GENETIC AND EVOLUTIONARY METHODS FOR MOBILE ROBOT MOTION CONTROL AND PATH PLANNING

Abdollah Homaifar, Edward Tunstel, Gerry Dozier, and Darryl Battle

20.1 INTRODUCTION

A variety of evolutionary algorithms, operating according to Darwinian concepts, have been proposed to solve problems of common engineering applications. Applications often involve automatic learning of nonlinear mappings that govern the behavior of control systems, as well as parallel search strategies for solving multiobjective optimization problems. In many cases, hybrid applications of soft computing methods have proven to be effective in designing intelligent control systems. This chapter presents two instances of such hybrid applications to problems of mobile robot control. In particular, evolutionary computation and fuzzy logic are combined to solve robot motion control and path planning problems. The first part of the chapter describes a methodology for applying genetic programming (GP) to design a fuzzy logic steering controller for mobile robot path tracking. Genetic programming is employed to learn the rules and membership functions of the fuzzy logic controller, and also to handle selection of fuzzy set intersection operators (t-norms). The second part of the chapter describes an application of fuzzy logic to enhance the performance of an evolutionary robot path planning system. In this case, fuzzy logic is employed in the selection phase of the simulated evolution process.

20.2 GENETIC PROGRAMMING FOR PATH TRACKING CONTROL

In applications of genetic and evolutionary methods, the data structures of individuals being evolved are different depending upon the specific type of evolutionary algorithm employed. Genetic programming is a method of program induction introduced by Koza [1]. It has been demonstrated to be useful as an approach to learning fuzzy logic rules for mobile robot control and navigation [2, 3]. It has also proven useful for the classical cart centering control problem [4]. The data structures undergoing adaptation in GP are not as hierarchically formed programs of a given host programming language. In the host language, individuals are represented as parse trees, which dynamically change size and structure during simulated evolution.

The set of possible structures produced by GP is primarily based on the set of all possible valid compositions that can be constructed from the set of \( n \) problem dependent functions defined in a function set, \( F = \{ f_1, f_2, \ldots, f_n \} \), and the set of \( m \)
terminals (function arguments, variables, and/or constants) defined in a terminal set, $T = \{ t_1, t_2, ..., t_m \}$.

In order to appreciate the utilization of GP for the design of fuzzy logic controllers, we introduce the steering control problem next and follow it by discussing some of the important implementation issues to be considered for the application of GP.

20.2.1 Path Tracking Formulation

The first of two control problems examined in this chapter is a path tracking problem, which was formulated by Hemami et al. [5, 6] for a class of low speed (less than 2 m/s) tricycle-model vehicles. Essentially, the control objective is to successfully navigate a mobile robot or automated guided vehicle along a desired path in a two-dimensional environment. We wish to automatically design a multiple input, single output fuzzy controller that will achieve this objective. The inputs consist of a measurable position error, $\varepsilon_d$, and a measurable orientation error, $\varepsilon_\theta$, associated with path following in the plane (see Figure 20.1). The output is the steering angle, $\delta$, which is the corrective control action that would cause the errors to approach zero and, thus, force the robot to follow the desired path. The position error is taken as the deviation of the center of gravity, $C$, or any other desired point of the robot from the nearest point on the path. The orientation error is the angular deviation of the robot from the tangent of the desired path.

![Figure 20.1: Tracking Control and Error Variables.](image)

Hemami et al. derived a state-space kinematic model for this robot where the state vector comprised the pose errors described. The resulting kinematic model is repeated herein for clarity in the discussion that follows. The reader is referred to reference [5] or [6] for details of the derivation, which culminates in the following:

$$
\begin{bmatrix}
\dot{\varepsilon}_d \\
\dot{\varepsilon}_\theta
\end{bmatrix} =
\begin{bmatrix}
0 & V_u \\
0 & 0
\end{bmatrix}
\begin{bmatrix}
\varepsilon_d \\
\varepsilon_\theta
\end{bmatrix} +
\begin{bmatrix}
MC / MP \\
1 / MP
\end{bmatrix} V_u \tan \delta \pm \frac{\dot{\eta}_d}{\eta_0} $$

(20.1)

where $V_u$ is forward linear velocity of the robot, and $\dot{\eta}_d$ and $\dot{\eta}_\theta$ are rates of change of the effects of path curvature. In Hemami et al. [6] it is concluded...
(based on dynamic analysis of the same vehicle) that for small steering angle, $\delta$
($\tan \delta = \delta$), Equation (20.1) approximates the slow dynamics of the vehicle
when its forward velocity is low. In the simulations presented later, we have
simplified the robot kinematic model by taking this small steering angle
approximation into account. Furthermore, we apply the controller to straight
line path following and, therefore, neglect the model effects of path curvature.
Such a simplification does not preclude autonomous tracking of reasonably
complicated paths since multisection paths can be defined to be piecewise
linear.

