International Journal of Innovative Computing, Information and Control
Volume **9**, Number **5**, May **2013**

ICIC International ©2013 ISSN 1349-4198
pp. 2185–2202

A GENETIC-BASED OPTIMIZED FUZZY-TABU CONTROLLER FOR MOBILE ROBOT RANDOMIZED NAVIGATION IN UNKNOWN ENVIRONMENT

Weria Khaksar[1], Tang Sai Hong[1], Mansoor Khaksar[2] and Omid Reza Esmaili Motlagh[3]

[1]Department of Mechanical and Manufacturing Engineering
Faculty of Engineering
University Putra Malaysia
43400 UPM, Serdang, Selangor, Malaysia
gs22153@mutiara.upm.edu.my; saihong@eng.upm.edu.my

[2]Department of Industrial Management
Faculty of Human Sciences
Islamic Azad University of Sanandaj
Sanandaj, Iran
Khaksar_mansoor@yahoo.com

[3]Department of Robotic and Automation
Faculty of Manufacturing Engineering
Universiti Teknikal Malaysia Melaka (UTeM)
Hang Tuah Jaya, 76100 Durian Tunggal, Melaka, Malaysia
omid.utem@gmail.com

Received December 2011; revised April 2012

Abstract. *Although sampling-based path planning algorithms have been shown robust performances in many cases, they still suffer from many drawbacks. Inefficiency in online/sensor-based planning purposes, generating far-from-optimum paths, high running time, failure in difficult environments and generating unstable variant results in different runs are some of the existing problems with sampling-based methods. We propose an online sampling-based path planning algorithm which employs a fuzzy-tabu controller to evaluate the generated samples and select better ones in order to improve the performance of the planner. Furthermore, a genetic algorithm-based optimization framework is developed to optimize the parameters of the controller. The genetic optimizer takes place before the planner starts any planning task and attempts to optimize the fuzzy-tabu controller parameters and also the parameters of the planner. We simulate the proposed algorithm to analyze its performance and to compare it with existing sampling-based algorithms. The proposed algorithm generates relatively short paths in less than few seconds while results variations between different runs are considerably low.*

Keywords: Online path planning, Sampling-based, Fuzzy-tabu controller, Genetic optimizer, Result variation

1. Introduction.

Path planning for a mobile robot is a procedure to move the robot from an initial position to a goal configuration inside an environment filled by arbitrary shaped obstacles, while avoiding any collision with them. Canny [1] proved that the path planning problem is NP-Complete. In most of the path planning applications, there is no prior information about the environment, e.g., positions of the obstacles and surrounding boundaries. This class of path planning problems is called sensor-based or online path planning. There are a variety of researches in this field resulting in different approaches, each with their specific characteristics, advantages and drawbacks [2-8]. In this class of path planning, the motion decisions are made as the robot moves and obtain new information from the environment.

Sampling-based approaches are a well-known class in the field of robot motion planning with important benefits and applications. In this type of algorithms, the planner generates collision free positions to capture the workspace connectivity and then, connects them in order to guide the robot from the start to the goal. Although, these approaches are not complete [9], they follow a weaker but still interesting form of completeness called Probabilistic Completeness which guarantees if a feasible path exists, the planner will eventually find it [10]. Sampling-based motion planning methods can be generally divided into two main groups namely, roadmap-based and tree-based approaches. Roadmap-based algorithms generate collision-free samples inside the configuration space and connect some of them together in order to capture the connectivity of the environment. Once the roadmap was created, the planner is ready to get the positions of the start and the goal configurations and find a collision-free path between them by means of a graph search technique, e.g., A* algorithm [11]. These algorithms are typically used as multi-query planner which can solve the motion planning problem for any given start and goal configurations. A well-known roadmap-based planner is probabilistic roadmap, PRM [12]. PRM approach generates a number of collision-free samples inside the workspace and connects each sample to its N closest neighbors by a straight line. Afterward, by adding the start and goal configurations to the graph, it will be used for any given query. Most of the other roadmap-based approaches are based on PRM. The Randomized Bridge Builder, RBB [13], attempts to find two close samples inside the obstacles configurations and add their midpoint to the roadmap if it belongs to the collision-free space. The Gaussian sampler [14] generates a sample inside the obstacles area and uses a Gaussian distribution to find a collision-free node close to it. There are many other roadmap-based algorithms which aim to increase the efficiency of the PRM method either by improving the sampling strategy or connection method [15-17].

Tree-based algorithms were developed for quickly solving one particular path planning problem. In this type of algorithms, the roadmap structure is replaced by a tree structure. The tree is normally rooted at the start position and grows by generating random samples and connecting them to one of the tree's nodes. One of the first and most important tree-based structures is Rapidly Exploring Random Tree, RRT [18]. RRT normally generates a collision-free sample and expands the tree in the direction of this sample through its closest node with a fixed step size, and repeats this procedure until the size of the tree reaches a desired amount. Then the constructed tree is used to find a feasible path. RRT approach can be improved by constructing two trees rooted at start and goal positions in order to shorten the convergence time of the algorithm. This extension of the original RRT is called RRT-bidirectional. Several extensions for tree-based planners were developed during the last decade. RRT-Connect [19], builds two RRTs rooted at start and goal positions and uses a heuristic to grow the trees. In the original RRT algorithm, the sampling domain is the Voronoi regions of each existing node of the tree while, DD-RRT [20] algorithm incorporates the Voronoi domain and the visible domain to improve the RRT method for solving Bug Trap problem. This approach defines a visibility radius for each generated node of the tree and generates new random nodes in the vicinity of tree's nodes. In the AD-RRT method [21], this visibility radius is subject to change during the tree construction to update its value and improve the performance of the planner as the number of calls for the collision detector is reduced. A simple tree-based planner was proposed in [22] which approximates connected regions of free space with volumes instead of points. This approach considers balls centered at each node and new samples

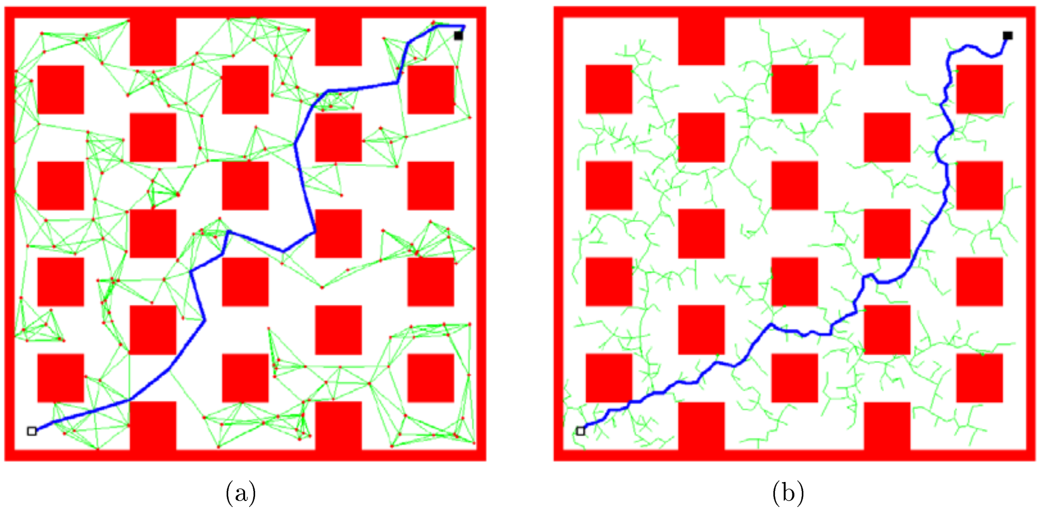


FIGURE 1. The performance of two sampling-based algorithms: (a) probabilistic roadmap (PRM) with 200 nodes and (b) rapidly exploring random trees (RRTs) with step size equal to 0.2. Start and goal configurations are illustrated by white and black squares respectively.

will be generated outside these balls. Figure 1 shows the performance of PRM and RRT algorithms in a chess board environment.

Over the last years, there has been a lot of work in improving sampling-based motion planning algorithms. Although the performance of the sampling-based algorithms are efficient in many cases and these planners have solved different path planning problems, they still have important drawbacks which result in poor performances in some situations.

The followings are some of the important inefficiencies of sampling-based methods.

(1) Sampling-based approaches are mostly designed for offline planning where the environments are completely known for the planner [9]. Hence, they cannot be used in their original form for planning in unknown environments.
(2) Sampling-based algorithms attempt to find a feasible path for a given query and they usually ignore the results optimality. In most of the cases, the resulted paths are far from optimum. There are some postprocessing packages in the literature which are used to shorten the resulted paths but they increase the running time of the planner and also, it is not possible to use a postprocessing method for online planning purposes [10].
(3) The running time of a sampling-based planner strongly depends on the planner parameters including the number of generated samples and the connection strategy. For instance, increasing the number of generated samples provides shorter paths while it increases the runtime of the planner extensively [23].
(4) There are some cases in which the sampling-based algorithms fail to return an answer in a reasonable amount of time. Difficult types of environments including narrow passages, mazes and bug traps are presented in the literature as examples of poor performance of the sampling-based methods.
(5) According to the randomized nature of these approaches, the results are not stable. The length of generated paths and the runtime of the planner vary in different executions of the algorithm.

