Fuzzy Contexts (Type C) and Fuzzymorphism to Solve Situational Discontinuity Problems

Kevin McCarty and Milos Manic

Abstract— **Generalized solutions to complex problems often suffer from being overly complicated. The main contribution of this paper is to describe an architecture that allows for greater problem generalization without the traditional corresponding increase in complexity. The architecture extends traditional fuzzy logic and is called Fuzzy Contexts or Fuzzy Logic Type-C. Fuzzy logic permits partial membership and values can belong to multiple fuzzy sets. By breaking down a problem space into smaller contexts and allowing algorithms themselves to have relaxed memberships in those contexts, a Type-C solution can support multiple solutions to complex problems. This paper describes how problem spaces may be decomposed into smaller, more easily solvable components and fuzzified together under a Type-C hierarchy. Test results with a simulated robotic navigation system demonstrates how a Type-C implementation is able to improve upon a generalized fuzzy controller.**

I. Introduction

An algorithm is "any well-defined computational procedure that takes some value or set of values as input and produces some value or set of values as output" [1]. In the decades since the introduction of the first digital computer, many kinds of algorithms were developed in order to solve specific classes of problems. For instance, when faced with a list of names, a developer may decide it necessary to sort them using a sorting algorithm such as a quicksort.

Quicksort is very useful for sorting problems [1] but would likely be a very poor application for a scheduling problem [2]. Hence, other algorithms are needed as problems and requirements change. Imagine a domain of problems in which some solutions are best served using a sorting algorithm and others using a scheduling algorithm as shown in fig 1.

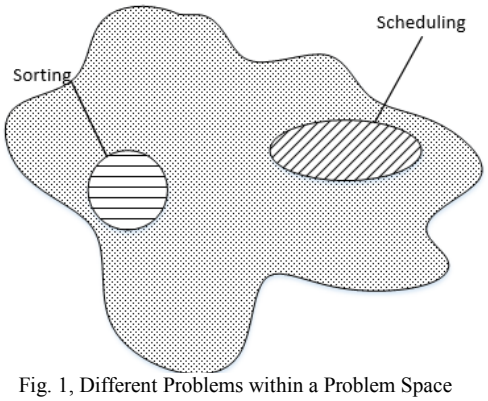

Fig. 1, Different Problems within a Problem Space

It is not uncommon in a larger domain to see different problem spaces overlap.

K. McCarty is with the University of Idaho, Idaho Falls, ID 83402 USA 208-282-7900; email:kmccarty@ieee.org.

M. Manic is with the University of Idaho, Idaho Falls, ID 83402 USA 208-282-7900; email:misko@uidaho.edu.

Even within a given problem space, circumstances can arise which introduce "situational discontinuity". Situational discontinuity occurs when the problem space, for example, a road, contains occurrences which change the resulting problem significantly, such as hitting a patch of ice. Because the subsequent behavior must be so different, it is effectively the same having a different problem space altogether. Living creatures are naturally well-equipped to adapt to Situational Discontinuity Problems (SDPs). A duck, for example, swims in the water and walks on land and flies through the air. Humans sweat when hot and shiver when cold and so on.

In the artificial world, handling SDPs becomes a matter of using different algorithms or generalizing a single approach. In data mining, for example, there are many algorithms used to find interesting information from huge, often disparate data sets [3]. An experienced data miner needs to be familiar with Decision Trees, Neural Networks, Linear Regression and a whole host of other algorithms, each of which has advantages depending upon the underlying patterns in the data [4]. Something as simple as a fuzzy thermostat may have some rules for temperature, other for humidity and still others for time of day in order to handle many different demands for climate control.