To allow for control of the mobile robot, some means of measuring the input
information is needed to feed into the system in order to generate a desired
output. Thus the system under control is assumed to have some suitable sensory
apparatus. For our implementation, we assume that the robot has dead-
reckoning/odometry sensors that provide access to the error states at all times, or
permit calculations thereof. This sensory input data is then mapped to control
outputs according to the desired control policy. In path following simulations,
the position and orientation errors in Equation (20.1) are updated using the
fourth-order Runge-Kutta method, which is widely used in computer solutions
to differential equations [7].

20.2.2 GP Solution

The path tracker to be learned by GP is a two input, single output fuzzy
controller that will map the error states into a proper steering angle at each time
step. A population of candidate solutions is created from which a solution will
emerge. The allowance for rule bases of various sizes enhances the diversity of
the population. That is, the GP system creates individuals in the initial
population that each have possibly different numbers of rules within a finite
range (15-30) specified before a run. In the process of learning fuzzy control
rules and membership functions, GP manipulates the linguistic variables directly
associated with the controller. Given a desired motion behavior, the search
space is contained in the set of all possible rule bases that can be composed
recursively from a set of functions and a set of terminals. The function set
consists of membership function definitions (describing controller inputs),
components of the generic fuzzy if-then rule, and common fuzzy logic
connectives. More specifically, these include functions for fuzzy sets, rule
antecedents and consequents, fuzzy set intersection and union, and fuzzy
inference. The terminal set is made up of the input and output linguistic
variables and the corresponding membership functions associated with the
problem.

Selection of appropriate t-norms is automated, thereby giving the GP system
greater control of the evolutionary design. That is, the influence of GP is
extended to include selection of the type of t-norm employed to compute the
conjunction of fuzzy propositions in the antecedent of a rule. The two most
commonly used t-norms for fuzzy control are Mamdani's $\min$ and Larsen's
product [8]. T-norms for each conjunctive rule are selected at random by GP for
rule bases in the initial population, and are carried along based on fitness in successive generations.

To achieve the goal of evolving membership functions and rules for FLCs, the GP system must conform to strong syntactic constraints when breeding individuals. Special rules of construction were introduced in Tunstel and Jamshidi [2] and later extended using algorithms described in references [9] and [10]. We refer the reader to Homaifar et al. [11] for a detailed description of the resulting syntactic rules, the full design algorithm, and other GP implementation issues related to fuzzy controller design.

20.2.2.1 Controller Fitness Evaluation

Each rule base in the current population is evaluated to determine its fitness value for steering the robot from initial locations near the desired path to final locations on the path such that steady state and final pose errors are minimized. This evaluation involves frequent simulation of the robot's motion from each of a finite number of initial conditions until either the goal state is achieved or the allotted time expires. The initial conditions are referred to as fitness cases in the GP community. For this problem we use eight different initial conditions, which is a logical choice given the pair-wise symmetry of the possible error categories illustrated in Figure 20.2. Consider error category (d), which represents a case where the robot is located on the left of the desired path with a negative heading orientation. There also exists a symmetric case where the robot is located on the right of the desired path with a positive heading orientation. These symmetric cases are each represented by error category (d). The same holds for categories (a), (b) and (c) illustrated in the figure, yielding a total of eight fitness cases that fully describe the possible combinations of errors with respect to the path.

The fitness function is a measure of performance used to rank each individual relative to others in the population. We compute path tracking performance by summing the Euclidean norms (normalized) of the final error states plus the average control effort \( \bar{\delta} \) over all eight fitness cases. Thus, the following fitness function drives the evolution process

\[
\text{Raw Fitness} = \delta \sum_{j=1}^{8} \sqrt{\epsilon_{d,j}^2 + \epsilon_{\theta,j}^2 + \bar{\delta}^2},
\]

where \( \epsilon_{d,j} \) and \( \epsilon_{\theta,j} \) are the position error and orientation error existing at the end of each fitness case simulation. The objective of this fitness function is to minimize final path tracking errors as well as the control effort expended. As such, a perfect fitness score is zero and, in general, lower fitness values are associated with better controllers.

Simulations show that adding average control effort as part of the path tracking metric significantly reduces undesired steering oscillations. Fitness functions based solely on final error states sometimes yielded impractical controllers that exhibited rapid oscillations in the steering control signal, which would cause damage to the steering mechanism of a real mobile robot.

