A Database Driven Memetic Algorithm for Fuzzy Set Optimization

Kevin McCarty[1], Milos Manic[1]
[1]University of Idaho, Idaho, USA,
kmccarty@ieee.org, misko@ieee.org

Abstract— **Fuzzy logic provides a natural and precise way for humans to define and interact with systems. Optimizing a fuzzy inference system, however, presents some special challenges for the developer because of the imprecision that is inherent to fuzzy sets. This paper expands upon an earlier development of a fuzzy framework, adding components for dynamic self-optimization. What makes this approach unique is the use of relational database as a computational engine for the memetic algorithm and fitness function. The new architecture combines the power of fuzzy logic with the special properties of a relational database to create an efficient, flexible and self-optimizing combination. Database objects provide the fitness function, population sampling, gene crossover and mutation components allowing for superior batch processing and data mining potential. Results show the framework is able to improve the performance of a working configuration as well as fix a non-working configuration.**

I. Introduction

Describing the behaviors of complex system presents many challenges for the software architect and developer. Foremost among them is the ability to model behaviors that are by their very nature imprecise [1, 2]. Fuzzy logic, also known as Type-1 or Type-2 Fuzzy Logic, introduced by Lofti Zadeh, addresses these problems by approximate, rather than precise descriptions for terms and allows for polyvalent membership definitions. A fuzzy system, with fuzzy sets, terms and rules, is called fuzzy controller or Fuzzy Inference System (FIS).

Optimization of an FIS can prove challenging because of the imprecise nature of fuzzy sets and how they are constructed [3]. Optimization of any algorithm usually involves finding and testing alternative configurations and choosing which configuration most closely produces the desired result.

Since fuzzy sets are essentially defined by human experts with imperfect knowledge, an FIS is often not running optimally. In such a situation, the fuzzy system will have a tendency to overcompensate again and again. A problem thermostat might make dramatic switches from HEAT to COOL in an attempt to keep a temperature level while a navigation unit might make wide swings to the left and right to overcompensating each time. So in addition to simply having a working FIS, one must try to make the FIS as efficient as possible.

Genetic algorithms have been shown to help with this task [4] by trying a myriad of different configurations using random sampling, cross-fertilization and mutation. However, using a genetic algorithm means also finding a fitness function which can be used to measure the effectiveness of a particular test gene. Managing the genetic algorithm and determining the fitness function requires managing large numbers of measurements from the FIS along with samples of genes and results. Relational databases are very good for managing large sets of data and batch operations on that data can be performed efficiently using the database framework [5]. Therefore, by combining the individual computation capabilities of a FIS with the batch capabilities of a database, a fuzzy application can possess a great deal of power for problem solving. By providing an underlying database framework, resulting inputs and outputs can be data mined, which may also assist with learning and optimization.

This paper discusses work done on extensions to an existing fuzzy framework [6], supporting both Type-1 and Type-2 fuzzy inference systems, with focus on the advantages of using a relational database for learning, optimization and analysis. The extensions include runtime capture of fuzzy results, table-driven and dynamic configuration objects which allow the framework to reconfigure itself on the fly. A memetic algorithm is also added to the framework, with the database providing the objects for gene sampling, random mutation, crossover mutation and the fitness function.

The remainder of this article is structured as follows: Part II introduces the problem statement and discusses prior work. Part III describes the software architecture, database objects and schema. Part VI describes how the memetic algorithm and database batch processes are used to create a self-optimizing component. Part V presents some test examples and results. Part VI presents conclusions and future work.

II. Problem Statement

Consider a robot car whose goal is trying to navigate a simple maze as illustrated in fig. 1.