Intuitively, it seems obvious that different classes of problems require different approaches, but the problem with SDPs is that they tend to be ambiguous, hence it can be difficult to determine when an SDP has occurred and what to do about it. A fuzzy controller trying to navigate a maze must already deal with a number of navigation problems without also having to negotiate obstacles such as ice and potholes that it may or may not encounter. Ideally there would be a generalized contextual approach capable of handling all the underlying SDPs encountered; one that was efficient, easy to understand and implement. Psuedocode for such an approach might look something like the following:

result TYPE-C_EVALUATE (contexts, tuple)
inputs: *contexts*, a set of fuzzy contexts in which each context represents a problem scenario, such as ice, potholes, smooth, etc.
: *tuple*, set of values representing measurements or state of process
output: crisp result

Step 1: Test each context to see if it is valid for this state
Step 2: For each valid context:
Determine the corresponding weighting of this context
Determine the membership value for this context
Run the corresponding context algorithm against *tuple*
Step 3: Combine algorithm results, weight and membership values to get ***result.*** Return ***result***

Fig. 2, Pseudocode for a Generalized Contextual Approach

This approach is useful to avoid the complexity problems of generalized algorithms [5]. This paper presents a novel implementation of this approach called Fuzzy Logic Type-C and is organized as follows: Section II presents the problem statement, limitations and prior work in a number of Fuzzy techniques. Section III introduces the Fuzzy Context and how it can be applied to solve the SDPs in a complex problem space and describes the architecture of a Type-C application. Section IV presents test results demonstrating the advantages of a Type-C based implementation. Section V presents conclusions and future work.

II. Problem Statement and Prior Work

Describing the behaviors of complex systems present many challenges for the software architect and developer. Foremost among them is the ability to model behaviors that are by their very nature imprecise [6]. In the "crisp" world, this is a particularly difficult task since even a small number of inputs requires a complex equation in order to create a smooth, continuous result. In particular, crisp solutions have difficulty properly describing behavior at boundaries [7]. Consider what the graph of a thermostat temperature might look like using crisp definitions of COOL, WARM and HOT as seen in fig 3.

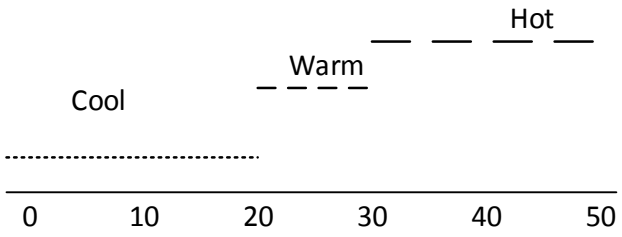

Fig. 3, Crisp Definitions for a Thermostat Control

Discontinuity at the boundaries must be smoothed in order for the function to prevent hyper-oscillation around those values. Fuzzy logic, also known as Type-1 Fuzzy Logic, introduced by Lofti Zadeh [6]-[8] addresses these problems by approximate, rather than precise, descriptions for terms and allows for polyvalent membership definitions. In the crisp definition above, 19.999 degrees is COOL while 20 degrees is WARM. In a fuzzy definition, 19.999 degrees is, to an extent, COOL, and also, to an extent, WARM. As compared to a crisp controller, a fuzzy controller allows for greater linguistic precision in describing a complex system behavior while at the same time relaxing precision around the boundary points and elsewhere. This is done through the use of a membership function μ whose output, instead of the traditional FALSE (0) and TRUE (1) allows for output of 0, 1 and all values in between. Thus, for a domain *D*

$$\mu(x) \rightarrow [0,1], x \in D \quad (1)$$

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

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

A fuzzy set can take any convex shape, with each fuzzy set depending upon its membership function. Triangles are one common shape. The prior crisp definition for the thermostat temperature becomes a union of fuzzy sets as shown in fig 4.

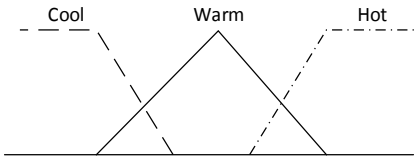

Fig. 4. Fuzzy Sets as Triangles

Each fuzzy set contributes partially to the final result depending upon the resulting $\mu(x)$. From the example above, 19.99 degrees centigrade might be classified as 60% COOL and 40% WARM since the sets overlap.

