Fuzzy Algorithms: Applying Fuzzy Logic to the Golden Ratio Search to Find Solutions Faster

Stephany Coffman-Wolph
West Virginia University Institute of Technology
405 Fayette Pike, Montgomery, West Virginia 25136
sscoffmanwolph@mail.wvu.edu

Abstract

Applying the concept of fuzzy logic (an abstract version of Boolean logic) to well-known algorithms generates an abstract version (i.e., fuzzy algorithm) that often results in computational improvements. Precision may be reduced but counteracted by gaining computational efficiency. The trade-offs (e.g., small increase in space, loss of precision) for a variety of applications are deemed acceptable. The fuzzification of an algorithm can be accomplished using a simple three-step framework. Creating a new fuzzy algorithm goes beyond simply converting the data from raw data into fuzzy data by additionally converting the operators and concepts into their abstract equivalents. This paper demonstrates: (1) how to apply the general framework by developing a fuzzy algorithm for a simple linear search algorithm and (2) the success of this process through the development of the Fuzzy Golden Ratio Section Search.

Background

Fuzzy logic is a set of rules and techniques for dealing with logic beyond a two-value (yes/no, on/off, true/false) system. In other words, fuzzy logic is an abstraction of two-value logic and allows for not only multiple values but also an overlap of values between fuzzy sets. Therefore, fuzzy logic can mimic a more human-like approach to decision making. L. Zadeh introduced fuzzy logic in the 1960s as an expansion of Boolean logic in his paper entitled "Fuzzy Sets" (Zadeh 1965).

Applying the concepts of fuzzy logic to an algorithm essentially creates an abstract version of the original algorithm. The abstract version allows for faster processing by taking advantage of integer calculations, reduced search trees, etc. A fuzzy algorithm can (potentially) find multiple similar solutions while the non-fuzzy version of the algorithm generates only one solution. Fuzzification of an algorithm is extremely advantageous when components or solutions naturally contain a high level of similarity/symmetry or form natural groups/clusters.

This process has already been successfully applied to the following algorithms: (1) a concept-oriented fuzzification for finding fuzzy patterns (Coffman-Wolph & Kountanis 2013c), (2) a fuzzification of both data and the operators for a fuzzy process particle swarm optimization (FP2SO) (Coffman-Wolph & Kountanis 2013a), (3) a fuzzification of multiple algorithm components to find strategies for adversarial situations from game theory (Coffman-Wolph 2013), and (4) a fuzzification of the simple simplex method for the transportation problem (Coffman-Wolph & Coffman, Jr 2014). All of the above fuzzy algorithms were generated using a simple three-step framework (Coffman-Wolph & Kountanis 2013b). Similarly, other researchers have used fuzzy logic controls with various evolutionary algorithms (Sabzi, et al 2016). The author previously asserted that the general framework technique could be successfully applied to many other algorithms to take advantage of the power of fuzzy logic. This paper demonstrates the successful application of the framework to one-dimensional search algorithms with the Golden Ratio Section Search used to illustrate the process.

Fuzzification of Algorithms

There are three main categories of components within an algorithm that can be fuzzified: (1) data, (2) operators, and (3) concepts. As stated earlier, fuzzification is a method of adding abstraction. The fuzzification of data is the process of taking "raw"/non-fuzzy data and converting it into fuzzy data. The fuzzification of operators is the process of converting a mathematical, logical, or comparative operator to its fuzzy counterpart, which operates on fuzzy sets instead of pure numbers. The fuzzification of concepts, the most difficult of the three, is the conversion of an idea into a similar fuzzy idea. These three techniques, together, will be used within the framework to create a fuzzy algorithm.

Fuzzification of Data

The process of fuzzifying data (also known as fuzzification of data) is a simple two-step process. The first is scaling or normalizing the data. The second is assigning each data point to a fuzzy set. When the process is complete the data is ready to be used within a fuzzy algorithm.