In this paper we propose a sampling-based path planning algorithm which incorporates the data of the robot's sensory system in the sampling procedure. The proposed algorithm improves the performance of the sampling-based methods in the abovementioned terms. The first drawback is improved by restricting the sampling area to the visible region for the robot and the planner, only considers the points on the vision circle of the robot for sampling. Then, a fuzzy controller is applied which uses the tabu search heuristic rules in order to evaluate the generated samples and shorten the generated path's length and runtime of the planner. Furthermore, a genetic algorithm optimization framework is designed to optimize the parameters of the fuzzy-tabu controller. The optimized controller will improve the performance of the planner in difficult environments and also reduces the variations of the results in different executions of the planner.

Designing a fuzzy logic controller with evolutionary algorithms is an extensively studied field. These studies can be divided into three categories [24] including genetic-fuzzy systems, particle swarm optimization and ant colony optimization. Genetic-fuzzy systems employ genetic algorithm for designing and optimizing a fuzzy system. A relatively general classification of genetic-fuzzy systems was provided in [25] where these approaches were divided into three main groups namely genetic tuning, genetic learning and a hybrid model of both tuning and learning.

In this paper we design a hybrid optimization model which deals with both tuning and learning. This hybrid model takes place before the algorithm starts the path planning. It optimizes the parameters of the membership functions of inputs and output of the fuzzy-tabu controller and also rearranges the rules in order to improve the efficiency of the overall approach. Also, the hybrid genetic optimization model will consider some of the path planning algorithm parameters in the optimization process. The rest of the paper is organized as follows. In Section 2, the performance of the fuzzy-tabu controller is described. The hybrid genetic optimization model is proposed in Section 3. Simulation studies and analysis are illustrated and studied in Section 4 and a conclusion is given in Section 5.

2. Fuzzy-Tabu Controller.

We consider a circular robot which is equipped with a number of range sensors. As illustrated in Figure 2, the sensory system of the robot is able to perform a complete 360° scan of the surrounding area inside the vision range circle (*VRC*) of the sensors and determine free space (S^F) and obstacle's space (S^O). We formulate the vision of the robot based on [9], as follows.

$$V_S(x_C, y_C, \theta) : (\mathbb{R}^2 \times \phi) \to \mathbb{R}, \quad \phi = [0, 2\pi] \tag{1}$$

$$V_S(x_C, y_C, \theta_j) = \begin{cases} \min_\gamma(\gamma \times [\cos(\theta_j), \sin(\theta_j)]^T) \text{ if } \gamma \times [\cos(\theta_j), \sin(\theta_j))]^T \leq VR_{\max} \\ \quad (\text{such that, } [x_C, y_C] + \gamma \times [\cos(\theta_j), \sin(\theta_j)]^T \in S^F) \\ VR_{\max} \qquad \text{otherwise} \end{cases} \tag{2}$$

where $[x_C, y_C]$ is the coordinates of the robot's current configuration, $VR_{\max}$ is the sensor's maximum vision range and γ is a positive scalar. A plot of V_S versus θ is presented in Figures 2(b) and 2(d) and in Figures 2(c) and 2(e) the differential of resulted curves $[\partial(V_S)/\partial(\theta)]$ is plotted to illustrate the visible vertices of obstacles and boundaries of the environment as the sharp peaks of the differential curves.

$$\text{Vertices} = \{x_v, y_v\} \tag{3}$$

$$\text{such that: } \begin{cases} \begin{bmatrix} x_V \\ y_V \end{bmatrix} = \begin{bmatrix} x_C \\ y_C \end{bmatrix} + V_S(x_C, x_C, \theta) \times \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} \\ \partial(V_S)/\partial(\theta) = 0 \end{cases} \tag{4}$$

The sampling area is the visible and collision-free parts of the VRC, using uniform sampling distribution. After the sampler succeeds in finding a collision-free sample, the fuzzy-tabu controller (FTC) takes place to evaluate the generated samples.

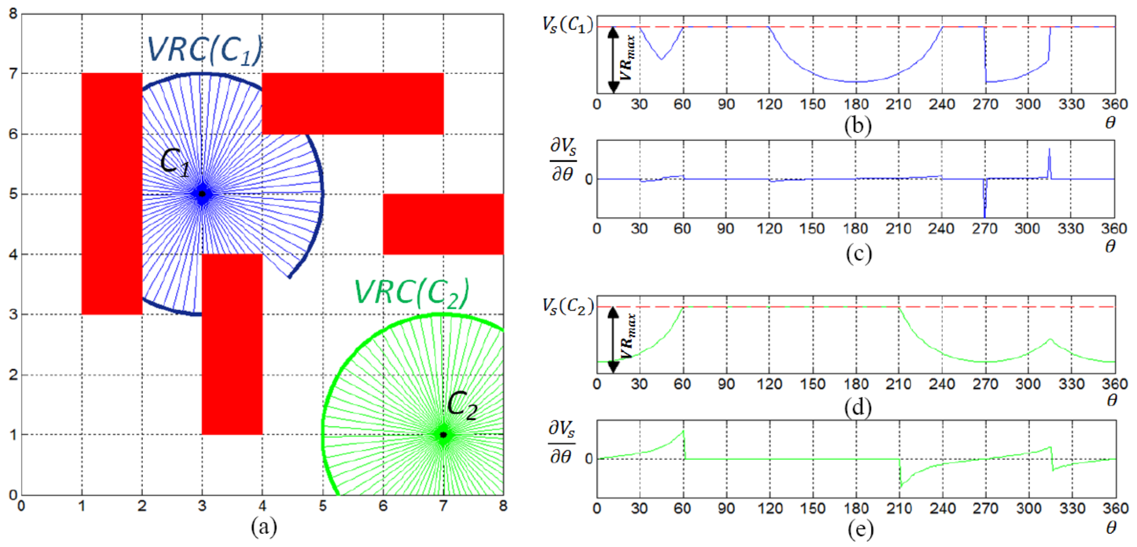


Figure 2. The performance of the robot's sensory system: (a) the vision range in two points C_1 and C_2, (b) and (d) the magnitudes of rays emitted from the robot's at C_1 and C_2 versus the angle respectively, (c) and (e) the differential of the V_S curve represents the visible vertices of the obstacles and workspace boundaries.

FTC employs the structure of tabu search heuristic to guide the search procedure by creating two tabu lists including short tabu list (STL) and long tabu list (LTL). STL includes three fuzzy variables which should be calculated for the generated sample including μ_G^+ which is the normalized subtraction of the sample's distance to the goal and robot's current position distance to the goal; μ_P^+ as the normalized distance between the generated sample and the previous position of the robot and μ_S^+, which is calculated as the normalized subtraction of the sample's distance to the start position and robot's current position distance to the start position. These variables are being calculated for the generated sample as follows. Note that $\mathbb{D}[A, B]$ is the Euclidean distance between points A and B. Figure 3 illustrates the elements of STL.

$$\mu_G^+ = 0.5 + (\mathbb{D}[Sample, Goal] - \mathbb{D}[Current, Goal])/2 \times VR_{\max}, \quad \mu_G^+ \in [0,1] \quad (5)$$

$$\mu_P^+ = \mathbb{D}[Sample, Current]/VR_{\max}, \quad \mu_P^+ \in [0,1] \quad (6)$$

$$\mu_S^+ = 0.5 + (\mathbb{D}[Sample, Start] - \mathbb{D}[Current, Start])/2 \times VR_{\max}, \quad \mu_S^+ \in [0,1] \quad (7)$$

$$STL = \{\mu_G^+, \mu_P^+, \mu_S^+\} \quad (8)$$

The elements of LTL are the last m successfully visited positions of the robot which cannot be visited again. This list is updated during the planning as follows.

$$LTL = \{P_1, P_2, \ldots, P_m\},\ P_j \text{ is the last } j\text{th successfully visited position of the robot.}$$

FTC uses these two lists to evaluate the generated sample. The input of the fuzzy system is the elements of STL and its output will be the risk of the moving the robot to the new sample. Figure 4 presents the membership functions and decision surfaces of the fuzzy system.

The planner uses the input variables to apply three simple rules. First, the robot's distance to the goal needs to be decreased continuously; second, the robot is prohibited to get close to its previous position and third, the robot is forced to get farther from

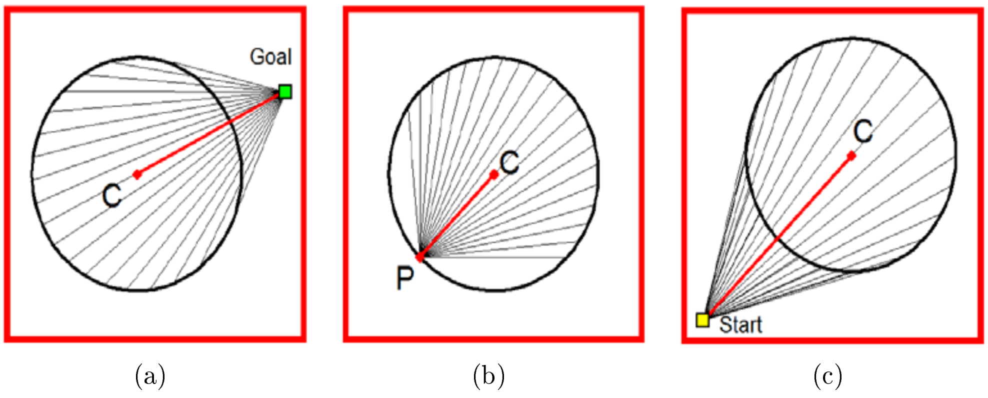


Figure 3. The fuzzy-tabu controller variables when the robot is placed at C: (a) calculating the value of μ_G^+ by measuring the distance of the sample to the goal (black lines), (b) μ_P^+ is calculated by measuring the distance between the sample and the previous position of the robot (black lines), (c) μ_S^+ is formed by measuring the distance of the sample to the start position (black lines).

