The Hunch Factor: Exploration into Using Fuzzy Logic to Model Intuition in Particle Swarm Optimization

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

Abstract

Particle Swarm Optimization (PSO) is a powerful biology-based optimization search strategy. This paper explores the addition of intuition into the PSO algorithm to improve the speed and number of iterations required to find solutions to the problem. This intuition will be modeled using a fuzzy variable called the Hunch Factor. It will act as memory for the system and influence the choices of the algorithm. Thus, allowing the algorithm to make more human-like decisions. This paper is an early exploration into the hunch factor via several experiments of the hunch factor with a simple optimization problem.

Background

L. Zadeh introduced the concept of fuzzy logic as an expansion of Boolean logic in his monumental paper entitled "Fuzzy Sets" (Zadeh 1965). Fuzzy logic is a set of rules and techniques for dealing with logic beyond a two-value (yes/no, on/off, true/false) system. Therefore, fuzzy logic, on a basic level, is an abstraction of traditional, two-value logic. Thus, fuzzy logic mimics a more human like approach to decision making. Fuzzy logic differs from traditional mathematical sets because it allows for an overlap of values between fuzzy sets.

Particle Swarm Optimization (PSO) is considered to be a highly successful and widely used problem-solving method and a subfield of swarm intelligence (Kennedy and Eberhart 2001). PSO is a biology-based optimization search strategy. It uses multiple independent particles to search the solution space of a given optimization problem. In PSO each individual particle stores the current candidate solution and refines the solution during the execution of the algorithm. PSO was based on the social behavior of animals and insects that regularly exist in groups.

Copyright held by the author.

The hunch factor supplies a human "hunch-like" element into the decision-making processes of the PSO algorithm. The hunch factor, represented by a membership function, is used to represent the innate ability of the system to derive guesses that influence the decisions made by the system. Additionally, the hunch acts as a fuzzy learning component for the algorithm since the hunch is continually altered during PSO execution. The hunch factor was first introduced in the author's dissertation (Coffman-Wolph 2013).

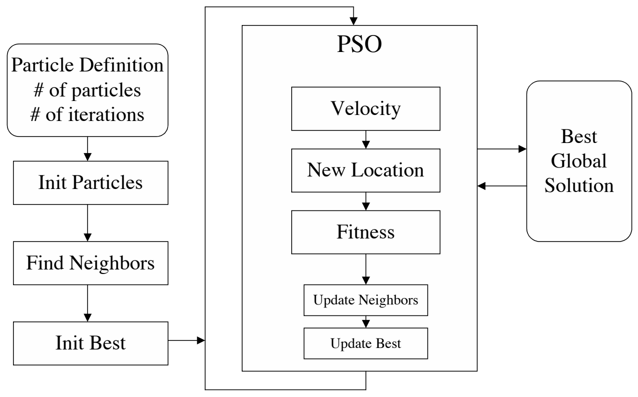

Figure 1: PSO Flowchart

The PSO Algorithm

The Particle Swarm Optimization (PSO) algorithm used in this paper is based on the original written by Kennedy and Eberhart (Kennedy and Eberhart 2001). The algorithm is a simple biology-inspired mathematical algorithm containing three main functions: fitness, velocity, and new location. Each of the three functions is processed on each particle individually until either the number of iterations is complete or a target value is reached. The fitness function determines how "good" the current candidate solution is, the velocity function determines the direction and speed the particle should head during the next iteration, and the new location function uses the velocity and previous location information to find the next candidate solution. Figure 1 provides the basic flowchart for the PSO algorithm.

Particles

Each particle has the following elements:

- x_i: A candidate solution
- v_i^{old}: Previous calculated velocity
- p_i: Particle's best solution so far
- N_i: List of neighbors
- p_n: Neighbor's best solution, $n \varepsilon N_i$

Fitness Function

The fitness equation is always problem dependent. However, it will always be a method for the evaluation of the candidate solution information. The number of variables in the equation(s) equals the size of the candidate solution vector. Any problem that can be formulated into an optimization problem can be used as a fitness function for the PSO.