The path tracking success of an individual in the population is also based on its ability to minimize the error states to within the following specified
tolerances, $|\varepsilon_{d}| < 0.15\text{m}$ and $|\varepsilon_{\theta}| < 0.26\text{ radians}$, for each fitness case. A fitness case simulation in which these tolerances are satisfied is considered a hit, or successful trial. Thus, each individual has the potential of receiving a total of eight hits during fitness evaluation for this path tracking problem.

<table>
<thead>
<tr>
<th>tangent to path</th>
<th>tangent to path</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) $\varepsilon_{d} = 0$, $\varepsilon_{\theta} &lt; 0$</td>
<td>(b) $\varepsilon_{d} &lt; 0$, $\varepsilon_{\theta} = 0$</td>
</tr>
<tr>
<td>(c) $\varepsilon_{d} &lt; 0$, $\varepsilon_{\theta} &lt; 0$</td>
<td>(d) $\varepsilon_{d} &gt; 0$, $\varepsilon_{\theta} &lt; 0$</td>
</tr>
</tbody>
</table>

Figure 20.2. Error Categories for Path Tracking Control Problem.

20.3 PATH TRACKING SIMULATION RESULT

In this section, we present representative results of simulated path tracking performance for an evolved controller. Results are presented, in particular, for a fuzzy controller designed with t-norms selected randomly during co-evolution of rules and membership functions. Selection of appropriate t-norms is one of several design decisions that could lengthen the manual trial-and-error procedure typically used by FLC designers. We elect to automate this decision and thereby give the GP system greater control of the evolutionary design. To achieve this, the GP system is allowed to choose at random between the two common t-norms mentioned above. T-norms for each conjunctive rule are selected at random by GP for rule bases in the initial population, and are thus carried along based on fitness in successive generations.

The simulated robot is based on Hemami's kinematic model with dimensions taken from the Heathkit Hero-1 mobile robot. The Hero-1 has a tricycle wheel configuration in which the front wheel is driven by a DC motor and steered by a stepper motor. Its two rear wheels are passive. Dimensions employed are 0.3 m for the wheelbase, and 0.2 m for the offset from the rear axle to the front wheel. These dimensions correspond to the constant lengths $2d$ and $MP$ of Figure 20.1, respectively. All simulations were conducted assuming a controller sampling-rate of 20 Hz and run for a maximum of 10 seconds. In each case, the robot travels at a constant nominal forward speed of 1.5 m/s unless otherwise stated.

All GP runs for the path tracking problem were executed on a 260 MHz MIPS DECstation using a restructured version of the simple genetic programming in C (SGPC) system [12]. Five consecutive runs (initialized using different random number generator seeds) were executed using the GP control
parameters listed in Table 20.1. About one hour of computation time is required for a run of this magnitude.

**Table 20.1: GP Control Parameters.**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>200</td>
</tr>
<tr>
<td>Number of generations</td>
<td>50</td>
</tr>
<tr>
<td>Mutation probability</td>
<td>0.001</td>
</tr>
<tr>
<td>Crossover probability</td>
<td>0.600</td>
</tr>
<tr>
<td>Maximum mutation depth</td>
<td>4</td>
</tr>
<tr>
<td>Reproduction probability</td>
<td>0.399</td>
</tr>
<tr>
<td>Maximum new tree depth</td>
<td>5</td>
</tr>
<tr>
<td>Maximum depth after crossover</td>
<td>7</td>
</tr>
</tbody>
</table>

A rule base of 25 rules emerged as the fittest among all five runs. This rule base used five conjunctive rules, three employing the Mamdani t-norm and two employing the Larsen t-norm. The evolved input membership functions associated with the best rule base are shown in the left half of Figure 20.3.

**Figure 20.3: Co-Evolved Input Membership Functions and Fixed Output Membership Functions.**

Co-evolved rules are listed in Table 20.2, where the notations NB, NS, Z, PS, and PB represent fuzzy linguistic terms of “negative big,” “negative small,” “zero,” “positive small,” and “positive big,” respectively. Terms describing the inputs, $\varepsilon_d$ and $\varepsilon_b$, are preceded with the prefixes “p” and “o,” respectively. The fixed output membership functions are shown in the right half of Figure 20.3, where the linguistic terms are labeled without prefixes.
Table 20.2: Best Evolved Rule Base.