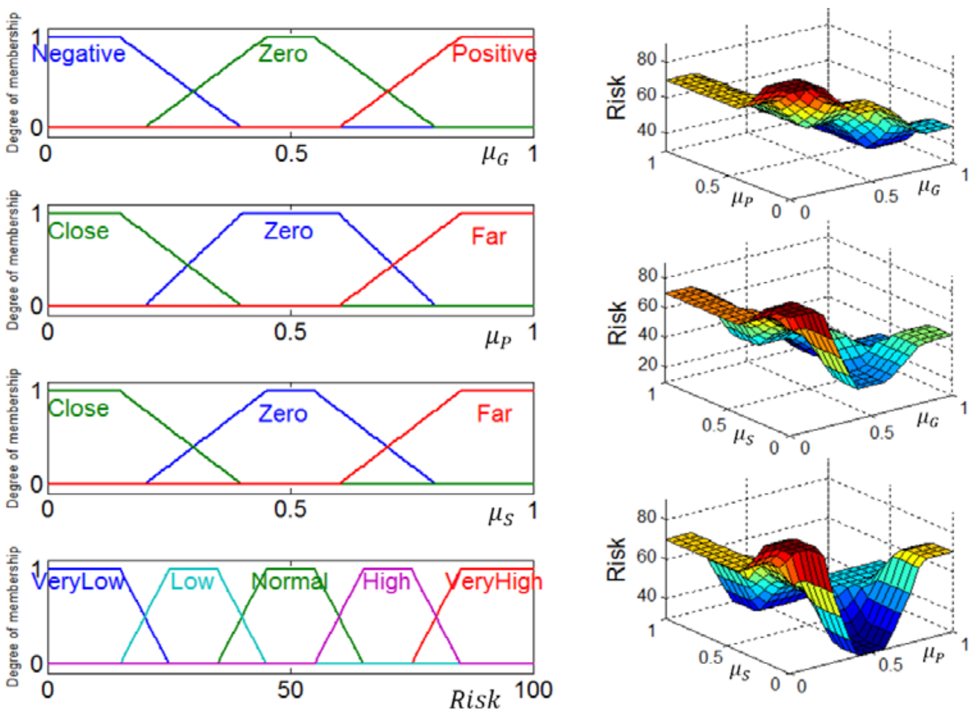


Figure 4. Membership functions and decision surfaces for the fuzzy-tabu controller. All membership functions are trapezoid.

the start position continuously. Using the elements of LTL helps the robot to visit each position once.

After the evaluation, if the output of the fuzzy controller is less than a predefined value $R_{\max}$, and it does not belong to the LTL, then the sample will be considered as the next position of the robot. $R_{\max}$ is the maximum allowed risk for robot's destinations. This phase of the algorithm is based on the local search phase of the conventional tabu search.

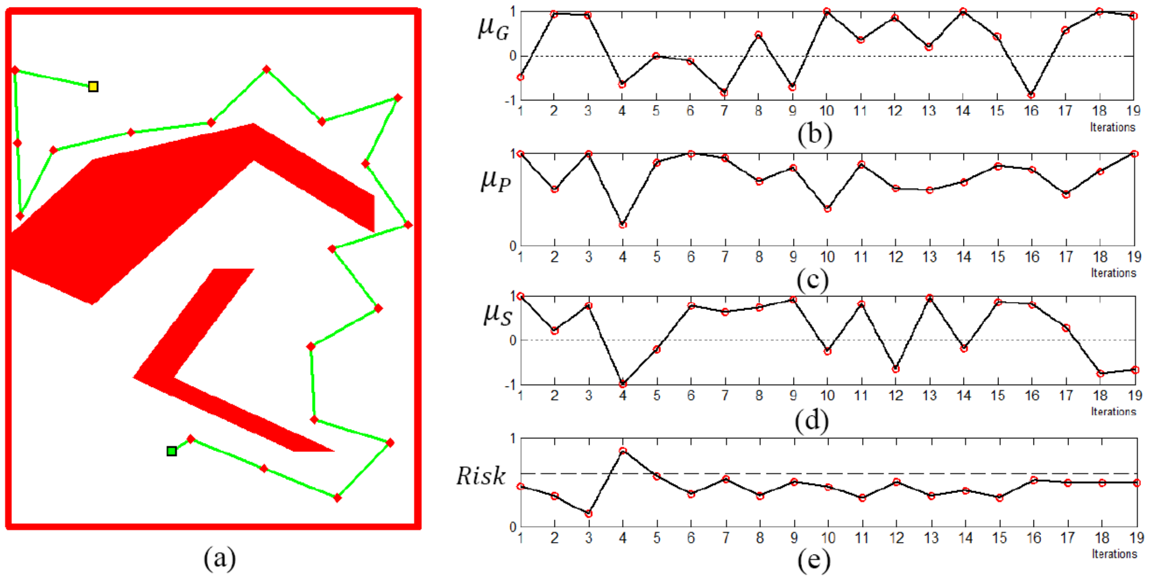


FIGURE 5. The performance of the fuzzy-tabu controller: (a) the resulted path for a given path planning problem, (b)-(e) the changes in decision variables during the planning process

In most of the cases, by the end of this phase, the planner succeeds in finding a new position for the robot. Otherwise, based on the rules of tabu search, the search is intensified and the elements of STL should be discarded one by one in order to expand the search area. In this situation, μ_G^+, μ_P^+, and μ_S^+ components will be ignored respectively, hoping to find a sample with a suitable risk. At the end of this level, all of the STL elements have been discarded and the only restriction on the sampling area is the elements of LTL. Finally, if the algorithm failed to find a sample, then search is diversified and all restriction criteria will be ignored and the algorithm performs similar to the conventional sampling-based methods. Figure 5 shows a path planning problem which has been solved by the proposed algorithm, along with the controller variable plots which show their changes during the path planning.

As presented in Figure 5, only in the fourth iteration, search is diversified as the Risk of the fourth iteration is more than the $R_{\max}$ while in other iterations, the controller succeeded in finding samples with risk below the maximum allowed risk. More detail about using tabu search in sampling-based path planning is presented in [26].

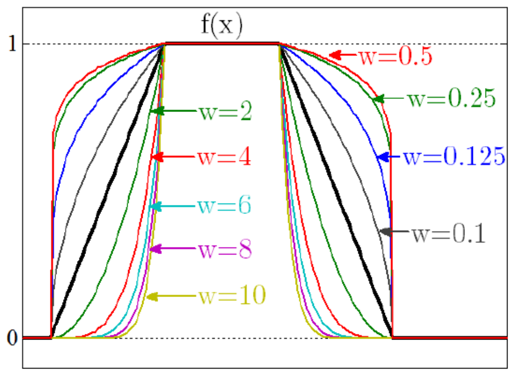
3. Genetic Optimization.

For any fuzzy system, there are two important concepts including interpretability and accuracy. These concepts are inconsistent and attempting to improve one of them may result in worsening the other [27]. Some of the optimization methods focus on improving the interpretability while, some others aim to improve the accuracy of the fuzzy systems. For designing an optimization system which consider both concepts, first, the main objective (interpretability or accuracy) is tackled defining specific model structure to be used and therefore, setting the fuzzy modeling approach and deriving the model. Then, the modeling components are improved to compensate for the initial difference between both concepts. We consider an optimization framework which attempts to improve both interpretability and accuracy. Accuracy is improved by tuning the parameters of trapezoid fuzzy membership functions and also using nonlinear scaling factors to change the shape of the membership functions. For improving the interpretability of the fuzzy model, we design a rule selection process which optimizes the number of rules in the reasoning procedure.

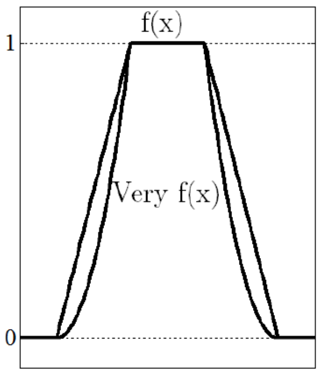
3.1. **Tuning the fuzzy membership functions.** As mentioned before, in the initial fuzzy-tabu controller, we have three input variables and one output, all with trapezoid membership functions. The genetic optimization model considers the parameters of these functions as a part of the optimization variables. Generally, a trapezoid membership function can be defined as follows.

$$\mu_{trap}(x) = \begin{cases} (x-\alpha)/(\beta-\alpha) & \text{if } \alpha \le x \le \beta \\ 1 & \text{if } \beta \le x \le \gamma \\ (\delta - x)/(\delta-\gamma) & \text{if } \gamma \le x \le \delta \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

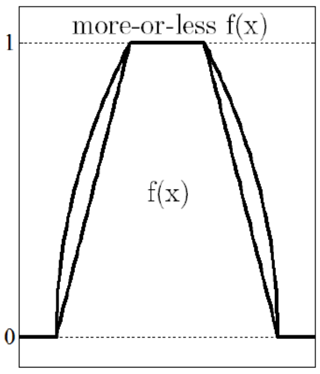
As presented in Figure 4, for each input variable we have eight tuning parameters and for the output, we have fourteen parameters. Hence, we have 28 parameters for tuning. We also use a nonlinear scaling factor (*NSF*) for each membership function to vary its compatibility degree to the fuzzy set by raising the membership function value to a positive scalar. We choose this nonlinear scaling factor from the interval of $(0, 10]$. Figure 6 presents the effect of the *NSF* on a trapezoid membership function.