Fuzzy algorithms are very good at approximating complex polynomials and provide stronger mechanisms for handling noise and uncertainty along with variations among "expert" definitions than their crisp cousins.

However, fuzzy logic also has limitations that pose new problems. Whereas the crisp algorithm has difficulty with the discontinuity at a boundary, likewise a fuzzy algorithm has trouble handling an SDP such as when an obstacle presents itself.

In the thermostat problem, a crisp solution could be improved by adding additional temperature tiers [7]. Likewise an SPD could be improved by the addition of fuzzy rules. However, adding tiers makes the temperature algorithm significantly more complex; likewise the addition of fuzzy rules adds significant additional complexity to a fuzzy solution [5]. Just as Fuzzy Logic was necessary to solve the crisp boundary discontinuity problem, so there is a need for an approach to solve situational discontinuities within SDPs.

The underlying problem within fuzzy systems, and more generally, all approaches, is that certain problem domains are more solvable using certain approaches than others. Within each of these specific problem areas often lies even more specific issues which require ever more specialized techniques.

For instance [9, 10] demonstrates how a Fuzzy Type-1 controller was superior navigating around corners but inferior to a Fuzzy Type-2 controller navigating smoother surfaces. Even within a particular problem domain, one configuration of a Fuzzy Inference System (FIS) will be superior for handling a simple maze while another FIS is more appropriate elsewhere for obstacles.

A number of methods were introduced to extend fuzzy systems while also trying to limit the corresponding increase in complexity. Prior work in this area involves the use of Fuzzy Type-2 [9]-[12] and Nonstationary Fuzzy [13] sets and Polymorphic Fuzzy Signatures [5], [14].

Fuzzy Type-2 introduces uncertainty into the fuzzy sets themselves, in effect relaxing the boundaries of the membership function μ_2, so in contrast to Equation 1:

$$\mu_2 = \{((x, \mu), \mu_2(x, \mu)) | \forall x \in D, \mu \in [0,1]\} \quad (3)$$

Note also that output of μ_2 is also member of the set [0, 1]. Whereas a Type-1 fuzzy set is a 2-dimensional object, Type-2 fuzzy sets are surfaces as demonstrated in fig 5.

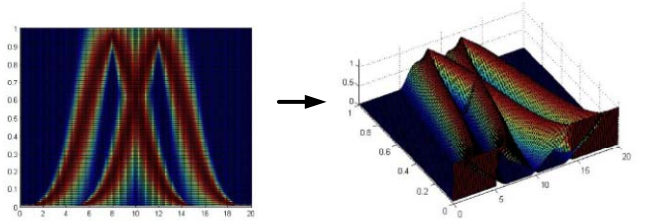

Fig. 5 Type-2 Fuzzy Sets

A Type-2 fuzzy interference system is useful in dealing with problems such extensive noise or smoothing out erratic behaviors that plague Type 1 controllers.

Nonstationary Fuzzy Sets (NFS) introduces the notion of variability of fuzzy sets over some dimension such as time, location, or even noise. The NFS nf_s is described as

$$nf_s = \int_{d \in D} \int_{x \in X} \mu_{fs}(d, x)/x/d \qquad (4)$$

Where d is some value along a dimension of the problem domain D and x is a tuple or point within the set of possible inputs X. Nonstationary Fuzzy Sets allow for a dynamic fuzzy membership function (and sets) able to accommodate significant changes to the problem space. A perturbation function adjusts the underlying membership functions as needs change. The NFS is then able to generate a variable FIS to handle changes in the problem space which otherwise might cause difficulties to a static Type-1 or Type-2 FIS.