<table>
<thead>
<tr>
<th></th>
<th>Rule Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IF oZ THEN NS</td>
</tr>
<tr>
<td>2</td>
<td>IF pPB THEN Z</td>
</tr>
<tr>
<td>3</td>
<td>IF pNB THEN Z</td>
</tr>
<tr>
<td>4</td>
<td>IF pPS THEN NB</td>
</tr>
<tr>
<td>5</td>
<td>IF pNS and oPS THEN NS (Mamdani’s min)</td>
</tr>
<tr>
<td>6</td>
<td>IF pNB THEN PB</td>
</tr>
<tr>
<td>7</td>
<td>IF oNS THEN Z</td>
</tr>
<tr>
<td>8</td>
<td>IF oNB THEN PS</td>
</tr>
<tr>
<td>9</td>
<td>IF pNS THEN NS</td>
</tr>
<tr>
<td>10</td>
<td>IF pNS and oZ THEN PB (Larsen’s prod)</td>
</tr>
<tr>
<td>11</td>
<td>IF oPB THEN NB</td>
</tr>
<tr>
<td>12</td>
<td>IF pNS and oPB THEN NB (Larsen’s prod)</td>
</tr>
<tr>
<td>13</td>
<td>IF pPS THEN NS</td>
</tr>
<tr>
<td>14</td>
<td>IF oNS THEN PB</td>
</tr>
<tr>
<td>15</td>
<td>IF pPB THEN NB</td>
</tr>
<tr>
<td>16</td>
<td>IF oZ THEN PS</td>
</tr>
<tr>
<td>17</td>
<td>IF oNB THEN PB</td>
</tr>
<tr>
<td>18</td>
<td>IF pNS and oNS THEN NB (Mamdani’s min)</td>
</tr>
<tr>
<td>19</td>
<td>IF pNS THEN Z</td>
</tr>
<tr>
<td>20</td>
<td>IF oPS THEN NB</td>
</tr>
<tr>
<td>21</td>
<td>IF pZ THEN PS</td>
</tr>
<tr>
<td>22</td>
<td>IF pPB and oZ THEN Z (Mamdani’s min)</td>
</tr>
<tr>
<td>23</td>
<td>IF pPB THEN PS</td>
</tr>
<tr>
<td>24</td>
<td>IF oPS THEN PS</td>
</tr>
<tr>
<td>25</td>
<td>IF oNS THEN PS</td>
</tr>
</tbody>
</table>

Figure 20.4: Co-evolved FLC Path Tracking Performance.

The evolved controller received a raw fitness of 0.1091 with 8 hits. In Tunstel and Jamshidi [2], an FLC designed manually, through a lengthy process of trial and error, is presented which also used 25 rules. Hours of iterative refinement of membership functions and rules were invested before arriving at a
suitable design. In comparison, the hand derived FLC received a comparable raw fitness (0.08 with 8 hits) for the identical tracking problem. Figure 20.4 shows the temporal responses of position error, orientation error, and control effort for the evolved controller and for the hand derived controller. This result corresponds to error category (d) of Figure 20.2, with initial conditions of $e_d = 0.8 \text{ m}$ and $e_\theta = -0.9 \text{ rad}$. As was shown in Hemami et al.[6], this error category is the most general for studying path tracking by tricycle-type vehicles, in the sense that corrective vehicle steering from states in other error categories ultimately leads to vehicle error status in category (d) or its counterpart. The evolved controller achieved comparable response characteristics to those of the hand derived controller using an equivalent number of rules.

20.3.1 Evolved Controller Robustness

Given the capability to evolve FLCs that can effectively follow paths, an important next step is to examine their robustness to practical perturbations. To test the noise robustness of the evolved controller, simulations were performed with the imposition of a noise signal upon the sensor measurement related to heading (orientation). We assume that the error states are derived from sensor measurements which, due to their imperfect nature, introduce an additive sinusoidal noise signature of small amplitude and low frequency (relative to the controller sampling frequency) that corrupts the orientation error. For this investigation we impose the sensor noise signal, $n(t) = 0.15\cos(3t)$ with $t = kT$, where $k = 1, 2, 3, \ldots$ is the sampling instant, and $T$ is the sampling period. Thus, the noise amplitude is bounded by 0.15 radians ($10^\circ$), and at any sampling instant the corrupted orientation error signal lies in the range of $(e_\theta \pm 0.15)$ radians.

In addition to the additive noise, we also increased the constant nominal forward speed of the robot by 20%, which resulted in a simulated speed of 1.8 m/s. A typical result is shown in Figure 20.5, which illustrates the performance of both the evolved controller and the hand derived controller when induced with noise and an increased vehicle speed. While the oscillatory effects of the added noise are clearly evident in the steady state response, the controller successfully navigates the robot onto the path and maintains the steady state errors within the tolerances specified earlier. Thus, this evolved fuzzy controller exhibits path tracking robustness to the imposed perturbations. This result is representative of temporal responses for each of the remaining fitness cases. In simulations completed thus far, the most robust fuzzy controllers were those evolved when GP was allowed to randomly select t-norms.