Begin by scaling each of the data points. For easier conversion from raw data to fuzzy data, scale or normalize the data first. There are many ways to scale data. For example, scale each data point x_i, using the following standard normalization equation:

The Normalized value:
$x_{inorm} = ((x_i - \min) / (\max - \min))$,
Where:
max is the maximum value of all data points
min is the minimum value of all the data points

After the data is normalized, the fuzzification process can begin. There are two important decisions that need to be made during this phase: (1) the total number of fuzzy sets and (2) the amount of overlap between the sets. The higher the number of fuzzy sets and the smaller the overlap between the sets, the more precise the system is (i.e., less fuzziness). If each fuzzy set only represents one value and there is no overlap, then the system is essentially the same as the original non-fuzzy version. Often these two values are left as user defined parameters in the final algorithm.

There are numerous ways to define fuzzy sets. It is common to define the fuzzy sets using an equation. It is possible to have "uneven" fuzzy sets (i.e., the fuzzy set size is not a constant or the overlap is not consistent). However, the most common approach to create fuzzy sets is to divide the range of values evenly with consistent overlap.

Fuzzification of Operators

Any calculation using fuzzy data requires that all operators are also fuzzified. Changing all the operators from the traditional version to the equivalent fuzzy version causes the end calculation to be more abstract. The term operator does not simply refer to mathematical operators, but also logical and comparative operators.

Writing fuzzy operators can be challenging on a number of levels and requires many decisions to be made. For example, when defining the comparative operators, decisions will need to be made regarding what makes two sets equal, not equal, less than, greater than, etc. Additionally, a fuzzy operator needs to deal with the overlap that can exist between fuzzy sets.

Within fuzzy logic there exists several predefined operators – the majority of them are logical operators. Each of these well-defined fuzzy operators' behavior is based on the equivalent operators from traditional set theory. Operators are most commonly associated with numbers, but in the fuzzy world, operators can deal with a greater range of data representations. Often the user needs to expand these predefined operators into special operators specifically defined for the context of the data.

Fuzzy operators can either be written as an equation or as a set of if/then rules. Both methods are common. Equations are computationally faster than if/then rules, providing greater speed advantages within an algorithm. On the other hand, if/then rules are often easier to write, providing an advantage design-wise.

Fuzzification of Concepts

The fuzzification of a given concept is the most challenging of the three presented in this paper mainly because a concept can be difficult to define. A concept is an essential element from the algorithm. Like the fuzzification of data or an operator, the purpose is to create an abstract version of the non-fuzzy "concept". To identify concepts that are candidates for the fuzzification process, look for places where items could be represented by a set or look for items that only differ by a small amount. Another way to identify concepts that would benefit from being fuzzified is to notice repeated information or items that could be categorized. Fuzzification (i.e., the creation of fuzzy sets) lends itself well to these types of situations.

Fuzzy Algorithm

Many researchers refer to a "fuzzy algorithm" as one in which the data has been fuzzified (for example: Koteshwariah et al 2015). This paper excludes algorithms where only the data has been fuzzified resulting in a more strict definition of a fuzzy algorithm. Specifically, to be considered fuzzy, an algorithm must go beyond simply using fuzzy data. To truly make the algorithm fuzzy, either the operators and/or the concepts within the algorithm need to be fuzzified. When the algorithm is modified to includes a fuzzy concept, the algorithm is undeniable a fuzzy algorithm. It is trickier to determine if the algorithm is fuzzy when only the operators have been fuzzified because of the fine line between a traditional operator and a fuzzy operator – even when operating on fuzzy data. A fuzzy operator is differentiated from the basic operator by being re-written to accommodate the meaning of the fuzzy data.

Fuzzy Algorithm Framework

The following describes the general framework for the fuzzfication of any algorithm and provides the steps to convert any algorithm from a traditional non-fuzzy version to a fuzzy version. The framework is written as a general procedure so it can be applied to a broad spectrum of algorithm types.

The three steps of the framework:

1. Decide what can/should be fuzzified and determine the category of each piece to be fuzzified
2. Fuzzify each piece (identified in step 1) based on the category
   a. Scale/normalize the data, then fuzzify the data
   b. Fuzzify the operators
   c. Fuzzify the concepts
3. Defuzzification (if needed/applicable)

The first step is to make an in-depth examination of the algorithm (which is to be fuzzified) and decide which elements to fuzzify. It is helpful to determine what the solution will look like in the fuzzy form. This will lay the foundation for the entire fuzzified algorithm. Therefore, it can be helpful to work backwards through the algorithm to determine what elements (data, operators, or concepts) need to be changed in order for the solution to be produced.