Finally, Polymorphic Fuzzy Signatures (PFS) describe a multidimensional fuzzy tree of fuzzy sets where each leaf contains a specific fuzzy membership function. Fuzzification occurs by traversing the branches whose meta-fuzzy signature indicates membership and combining the applicable membership functions at the leaves using traditional fuzzy functions such as *max* and *min*. The polymorphic fuzzy signature is described as:

$$\mu_{sig}: X \to [c_i]_{i=1}^{k} \left(\equiv \prod_{i=1}^{k} c_i\right) \qquad (5)$$

$$\text{where } c_i = \begin{cases} [c_{ij}]_{j=1}^{k}; if\ branch\ (k_i > 1) \\ [0,1]\,; if\ leaf \end{cases}$$

Polymorphic fuzzy signatures allow one to break down an SDP into smaller, easier to describe, components. Each component is then associated with a particular FIS and signature. Each FIS output is fuzzified, with the resulting defuzzification using a traditional methods.

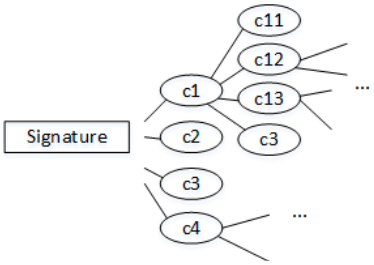

Fig. 6 Polymorphic Fuzzy Signature Tree

III. Fuzzy Contexts

Fuzzy Contexts, or Fuzzy Type-C extends the concepts of the Fuzzy Logic, PFS and NFS into a rich framework supporting the use of multiple, distinct approaches to solve varied problems.

Traditional approaches take an existing algorithm and try to generalize over a larger problem domain. Typically this is accomplished by adding complexity to the algorithm. For instance, a neural network adds neurons while a fuzzy controller adds fuzzy rules, both at a cost of complexity to an algorithm with correspondingly diminishing returns [5]. For example a simple fuzzy controller designed to solve a navigation problem can perform quite well with a small number of rules. Adding new rules gives the controller more capability, but each new rule expands the solution on a smaller and smaller scale. Conversely each new rule greatly increases the system's complexity [14]. Even within a problem space suited for a particular approach, an algorithm can still fall victim to problems of complexity and diminishing returns as illustrated by fig 7.

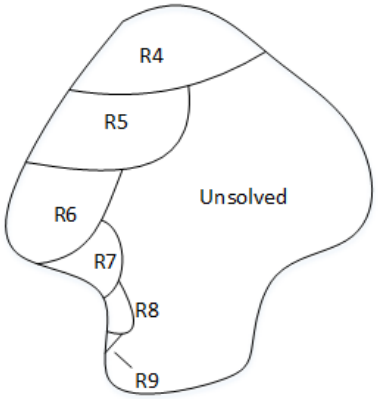

Fig. 7 Adding Fuzzy Rules Results in Diminishing Returns

Furthermore, there are times when multiple approaches may be equally worthy at certain stages of a process. Consider the example of a scheduler - one with a small number of possible configurations may be best served with a global ranking system; while a larger number of configurations may require some sort of local search technique [15]. The effectiveness of these different approaches can overlap, creating an intersection of subdomains as demonstrated in fig 8.

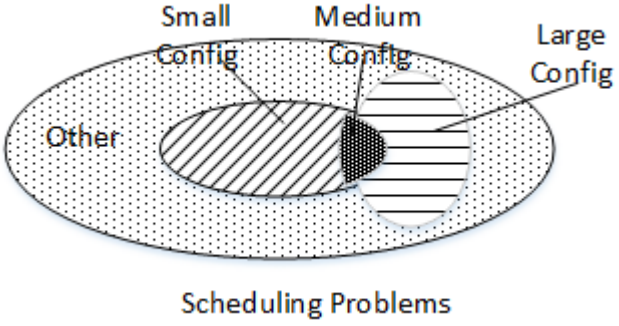

Fig. 8, Overlap of Techniques for a Scheduler

In this situation, either approach is acceptable. More importantly, the union of the two spaces gives the combined algorithms a larger surface area with less overall complexity than trying to extend either approach separately. The problem lies in determining the situation, or "context" in which to apply one algorithm or the other. For another example of how a context applies, albeit in a different way, consider an inventory control problem at a department store. Seasonal contexts dictate which items are most import to maintain inventory and how much. As before with the scheduler, an inventory control system needs to account for the season, or context, in order to be most efficient, this time at maintaining inventory levels.