The performance assessment of the evolved controller with regard to robustness is based upon the assumption that low frequency oscillations within the control signal of amplitude less than 0.026 radians ($1.5^\circ$) are practical. In light of this assumption, the results indicate that the evolved FLC was able to navigate the robot along the desired path with the imposed perturbation of sensor noise and the increase in the robot’s nominal speed.
Figure 20.5: Co-evolved FLC Response to Sensor Noise and Increased Forward Speed.

20.4 EVOLUTIONARY PATH PLANNING

Thus far, we have discussed genetic programming techniques for solving a mobile robot tracking problem. We now move on to a related application of evolutionary methods for mobile robot motion planning. Mobile robots that are capable of tracking paths can be used effectively in mapped environments where specific paths from location to location can be designated. If autonomy is desired, the robot software should be capable of using map-based information to plan suitable paths in the operating environment. In the very least, a facility for offline path planning should be available to generate suitable paths for the robot. What constitutes a suitable path depends on the specific features of the application and robot functionality. A single objective, or multiple objectives, may be imposed to define suitable paths for a given application. Many path planning systems consider shortest paths as the primary criterion. However, the shortest path may not always be the most efficient. For example, path smoothness could be of considerable importance. In general, the path planning problem can often be posed as a multiobjective optimization problem. Depending on the nature of the objectives to be met for suitable paths, the formulation of an effective closed-form, multiobjective function to be optimized could be quite difficult. As a way to circumvent such difficulties, a multiobjective selection method has been developed for use with an evolutionary path planning system.

The remainder of this chapter presents an approach to path planning that employs evolutionary methods to find suitable paths in a robot’s operating environment. The main attributes and evolutionary mechanisms of the path planner are described. In addition, a technique for enhancing path planning performance using fuzzy logic in the evolutionary process is presented.
20.4.1 Evolutionary Path Planning System

In this section, we present the salient attributes of an evolutionary path planning system called GEPOA (global evolutionary planning and obstacle avoidance system), which has been applied to robot planning problems [13]. GEPOA uses steady state reproduction, flat crossover [14] with Gaussian mutation, and uniform mutation in an effort to develop feasible paths. In each generation of path evolution, two parents are selected using tournament selection with a tournament size of two. If the first parent selected represents an infeasible path, it is repaired 50 percent of the time using a method called visibility-based repair (VBR), described below. If the first parent selected is feasible then the two parents create one offspring, which replaces the worst individual in the population. The following attributes of GEPOA will be briefly described: environment and path representation, visibility-based repair, path evaluation and selection functions, and evolutionary operators.

20.4.1.1 Environment and Path Representation

An obstacle within the robot’s environment is represented as a set of intersecting line segments. Each line segment connects two distinct vertices. Associated with each vertex within the environment is a value, which represents the number of obstacles that contain it. This value is referred to as the containment value (CV) of a vertex. If a vertex lies along the boundary of an environment its CV is assigned a value of infinity. Figure 20.6 provides an example of how obstacles are represented in GEPOA. Notice that the four-sided obstacle (Obstacle 1) is represented by only two lines in GEPOA.

An individual in the evolving population of candidate paths (CPs) contains four fields. The first field is a chromosome, which contains a gene corresponding to the Cartesian coordinates of each node of the path (where nodes of a path are connected by a straight-line segment). The second field is called the seed. The seed of an individual is the gene that will be crossed or mutated to created an offspring. Initially, an individual will have only three genes: the start gene, the seed gene, and the destination gene. Repair genes are inserted into the chromosome by the VBR algorithm each time a straight-line segment of an individual is found to pass through an obstacle. The third field is a value referred to as the violation distance. The violation distance represents the Euclidean distance of the CP, which cuts through one or more obstacles. The fourth field records the Euclidean distance of the path from the start to destination genes.

20.4.1.2 Visibility-Based Repair of Candidate Paths

VBR facilitates construction of valid paths through free space and is performed as follows. When an obstacle, $o$, lies along a straight-line segment between two nodes $P$ and $Q$, each line of $o$ is checked to see if it is intersected by $PQ$. If a line of $o$ is intersected by $PQ$, then a repair node is created using the following set of rules:

1. if the CVs of a line's vertices are both equal to one, then the repair node
is selected to be a point along an extension (the distance outside an obstacle at which a repair node is placed is a user-specified parameter);
2. if the CVs of a line's vertices are different, then the repair node is selected to be a point just outside the vertex which has the lower CV;
3. if the CVs of a line's vertices are greater than one and equal, then the repair node is selected to be a point just outside of the vertex which is farther from the point of intersection.