Velocity Function

The velocity function determines both the direction and speed the particle should move to form the next candidate solution. The velocity is calculated for each element of the candidate solution using the following formula:

$$v_i = \alpha * v_i^{old} + \phi_1 r()*(p_i - x_i) + \phi_2 r()*(p_n^* - x_i)$$

where:

- α = Inertia [0,1]
- ϕ_1 = Learning factor 1
- ϕ_2 = Learning factor 2
- $[\phi_1 + \phi_2 = 4]$
- r() = Random number function [0,1]
- x_i: Current position/candidate solution of particle i
- v_i^{old}: Previous calculated velocity
- p_i: Particle's best solution so far
- p_n: Neighbor's best solution, $n \varepsilon N_i$

New Location Function

The new location function updates the particle's candidate solution. It uses the values calculated by the velocity function. The equation can be problem specific. The general equation for finding the new location for particle i is as follows:

$$x_i = x_i + v_i$$

where:

- x_i: Current position/candidate solution of particle i
- v_i: Calculated velocity

Nearest Neighbors Calculation

The nearest neighbors are calculated using the traditional distance equation. The neighbor list is part of the initial start up calculations. Additionally, the calculations are re-run and updated after an iteration of the algorithm (i.e., all the particles have been updated). The distance equation is as follows:

$$\text{distance} = \sqrt{(x_o - n_{x0})^2 + \ldots + (x_n - n_{xn})^2}$$

where:

- x_i: Current position/candidate solution of particle i
- n_{xi}: Current position/candidate solution of neighbor particle n_i

The Hunch Factor

As stated earlier the hunch factor is represented by a membership function. It is used to represent the innate ability of the system to derive guesses that influence the decisions made by an algorithm. The hunch provides memory for the system and acts as a learning element. The hunch membership function is updated during runtime with information gained during the run of the algorithm. Specifically for the PSO, the hunch will be updated for every particle each iteration of the algorithm.

The hunch factor will be applied directly to the velocity function and, thus, influence the values of the next candidate solution. Basically, the hunch factor will be an additional factor to the direction and speed the particle will move in based on previous history of either the particle and/or all the particles (depending on the experiment set).

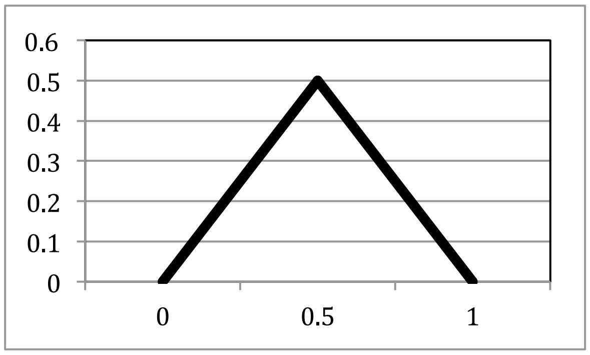

Figure 2: Initial Hunch Factor

Hunch Factor Representation

The hunch factor is stored as a fuzzy value and represented by a membership function. For simplicity, the hunch factor will be represented by a triangle membership function for these experiments. It is a fuzzy value and will be treated as such (i.e., storage and manipulation) during the execution of the algorithm. The hunch factor will be defuzzified and applied to the non-fuzzy velocity function calculation. (Defuzzification, in this case, will be based on the center of mass for the membership function).

To illustrate this concept, we can compare this to the childhood game of hot and cold. A group of players are attempting to find an object within a given room based on a leader calling out hot and cold to various players as they move around the room. The players begin by wanding around the room at random. As a player get positive reinforcement (i.e., warmer) they move in that postive direction until they get negative reinforcement (i.e., colder) and they change their path. The hunch factor is the accumulation of the warmer and colder clues received throughout the game and, thus, use the entire history instead of just the previous limited information. The hunch factor assists in speeding up the processes to finding the correct solution.

Hunch Factor Manipulation

After each iteration of the PSO algorithm and for each particle, the difference between the old fitness value and the new fitness value is used to alter the hunch accordingly. If the new fitness value is greater than the old fitness value, then the hunch is altered in a positive manner - the width of the membership function is decreased and the height increased. If the new fitness value is less than the old value, then the opposite changes are made to the hunch. The magnitude of the difference between the values is also taken into consideration.