Living things incorporate contexts quite well, we are naturally in tune with our external situation; but applying contexts to artificial processes requires a mechanism to both identify a given context as well as the best algorithmic behavior to apply, along with behavior at transition points where the "best" algorithm is ambiguous. Fuzzy logic provides a useful foundation for exploiting this imprecision and creating and using contexts [8]. With Fuzzy Type-C, diverse problem spaces can be combined without sacrificing the simplicity and power of individual problem solving techniques. Consider a problem space *P* over a domain *D*. It consists of a collection of states *s*, which is a tuple of s_i values, each s_i value belonging to *D*.

$$P \equiv \{s = s_1, .., s_n, s_i \epsilon D\} \quad (6)$$

An algorithm *a*, such as a Type-2 FIS operates on *P* taking as input an s_p and generates a result r_p.

$$a \equiv f(s), f(s_p) = r_p \quad (7)$$

A fuzzy "signature" is a collection of problem states, upon which the algorithm works efficiently, hence:

$$f_{sig} = \sum_{i=1}^{n} a_{s_i}, s \in P, s_i \in D \quad (8)$$

The "context" is the combination of the algorithm, the signature and all the associated states along with a membership function μ_c. μ_c determines membership within a given context of any particular state s_i.

$$C_i = \{f_{sig_i}, a_i, \sum s_i, \mu_{c_i}, s_i \in P, \mu_{c_i}(s_i) \in [0,1]\} \quad (9)$$

The Type-C FIS contains all the contexts associated with *P*.

$$C = \sum_{i=1}^{n} C_i \quad (10)$$

Fig 9 demonstrates how this approach might handle a complex problem domain.

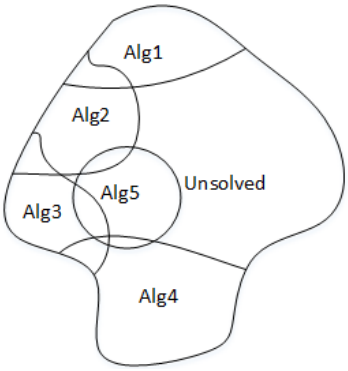

Fig. 9 Solving a Problem Space Using Multiple Approaches

Hence, Fuzzy Type-C encapsulates multiple problem solving approaches by associating a "signature" of an environment with a distinct Type-1, Type-2 FIS or other algorithm and all the potential states the algorithm was designed to address.

Because the new contexts can be added as a problem space expands, a Type-C FIS allows an expansion of a problem space into a larger domain without having to overly generalize.

The Type-C architecture consists of the following major components:

1. A set of inputs as a tuple
2. A series of contexts. Each context consists of:
    a. Fuzzy signature
    b. Membership function
    c. Algorithm that receives the input tuple and returns a result.
3. Results fuzzifier
4. Optional optimizer/contextualizer for dynamic optimization and automated learning. It determines if the error rate is acceptable, otherwise will strive to optimized an existing context or generate a new context.
5. Defuzzifier that takes fuzzified output and generates a crisp result.

The architecture is illustrated by fig 10.