(a)



(b)



(c)

FIGURE 6. The effect of nonlinear scaling factor (*NSF*) on the fuzzy membership functions: (a) different values for w creates different membership functions, (b) when $w = 2$, we have the '*very*' hedge and (c) for $w = 0.5$, the resulted hedge is called '*more-or-less*'.

As presented in Figure 6, when this factor is equal to 0.5 and 2, we have the *more-or-less* and *very* hedges respectively. Totally we have 14 membership functions for input and output variables and hence, we need 14 variables as the nonlinear scaling factors.

$$\mu_{NSF}(x) = [\mu(x)]^w, \quad w \in [0, 10] \tag{10}$$

$$\mu_{very}(x) = [\mu(x)]^2 \tag{11}$$

$$\mu_{more\text{-}or\text{-}less}(x) = \sqrt{\mu(x)} \tag{12}$$

Totally, for the tuning part of the genetic optimizer, there are fifty four parameters in which the first forty are being used for tuning the positions of the trapezoid membership functions and the last fourteen will change the shapes of them. We need to define some linear constraints for these variables in order to avoid any variable domain error. We define two sets of linear constraints including range constraints and position constraints. Range constraints keep the variables inside the interval of membership functions as follows.

$$\alpha_i \in [0, 1], \quad \text{and } i = 1, 2, \ldots, 40 \tag{13}$$

$$\alpha_i \in [0, 10], \quad \text{and } i = 41, 42, \ldots, 54 \tag{14}$$

Position constraints limit the variables to keep the structure of the resulted membership functions true.

$$\mu_G^+ : \quad \{\alpha_1 \le \alpha_2, \alpha_3 \le \alpha_4, \alpha_4 \le \alpha_5, \alpha_5 \le \alpha_6 \text{ and } \alpha_7 \le \alpha_8\} \tag{15}$$

$$\mu_P^+ : \quad \{\alpha_9 \le \alpha_{10}, \alpha_{11} \le \alpha_{12}, \alpha_{12} \le \alpha_{13}, \alpha_{13} \le \alpha_{14} \text{ and } \alpha_{15} \le \alpha_{16}\} \tag{16}$$

$$\mu_P^+ : \quad \{\alpha_{17} \le \alpha_{18}, \alpha_{19} \le \alpha_{20}, \alpha_{20} \le \alpha_{21}, \alpha_{21} \le \alpha_{22}, \text{ and } \alpha_{23} \le \alpha_{24}\} \tag{17}$$

$$\begin{aligned} Risk: \quad &\{\alpha_{25} \le \alpha_{26}, \alpha_{27} \le \alpha_{28}, \alpha_{28} \le \alpha_{29}, \alpha_{29} \le \alpha_{30}, \alpha_{31} \le \alpha_{32}, \alpha_{32} \le \alpha_{33}, \\ &\alpha_{33} \le \alpha_{34}, \alpha_{35} \le \alpha_{36}, \alpha_{36} \le \alpha_{37}, \alpha_{37} \le \alpha_{38}, \text{ and } \alpha_{39} \le \alpha_{40}\} \end{aligned} \tag{18}$$

Totally, we have 40 range constraints and 26 position constraints in the final genetic optimization model.

3.2. **Rule selection.** Tuning of membership functions usually needs an initial model with a large number of rules to get an appropriate level of accuracy. Generally, to obtain a good number of initial rules, methods ensuring covering levels higher than needed are used. In this way, we could obtain rules that are being needed at first and could be unnecessary once the tuning is applied or rules that could impede the tuning of the remaining ones in order to obtain the global optimum in terms of the accuracy. Thus, we can define the following classes of rules with respect to this global optimum in the complete set of rules. Erroneous rules degrade the system performance (rules that are not included in the most accurate final solution); Redundant or irrelevant rules do not significantly improve the system performance; Complementary rules complement some others slightly improving the system performance; and important rules should not be removed to obtain a reasonable system performance [28].

For improving the interpretability of the fuzzy model, we designed a set of rule selection binary variables as the weight of each fuzzy rule. If the corresponding variable for a rule is equal to zero, then that rule will not be considered in the final fuzzy reasoning system. As described before, our fuzzy controller has 3 input variables and for each one, there are three membership functions which make 27 rules and accordingly 27 rule selection binary variables.

$$\alpha_i \in \{0, 1\}, \text{ and } i = 55, 56, \ldots, 81 \tag{19}$$

Based on the fitness functions which will be described later, the algorithm is able to distinguish between different types of rules.

3.3. **Path planner parameters.** Beside the tuning and rule selection parameters, we include two parameters of the path planner in the genetic optimization model including $R_{\max}$ and $Iter_{\max}$ which are the maximum allowed risk for a generated sample and the maximum number of iterations before the algorithm returns failure respectively.

$$R_{\max} \in [0, 1], \quad Iter_{\max} \in [10, 10^5] \tag{20}$$

Choosing the values for these parameters is an important part of the planning. As presented in Figure 7, these two parameters are contradictory and by increasing the value of one of them, we need to decrease the other one in order to keep the path planner efficient.

Choosing higher values for $R_{\max}$ results in longer paths with shorter running time while, higher values for $Iter_{\max}$ creates shorter paths with high computation and long running time. Therefore, we include these two parameters in the optimization model so they can change with the rest of the algorithm's parameters and their optimum values will be determined along with the other parameters of the planner. Totally, the genetic optimization model includes 83 variables covering tuning variables $(\alpha_1, \alpha_2, \ldots, \alpha_{40})$, nonlinear scaling

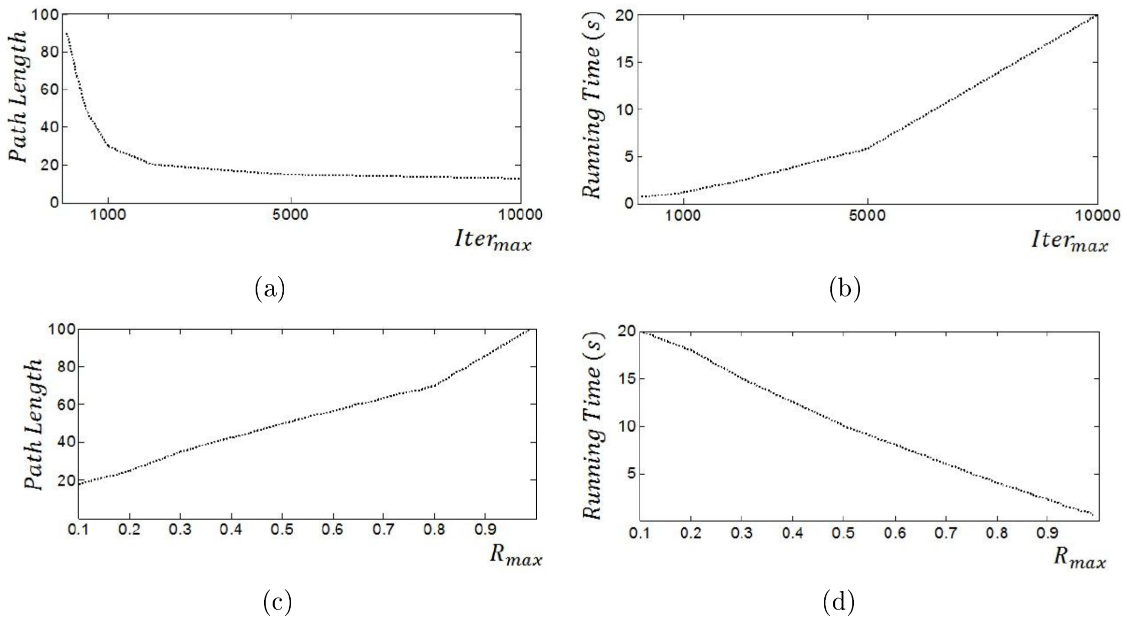


FIGURE 7. The changes in path length and running time of the planner for different values of $R_{\max}$ and $Iter_{\max}$: (a) and (b) increasing the value of $Iter_{\max}$ will decrease the path length while it increases the running time of the planner, (c) and (d) higher values for $R_{\max}$ results in longer paths but with lower running times.

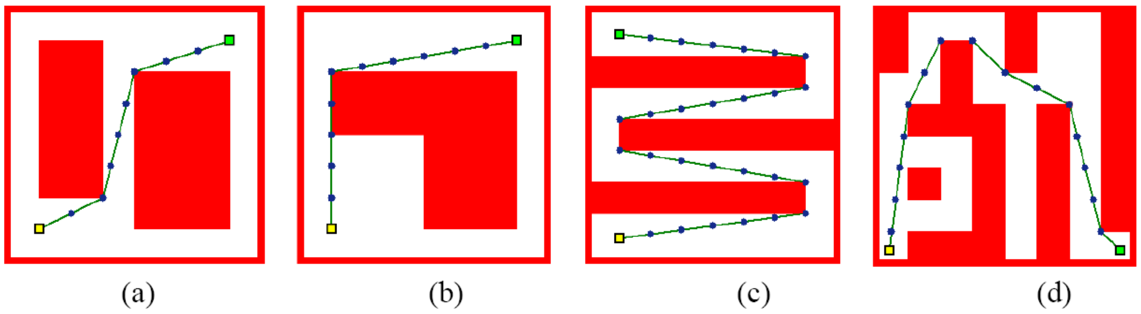


FIGURE 8. Test environment for defining the fitness function: (a) convex, (b) concave, (c) maze and (d) Mix workspaces

factors $(\alpha_{41}, \alpha_{42}, \ldots, \alpha_{54})$, rule selection variables $(\alpha_{55}, \alpha_{56}, \ldots, \alpha_{81})$, and path planner variables $(\alpha_{82}, \alpha_{83})$.

