water), a straightforward application of the Dijkstra's shortest path algorithm does not work in this context.

3 Path Planning

First, we need to choose a coding which maps a path from start s to destination d into a binary string. Although we can represent an arbitrary path by using a binary string of "variable length", it is much more desirable to use fixed-length strings. Thus, we assume that a path from s to d is either x-monotone or y-monotone (but not necessarily both). Obviously, not all paths are monotone. Hence this assumption theoretically implies the possibility of excluding an optimum path in a terrain with extremely crowded obstacles like a maze. As shown in examples below, however, monotone paths are reasonably powerful to express complicated paths.

An x-monotone (or y-monotone) path can be represented by a column-wise (or row-wise) sequence of $n-1$ pairs of direction and distance such that each pair corresponds to each column (or each row, respectively). Thus, the path can be encoded into a binary string whose length is proportional to n and fixed.

The first bit α indicates that a path is x-monotone if $\alpha = 0$; and it is y-monotone if $\alpha = 1$. A block of $3+\lceil \lg(n+1) \rceil$ bits represents direction and distance on each column or row. The first 2 bits of a block denote the direction, e.g., 00 (vertical), 01 (upper diagonal), 10 (horizontal), and 11 (lower diagonal) in case of $\alpha = 0$; and 00 (horizontal), 01 (left diagonal), 10 (vertical), and 11 (right diagonal) in case of $\alpha = 1$. The other bits of the block denote the distance as a signed integer if the direction is 00; otherwise they are ignored. Thus, the amount of space needed to store a population is $p(1+(n-1)(3+\lceil \lg(n+1) \rceil)) = O(pn \log n)$ bits, where p is the population size.

To interpret binary strings as paths and evaluate their fitness values, we use the following convention which produces more "valid" paths in generations and hence improves the performance of our GA. If a block of a binary string denotes the direction and distance that make the corresponding path go beyond the en-

tire grid at a boundary cell, the rest of the path is regarded as straight line segments along the boundary from the boundary cell to the destination cell Note that we leave the binary string as it is, since the uninterpreted substring may become valid and useful later if the substring is inherited to new generations by genetic operators.

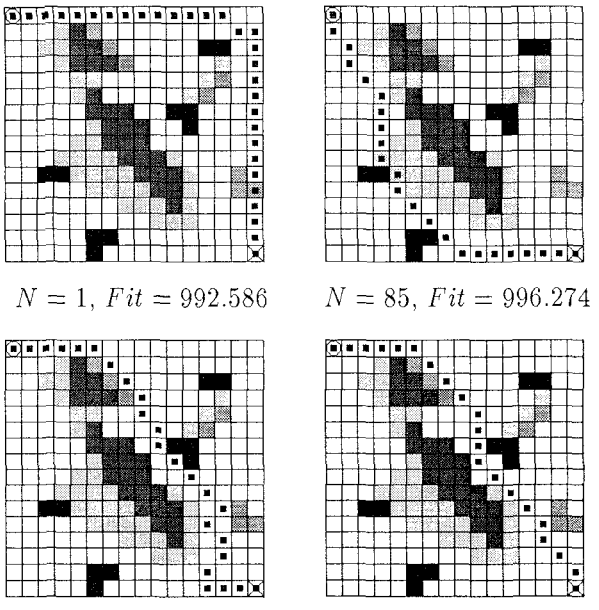
This coding has two advantages. First, as mentioned above, binary strings are of the same fixed length. If binary strings are of variable length, we need to carefully customize genetic operators such as crossover and mutation. For example, the operators must be able to generate strings of any length from an arbitrarily given initial population. Second, straightforward applications of genetic operators to strings always produce syntactically valid strings. This simplifies a process of recombination and makes the entire GA efficient. Therefore, with our coding, a *simple GA* [1] works well without any special operator.

We have implemented the GA for path planning by using the *GA Toolkit* [5] which we developed for design and prototyping of GAs on WWW.[2] For convenience, we changed minimization of a path into maximization by defining the fitness function as $(1 + w_{max})n^2 -$ (the length of a weighted path) if the path does not intersect any solid obstacle and otherwise 1, where w_{max} is the maximum weight.