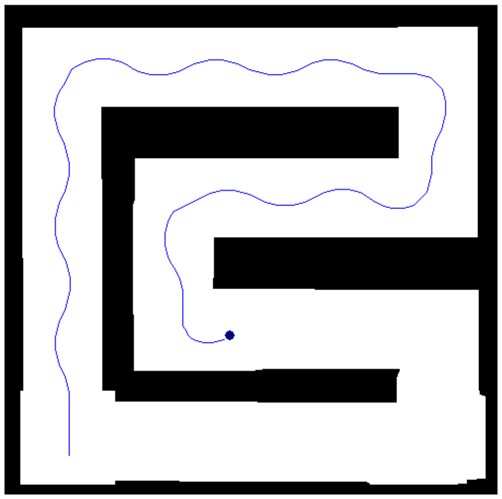


Fig. 1. A Robot Car Navigating a Maze

The car must figure out how to move through the maze while also maintaining an efficient path and avoiding walls and obstacles. Doing this requires continual adjustment of its direction in order to avoid coming too close.

But what does "too close" mean? The answer might depend upon the skill of the driver, road conditions, speed and other factors. Using crisp logic requires the creation of a complex polynomial or long series of conditionals to handle all the inputs and create a smooth continuous output; not an easy task! Even something as simple as a thermostat control can pose a challenge in crisp mathematical terms. This is due to the bivalent nature of traditional crisp sets in which membership for a point along the domain is mutually exclusive [7]. A crisp definition for temperature might look like the following:

IF TEMP < 20^o THEN ROOM IS COOL
IF TEMP IS 20^o TO 30^o THEN ROOM IS WARM (1)
IF TEMP > 30^o THEN ROOM IS HOT

The difficulty becomes particularly pronounced around boundary values. Suppose WARM is defined as ranging from 20-30 degrees centigrade. In crisp logic, 19.999 degrees is considered COOL. Moving from a COOL state to a WARM one with a .001 degree change does not square with a typical, more ambiguous "human" interpretation. There is also the issue of sensor noise, which also has the notable effect of introducing uncertainty into an affected system via measurement error. Hence, it may be practically impossible to determine with precision when that transition from WARM to COOL occurs.

Even if measurements were totally precise, there is the additional difficulty in modeling behavior around boundary points in a bivalent universe. A thermostat set to simply turn ON when the room is COOL can waste a lot of effor trying to modulate around .001 degree. A complex polynomial function can help smooth out the modulation, but it may not adequately address differences in "expert" opinion. Experts may define WARM over a range of temperatures, not just 20-30 degrees. As a result, a "precise" polynomial may actually make it more, not less, difficult to model precise behavior in the thermostat and describe that behavior in humanistic terms.

A fuzzy controller, in contrast, allows for greater linguistic precision in describing a complex system behavior while at the same time relaxing precision around the boundary points. This is done through the use of a membership function μ whose output, instead of the traditional, bivalent FALSE (0) and TRUE (1) allows for output of FALSE, TRUE and all values in between. So for a domain D

$$\mu(x) \rightarrow [0,1], x \in D \tag{1}$$

A fuzzy membership function defines a fuzzy set f_s, which can be described using a linguistic term such as WARM. A fuzzy set f_s is then a set of ordered pairs

$$f_s \equiv \{\langle x, \mu(x)\rangle | x \epsilon D\} \tag{2}$$

A linguistic term is applied to a fuzzy set as an identifier. Fuzzy sets come in many convex shapes, such as a triangle or trapezoid, depending upon its membership function demonstrated in fig 1.

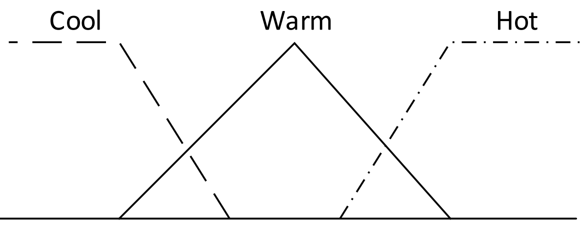


Fig. 2. Fuzzy Sets in the Framework

Each fuzzy set contributes partially to the final result depending upon the resulting μ(x). From the example above, 19.99 degrees centigrade might be classified as partially COOL and partially WARM since the sets overlap as shown in fig 2.