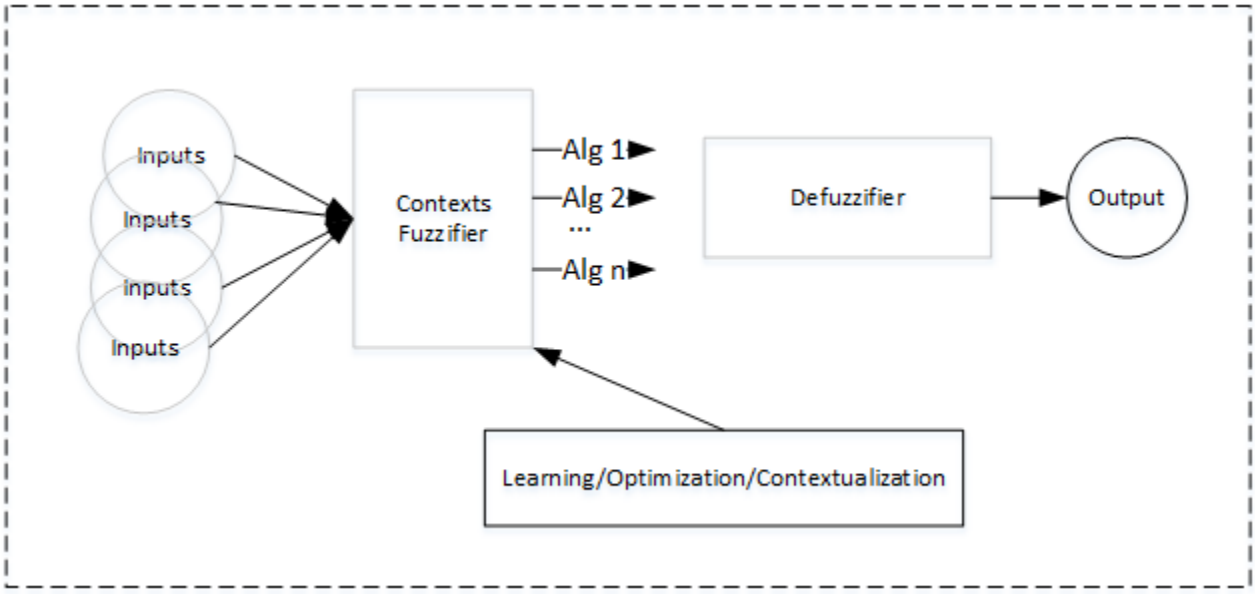

Fig. 10 Fuzzy Context Architecture

The Contexts Fuzzifier component uses a technique similar to that of fuzzy classification to determine membership of a context. Unlike in crisp sets where a data point is either in or not in a set, fuzzy classifications allow a point to have membership in multiple sets as shown in fig 11.

Fuzzy clusters are very useful in creating "transition" sets from one problem space to another within a domain. Likewise fuzzy clusters can determine membership of a given tuple with a context. Cluster, and context, creation and membership is determined using techniques such as discussed in [16].

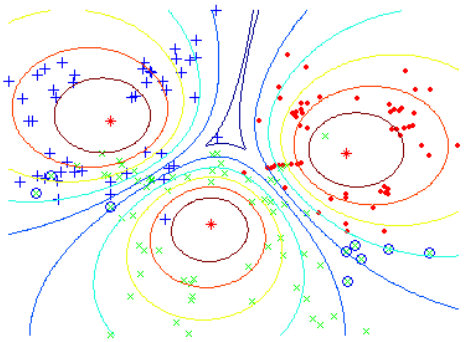

Fig. 11 Points in Fuzzy Clusters

At runtime a Type-C FIS determines the "contextualization" of each input tuple using each context's corresponding membership function. Any context whose membership value is greater than zero will have its corresponding algorithm run and its output fuzzified. Defuzzification is achieved using traditional fuzzy methods

$$\Re = \frac{\sum_{i=0}^{n} w_i a_i(s_i)\mu_{c_i}(s_i)}{\sum_{i=0}^{n} \mu_c(s_i)}, s_i \in D, \mu_C(s_i) > 0 \qquad (12)$$

Where $\Re$ is crisp result, s_i is an input tuple in a domain D, w_i is the weight, μ_c is the context membership and a_i is the intrinsic function for any context whose μ_c is greater than zero.