Next, we show example runs of the GA, where $n = 16$, the population size $p = 30$, *crossover rate* $\gamma = 0.8$ that is the ratio of individuals involved in the crossover over the entire population, and *mutation rate* $\mu = 0.01$ that is the probability of bit alternation per bit on a string. Note that the start cell s is depicted by a circle at the upper left corner, the destination cell d is depicted by X at the lower right corner, N denotes the generation number, and Fit denotes a fitness function value. Figure 1 shows how a path was evolved during an example run, where black cells denote solid obstacles and three kinds of shaded cells denote hazardous obstacles with weights (i.e., extra length) 1, 2 and 3 in increasing order of their darkness. Note that there are at least 6 local optima within 0.4% fitness difference. The GA examined some of the local optima, which are completely different from each other, and eventually found an optimum path.

One of the important advantages of our GA approach is adaptivity. A path may need to be changed when a mobile robot detects an unknown obstacle or inaccuracy of the map information used for generation of the path. GA is a dynamic system in the sense that

[2] The GA Toolkit is available at the following URL. `http://www.ics.hawaii.edu/~sugihara/research/ga-updates.html` It requires a Java-enabled Web browser such as Netscape 3.0.



$N = 1$, $Fit = 992.586$ $N = 85$, $Fit = 996.274$

$N = 520$, $Fit = 996.446$ $N = 645$, $Fit = 999.86$

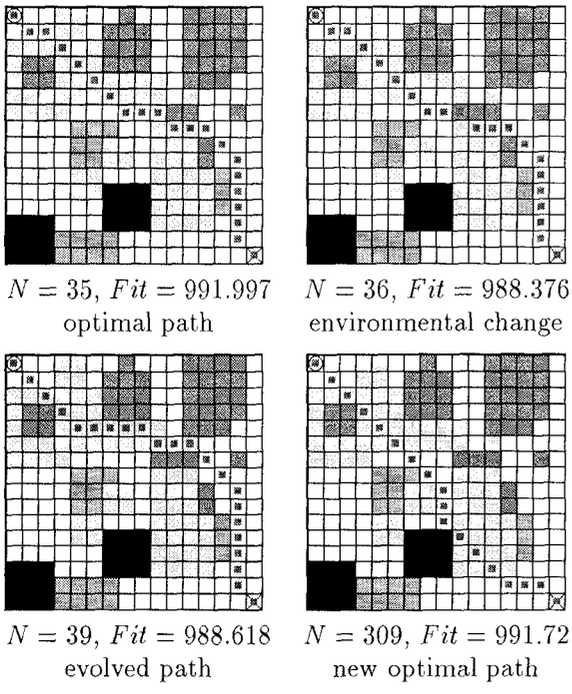
Figure 1: An example run of the path planning GA.

it starts changing a population once an environment changes and tries to adapt for the change. In other words, GA works even if input data is changed on fly. Therefore, it is suitable for not only off-line path planning but also on-line path planning.

Figure 2 shows an example for the adaptivity of the GA. Suppose that the current path is the one (which is optimal) shown in the top left of Figure 2. Note that there are several suboptimal paths with fitness values very close to the optimal fitness (at most 0.04% difference) and they are quite different from each other. Then, a mobile robot detects an unknown hazardous obstacle on the path (see the top right of Figure 2). Insertion of the new obstacle on the previously optimal path creates a new optimal path which was one of the suboptimal paths. In the example run, the GA adapted for this environmental change as shown in the bottom left and right of Figure 2.

4 Simulation Results

We have conducted simulation study [6] of the proposed GA by using the GA Toolkit [5]. The GA was first tuned by preliminary simulation on the terrain in Figure 1 and two more terrains of similar sizes. The simulation examined all combinations of 3 selection operators, 3 genetic operators, 1 mutation operator, and various parameter values for each operator with respect to several performance measures [6] such as



$N = 35$, $Fit = 991.997$ optimal path

$N = 36$, $Fit = 988.376$ environmental change

$N = 39$, $Fit = 988.618$ evolved path

$N = 309$, $Fit = 991.72$ new optimal path

Figure 2: A run on a terrain with 25 obstacles.

the *average fitness value* $\bar{f}$ and *likelihood of optimality* $L_{opt}(k)$ which is the estimated probability of finding an optimal solution within k generations.