3.4. **Fitness function.** The optimization procedure, attempts to optimize the performance of the fuzzy controller for path planner. Hence, the fitness function needs to be related to the path planning problem. On the other hand, our path planning algorithm is designed for handling unknown environments where there is no prior information about the parameters of the environment and it is not possible to run the optimization during the path planning process. For setting a suitable fitness function, we design four test environments as presented in Figure 8, including convex, concave, maze-like and Mix environment.

The fuzzy path planner parameters will be optimized in these test environments and the resulted optimized fuzzy controller will be used for path planning tasks in the actual cases.

We design two scenarios for fitness function. In the first scenario (*MinRisk*), the fitness function is defined as the sum of the risks of the generated samples when the planner is producing optimum path obtained from the Visibility Graph approach. In Figure 8, the optimal paths (lines) and different positions of the robot on these paths (circles) are shown on the optimum paths for each test environment. The genetic optimizer attempts to minimize the total risks of the generated positions based on the visibility graph's resulted paths.

$$\mathit{Fitness\ Function}^{\mathit{MinRisk}} = \sum\nolimits_{i=1}^{4} \sum\nolimits_{j=1}^{m_i} \mathit{Risk}_{ij} \tag{21}$$

where m_i is the number of iterations in the ith environment and Risk_{ij} is the output of the fuzzy controller for the ith iteration of the jth environment.

In the second scenario (*OptPath*), the path planner will be executed in each test environments and the fitness function is defined as the mean square error of the path lengths of the algorithm in each test environments.

$$\mathit{Fitness\ Function}^{\mathit{OptPath}} = 1/8 \left[\sum\nolimits_{i=1}^{4} \left(PL_i - PL_i^{Opt} \right)^2 \right] \tag{22}$$

where PL_i and PL_i^{Opt} are the path length of the algorithm's results and the optimal path length in the ith test environment respectively. Figure 9 shows the optimization results in each test environment.

As can be seen in Figure 10, the genetic optimizer tuned the membership functions for both input and output variables. Also, the optimizer changed the shape of the membership functions by means of the nonlinear scaling factors. The optimal values for $R_{\max}$ and $\mathit{Iter}_{\max}$ are presented in Table 1.

Like any algorithm, our proposed model is sensitive to some of its parameters and one of the main reasons for developing an optimization method is to determine the optimum values for these parameters in order to have the best performance of the model. There are some other parameters like the shapes and numbers of the membership functions which the algorithm seems to have a robust performance regarding to them. As the genetic

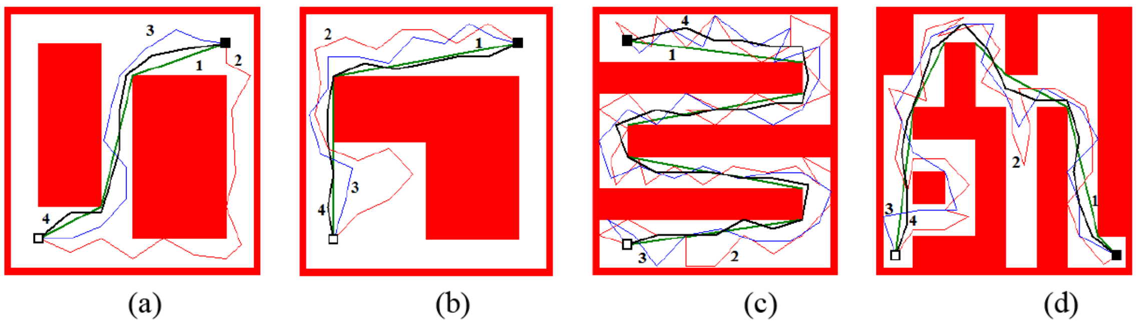


FIGURE 9. The optimization results in test environments. Line 1 are the optimal paths, Line 2 are the resulted paths from the original controller, Line 3 correspond to the first fitness function (*MinRisk*) and Line 4 are resulted from the second fitness function (*OptPath*).

TABLE 1. The optimized values for planner variables

	$R_{\max}$	$\mathit{Iter}_{\max}$
Initial FLC	60 %	100
MinRisk FLC	51.83 %	657
OptPath FLC	43.41 %	218

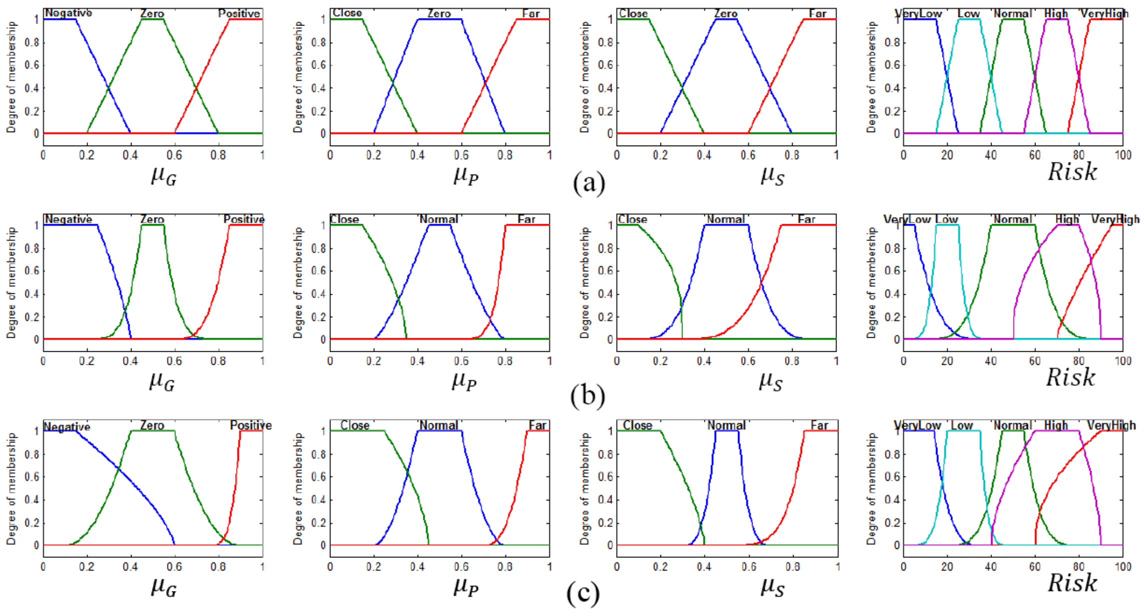


FIGURE 10. The results of the genetic optimization process on the fuzzy membership functions: (a) the initial controller, (b) the optimized controller based on *MinRisk* fitness function and (c) the optimized controller based on the *OptPath* fitness function

algorithm normally uses a random initial answer, the set-up procedure for the algorithm's parameters is taking place automatically as the optimizer keeps running the algorithm.

The genetic optimizer increases the value of $Iter_{\max}$ while it reduces $R_{\max}$. Figure 11 presents the decision surfaces for initial, *MinRisk* and *OptPath* FLCs.

4. Simulation Studies.

We simulate our proposed approach in five classes of environments including convex, maze, narrow passage, bug trap and concave environments. As described before, the proposed path planning algorithm is based on randomized sampling and therefore, the generated paths for a given problem will be different in each execution as the sampler chooses random positions for evaluation. For each class, the algorithm runs 100 times. The simulations were in MatLab R2008b using a 2 GHz Intel Core 2 Duo Processor. As presented in the flowchart of the proposed approach in Figure 12, the genetic optimizer modifies the fuzzy controller of the path planner and takes place before the planner starts any navigation tasks.

We use the initial, *MinRisk* and *OptPath* fuzzy controllers in the sampling-based path planner in all designed environments to analyze the effects of the genetic optimizer on the generated path length and running time of the algorithm. Figure 13 shows the results of the path planning in some environments using the optimized fuzzy-tabu controller with the second fitness function (*OptPath*). The resulted algorithm generates safe and relatively short paths without getting trapped in any local minima. Unlike the studied sampling-based algorithms, the proposed approach solves the path planning problems in difficult environments without any failure.

Table 2 shows the comparison results for our algorithm and 8 different sampling-based approaches including PRM, RBB, Gaussian sampling, RRT, RRT-bidirectional, RRT-Connect, DDRRT-bidirectional and ADRRT-bidirectional for 100 runs. We choose these algorithms for comparison studies because their main objective is to improve the performance of the sampling-based approaches in terms of path length and running time.