The Velocity Function with Hunch Factor

As stated, the hunch factor is applied to the velocity function. This allows the hunch factor to alter the speed and direction of the particle. The velocity function is, therefore, altered slightly from the original provided in the PSO algorithm. The following shows the altered velocity formula:

$$v_i = h * \alpha * v_i^{old} + (\phi_1 r()*(p_i - x_i) + \phi_2 r()*(p_n^* - x_i))$$

where:

- h = hunch factor value
- α = Inertia [0,1]
- ϕ_1 = Learning factor 1
- ϕ_2 = Learning factor 2
- $[\phi_1 + \phi_2 = 4]$
- $r()$ = Random number function [0,1]
- x_i: Current position/candidate solution of particle i
- v_i^{old}: Previous calculated velocity
- p_i: Particle's best solution so far
- p_n: Neighbor's best solution, $n \varepsilon N_i$

Experiments

In this paper, several hunch factor experiments will be carried out. One set of experiments will be focused on the number of hunch factors considered: one global hunch factor vs. a hunch factor specific to an individual particle. Another set of experiments will focus on the level of influence the hunch factor has on the velocity function. The final set of experiments will focus on the manipulation of the hunch factor during the runtime of the function.

Experiment #1: Number of Hunch Factors

As mentioned in an earlier section, the preliminary hunch research began in the author's dissertation (Coffman-Wolph 2013). In this work, a single global hunch factor was maintained for the system. This simple hunch factor showed some promise in positively influencing the system. This first experiment directly expands on the original experiment. Thus, experiment #1 will compare the results of having a global hunch factor for the system with individual hunch factors associated with specific particles. The membership functions will be constant and the update methods will be exactly the same, just the number of hunch factors will be different.

Experiment #2: Hunch Factor and Velocity

The hunch factor is applied directly to the velocity equation calculation. Initially, the hunch factor is simply applied to the velocity as an additional factor. In this experiment the hunch factor will be applied at various partial levels and compared directly to the unaltered full hunch application. Additionally, the hunch factor will be moved from the first term of the velocity equation (i.e., the old velocity or the old information) to the second and third terms of the velocity equation (i.e., the best solutions seen by the particle and neighboring particles or the new information) to observe the effects.

Experiment #3: Hunch Factor Manipulation

The final set of experiments will focus on the manipulation of the hunch factor. In the first two experiments, the hunch factor will be manipulated in a very simple manner. If the fitness value improves, the hunch factor height will be increased by 25 percent and width be decreased by 10 percent. If the fitness value does not improve, the hunch factor height will be decreased by 25 percent and width increased by 10 percent.

In experiment #3 the hunch factor height and width will be increased/decreased using various percentages. Also, the positive and negative influences will not remain consistent (i.e., more "punishment" for wrong, less "reward" for correct). The combinations that will be tried are as follows:

- Height and width 10% increase/decrease
- Height 10% increase when correct, 20% decrease when incorrect, width 10% increase/decrease
- Height and width 10% increase when correct, 20% decrease when incorrect
- Height 20% increase when correct, 10% decrease when incorrect, width 10% increase/decrease
- Height and width 20% increase when correct, 10% decrease when incorrect

Testing Problem

For demonstration purposes in this paper, a simple optimization problem is used. This problem is defined as follows (Hillier and Lieberman 1990):

Maximize $Z = 2x_0 * x_1 + 2x_1 - x_0^2 - 2x_1^2$

where x_0 and $x_1 \geq 0$

Testing Environment

The programming code for the PSO was written in Java. The testing environment is as follows: Eclipse SDK 4.2.1 using Java version 1.6 on a MacBook Pro running OS X version 10.8.2 with 3 GHz Intel Core i7.

Results

The following sections cover the results of the three experiments outlined in detail earlier in this paper. Experiment #1 dealt with the difference between having a single global hunch factor versus a hunch factor tailored to each particle. Experiment #2 dealt with how the hunch factor was applied to the velocity equation. Experiment #3 explored various manipulations of the hunch factor during runtime.

Data Collected

The following data was collected during various runs for all three experiments with the simple optimization problem provided earlier.