The preliminary simulation with 100 runs for each combination suggested the following tuning [6]: A sequence of roulette tournament selection, 1-point crossover and multi-point mutation; population size $p = 30$; mutation rate $\mu = 0.04 \sim 0.05$; crossover rate $\gamma = 0.8$; and *win rate* $\omega = 0.95 \sim 0.8$ which is the probability that a better individual wins at each binary tournament in the roulette tournament selection. With this simulation, we could identify p, μ and ω as primary factors and narrow down the range of each parameter's value for fine tuning.

For the fine tuning, we observed 1,000 runs on the terrain in Figure 1 for each combination of μ and ω, where $p = 30$, $\gamma = 0.8$ and the number of generations $k = 1000$. Simulation results are summarized in Table 1. Note that the 95% confidence interval for 1,000 runs is about 0.03. Thus, it is reasonable to expect $L_{opt}(1000) \geq 0.45$. This implies that the GA can find an optimal solution with high probability if it is executed repetitively, e.g., 95% for 5 runs of 1,000 generations and 99.7% for 10 runs of 1,000 generations.

In order to evaluate the average performance of our GA over various terrains (rather than a few terrains), we observed simulation of the GA with $\mu = 0.04$ and $\omega = 0.95$ on 25 terrains for 100 runs each. The 25 terrains were randomly generated as follows. We consid-

Table 1: $L_{opt}(1000)$ [%] for combinations of μ and ω.

$\mu \setminus \omega$	0.95	0.9	0.85	0.8	0.75
0.04	40.3	45.3	47.2	47.5	45.6
0.045	43.5	48.0	48.8	48.2	44.6
0.05	45.6	44.6	45.0	42.6	35.4

Table 2: Performance on randomly generated terrains.

No. obstacles	$L_{opt}(1000)$ [%]	$\bar{f}/f_{opt}$ [%]	$\bar{N}_{opt}$
10	79.4	99.999	125
15	44.2	99.813	212
20	22.2	99.821	378
25	29.0	99.796	295
30	20.4	99.797	439
overall	39.0	99.843	231

ered 5 levels of the terrain complexity in terms of the number of obstacles: 10, 15, 20, 25 and 30 obstacles. Each obstacle is generated by choosing its location, size and weight randomly. The weight ranges between 1 and 4 (we consider 4 as a solid obstacle) uniformly. The size ranges between 1 and 5 uniformly for up to 20 obstacles. For 25 or 30 obstacles, the size ranges between 1 and 4. This is because a simple estimate of obstacles' occupancy rate exceeds 100% (i.e., too crowded in a terrain) if the size ranges between 1 and 5, although the actual occupancy rate may be a little less than the estimate due to overlapping of obstacles.

A summary of the performance evaluation is presented in Table 2, where f_{opt} is the optimal fitness and $\bar{N}_{opt}$ is the average number of generations to reach an optimal solution. It is generally consistent with the intuition that the higher occupancy rate, the more difficult to solve. Since the standard deviation of the number of generations to reach an optimal path was less than 300, the *cut-off generation* 1,000 in each run seems to be large enough. The results suggest that even for the most complex terrains with a nearly 90% expected occupancy rate of obstacles, the GA likely finds an optimal path with $1 - (1 - 0.204)^{10} \simeq 90\%$ probability in 10 runs of 1,000 generations each.

Next, we present simulation results on the adaptivity of the GA. The adaptivity is defined as how quickly the GA accommodates an environmental change. In this paper, we measure it by the number of generations taken to find a new optimal path after insertion or deletion of an obstacle. The GA was executed on 5 terrains which were chosen, one each of the 5 levels of terrain complexity, from the terrains randomly generated for the above performance evaluation. We ob-

Table 3: Adaptivity of the GA.