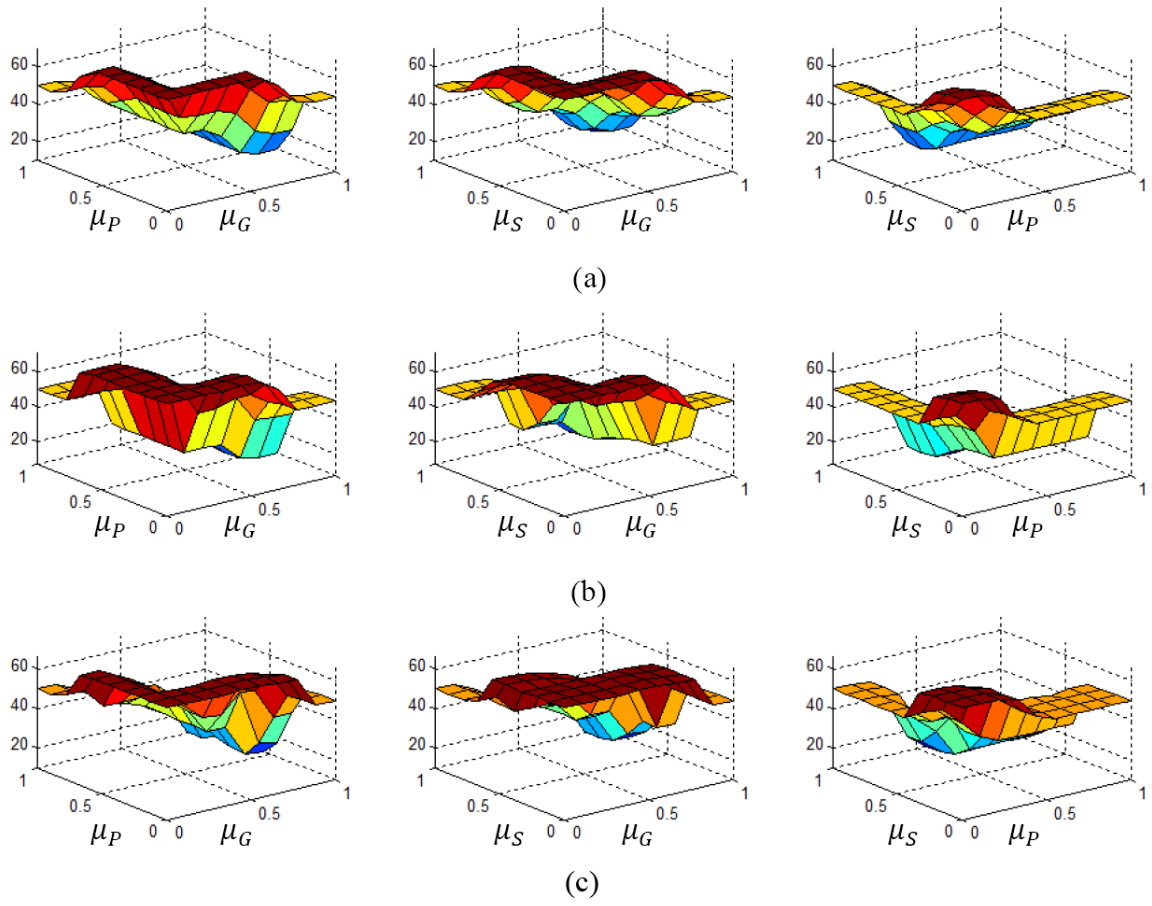


Figure 11. The results of the genetic optimization process on the fuzzy decision surfaces: (a) the initial controller, (b) the optimized controller based on *MinRisk* fitness function and (c) the optimized controller based on the *OptPath* fitness function

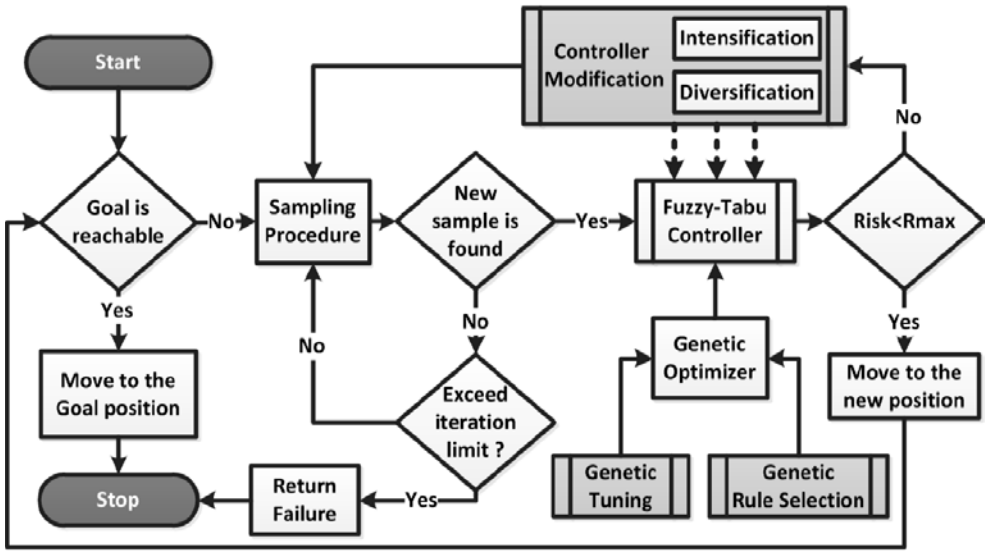


Figure 12. The flowchart of the proposed algorithm

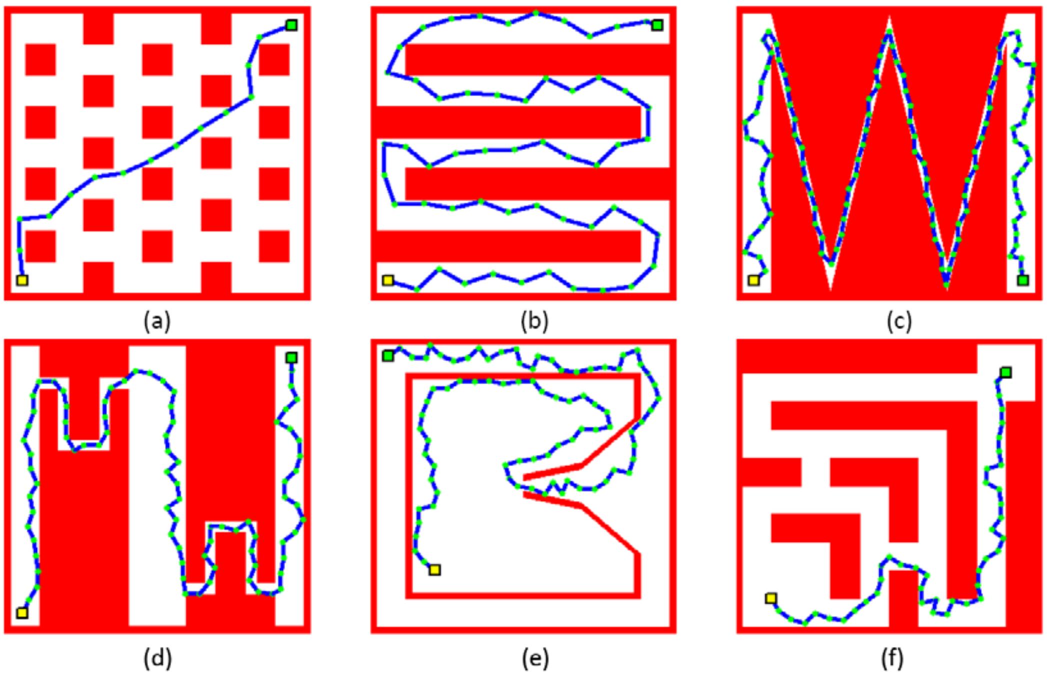


FIGURE 13. Simulations of the optimized algorithm with the second fitness function (*OptPath*) in 6 different workspaces including: (a) chess plate, (b) maze, (c) narrow 1, (d) narrow 2, (e) bug trap and (f) local minima