As mentioned in previous sections, data, operators, and concepts can be fuzzified. Simply fuzzifying the data is not necessarily a complete fuzzification of an algorithm. Conversely, not everything needs to be fuzzified. The challenge is selecting which elements should be fuzzified and which elements should not. Almost everything numeric could be fuzzified (by the process provided in the section on fuzzification of data). Elements that are essential to the fundamentals of the algorithm are generally not fuzzified. For example, in an algorithm for finding the shortest path from a starting location to a destination location, the starting and ending locations should not be fuzzified. However, the distances between locations could be fuzzified and, additionally, the concept of ordering might be fuzzified. Of course, these decisions may depend greatly on the application of the algorithm.

The final optional step is defuzzification. Defuzzification, the opposite of fuzzification, is used to convert the result back to a non-fuzzy result. In some cases, this step might be unnecessary. The determination of needing defuzzification is both problem dependent and usage dependent. If the algorithm is being used to narrow the solution space and the information will be fed into a non-fuzzy algorithm, then it is important to develop a method of defuzzification. Defuzzification can be applied to data or concepts.

The Golden Ratio Section Search

In this paper, the Golden Ratio Section Search will be fuzzified. The Golden Ratio Section Search belongs to a class of problems known as the one-dimensional search algorithms. One-dimensional search methods are used to find the minimum or maximum of a function within a particular interval. Other similar searches include: Dichotomous Search (bisection method), Uniform Search Method, Equal Interval Method, Fibonacci Search Method, Quadratic (or polynomial) Interpolation, and Newton's Method. The Golden Ratio Section Search has the advantage of faster convergence than most simple algorithms without requiring function derivatives making the method more versatile.

For this paper, two problems will be run on both a fuzzy and traditional version of the Golden Ratio Section Search. The two versions will be compared in terms of computation complexity, number of iterations, and an evaluation of advantages and disadvantages. A walkthrough will be provided so the reader and compare the fuzzy and non-fuzzy versions of the algorithm.

Application of One-dimensional Search Algorithms

There are many types of problems that can be formulated so that searching for a minimum or maximum of a function yields the desired result. Thus, this type of search has far reaching applications. One example is finding the roots of a polynomial, which can be challenging if $n>3$. One-dimensional searches are used to find the "fixed point" of a function (i.e., given a function $f(x)$, then a fixed point c is a point that satisfies $f(c) = c$). Similarly, it can be used to find the "zeros" of a function (i.e., given a function $f(x)$, then a zero of the function is any point where $f(x) = 0$). In non-linear constrained or unconstrained optimization, one-dimensional search algorithms are used to find the step size to move in the "improving" direction (e.g., in steepest descent, the improving direction is the gradient).

The (Non-Fuzzy) Algorithm

Golden Ratio Section Search works similar to most bisection searches but there is more than one internal point. (And, thus, more than two segments). One common application of this algorithm is to find the absolute minimum (or maximum) of a mathematical function. The algorithm begins by finding (or calculating) the upper and lower bounds of the area that contains the minimum (or maximum) value. Calculate the two interior points using the bounds forming three sub-segments. The algorithm then decides which sub-segment contains the minimum (or maximum) value. The upper and lower bounds are narrowed and the process is repeated until the minimum (or maximum) value is located within a specific tolerance. The Golden Ratio is defined as follows:

Golden Ratio = $\varphi = (1+\sqrt{5})/2 \approx 1.6180339$

The Golden Ratio Section Search uses τ:
$\tau = (\sqrt{5}-1)/2 \approx .6180339$
(Which is the Golden Ratio minus one)

This value, τ, will be used to subdivide the line such that the ratio of the lengths of the segments is the Golden Ratio. With a lower bound of LB and an upper bound of UB, calculate the two points x_1 and x_2 that subdivide the line into three segments. The general formulas for this are:
x_1 = LB + (1 - τ)(UB-LB)
x_2 = LB + τ(UB-LB)

The Fuzzy Algorithm

As mentioned in an earlier section, the preliminary research for fuzzy algorithms began with a fuzzy PSO for solving the Traveling Salesperson problem (Coffman-Wolph & Kountanis 2013a) and continued in the author's dissertation (Coffman-Wolph 2013). This research continues exploring what algorithms can benefit from being fuzzifed using the framework.