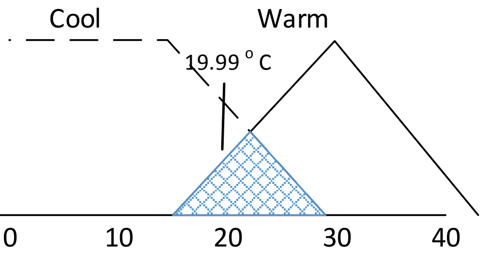


Fig. 3. Fuzzy Set Definition Overlap

Traditional Zadeh rules for fuzzification of a Type-1 FIS consider membership of a fuzzy variable is equal to the minimum membership of the corresponding fuzzy sets. This is also referred to as a fuzzy intersection of fuzzy sets.

$$\cap \mu_{C_i} = \min(\mu_{C_1}, \mu_{C_2}, .., \mu_{C_n}) \tag{5}$$

In order to defuzzify a result, one takes a fuzzy union across all intervals in the domain by taking the maximum across the sets of intervals and their corresponding memberships.

$$\cup \mu_{C_i} = \max(\mu_{C_1}, \mu_{C_2}, .., \mu_{C_n}) \tag{6}$$

A centroid, or center of gravity is calculated by determining a weighted mean across the fuzzy region [9]. The fuzzy solution region $\Re$ is calculated by the following:

$$\Re = \frac{\sum_{i=0}^{n} d_i \mu_A(d_i)}{\sum_{i=0}^{n} \mu_A(d_i)} \tag{7}$$

where d is the ith domain value and $\mu(d)$ is the membership value. $\Re$ can either be a standalone result, or, in the case of a dynamic control such as a thermostat or navigation system, it represents the change applied to the state of the FIS.

$$\Re \equiv \delta_{FIS} \tag{8}$$

An additional benefit of fuzzy logic is that it also provides stronger mechanisms for handling noise and uncertainty along with variations among "expert" definitions [2], [8]. Some experts may find that 22^oC is COOL, while others find the 18^oC is WARM. Fuzzy logic allows one to create fuzzy sets which encompass multiple definitions under a single set/term. In cases where the definition is clear, membership is defined by the μ of one set; but where there is overlap, the resulting membership is a contribution from every fuzzy set for which the corresponding μ > 0.

Type-2 fuzzy logic extends Type-1 by introducing fuzzy sets whose membership functions are themselves fuzzy sets. (Mendel) A Type-2 membership function might be considered a second order fuzzifier expressed as a set of ordered pairs

$$\mu_2 = \{((x, \mu), \mu_2(x, \mu)) | \forall x \in D, \mu \in [0,1]\} \qquad (4)$$

Note also that output of μ_2 is a member of the set [0, 1]. A type-2 fuzzy interference system then allows an additional level of uncertainly which is useful in dealing with problems such extensive noise or smoothing out erratic behaviors that plague Type 1 controllers [8], [10]. Whereas a Type-1 fuzzy set is a 2-dimensional object, Type-2 fuzzy sets are surfaces as demonstrated in fig 3.

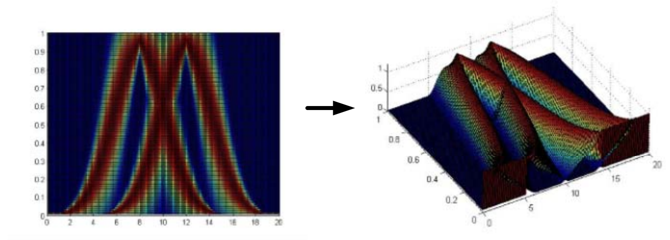


Fig. 4 Type-2 Fuzzy Sets

In many cases, testing every alternative configuration of fuzzy sets is impractical due to the huge number of potential configurations. One of the approaches to dealing with an impractical number of possibilities is using Local Search [11]. Local search takes a random starting point and follows the gradient of a result in order to find a local maximum (or minimum). Think of it as a blind man trying to climb a mountain. While he may not be able to see *the* peak, he knows in which direction *a* peak lies by following the gradient or slope of the ground. A problem occurs when our blind traveler finds himself on a maxima trap where every direction is down but he is not on the true peak as shown in fig 5.