![Figure 20.6: Obstacle Representation in GEPOA.](image)

![Figure 20.7: Visibility Based Repair of Paths.](image)
Figure 20.7 shows an example of how VBR can be used to transform an infeasible path into one that is feasible. In Figure 20.7a, an infeasible path $XPT$ is shown. The path $XPT$ is infeasible because the line segment $XP$ passes through Obstacle 1 and the line segment $PT$ passes through Obstacle 3. Before proceeding further, notice that each vertex in the environment shown in Figure 20.7a has a CV of one.

Using VBR, the line segment $XP$ can be repaired to $XAP$. Since $XP$ intersects Line 1 of Obstacle 1, a repair node corresponding to a point just outside either $v(1,1)$ or $v(1,3)$ must be selected. By applying Rule 1, Node $A$, which corresponds to a point just outside vertex $v(1,1)$, is selected as the repair node. Similarly, the line segment $PT$ can be repaired to $PBCT$. Again Rule 1 must be applied to Line 1 and Line 2 of Obstacle 3. The repair node that results from the intersection of $PT$ and Line 1 is Node $B$. The repair node that results from the intersection of $PT$ and Line 2 is Node $C$. Figure 20.7b shows the result of using VBR on $XPT$. The repaired, feasible version of $XPT$ is $XAPBCT$.

Given a candidate path, the VBR algorithm used by GEPOA works as follows. Each obstacle within the environment is checked with each straight line segment from the start gene to the destination gene of the candidate path until a segment is found that passes through the obstacle. The infeasible segment is repaired via VBR and the process is repeated using the next obstacle. As an example of how this repair algorithm works, notice once again Figure 20.7. When given the path $XPT$ the algorithm works as follows. Obstacle 1 is checked to see if it is violated by segment $XP$. Since it is, a repair gene (Node $A$) is generated and Obstacle 2 is then considered. Obstacle 2 is checked to see if it is “cut” by segment $XA$. Since it is not cut by segment $XA$, Obstacle 2 is checked with segment $AP$ then segment $PT$. Since there are no more segments to inspect, Obstacle 3 is considered. Obstacle 3 is checked to see if it is cut by segments $XA$ and $AP$. Finally, Obstacle 3 is checked to see if it is cut by $PT$. Since it is, two repair genes are generated (Nodes $B$ and $C$) and the algorithm terminates.

Since this repair algorithm considers an obstacle only once, it is possible for a repair gene to be generated that creates a line segment that cuts through a previously considered obstacle. Therefore, a candidate path may need to be repaired by this VBR algorithm more than once.

20.4.1.3. Path Evaluation, Selection, and Evolutionary Operators

The evaluation function computes the Euclidean distance of each straight line segment of the path that an individual represents as well as the violation distance. GEPOA uses a modified version of tournament selection, with a tournament size of two, to select individuals to become parents. The selection process is as follows. Two individuals are randomly selected from the current population. If the violation distances of the two are different, then the individual with the smaller violation distance is selected to be a parent. If the violation distances are the same then the individual with the smaller overall distance is selected to be a parent.
GEPOA uses two operators along with VBR to create and/or refine individuals. The two operators are as follows: (1) a version of Radcliffe’s flat crossover [14], which we refer to as seed crossover and (2) a version of uniform mutation, which we refer to as uniform seed mutation. Seed crossover proceeds as follows. Given two seed genes $s_1 = (x_1, y_1)$ and $s_2 = (x_2, y_2)$, a seed gene for an offspring is created as follows:

$$s_{off} = (rnd(x_1, x_2) + N(0, 4.0) \cdot rnd(y_1, y_2) + N(0, 4.0))$$

(20.3)

where $rnd$ is a uniform random number generator and $N(0, 4.0)$ is a Gaussian random number with zero mean and a standard deviation of 4.0. The resulting offspring has a chromosome containing three genes: a gene corresponding to the node representing the current position of $R(X)$, the seed node, and the destination node. The offspring then undergoes VBR and may have additional repair genes added by the VBR algorithm. In uniform seed mutation, either the $x$ or $y$ coordinate of a parent is mutated using uniform mutation to create a seed gene for an offspring. A resultant offspring created by seed mutation is similar to one created by seed crossover in that it also has a chromosome containing three genes. Once again the offspring undergoes VBR and may have additional repair genes added by the VBR algorithm.