TABLE 2. Simulation results for studied variables

Algorithm \ Environment		**Chess** (OPL = 13.9)		**Maze** (OPL = 45.2)		**Narrow 1** (OPL = 44.2)		**Narrow 2** (OPL = 35.4)		**Bug Trap** (OPL = 18.8)		**Local Minima** (OPL = 15.9)	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
PRM (A^*) $N = 300, K = 20$	Path Length	15.03	0.93	50.86	1.01	48.64	1.29	37.62	0.46	22.82	3.14	18.60	0.62
	Runtime (s)	10.55	0.19	7.47	0.15	7.40	0.05	7.58	0.12	7.25	0.17	7.64	0.05
	Failure	0%		0%		65%		80%		20%		0%	
RBB (A^*) $N = 300, K = 20,$ $P(\text{Bridge}) = 0.8$	Path Length	14.99	0.82	50.52	0.73	48.97	1.06	37.81	1.01	30.50	0.81	19.35	0.74
	Runtime (s)	11.41	0.24	7.49	0.15	7.54	0.09	7.65	0.05	7.36	0.05	2.30	0.02
	Failure	0%		0%		20%		60%		10%		0%	
Gaussian (A^*) $N = 300, K = 20$ $\sigma = 0.1$	Path Length	18.45	1.80	57.51	1.62	51.47	0.99	42.66	2.58	23.73	2.55	23.57	1.60
	Runtime (s)	14.40	0.62	7.59	0.05	7.66	0.07	7.80	0.06	7.13	0.09	7.71	0.06
	Failure	0%		1%		0%		15%		25%		0%	
RRT Step = 1	Path Length	18.24	1.86	57.41	1.19	56.25	2.39	48.63	1.13	32.45	2.63	23.37	2.23
	Runtime (s)	12.32	7.95	137.87	6.11	316.32	6.58	250.69	3.08	316.25	3.08	8.74	3.96
	Failure	0%		23%		83%		23%		55%		0%	
RRT-bi Step = 1	Path Length	18.46	2.45	61.46	2.50	55.31	1.09	47.13	2.95	38.55	4.47	27.30	3.08
	Runtime (s)	3.64	0.95	26.25	9.25	211.07	5.03	102.51	108.30	22.42	19.49	3.40	0.96
	Failure	0%		0%		68%		0%		0%		0%	
RRT-Connect	Path Length	22.09	3.83	62.43	2.01	56.04	1.84	42.74	2.35	38.67	7.60	25.29	4.11
	Runtime (s)	1.63	1.03	4.24	1.81	168.32	4.35	53.27	4.19	8.13	9.57	0.44	0.19
	Failure	0%		0%		63%		0%		0%		0%	
DDRRT-bi R = 0.1	Path Length	16.58	2.27	63.82	2.88	62.06	3.14	54.76	2.93	38.97	3.59	22.99	2.71
	Runtime (s)	2.88	0.65	8.68	4.44	173.55	2.22	23.52	3.02	3.66	1.38	2.54	0.38
	Failure	0%		0%		44%		0%		0%		0%	
ADRRT-bi $\alpha = 0.05$	Path Length	16.05	0.69	57.87	2.86	59.37	2.36	44.51	1.17	37.64	3.44	22.42	2.49
	Runtime (s)	2.64	0.53	6.22	1.84	143.18	2.03	9.78	0.73	2.71	0.34	2.31	0.21
	Failure	0%		0%		36%		0%		0%		0%	
Our Algorithm Step = 0.5	Path Length	**14.62**	**0.34**	**47.17**	**0.45**	**46.74**	**0.77**	**36.79**	**0.04**	**21.16**	**0.81**	**17.50**	**0.24**
	Runtime (s)	**0.07**	**0.02**	**0.81**	**0.02**	**0.86**	**0.04**	**0.87**	**0.03**	**1.12**	**0.08**	**0.12**	**0.02**
	Failure	0%		0%		0%		0%		0%		0%	

As the second fitness function provides better solutions, we only add its results in Table 2 and ignore the resulted controller from the first fitness function. As can be seen in Table 2, each of the studied algorithms has poor performance at least in one environment while, the proposed method performs efficiently in all environments without any failure. The average path lengths, average runtimes and standard deviations of the proposed algorithm

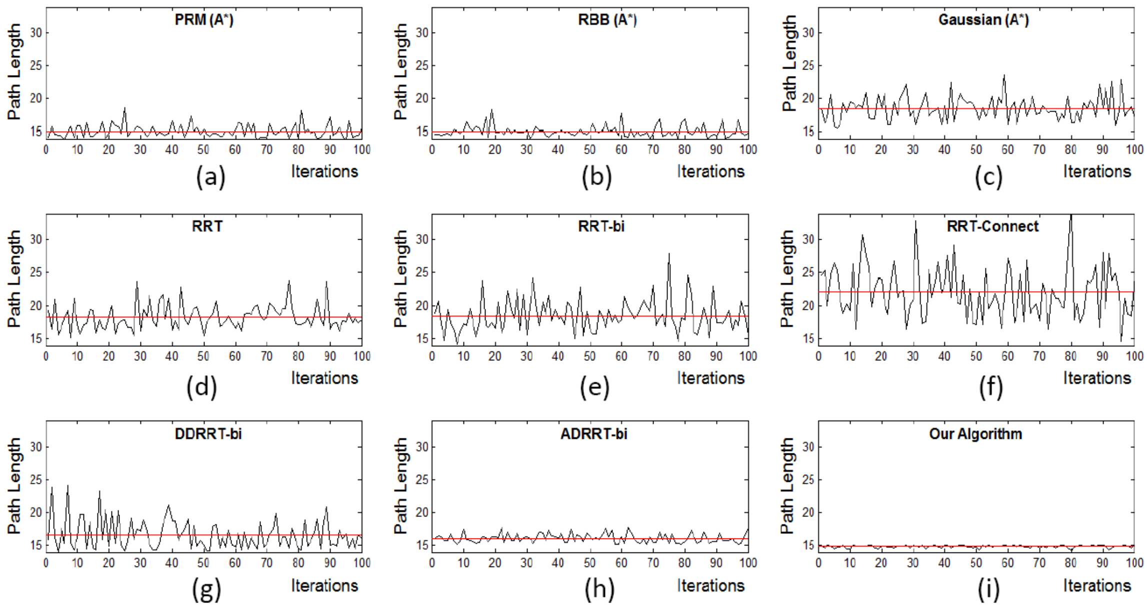


Figure 14. Path length variations for 100 executions of the studied planners for the chess plate environment as presented in Figure 13(a). Note that all graphs are plotted in the range of $[0, 35]$ for the vertical axis.

are considerably lower than the other studied algorithms. The standard deviation (Std) of the studied algorithm's path length and runtime shows the stability of the proposed algorithm. Figure 14 shows the variations of generated path lengths for studied algorithms for 100 executions in the workspace of Figure 13(a).

According to Table 2, in a simple convex environment like the chess board, all of the simulated algorithms generate relatively short paths in reasonable amounts of time. Although the RRT-Connect algorithm generates longer paths than the others because of its greedy heuristic rule which reduces the runtime but results in longer paths. In a maze-like environment, the average runtime of the Gaussian sampling approach shows that the near obstacle sampling strategy of this approach increases the runtime of the planner. Other algorithms can solve a maze problem in a short time with relatively short paths. In narrow passages, most of the algorithms have some problems in finding a feasible answer in a reasonable amount of time. The only successful approaches in this class of problems are RBB and Gaussian sampling. In a bug trap, only DDRRT-bi and ADRRT-bi have successful performances while the other algorithms fail to find an answer and if they don't, the results are far from the optimum values. Considering the facts that each of the studied approaches has poor performances in some of the environments and they cannot be applied without complete information about the environments, we can see that the proposed algorithm has a robust ability to deal with the lack of information about the environments and to plan in difficult environments. Moreover, the proposed algorithm returns near-optimal paths in less than few seconds while the variation of the results in different executions in considerably low.

The variation in the length of the generated paths for our proposed algorithm is less than the other studied algorithms because of the performance of the fuzzy-tabu controller. This controller always selects the positions with the risk less than the maximum allowed risk ($R_{\max}$) and therefore, it guides the sampling-based planner to generate paths with relatively close lengths in different runs.

Another important factor is the variation of running time of the planner in different runs. Figure 15, presents the variation of the running time for all studied algorithms in

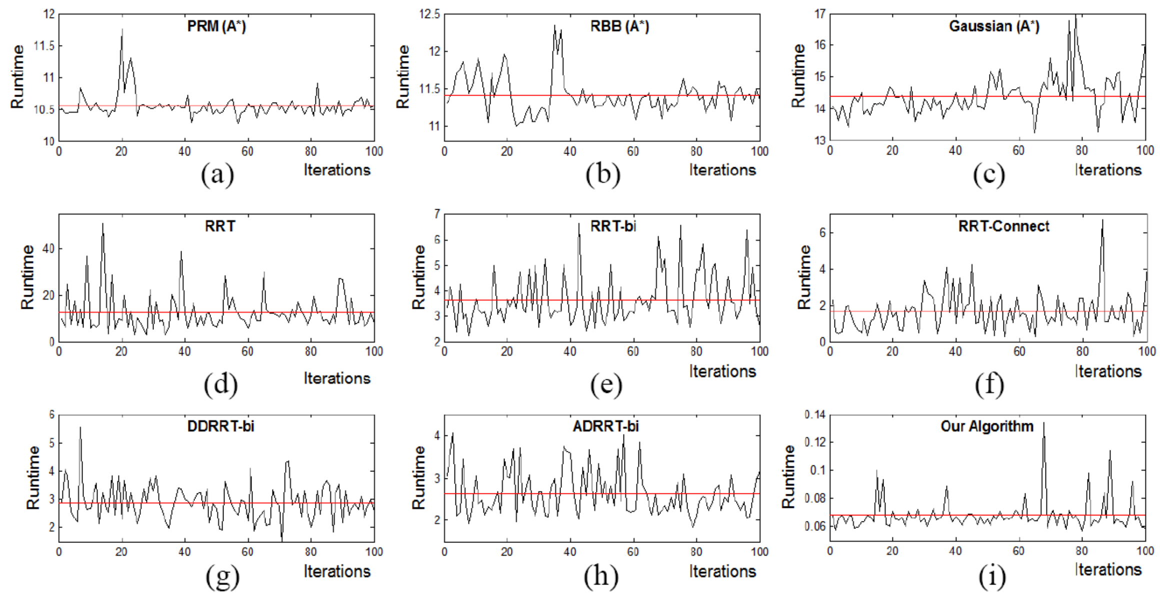


FIGURE 15. Running time variations for 100 executions of the studied planners for the chess plate environment as presented in Figure 13(a). Note that each graph is plotted in the range of its own vertical bounds.

TABLE 3. Performance comparison of the original controller, the optimized controller with the first fitness function (*MinRisk*) and the optimized controller with the second fitness function (*OptPath*). OPL is the length of the optimal path.