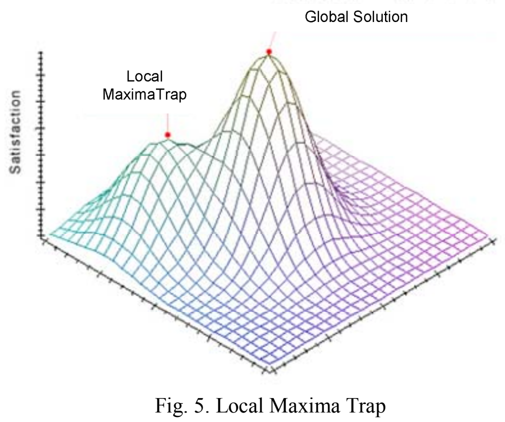


Fig. 5. Local Maxima Trap

There are many different Local Search techniques, from a simple hill climbing technique to more sophisticated techniques such as simulated annealing, in which the climber is "shaken" out of a local maximum through the annealing process. Evolutionary computation provides another form of local search in which configurations "evolve" in a fashion similar to natural processes.

In one type of genetic algorithm, a set of "genes" or configurations is built. A pair of genes is selected at random and compared using a fitness function. Winners and losers in the sample can then be crossed and/or mutated and compared with the winning configuration. Through the evolutionary process, new genes come into existence which have the potential to be superior to even the best gene in the original sample. A memetic algorithm takes the evolutionary process a step further by incorporating a local search technique post-evolution. This allows the gene to evolve and immediately search for a solution without having to resample.

All these processes combined serve to generate a large amount of data. Managing all this data requires a lot of effort and resources. Relational databases are the preferred way for storing large amounts of persistent data [12] but hooking one to an FIS will require building a number of objects to facilitate data transfer and analysis. Although not a necessary component, using a database presents several advantages over a proprietary file or memory based storage option:

1. A database allows third-party tools to examine or modify configuration and runtime values remotely.
2. A database allows for additional security when required
3. A database allows for advanced query techniques and superior storage of large datasets.

Though uncommon in traditional fuzzy frameworks, relational database extensions can help toward improving the usability and effectiveness of them. Even for generalized fuzzy operations, being able to store persistent data allows an algorithm to store, test and compare fuzzy sets, rules and their corresponding defuzzification values using a memetic algorithm. Having the powerful capabilities of a relational database storage and retrieval handy also means less work for the developer on that front plus relational databases also provide some additional advantages:

1. Relational databases are designed to efficiently run operations as batch processes. Batch processes can query large datasets for specific criteria more quickly than a row-by-row data search.
2. Many relational databases, such as Microsoft SQL Server, come with data mining tools. These tools can be used to mine fuzzy sets and rules to determine those which have greatest impact on desired result.

Hence, combining a traditional fuzzy framework with database hooks can increase the former's ability to analyze its prior performance. Furthermore, due to the relational storage, evolutionary algorithms and other local search techniques have plenty of data to do automated testing for maximums and minimums and discover optimizations that would be more difficult for humans or traditional algorithms.

III. Framework Architecture

As a minimum, implementation of a fuzzy controller requires building a Fuzzy Inference System (FIS). The FIS consists of a number of software components:

1. Fuzzy Sets (with Membership Functions)
2. Linguistic terms to define those sets
3. Collections of rules defining the behavior of a fuzzy system.
4. A Defuzzifier

The framework introduced in [6] implements basic Type-1 and Type-2 FIS and includes an XML-based configuration language. It was extended by adding database objects to capture intermediate results during the fuzzification and defuzzification process to a SQL Server database. In particular, results from each rule as it is evaluated along with the defuzzification results. The results are also used to test the efficacy of the optimization process.

IV. MEMETIC-BASED OPTIMIZATION

Local search routines such as hill-climb and simulated annealing are used for finding optimal solutions when it is impractical to review every possible configuration. By picking a random starting point and following a gradient to a nearby maximum or minimum, a local search routine is very useful for fuzzy set optimization. Evolutionary algorithms, such as the memetic algorithm take local search a step further by combining a gradient search with natural selection. [13]

One limitation with earlier approaches is the limited use of relational databases for fuzzy optimization [6]. By allowing the relational database to handle many of the processing tasks of the memetic algorithm, the FIS can dynamically create and test various fuzzy implementations automatically, evolve and determine which combinations are most optimal using batch operations instead of iterating measurement by measurement. Batch operations are designed for simultaneous processing of large datasets and are generally more efficient than traditional value by value calculations over large sets.