To fuzzify the Golden Ratio Search, we must consider what should and should not be fuzzified. A fuzzy algorithm produces a fuzzy solution (unless a defuzzification process is applied to the solution). The fuzzy solution for this type of problem is a fuzzy minimum (or maximum). Thus, we must ask ourselves if a precise minimum (or maximum) is required. Often, a precise minimum (or maximum) is not required. Generally, the Golden Ratio Search (and other bi-sectioning algorithms) are used as a feeder into other algorithms (e.g., curve fitting, finding minimum of a polynomial) and do not require the absolute best solution – a "good enough" solution is all that is required. In some cases, the fuzzy version of an algorithm can be used to eliminate a significant portion of the search space and then be used as a starting range in a precise non-fuzzy algorithm to find the absolute minimum (or maximum). We can conclude that the concept of fuzzy minimum (or maximum) is an appropriate concept to be fuzzified for the algorithm.

The next concept to consider for fuzzification is the calculated points. For the traditional Golden Ratio Section Search, we calculate 2 precise points (and thus create 3 exact segments). In the fuzzified version, these 2 precise points would be represented by 2 fuzzy sets (and 3 fuzzy segments). These sets are not necessarily distinct and may contain overlap. Earlier, the crisp calculations were provided to find the 2 precise points. These formulas, operators, and values will be converted to an abstract fuzzified version. (A subscript f will be used to denote a fuzzified value or operator. Note that 1_f is a "fuzzy" 1 and is actually a set). The value of φ and τ will remain crisp and non-fuzzified as these values are essential to the algorithm.

Golden Ratio = $\varphi = (1+\sqrt{5})/2 \approx 1.6180339$
$\tau = (\sqrt{5}-1)/2 \approx .6180339$
$x_{1f} =_f LB_f +_f (1-\tau) *_f (UB_f -_f LB_f)$
$x_{2f} =_f LB_f +_f (\tau) *_f (UB_f -_f LB_f)$

Example #1: Simple Polynomial

The Fuzzy Golden Ratio Section Search will first be tested using the problem: find the minimum of the polynomial: x2 - y +2 = 0 (see figure 1). The lower bound (LB) and upper bound (UB) will be set to -3 and 3 respectively. This polynomial will be used to provide the reader with a full example for comparison purposes of the non-fuzzy traditional algorithm and the fuzzified version.

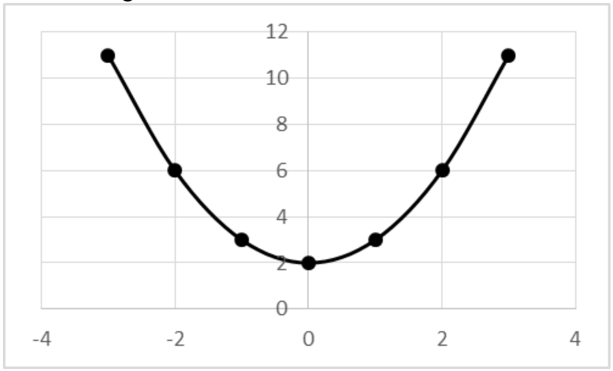

Figure 1: $x^2 - y + 2 = 0$.