Fuzzymorphism occurs when contexts overlap as might occur in a problem space similar to that in fig 9. As a state moves away from the center of one context and closer to the center of another, the resulting defuzzification will take on more of the characteristics of the underlying context algorithm, hence a Type-1 FIS might slowly morph to a Type-2 FIS for example. Fig. 12 illustrates the concept.

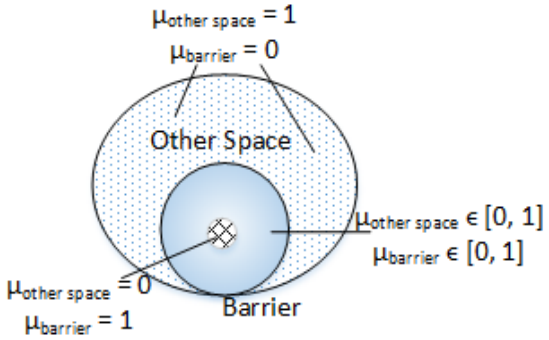

Fig. 12 Type-C membership over a Problem Space

Fuzzymorphism allows a Type-C based system containing multiple approaches to dynamically "morph" into the one most suitable for the problem at hand. In the case of multiple Type-1 FIS, Type-C performs similarly to a Nonstationary Fuzzy System. However, because Type-C is algorithm independent, the framework will support any algorithm capable of accepting the input tuple and producing a corresponding output, allowing for a much more diverse approach.

IV. Test Results

For test purposes an existing Fuzzy Type-1 framework [17], [18] with 6 fuzzy rules was used to navigate a robot car around a maze. The car has an easy time navigating the simple maze as shown in fig 12, but has an equally difficult time when obstacles are introduced, such as those shown in fig 13. The SDP, in the form of the obstacles, causes the car to act as if a walls exists and reverse course rather than navigate around. As shown in fig 13, the car failed to navigate the two small obstacles, effectively running in circles.

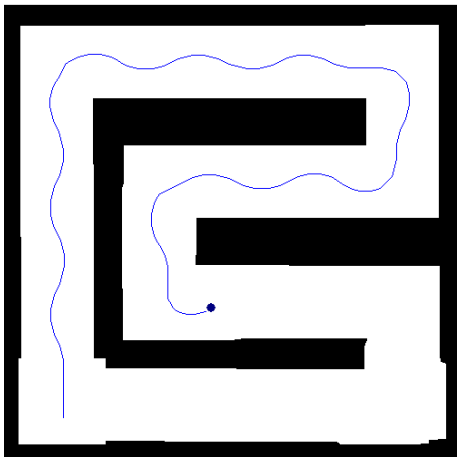

Fig. 12. A Robot Car Navigating a Maze

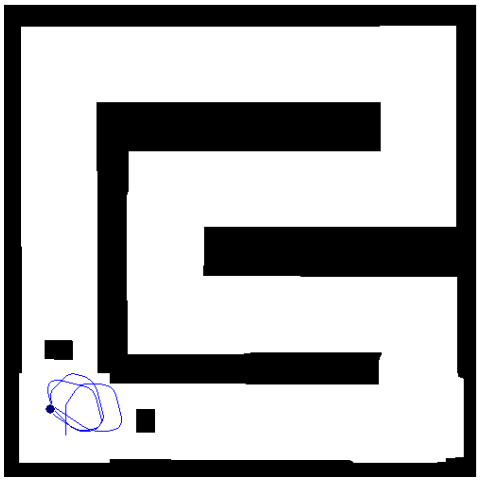

Fig. 13, Robot Car is Unable to Handle SDP in the Form of Obstacles

Next, the controller was modified using a Type-C architecture. A second controller was introduced, consisting of 4 fuzzy rules. Its job is to explore obstacles at close range. Two contexts were created, one named OBSTACLE with the second controller for operation near obstacles, the other named OTHER_SPACE for the original controller operating everywhere else. It is useful to have a context that is defined as the complement of all other defined contexts.