The database was built to handle four major tasks of the memetic algorithm

1. The Fitness Function
2. Gene Sampling and comparison
3. Crossover Mutation

Random MutationOptimization for this FIS involves reducing the back and forth motion correction in the navigation as demonstrated in fig 1. This means reducing the variations in the output values. The Fitness Function then must determine whether or not a particular decision is more optimal than another by comparing each FIS's decision and how far it deviates from an optimal path. Determining the optimal path requires the following steps in pseudocode:

```
result DETERMINE_OPTIMAL_PATH (values)
inputs: values, a set of inputs into FIS and final result
output: a set of points along the optimal path

Step 1: Record all changes as the FIS operates. In this
case, as the "car" tries to navigate the path.
Step 2: Determine each "peak" and "valley" in the
sequence.
Step 3: Take the midpoint between each peak/valley pair.
Step 4: Return the set of midpoints
```

For Step 1, the program logs all inputs to the FIS and resulting outputs to the database. Now think of a map with standard terrain features: peak, valley, plateau and depression. Step 2 defines a "peak" as any output value surrounded on both sides by lesser values. In other words, a peak is a local maximum. One problem is the existence of plateaus, where the same maximal value exists in a sequence indicating a "flat" surface. In that case, take the median value and declare it the peak. Do likewise for valleys and depressions using local minimums as shown in fig 6.

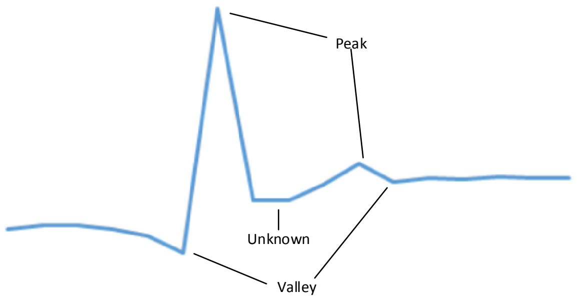


Fig. 6. Determining Peaks and Valleys

Note that each peak must be surrounded by valleys and vice versa so whenever peaks are adjacent to peaks such as the situation shown in fig 7 the greater peak will eliminate the lessor. Likewise for valleys.

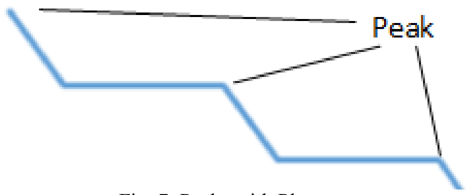


Fig. 7. Peaks with Plateaus

The final result is a set of alternating peaks and valleys like those shown in fig 8.

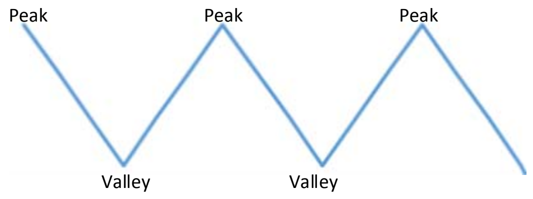


Fig. 8. Ordered Peaks and Valleys after Processing

Step 3 takes the midpoint between each peak and valley and connects them using a line. The path between the midpoints is the "optimal" path of the FIS show in fig 9.

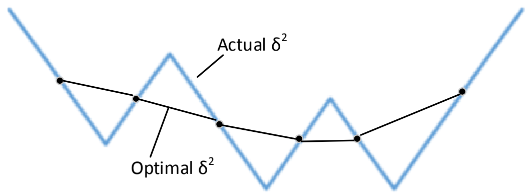


Fig. 9. Creating an Optimal Gradient

Calculate the "error" by taking the difference between the value of the optimal line returned from Step 4 at a given point and the actual value. This error is useful in the Fitness Function to determine the efficacy of a given FIS configuration.