Non-Fuzzy Walkthrough

Using the bounds -3 and 3, begin by calculating the two points x_1 and x_2 to subdivide the line:
x_1 = LB + (1 - τ)(UB-LB) = -3 + (1-.618)(3-(-3)) = -.708
x_2 = LB + τ(UB-LB) = -3 + .618(3-(-3)) = .708

There are now three line segments (-3 to -.708), (-.708 to .708), and (.708 to 3). Evaluate the f(UB), f(x_1), f(x_2), and f(LB):
f(UB) = 11
f(x_1) = 2.501
f(x_2) = 2.501
f(LB) = 11

Since both f(x_1) and f(x_2) are less than both f(LB) and f(UB), the minimum must be between x_1 and x_2 and, thus, they will become the new LB and UB respectively. The algorithm continues by repeating the previous process to find the next x_1 and x_2 but using the new more restricted lower and upper bounds. Using the same formulas, calculate the next x_1 and x_2:

$x_1 = -.708 + (1 - .618)(.708-(-.708)) = -0.167$
$x_2 = -.708 + .618(.708-(-.708)) = .167$

Next we evaluate the f(UB), f(x_1), f(x_2), and f(LB):

f(UB) = 2.501
f(x_1) = 2.02
f(x_2) = 2.02
f(LB) = 2.501

Using these values, the new lower and upper bounds are determined to be:

LB = -.167
UP = .167

The algorithm proceeds by calculating the next new values of x_1 and x_2 and the corresponding evaluations:

$x_1 = -.167 + (1-.618)(.167-(-.167)) = -.039$
$x_2 =-.167 +.618(.167-(-.167)) = .039$
f(x_1) = 2.001
f(x_2) = 2.001

Given the above x_1 and x_2, the new lower and upper bounds become:

LB = -.039
UP = .039

The algorithm will make another pass to further tighten the lower and upper bounds to find the minimum of the polynomial. The algorithm will calculate the next values of x_1 and x_2 and the corresponding evaluations:

$x_1 = -.039 + (1-.618)(.039-(-.039)) = -.009$
$x_2 =-.039 +.618(.039-(-.039)) = .009$
f(x_1) = 2.000
f(x_2) = 2.000

Given the above x_1 and x_2, the new lower and upper bounds become:

LB = -.009
UP = .009

One can continue the algorithm further to find a more precise results. Otherwise, the algorithm can be stopped at this point. As can been seen from figure 1, the minimum value of the polynomial is 2 at the point x = 0. The value we calculated is approximately 2.000. For many purposes (as mentioned earlier), this value is accurate enough and we can consider the algorithm complete.

Fuzzy Walkthrough

The author will now proceed with the Fuzzy Golden Ratio Section Search to find the minimum of the polynomial: x^2 - y +2 = 0 with the initial LB (lower bound) and UB (upper bound) of -3 and 3, respectively. All fuzzy data and operators will be denoted with a subscript f. It is important to remember that the fuzzy values represent a set, but for simplification purposes will be represented in the write-up as a single integer value with a subscript f.

During the setup of the algorithm, a few decisions need to be made: (1) size of the fuzzy sets and (2) allowing overlap between fuzzy sets. Both of these values control the level of precision for the algorithm. (For example: crisp, traditional numbers could be represented with a fuzzy set size of only one value with no overlaps). Additionally, the size of the fuzzy sets does not need to remain constant throughout the algorithm execution. Keeping in the spirit of the Golden Ratio Section Search algorithm, this example will hold the size of the fuzzy sets constant at +/- τ and, thus, create an overlap of approximately .236 between fuzzy sets. For example, -1_f is defined as -1.618 to -.382, 0_f is defined as -.618 to .618, and 1_f is defined as .382 to 1.618.

The algorithm begins the same as the traditional algorithm, with the calculation of the internal two points based on the fuzzified lower and upper bounds of -3_f and 3_f. (As previously mentioned: the value of τ will remain crisp as well as the calculation of 1- τ. The value of τ is essential to the original algorithm and not fuzzified. Additionally, this value is essential to the definition of the fuzzy set size). The values are calculated as follows:

$x_{1f} =_f LB_f +_f (1 - \tau) *_f (UB_f -_f LB_f)$
$=_f -3_f +_f (1- .618) *_f (3_f -_f -3_f)$
$=_f -1_f$
$x_{2f} =_f LB_f +_f (\tau) *_f (UB_f -_f LB_f)$
$=_f -3_f +_f (.618) *_f (3_f -_f -3_f)$
$=_f 1_f$