100 particles, 1000 iterations		
	Time	Fitness
PSO	1124	0.8542
PSO, Hunch New	1162	0.8669
PSO, Hunch Old	1140	0.9777
PSO, Multi Hunch	1060	0.8778

Figure 3: Data for 100 particles, 1000 iterations

100 particles, 2000 iterations		
	Time	Fitness
PSO	2233	0.8576
PSO, Hunch New	2381	0.8850
PSO, Hunch Old	2246	0.9806
PSO, Multi Hunch	2209	0.9199

Figure 4: Data for 100 particles, 2000 iterations

100 particles, 3000 iterations		
	Time	Fitness
PSO	3328	0.9690
PSO, Hunch New	3362	0.7806
PSO, Hunch Old	3350	0.9636
PSO, Multi Hunch	3225	0.9309

Figure 5: Data for 100 particles, 3000 iterations

100 particles, 4000 iterations		
	Time	Fitness
PSO	4507	0.9629
PSO, Hunch New	4642	0.9392
PSO, Hunch Old	4659	0.9513
PSO, Multi Hunch	4222	0.9140

Figure 5: Data for 100 particles, 4000 iterations

200 particles, 1000 iterations		
	Time	Fitness
PSO	6639	0.9236
PSO, Hunch New	6794	0.9376
PSO, Hunch Old	6748	0.9203
PSO, Multi Hunch	6369	0.9012

Figure 6: Data for 200 particles, 1000 iterations

300 particles, 1000 iterations		
	Time	Fitness
PSO	21070	0.9274
PSO, Hunch New	20780	0.9706
PSO, Hunch Old	21060	0.9600
PSO, Multi Hunch	20154	0.9480

Figure 7: Data for 300 particles, 1000 iterations

	100, 1000	100, 4000
Percentage (%)	Fitness	Fitness
10	0.9837	0.958
20	0.8941	0.9697
30	0.9641	0.9259
40	0.9870	0.9300
50	0.9152	0.9855
60	0.9223	0.8685
70	0.8756	0.9399
80	0.9535	0.9560
90	0.9541	0.9390
100	0.9777	0.9513

Figure 8: Data for Hunch % Applied

Set Up	PSO Fitness	PSO Hunch Old	PSO Hunch New
100 p, 1000 it	0.8542	0.9777	0.8669
100 p, 2000 it	0.8576	0.9806	0.8850
100 p, 3000 it	0.9690	0.9636	0.7806
100 p, 4000 it	0.9629	0.9513	0.9392

Figure 9: Data for Hunch Placement

Set Up	PSO Fitness	PSO Hunch Old	PSO Hunch New
100 p, 1000 it	0.8542	0.9777	0.8669
200 p, 1000 it	0.9236	0.9203	0.9376
300 p, 1000 it	0.9274	0.9600	0.9706

Figure 10: Data for Hunch Placement

Iterations	Normal	Case #1	Case #2	Case #3	Case #4	Case #5
1000	0.9777	0.9210	0.9282	0.9401	0.9659	0.9919
2000	0.9806	0.9797	0.9683	0.7785	0.9871	0.9492
3000	0.9636	0.9817	0.9535	0.9362	0.9684	0.9455
4000	0.9513	0.9786	0.9759	0.9314	0.9856	0.9456

Figure 11: Data for Hunch Test Cases

Experiment #1 Results: Number of Hunch Factors

As stated earlier, experiment #1's focus was to compare the effects of having one global hunch factor for the PSO verses having a hunch factor for each particle. The following diagrams demonstrate the various results found for each test problem and various test cases (i.e., various particle and iteration sizes).

The analysis of the results begins by focusing on increasing the number of iterations. Figure 12 shows that there was negligible difference in execution time for just the PSO, the PSO with one hunch factor, and the PSO with a separate hunch factors for each particle. Figure 13 shows the average fitness values found for the PSO, the PSO with 1 hunch factor, and the PSO with a separate hunch factor for each particle.