This is where a relational database brings with it some formidable advantages. A typical algorithm might look at each point and then use neighboring points to determine peaks, valleys, plateaus and depressions, going back and forth in order to eliminate false peaks and valley as well as determine plateau and depression points. With some modifications that process can run over the entire set of results as a batch. The corresponding batch algorithm steps is as follows:

```
result DETERMINE_OPTIMAL_PATH (values)
inputs: values, a set of inputs into FIS and final result
output: a set of points along the optimal path

Step 1: Record all changes as the FIS operates.
Step 2a: Take the UNION of all known terrain features, unknowns, and the second point.
Step 2b: Rank the resulting set by value, epoch and type
Step 2c: Clump adjacent peaks and valleys using rank and epoch
Step 2d: Dense rank each clump
Step 2e: Get max and min rows for each clump
Step 2f: Assign unknowns to their corresponding clump
Step 2g: Order clumps to get local maximum or minimum
Step 3: Take the midpoint between each peak/valley pair.
Step 4: Return the set of midpoints
```

Step 1 is as before. Step 2a creates the UNION F of the following sets of points from FIS results P:

$$F = K \cup V \cup X \cup p_2 \qquad (9)$$

K consists of true peaks, all left plateaus and right plateaus

$$K = \bigcup \begin{cases} \{p_n | p_n > p_{n-1}, p_n > p_{n+1} \forall p \in P\} \\ \{p_n | p_n > p_{n-1}, p_n = p_{n+1} \forall p \in P\} \\ \{p_n | p_n = p_{n-1}, p_n > p_{n+1} \forall p \in P\} \end{cases}$$
(10)

V consists of true valleys all left depressions and right depressions.

$$V = \bigcup \begin{cases} \{p_n | p_n < p_{n-1}, p_n < p_{n+1} \forall p \in P\} \\ \{p_n | p_n < p_{n-1}, p_n = p_{n+1} \forall p \in P\} \\ \{p_n | p_n = p_{n-1}, p_n < p_{n+1} \forall p \in P\} \end{cases} \qquad (11)$$

Another set of points exists where the middle point is equal to both the right and left adjacent points. There is no direct way to tell whether the point belongs to a plateau or depression. Call the set of these unknown points X.

$$X = \{p_n | p_n = p_{n-1}, p_n = p_{n+1} \forall p \in P\} \qquad (12)$$

Perform one final operation. The first epoch value has no previous entry, so it is excluded in the initial query. Label the second point by doing a compare with point 1 and assign peak or valley accordingly.

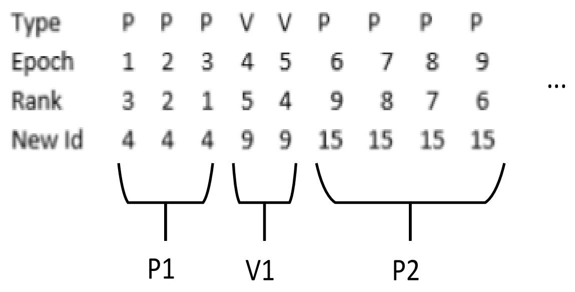
Step 2b ranks the results (number the rows 1, 2…) by value, epoch id and type (peak, valley or unknown). By adding the rank and original epoch ID, Step 2c is able to generate a new ordering. This causes the adjacent peaks and valleys to "clump" together.

Step 2d creates a dense rank by type for each clump. This creates a unique identifier for each clump and also forces unknowns between their corresponding peaks and valleys.

Step 2e calculates the endpoints for each clump by taking their corresponding maximum and minimum epochs.

Step 2f assigns unknowns to their corresponding peaks or valleys. If an unknown lies between the endpoints of a "peak clump" they are peaks, otherwise they are valleys. Take the midpoint of each unknown and declare that the peak or valley. Now all points are classified as peaks or valleys.