Environment	Our Algorithm (Original)					Our Algorithm (*MinRisk*)					Our Algorithm (*OptPath*)				
	Path Length		Runtime		$\frac{\text{PL}}{\text{OPL}}$	Path Length		Runtime		$\frac{\text{PL}}{\text{OPL}}$	Path Length		Runtime		$\frac{\text{PL}}{\text{OPL}}$
	Mean	Std	Mean	Std		Mean	Std	Mean	Std		Mean	Std	Mean	Std	
Chess	15.11	0.41	0.08	0.10	**108.71**	14.92	0.38	0.12	0.03	**107.34**	14.62	0.34	0.07	0.02	**105.18**
Maze	48.33	0.39	0.83	0.09	**106.92**	47.84	0.46	0.83	0.07	**105.84**	47.17	0.45	0.81	0.02	**104.36**
Narrow 1	48.04	0.81	0.91	0.14	**108.69**	47.49	0.74	0.88	0.06	**107.44**	46.74	0.77	0.86	0.04	**105.75**
Narrow 2	38.57	0.11	0.97	0.13	**108.95**	37.23	0.09	0.90	0.05	**105.17**	36.79	0.04	0.87	0.03	**103.93**
Bug Trap	23.81	0.83	1.13	0.14	**126.65**	22.07	0.73	1.18	0.09	**117.39**	21.16	0.81	1.12	0.08	**112.55**
Local Minima	18.08	0.34	0.21	0.09	**113.50**	17.93	0.31	0.19	0.08	**112.55**	17.50	0.24	0.12	0.02	**109.86**

the same environment as Figure 13(a). The standard deviation of the runtime for our algorithm in the Chess environment is 0.02 which is the smallest Std among all considered algorithms. The main reason for this low runtime is the fact that unlike the other sampling-based algorithms, our planner does not attempt to capture the connectivity of the environment completely. It only samples inside the visible area and tries to move the robot closer to the goal, farther from the start position, and farther from the previous positions.

Table 3 shows the simulation results for the initial fuzzy-tabu controller, the optimized controller with the first fitness function (*MinRisk*) and the second fitness function (*OptPath*).

Based on the simulation results, the second fitness function (*OptPath*) performs more efficiently. It generates path in less than 2 seconds while the path lengths are relatively close to the optimal path lengths.

The simulation results indicate that the optimized controller improves the performance of the proposed online sampling-based approach and succeeds in guiding the robot in any type of environment with very low variation of the results in different runs.

5. **Conclusion.**

In this paper, we proposed a novel sampling-based path planning algorithm for guiding a mobile robot in an unknown environment. The drawbacks of the existing sampling-based algorithms were classified into five groups including (1) inability of planning in unknown environments, (2) the optimality problem of the generated paths, (3) high running time of the planner, (4) failure in difficult situations like narrow passage and bug trap, and (5) high variation of the results in different runs. We incorporated several intelligent components to overcome the abovementioned drawbacks. The proposed algorithm acquires information from the robot's sensory system and restricts the sampling area to the visible regions for the robot. A fuzzy controller was designed to evaluate the generated samples and choose better ones in terms of path length and runtime. This fuzzy controller employs the tabu search heuristic rules in the evaluation procedure to handle planning in difficult environments. Furthermore, a genetic-based optimization framework was designed to optimize the controller parameters. The resulted algorithm performs successfully in any type of environments without failure while the average path lengths and runtimes are considerably lower than the other sampling-based algorithms. Also, the variations of the results in different runs are tremendously lower than the other studied approaches.

REFERENCES

[1] J. Canny, *The Complexity of Robot Motion Planning*, The MIT Press, USA, 1988.

[2] S. Thrun, M. Montemerlo and J. D. D. Dolgov, Path planning for autonomous vehicles in unknown semi-structured environments, *The International Journal of Robotics Research*, vol.29, no.5, pp.485-501, 2010.

[3] M. Saad, H. Saliah-Hassane and J. Sfeir, An improved artificial potential field approach to real-time mobile robot path planning in an unknown environment, *IEEE International Symposium on Robotic and Sensors Environments*, pp.208-213, 2011.

[4] X. Huang, M. Wang and P. Li, A new hybrid method for mobile robot dynamic local path planning in unknown environment, *Journal of Computers*, vol.5, no.5, pp.773-781, 2010.

[5] M. Deng, A. Inoue and L. Jiang, Obstacle avoidance and motion control of a two wheeled mobile robot using svr technique, *International Journal of Innovative Computing, Information and Control*, vol.5, no.2, pp.253-262, 2009.

[6] M. Xie, G. Wei, S. Zhang and D. Zhang, Improved A* algorithm for robot path planning in unknown environment, *The 3rd IEEE International Conference on Broadband Network and Multimedia Technology*, pp.1154-1158, 2010.

[7] N. N. Buniyamin, Z. Mohamad and W. A. J. Wan, Point to point sensor based path planning algorithm for autonomous mobile robots, *The 9th WSEAS International Conference on System Science and Simulation in Engineering Conference on System Science and Simulation in Engineering Conference on System Science and Simulation in Engineering*, Iwate, Japan, 2010.

[8] W. Ngah, W. A. J. Sariff, N. Mohamad and Z. Buniyamin, A simple local path planning algorithm for autonomous mobile robots, *International Journal of Systems Applications, Engineering and Development*, vol.5, no.2, 2011.

[9] H. Choset et al., *Principles of Robot Motion: Theory, Algorithms, and Implementations*, H. M. Choset (ed.), The MIT Press, Massachusetts, USA, 2005.

[10] S. M. Lavalle, *Planning Algorithms*, Cambridge University Press, Cambridge, USA, 2006.

[11] P. Judea and J. D. Rina, Generalized best-first search strategies and the optimality of A*, *Journal of ACM*, vol.32, no.3, pp.505-536, 1985.

[12] P. Svestka, J. C. Latombe, M. H. Overmars and L. E. Kavaraki, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, *IEEE Trans. on Robotics and Automation*, vol.12, no.4, pp.566-580, 1996.

[13] T. Jiang, J. Reif, Z. Sun and D. Hsu, The bridge test for sampling narrow passages with probabilistic roadmap planners, *IEEE International Conference on Robotics and Automation*, pp.4420-4426, 2003.

[14] M. H. Overmars, A. F. Stappen and V. Boor, The Gaussian sampling strategy for probabilistic roadmap planners, *IEEE International Conference on Robotics and Automation*, vol.2, pp.1018-1023, 1999.

[15] N. M. Bayazit, O. B. Dale, L. K. Jones and V. D. Amato, OBPRM: An obstacle-based PRM for 3d workspaces, *Proc. of the 3rd International Workshop on the Algorithmic Foundations of Robotics*, Houston, TX, pp.156-168, 1998.

[16] D. Kavraki, L. E. Latombe, J.-C. R. Motwani and S. S. Hsu, On finding narrow passages with probabilistic roadmap planners, in *Robotics: The Algorithmic Prespective*, P. Agarwal et al. (eds.), Wellesley, MA, 1998.

[17] C. Holleman, L. E. Kavraki and L. J. Guibas, A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach, *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp.254-260, 1999.

[18] S. M. Lavalle, Rapidly-exploring random trees: A new tool for path planning, *Technical Report TR 98-11*, Computer Science Department, Iowa State University, 1998.

[19] S. M. Lavalle and J. J. Kuffner. Jr, RRT-connect: An efficient approach to single-query path planning, *IEEE International Conference on Robotics and Automation*, pp.995-1001, 2000.

[20] L. Jaillet, T. Simeon, S. M. Lavalle and A. Yershova, Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain, in *IEEE International Conference on Robotics and Automation*, Barcelona, Spain, pp.3856-3861, 2006.

[21] A. Yershova, S. M. LaValle, T. Simeon and L. Jaillet, Adaptive tuning of the sampling domain for dynamic-domain RRTs, *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp.2851-2856, 2005.

[22] R. Tedrake and A. Shkolnik, *Sample-Based Planning with Volvolumes in Configuration Space*, http://groups.csail.mit.edu/robotics-center/public papers/Shkolnik11.pdf, 2011.

[23] I. A. Sucan, L. E. Kavraki and K. I. Tsianos, Sampling-based robot motion planning: Towards realistic applications, *Computer Science Review*, vol.1, no.1, pp.2-11, 2007.

[24] O. Karahan and Z. Bingul, A fuzzy logic controller tuned with PSO for 2 DOF robot trajectory control, *Expert Systems with Applications*, vol.38, pp.1017-1031, 2011.

[25] F. Herrera, Genetic fuzzy systems: Taxonomy, current research trends and prospects, *Evolutionary Intelligence*, vol.1, pp.27-46, 2008.

[26] W. Khaksar, T. S. Hong and M. Khaksar, A sampling-based tabu search approach for online path planning, *Advanced Robotics*, vol.26, no.8-9, 2011.

[27] O. Cordon, M. J. del Jesus, F. Herrera and J. Casillas, Genetic tuning of fuzzy rule deep structures preserving interpretability and its interaction with fuzzy rule set reduction, *IEEE Trans. on Fuzzy Systems*, vol.13, no.1, pp.13-29, 2005.

[28] M. J. Gacto, F. Herrera and R. Alcala, A multi-objective genetic algorithm for tuning and rule selection to obtain accurate and compact linguistic fuzzy rule-based systems, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol.15, no.5, pp.539-557, 2007.