As can be observed in Figure 12, when dealing with the standard PSO, as the number of iterations increases, the fitness value improves. However, when the global hunch factor is added to the PSO system, the hunch performs extremely well at lower iterations and becomes un-effective as the number of iterations increases. The system with individual hunch factors for each particle follows a pattern similar to the standard PSO system, but also demonstrates a performance issue with high iterations similar to the global hunch PSO system. However, it should be observed that both systems with the hunch outperformed the standard PSO in terms of fitness at low iteration values.

Figure 14 and Figure 15 demonstrate the effects of the number of hunch factors as the number of particles increases (while the number of iterations is held constant). Figure 14 (as with Figure 6) shows that the execution time is negligibly affected by the addition of the hunch into the system. Figure 15 illustrates the changes in the fitness value with the addition of the hunch. It can be observed, that overall the hunch (single global or multiple) is less effective as the number of particles increases. However, at low number of iterations, either system with the hunch out performs the standard PSO.

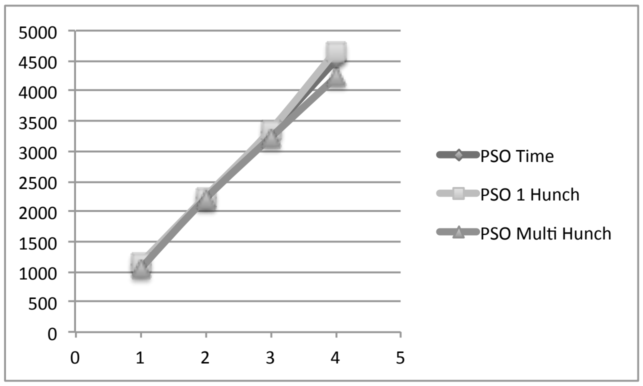

Figure 12: Comparison of Execution Time (Time in Milliseconds vs. Iterations in 1000s)

Experiment #2 Results: Hunch Factor and Velocity

Experiment #2 consists of two experiments. The first experiment is the level (i.e., percentage) of effect the hunch has on the velocity function. Figures 16 and 17 illustrate the results of these different levels. The second experiment focuses on the difference between applying the hunch to the "old information" vs. the "new information" and can be observed in Figures 18 and 19.

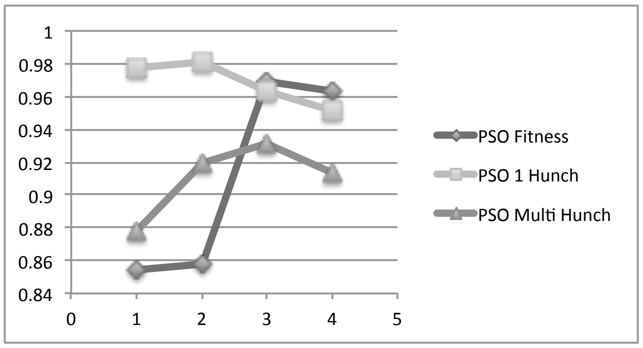

Figure 13: Comparison of Fitness Values (Fitness Value vs. Iterations in 1000s)

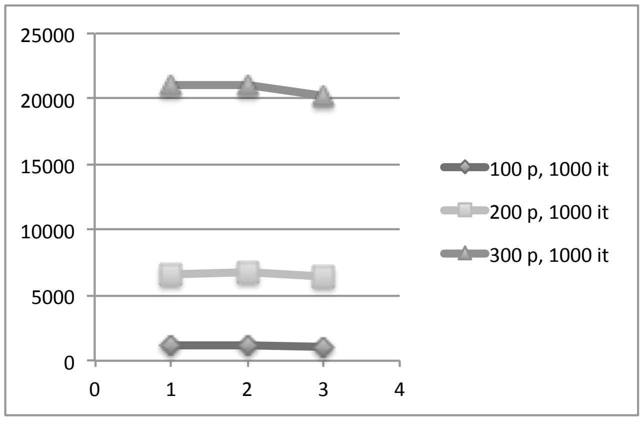

Figure 14: Comparison of Execution Time (Time in Milliseconds vs. Particles in 100s)