There are now three line segments (-3_f to -1_f), (-1_f to 1_f), and (1_f to 3_f). We now evaluate the f(UB), f(x_1), f(x_2), and f(LB):

$f_f(UB_f) =_f 11_f$
$f_f(x_{1f}) =_f 3_f$
$f_f(x_{2f}) =_f 3_f$
$f_f(LB_f) =_f 11_f$

Both $f_f(x_{1f})$ and $f_f(x_{2f})$ are less than both $f_f(LB_f)$ and $f_f(UB_f)$. The minimum must be between x_{1f} and x_{2f} and they will become the new LB_f and UB_f. The process repeats and the next x_{1f} and x_{2f} are calculated:

$x_{1f} =_f LB_f +_f (1-\tau) *_f (UB_f -_f LB_f)$
$=_f -1_f +_f (1-\tau) *_f (1_f -_f -1_f)$
$=_f 0_f$
$x_{2f} =_f LB_f +_f (\tau) *_f (UB_f -_f LB_f)$
$=_f -1_f +_f (\tau) *_f (1_f -_f -1_f)$
$=_f 0_f$

There are now three line segments (-1_f to 0_f), (0_f to 0_f), and (0_f to 1_f). Next, evaluate the f(UB), f(x_1), f(x_2), and f(LB):

$f_f(UB_f) =_f 11_f$
$f_f(x_{1f}) =_f 2_f$
$f_f(x_{2f}) =_f 2_f$
$f_f(LB_f) =_f 11_f$

The algorithm will be stopped at this point. As can been seen from figure 1, the minimum value of the polynomial is 2. The value we found is approximately 2 (i.e., the fuzzy set 2). (In general, the algorithm would continue, like the non-fuzzy version, until reaching the desired level of accuracy).

Example #2: More Complex Polynomial

In this example, the Fuzzy Golden Ratio Section Search will be tested with a more complex polynomial. The example was run using two simple Java programs: (1) the traditional Golden Ratio Section Search and (2) the Fuzzy Golden Ratio Section Search. The results of the fuzzy version will be compared to the traditional version.

This second example is to find the maximum of the polynomial: $f(x) = 12x - 3x^4 - 2x^6$ (Hillier and Lieberman 1990). The lower and upper bounds are defined as 0 and 2 (which are the two points where the function begins to be negative). The traditional algorithm ran the function as defined. The fuzzy version of the algorithm used a scaled version of the problem which allows the algorithm to take advantage of integer calculations. The algorithm can be scaled to any number of "decimal points" (denoted as d) of precision by dividing each x by 10^d and then multiplying the entire equation by 10^d.

The maximum value of the $f(x) = 12x - 3x^4 - 2x^6$ occurs in the range of 0.828 and 0.843 and is approximately 0.836 (Hillier and Lieberman 1990). The traditional version of the Golden Ratio Section Search required 5 iterations and computes the maximum to fall in the range of 0.833 and 0.839. The fuzzy version of the Golden Ratio Section Search uses only 3 iterations and computes that the maximum falls in the range of 8 and 9 (on the scaled problem) which is .8 and .9 on the non-scaled version.

The computation complexity of the algorithm is not increased by adding fuzzification to the algorithm. The Java code for the fuzzy algorithm contains only one additional method - a customized Java code to determine the fuzzy equivalence between fuzzy sets. This extra method is called only twice during each iteration and contains 8 lines of code most of which are a series of branching statements. This adds minimal execution time to the overall algorithm.

Testing Environment

The programming code for both the Fuzzy and Traditional version of the Golden Ratio Section Search was written in Java. The testing environment is as follows: Eclipse Mars, SDK 4.5.1 using Java version 1.8 on a HP Spectre running Windows 10 with 2.5 GHz Intel Core i7.

Discussion and Concluding Remarks

The Golden Ratio Section Search algorithm was successfully converted into a fuzzified version of the algorithm using the Framework for Fuzzification of Algorithms (Coffman-Wolph and Kountanis 2013b). For both example polynomials, the fuzzy version of the Golden Ratio Section Search performed better since it required fewer iterations to find a suitable range for the minimum or maximum solution. As stated before, these ranges are sufficient for many of the applications that use Golden Ratio Section Search and other similar one-dimensional search methods. For example, in each iteration of a non-linear optimization algorithm, one is required to use these types of techniques to estimate the best step size in the improving direction. The fuzzy version of the Golden Ratio Section Search takes advantage of the computer's natural high speed of integer calculations. (The traditional version uses the Java double data type for the calculations).