### 20.5 PATH EVOLUTION WITH FUZZY SELECTION

During evolution of candidate paths the selection of the parent paths that undergo reproduction is based on several objective criteria. This section describes a tournament selection procedure that employs fuzzy logical inference to enhance the performance of the GEPOA system. The fuzzy tournament selection algorithm (FTSA) selects CPs to be parents and undergo reproduction based on:

1. the Euclidean distance of a path from the origin to its destination,
2. the sum of the changes in the slope of a path,
3. the average change in the slope of a path.

As such, the overall objective of the FTSA is to allow evolutionary path planners to evolve CPs that feature minimal distances from start to destination, minimal sums of the changes in slope (SCS), and minimal average changes in slope (ACS). Given two candidate paths ($CP_1$ and $CP_2$) that are randomly chosen from the current population, the FTSA takes six inputs – the path distances, the SCS, and the ACS. It returns one output in the continuous interval $[-1, 1]$, which corresponds to the CP that should be selected to be a parent. Any output less than zero indicates that $CP_1$ is to be selected, while any output greater than zero indicates that $CP_2$ is to be selected.

#### 20.5.1 Fuzzy Inference System

Let $(d_1, s_1, a_1)$ and $(d_2, s_2, a_2)$ denote the distance, SCS, and ACS for $CP_1$ and $CP_2$, respectively. These six inputs are converted into three derived parameters,
$d$, $s$, and $a$, whose computed values lie in [-1,1] according to the following expressions:

$$\begin{align*}
\begin{cases}
d = \frac{d_1 - d_2}{d_1 + d_2} \\
s = \frac{s_1 - s_2}{s_1 + s_2} \\
a = \frac{a_1 - a_2}{a_1 + a_2}
\end{cases}
\end{align*}$$

(20.4)

Note that for values of $d$, $s$, and $a$, which are less than zero, the more desirable attribute belongs to $CP_1$ and vice versa for $CP_2$. Each of the derived inputs has a domain partitioned by three fuzzy subsets defined using overlapping membership functions. Figure 20.8 shows the three membership functions, where $x$ is $d$, $s$, or $a$. If the value of $x$ is nonpositive then it is a member of the fuzzy set “less than,” $LT$, which represents the set of all tuples $(x_1, x_2)$ such that $x_1 < x_2$. Similarly, values of $x$ that are nonnegative are members of the fuzzy set “greater than,” $GT$, and represent the set of tuples $(x_1, x_2)$ for which $x_1 > x_2$. All values of $|x| < X$ are members of the fuzzy set “equal,” $EQ$, representing the set of all tuples $(x_1, x_2)$ for which $x_1$ and $x_2$ are approximately equal. By varying the value of $X$, the FTSA has the ability to focus on optimizing a particular objective. In the sequel, $X$ is $D$, $S$, or $A$.

The fuzzy rules are formulated as listed below. For each of the seven rules, $P$ represents the singleton consequent of the rule. If the consequent of a rule is $(P = -1)$, then the rule has specified that $CP_1$ should be selected to be a parent. Similarly if a rule's consequent is $(P = 1)$ then it has specified that $CP_2$ should be selected. The defuzzification method used is the Mean of Maxima.

- IF $d$ is $LT$ THEN $P = -1$
- IF $d$ is $EQ(D)$ AND $s$ is $LT$ THEN $P = -1$
- IF $d$ is $EQ(D)$ AND $s$ is $EQ(S)$ AND $a$ is $LT$ THEN $P = -1$
- IF $d$ is $EQ(D)$ AND $s$ is $EQ(S)$ AND $a$ is $EQ(A)$ THEN $P = 0$
- IF $d$ is $EQ(D)$ AND $s$ is $EQ(S)$ AND $a$ is $GT$ THEN $P = 1$
- IF $d$ is $EQ(D)$ AND $s$ is $GT$ THEN $P = 1$
- IF $d$ is $GT$ THEN $P = 1$

![Figure 20.8. FTSA Membership Functions for $LT$, $EQ(X)$, and $GT$.](image-url)
20.5.2 Experimental Example

Here, we present an illustrative example of the utility of the FTSA for enhancing the GEPOA path planning system described above. For fuzzy tournament selection to be effectively used in this type of system, it must be able to adequately rank individuals of a population. Hereafter, let GEPOA+FTS denote GEPOA with fuzzy tournament selection.