Step 2g regroups again by type (peaks or valleys) as shown in fig 10 Type line, ordering the result by beginning endpoint of the peak or valley. This gives a sort of peaks and valleys by epoch order for each starting end point as demonstrated in fig 10 Epoch line. Identify clumps this time by ranking the types in reverse order – fig 10 Rank line. Because they are sequential, each type row number plus ordering will amount to a unique number allowing a batch process to identify each clump uniquely as shown in fig 10 New Id line. At this point each clump is either a peak or valley with no like adjacent type, i.e. a local maximum or minimum. The maximum or minimum point for each clump is the true local maximum or minimum and the process can continue as before.



Type	P	P	P	V	V	P	P	P	P	
Epoch	1	2	3	4	5	6	7	8	9	…
Rank	3	2	1	5	4	9	8	7	6	
New Id	4	4	4	9	9	15	15	15	15	

Fig. 10 Batch Operation to Isolate True Peaks and Valleys

Another benefit of a relational database is it can handle simultaneous requests. This allows multiple disparate, lightweight fuzzy controllers the ability to engage a high performance server in tandem to handle heavier computational loads. Finally the database is persistent storage which allows for an additional Tabu-like search option and data mining.

V. Test Examples

Example 1: Working FIS optimization. In this example, a working FIS is run through the optimizer. 500 epochs was chosen as the number of steps to evaluate. Results were saved to a SQL Server 2012 database. Deviation from the "optimal" path by the original FIS was 2011 pixels.

The memetic algorithm is a combination of stored procedures and uses a sample of 30 randomly generated variations of a given fuzzy set within an FIS. It adjusts the endpoints of losers via crossover mutation with the best in the sample. Finally it performs a random mutation before reevaluating the results with the sample and seeing how the closely the new path tracked the optimal one. The best sample deviated by 1824 pixels from optimal. The resulting track showed sharper turns and fewer back and forth adjustments as shown in fig 11 and 12.

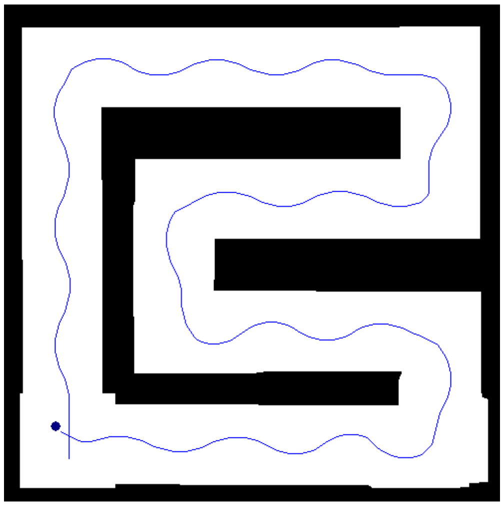


Fig. 11. A Robot Car Navigating a Maze, Typical FIS

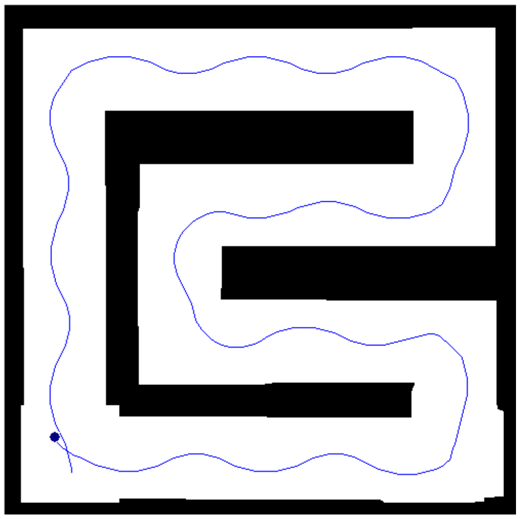
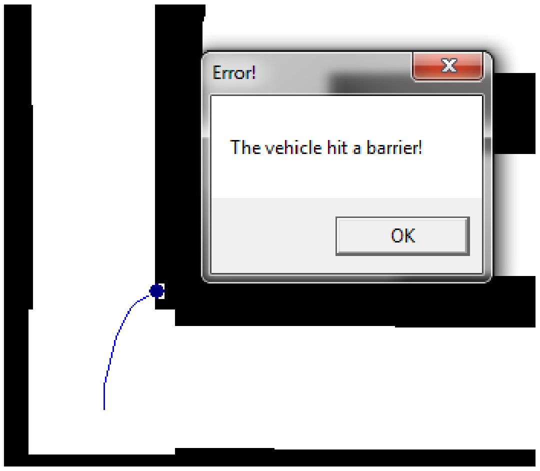