No. obstacles	ave. N_1	*std* of N_1	ave. N_2	*std* of N_2
10	3.7	2.61	18.1	12.34
15	57.6	43.36	88.1	61.94
20	16.1	16.50	23.8	30.54
25	356.4	287.84	261.6	234.56
30	4.4	5.43	18.4	19.46
overall	87.6	188.29	82.0	144.31

served 10 runs on each terrain in the following way. In each run, when the GA has found an optimal path on a terrain, a single-cell obstacle with weight 3 is added to a terrain so that it obstructs the optimal path in the middle. Once the GA finds a new optimal path, the GA continues its execution for about 20 generations in the new environment. Then, the inserted obstacle is now removed. The simulation run ends when the GA rediscovers the optimal path of the previous terrain.

Simulation results in total 50 runs are summarized in Table 3, where N_1 is the number of generations taken to find a new optimal path, N_2 is the number of generations taken to rediscover the previously optimal path after deletion of a new obstacle, and "*std*" stands for standard deviation. Note that the number N_1 of generations to find an optimal path in a new environment is likely much smaller than the number of generations to find it from scratch (see Table 2).

In addition, we observed that an alternative path (not necessarily optimal though) was produced in the next generation after an environmental change. This may be because the GA maintains the diversity of a population to some extent. Even if an optimal path becomes infeasible, another path in the population can be picked up. In many cases, a new optimal path to be found shares similar features with some paths in the current population and hence it is quickly produced by the GA. Of course, this is not always the case. If an environmental change drastically changes an optimal path and the current population does not cover features of a new optimal path, finding a new path is basically same as finding it from scratch.

Finally, we briefly mention about running time of the GA. So far, we have evaluated the computational time in terms of the number of generations, since it is independent of computational environments and convenient to compare different GAs for the same problem. The simulation that we have presented in this paper was conducted with a Java applet running on Netscape 3.0. The applet of the GA runs at the speed about 0.1 second per generation on a PC (166MHz).

5 Extensions to Motion Planning

The GA for path planning can be generalized for motion planning by considering a grid with 3 dimensions. The third dimension z is regarded as time in trajectory planning. For simplicity, we assume that a mobile robot does not have a particular orientation, e.g., the robot has a circular shape and can move in any direction.

Motion Planning Problem

Input: Obstacles in an n by n by ∞ grid

Output: A path between cells at (0,0,0) and $(n-1, n-1, T)$ for some integer T such that the total length of the weighted path in the 3D grid is minimum, subject to the constraints that (a) the path does not intersect any solid obstacle and (b) the path meets all the limitations of a robot's maneuverability [3]

In case of a static terrain, the input can be simplified to an n by n grid as in the path planning problem defined in Section 2.

A path in the 3D grid is said to be *xy-monotone* if no lines parallel to z-axis cross the path at two distinct points. Similarly, xz-monotone and yz-monotone are defined. A projection of a path on x-y plane is called *xy-projection* of the path. Similarly, *xz-projection* and *yz-projection* are defined. Note that xz-projection and yz-projection are always z-monotone in the motion planning problem, since the z-coordinate corresponds to time.

In order to solve the motion planning problem, we consider 3 possible coding schemes. The first coding scheme assumes that (1) a path is xy-monotone and (2) xy-projection of the path is x-monotone and y-monotone. Then, a path in the 3D grid can be represented by a pair of paths in a 2D grid. One is xz-projection and another is yz-projection. Note that these assumptions also imply that xz-projection is x-monotone and yz-projection is y-monotone.

The motion planning problem is now viewed as finding a pair of two 2D paths which minimizes the objective function. Thus, the coding for the 2D path planning problem can be generalized as follows. Let σ_1 be an x-monotone representation constructed from xz-projection by using the coding, except that the first bit is no longer needed, directions for decreasing time z are not allowed, and the distance value is always nonnegative. Similarly, σ_2 is an y-monotone representa-

[3] The limitations may include the maximum velocity and the maximum acceleration. If we take into account geometric features of a mobile robot and rotation of the robot, additional constraints on the maneuverability such as the minimum turning radius should be considered.

tion constructed from yz-projection. By interleaving the two strings σ_1 and σ_2 bit by bit, we obtain σ that is a binary representation of a path in the 3D grid. The reason for interleaving is that crossover can transform both xz-projection and yz-projection simultaneously.