There are two disadvantages to using the fuzzy algorithm over the traditional. The fuzzy algorithm often requires scaling (but this scaling can be controlled by the user to their desired level of precision). The second disadvantages is that in cases where precise answers are needed, the fuzzy algorithm would only be able to provide a starting lower and upper bound and is unable to provide extremely high levels of accuracy.

Future Work

The work done in this paper focuses on only two specific polynomials and provides only the beginning of exploration into the use of the fuzzy framework for this kind of search algorithm. Since these preliminary results show great potential, the next step would be to expand to larger and more varied polynomials as well as non-polynomial functions. Further work could be conducted on other similar algorithms including: Dichotomous Search (bisection method), Uniform Search Method, Equal Interval Method, Fibonacci Search Method, and Quadratic (or polynomial) Interpolation. Additionally, the author would like to expand to other even more general search algorithms (e.g., Tabu search) and sorting algorithms.

References

Coffman-Wolph, S. 2015. The Hunch Factor: Exploration into Using Fuzzy Logic to Model Intuition in Particle Swarm Optimization. In the Proceedings of the 26th Modern AI and Cognitive Science Conference (MAICS 2015).

Coffman-Wolph, S and Coffman, Jr, P. 2014. Fuzzification of the Special Simplex Method for the Transportation Problem. Conference Presentation. In the Proceedings of the INFORMS Annual Meeting, San Francisco, California.

Coffman-Wolph, S. 2013. Fuzzy Search Strategy Generation for Adversarial Systems using Fuzzy Process Particle Swarm Optimization, Fuzzy Patterns, and a Hunch Factor. Ph.D. diss., Department of Computer Science, Western Michigan University, Kalamazoo, MI.

Coffman-Wolph, S. and Kountanis, D. 2013a. Fuzzy Process Particle Swarm Optimization. In the Proceedings of the 43rd Southeastern Conference on Combinatorics, Graph Theory, & Computing. Winnipeg: Utilitas Mathematica Pub. Inc.

Coffman-Wolph, S. and Kountanis, D. 2013b. A General Framework for the Fuzzification of Algorithms. In the Proceedings of the 4th biennial Michigan Celebration of Women in Computing (MICWIC 2013).

Coffman-Wolph, S. and Kountanis, D. 2013c. Finding Strategies in Adversarial Situations. In the Proceedings of the 24th Modern AI and Cognitive Science Conference (MAICS 2013).

Hillier, F. S., and Lieberman, G. J. 1990. *Introduction to Operations Research*. McGraw-Hill.

Kennedy, J., and Eberhart, R. C. 2001. *Swarm Intelligence*. San Francisco, CA: Morgan Kaufmann Publishers, Inc.

Koteshwariah, C. B., Kisore, N. R., Ravi, V., 2015. A Fuzzy Version of Generalized DBSCAN Clustering Algorithm. In the Proceedings of the Second ACM IKDD Conference on Data Sciences (CoDS 15). New York: ACM.

Lasden, L. (1970) *Optimization Theory for Large Systems*. New York, NY: Macmillan Publishing Co, Inc.

Luenberger, D. 1973. *Introduction to Linear and Nonlinear Programming*. Reading, MA: Addison-Wesley Publishing Company.

Sabzi, H. Z., Humberson, D., Abudu, S., King, J. P., 2016. Optimization of Adaptive Fuzzy Logic Controller Using Novel Combined Evolutionary Algorithms, and Its Application in Diez Lagos Food Controlling System, Southern New Mexico. *Expert Systems With Applications*. 43: 154-164.

Wismer, D. and Chattergy, R. 1978. *Introduction to Nonlinear Optimization*. New York, NY: Elsevier North-Holland, Inc.

Zadeh, L.A. 1965. Fuzzy Sets. *Information and Control*. 8: 338-353.