Fig. 12. A Robot Car Navigating a Maze, Optimized FIS

Example 2: Non-working FIS repair. A fuzzy set is adjusted to force the car to crash. Then the FIS was run through the optimizer which was able to find and adjust the bad fuzzy set, creating a working FIS as demonstrated in fig 13 and 14.



Error!

The vehicle hit a barrier!

Fig. 13. Bad FIS cause robot car to hit barrier

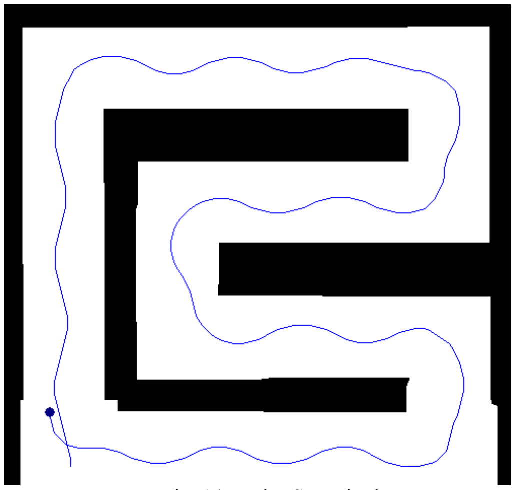


Fig. 14. Bad FIS repaired

VI. Conclusion and Future Work

The optimization method presented in this paper successfully improved the performance of both a working and non-working FIS. Further work needs to be done to allow greater flexibility and intelligence in how the memetic algorithm chooses which fuzzy sets to sample and mutate. Data mining methods could be used to determine which sets are the best candidates for sampling. Improvements to the fitness function are needed to handle those cases where optimal path fails to accurately represent errors, such as when the robot is navigating a donut shaped course.

VI References

[1] Zadeh, L A (1965) Fuzzy sets, information and control

[2] Zadeh, L (2008) Is there a need for Fuzzy Logic? Science Direct

[3] Omizegba EE, Adebayo GE (2009) Optimizing fuzzy membership functions using particle swarm algorithm IEEE International Conference on Systems, Man and Cybernetics, pp.3866-3870

[4] Martinez-Soto R, Castillo O, Aguilar LT, Melin P (2010) Fuzzy logic controllers optimization using genetic algorithms and particle swarm optimization, Advances in Soft Computing, Lecture Notes in Computer Science, Springer Berlin Heidleberg pp.475-486

[5] Han J, Kamber M, Pei J (2011) Data Mining: Concepts and Techniques 3rd Ed., Morgan Kaufmann Publishers

[6] McCarty K, Manic M, Gagnon A, (2013) A fuzzy framework with modeling language for type 1 and type 2 application development, 6th International Conference on Human System Interaction pp.334,341

[7] Surprise 96, Fuzzy Logic and Its Uses, Article 2 http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol2/jp6/article2.html, Accessed 29 Dec. 2013

[8] Mendel JM, John RIB (2002) Type-2 fuzzy sets made simple. IEEE Transactions on Fuzzy Systems C 10 (2) :117-127

[9] Cox E (1994) The fuzzy systems handbook, Academic Press, Chestnut Hill, MA

[10] Masaharu M, Kokichi T (1976) Some properties of fuzzy sets of type 2, Information and Control C 31 (4) :312-340

[11] Russell S, Norvig P (2009) Artificial Intelligence A Modern Approach, Prentice Hall 2009

[12] MacLennan J, Tang Z, Crivat B (2008) Data Mining with SQL Server 2008, Wiley Press

[13] Eiben AE, Smith JE (2007) Introduction to Evolutionary Computation, Springer-Verlag, Berlin