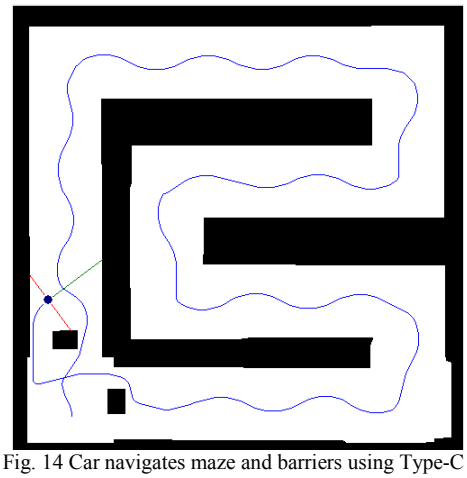

Fig. 14 Car navigates maze and barriers using Type-C

Once the second controller was added under the Type-C construct it became active and fuzzymorphically took over as the car neared the barrier, before it was forced into a 180 degree turn. Under Context OBSTACLE, the car instead explores along the edge of the obstacle. Upon finding an opening, it executes a sharp turn and proceeds until the space opens up again and it can fuzzymorphically revert back to the Context OTHER-SPACE and resume normal operations.

V. Conclusion and Future Work

Future work needs to be done to demonstrate Type-C utility with other algorithms, such as Fuzzy Type-2, Swarm or Ant Colony optimizations, artificial neural networks and others for different classes of problems. Currently work is being done to improve Type-C for machine learning and optimization, in particular for knowing when it is most appropriate to identify and generate new contexts. A memetic algorithm and database-driven error and fitness function are under development to determine an allowable error threshold which can be used to set the boundaries and describe the resulting context. Finally, additional work will incorporate other methods of fuzzification and defuzzification within a given context.

VI References

[1] Cormen T, Leiserson C, Rivest R, Stein C Introduction to Algorithms 3rd Ed., MIT Press 2009

[2] Sipser M, Introduction to the Theory of Computation 3rd Ed. Thompson Course Technology, 2102

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

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

[5] Mendis B, Gedeon T "Polymorphic fuzzy signatures," IEEE Conference on Fuzzy Systems July 2010

[6] Zadeh, L A (1965) Fuzzy sets, Information and Control

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

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

[9] O. Linda, M. Manic, "Fuzzy Force-Feedback Augmentation for Manual Control of Multi-Robot System," in IEEE Trans. on Industrial Electronics, Aug. 2011

[10] O. Linda, M. Manic, "Uncertainty-Robust Design of Interval Type-2 Fuzzy Logic Controller for Delta Parallel Robot," in IEEE Transaction on Industrial Informatics, Nov. 2011

[11] Mendel JM, John RIB (2002) Type-2 fuzzy sets made simple. IEEE Transactioins on Fuzzy Systems

[12] Hagras H, Wagner C (2012) Towards the wide spread use of type-2 fuzzy logic systems in real world applications. IEEE Computational Intelligence Magazine

[13] Garibaldi, Jonathan M., Marcin Jaroszewski, and Salang Musikasuwan. "Nonstationary fuzzy sets." Fuzzy Systems, IEEE Transactions on (2008)

[14] B. S. U. Mendis, "Fuzzy signatures: Hierarchical fuzzy systems and applications," Ph.D. dissertation, College of Engineering and Computer Science, The Australian National University, Australia, 2008.

[15] Russell S, Norvig P, Artificial Intelligence A Modern Approach, Prentice Hall 2009

[16] Ming-Chuan Hung; Don-Lin Yang, "An efficient Fuzzy C-Means clustering algorithm," 2001, Proceedings IEEE International Conference on Data Mining

[17] AForge.NET Framework. http://www.aforgenet.com/aforge/framework/ Accessed 12 Jan 2013

[18] K. McCarty, M. Manic, "A Fuzzy Framework with Modeling Language for Type 1 and Type 2 Application Development," in Proc. of IEEE 6th International Conference on Human System Interaction, 2012