Using a hypothetical environment consisting of obstacles distributed throughout enclosed free space, we compared paths evolved by GEPOA and GEPOA+FTS. The parameters for each of these algorithms were as follows: the population size was 20, the flat crossover rate with Gaussian mutation (standard deviation = 4.0) was 0.66, and the uniform mutation rate was 0.34. After the initial population was created, both algorithms were allowed to run for 500 generations, thus, creating a total of 520 individuals. For GEPOA+FTS, we set \( D = 0.15, S = 0.15, \) and \( A = 0.15. \)

Figure 20.9: Path Population of GEPOA: Generation 0 (left); after Generation 500 (right).

Figure 20.10: Path Population of GEPOA+FTS: Generation 0 (left); after Generation 500 (right).
The left halves of Figures 20.9 and 20.10 show the initial populations that were randomly generated by GEPOA and GEPOA+FTS, respectively. Since GEPOA and GEPOA+FTS use a visibility based algorithm to repair infeasible paths, it is not uncommon for feasible (but suboptimal) paths to appear in the initial population.

The right half of Figure 20.9 shows the population of paths developed by GEPOA after 500 steady state generations. Notice that GEPOA has converged upon the two equal and shortest paths; however, these paths are quite rugged. By contrast, the right half of Figure 20.10 shows the population of paths developed by GEPOA+FTS after 500 steady state generations. First of all notice that GEPOA+FTS has converged upon a number of good paths. Notice also that among the paths evolved with GEPOA+FTS, the shortest path is still represented. The fact that it is infeasible is not a major concern because it has a chance of being repaired! Not only does fuzzy tournament selection allow evolutionary search to converge upon the best path, but it also allows for a great deal of valuable and much needed diversity.

20.6 SUMMARY AND CONCLUSIONS

GP was successfully applied to discover FLCs capable of navigating a mobile robot to track straight-line paths in the plane. The overall performance of the best evolved rule bases was comparable to that of a manually designed rule base that utilized more rules in most instances. Instances of simultaneous evolution of membership functions and rules showed that GP was capable of evolving a FLC that demonstrated satisfactory responsiveness to various initial conditions while utilizing minimal human interface. Suboptimal solutions with respect to the employed fitness function were consistently found, demonstrating that GP performs well as a global adaptive search method. Further automatic improvement towards optimal solutions can be made by synthesizing a hybrid between GP and a localized search method such as hill climbing [15, 16].

GP was also applied to larger population sizes facilitated by the dramatic speed increase of our coding implementation in C vs. the previously investigated LISP implementation. The 82% increase in speed of evolution alone serves as a strong basis for practical application of GP in the controller design process. The approach provides a means for expeditious design of FLCs that can be directly applied to a physical system. Alternatively, human experts can use the rapidly evolved FLCs as design starting points for further manual refinement [4]. To assess the practicality of the GP solutions, robustness characteristics of evolved FLCs were examined. The controllers evolved with random selection of fuzzy t-norms were particularly robust when subject to imposed perturbations of sensor noise and an increase in nominal robot speed. The results support the notion that a genetically evolved fuzzy logic controller can have practical utility.

Fundamental features of the GP system include manipulation of linguistic variables directly associated with the fuzzy system (as opposed to numerical encoding/decoding), a syntactic structure that provides context preservation via
structure preserving genetic operators, and provision for evolving rule bases of various sizes in a single population. These features were inherited from our previous implementation in LISP. Beyond these, the implementation proposed herein provides several improvements and extensions that make GP a more powerful tool for FLC design. Namely, we have dramatically improved upon the required speed of evolution and extended the system to handle full FLC design, including evolution of the most appropriate t-norms for the controlled system.

In the second part of the chapter we presented an evolutionary algorithmic approach to robot path planning. It serves as an effective means of circumventing the difficulties associated with formulating complex multiobjective functions for suitable paths. A multiobjective selection method based on fuzzy logic was applied with an evolutionary path planning system. The fuzzy tournament selection algorithm can be used for multiobjective path planning by almost any evolutionary based motion planning system. Despite the simple nature of the fuzzy inference system employed, the FTSA exhibits complex behavior. The approach allows evolutionary search to converge upon a diversity of optimal and/or near optimal paths. The availability of alternative feasible paths is important in the event that a local navigation system cannot traverse a particular global path. This can happen, for example, when unfavorable conditions are sensed locally, replanning becomes necessary, or task constraints intervene.

ACKNOWLEDGMENTS

This work is partially funded by grants from NASA Autonomous Control Engineering Center (ACE) at North Carolina A&T SU under grant # NAG2-1196 and NASA Dryden Flight Research Center under grant # NAG4-131. The authors wish to thank the ACE Center and NASA Dryden for their financial support. A portion of the research described in this chapter was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

REFERENCES