This generalization preserves the advantage of the coding for 2D path planning, i.e., *fixed-length* coding. The only difference from the coding for 2D path planning is that we have to resolve potential inconsistency between σ_1 and σ_2. It is possible that the total time t_1 in σ_1 is less than the total time t_2 in σ_2 (or vice versa). In such a case, we use the following convention to interpret the interleaved string σ. After time t_1, a path stays on the same x-coordinate value.

The second coding scheme assumes only the first assumption (1) and removes (2). Then, a path can be represented by a triple of xy-projection, xz-projection and yz-projection. Each projection is mapped to a binary string by using the coding for 2D path planning. Note that the binary string representing xz-projection or yz-projection is of variable length, depending on the last z value on the projection. Then, the three strings are interleaved bit by bit after shorter paths are padded with dummy bits. Note that inconsistency must be resolved among three projections. In addition, the length of a binary string in this coding scheme is variable, since the duration time to be achieved in trajectory planning is unknown. Once we have to manipulate strings of variable length in a GA, a simple GA does not work and we need to carefully customize genetic operators for the coding scheme. The design of a GA with a variable-length coding is usually ad hoc and complicated.

The third coding scheme removes both the above assumptions (1) and (2), but assumes that the size of a cell in z-coordinate is chosen to be smaller than or equal to the time $\delta = d/(2V)$, where V is the maximum velocity and d is the size of a cell in x-coordinate or y-coordinate. There is no restriction on the shape of a path. Then, a path cannot cross two or more cells on the same z-coordinate value and hence has only 9 directions. Thus, a path can be represented by a sequence of directions. Although this coding scheme is very simple, yet powerful enough to express arbitrary paths in the 3D grid, it is a variable-length coding.

Therefore, the first coding scheme should be chosen if the assumptions (1) and (2) are acceptable.

It is also possible to extend the GA further to motion planning in 3D space. We are currently investigating a coding suitable for 3D path planning and designing a GA for motion planning of an *autonomous underwater vehicle* [7].

6 Conclusion

This paper proposed genetic algorithms (GAs) for path planning and trajectory planning of an autonomous mobile robot. We first presented a GA for 2D path planning and evaluated its performance by simulation. Our coding represents a path with a binary string of fixed-length. This is a key to the good performance. Simulation results were also presented to show how adaptive the GA is. Then, we discussed extensions of the path planning GA to GAs which solve both path planning and trajectory planning simultaneously. Since the GAs are adaptive so that they work even if an environment is time-varying or unknown, the GAs are suitable for both off-line and on-line motion planning.

References

[1] David Beasley, David R. Bull and Ralph R. Martin, "*An overview of genetic algorithms*," University Computing, Vol. 15, No. 2 and 4, pp.58–69 and 170–181, 1993.

[2] David E. Goldberg, "Genetic and evolutionary algorithms come of age," *Commun. ACM*, Vol. 37, No. 3, pp.113–119, March 1994.

[3] Young K. Hwang and Narendra Ahuja, "Gross motion planning — A survey," *ACM Compt. Surveys*, Vol. 24, No. 3, pp.219–291, Sept. 1992.

[4] Hoi-Shan Lin, Jiang Xiao and Zbigniew Michalewicz, "Evolutionary algorithm for path planning in mobile robot environment," *Proc. 1st IEEE Conf. on Evolutionary Computation*, Vol. I, Orland, FL, 1994, pp.211–216.

[5] John Smith and Kazuo Sugihara, "GA Toolkit on the Web," *Proc. 1st Online Workshop on Soft Computing*, Aug. 1996, pp. 93–98.

[6] Kazuo Sugihara, "Measures for performance evaluation of genetic algorithms," *Proc. 3rd Joint Conference on Information Sciences (JCIS '97)*, Vol. I, March 1997, pp.172–175.

[7] Kazuo Sugihara and Junku Yuh, "GA-based motion planning for underwater robotic vehicles," *UUST-10*, Durham, NH, Sept. 1997, to appear.

[8] H. Y. Xu and G. Vukovich, "Fuzzy evolutionary algorithms and automatic robot trajectory generation," *Proc. 1st IEEE Conf. on Evolutionary Computation*, Orland, FL, 1994, pp.595–600.