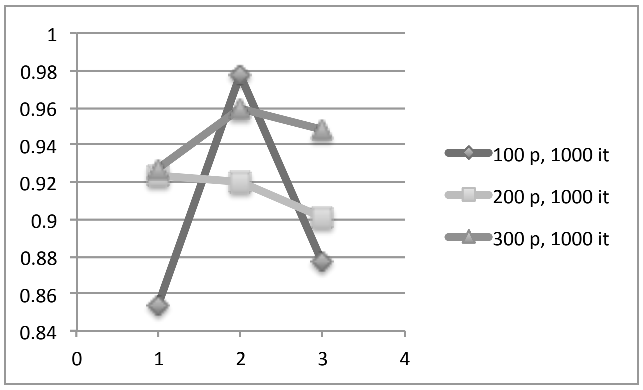

Figure 15: Comparison of Fitness Values (Fitness Value vs. Particles in 100s)

Applying only a percentage of the hunch into the velocity calculation had interesting results as seen in Figures 16 and 17. The "normal" mode was 100% (i.e., the point on the far right). Applying only a portion of the hunch was successful in the runs of 1000 and 4000 iterations. Middle ranges of the hunch were overall not successful. However, high percentages of the hunch were successful in both 1000 and 4000 iterations.

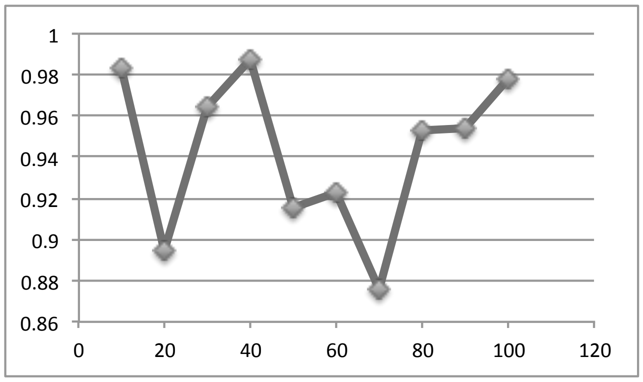

Figure 16: Hunch Percentage Applied - 100 particles, 1000 iterations (Fitness Value vs. Hunch Percentage)

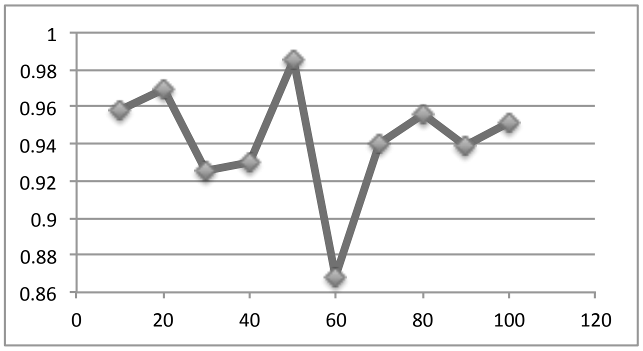

Figure 17: Hunch Percentage Applied - 100 particles, 4000 iterations (Fitness Value vs. Hunch Percentage)

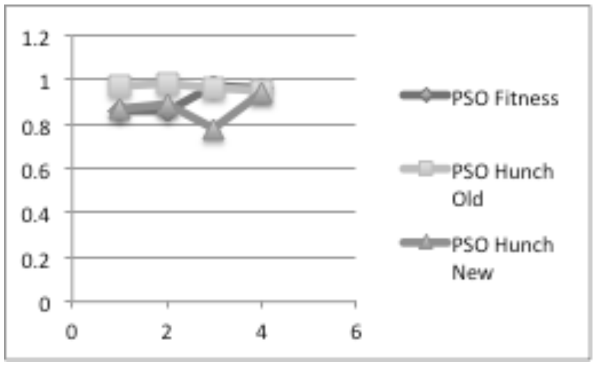

Figure 18: Hunch Applied Old Information (Fitness Value vs. Iterations in 1000s)

From Figures 18 and 19, it can be observed that the application of the hunch to either the old information or new information has very little effect on the fitness values found as the iteration size increased. However, as Figure

19 illustrates, the term of the velocity equation where the hunch was applied affects the results as the number of particles increases. The hunch on the new information matches the pattern for the PSO without the hunch. Overall the hunch being applied to the old information produces better fitness values.

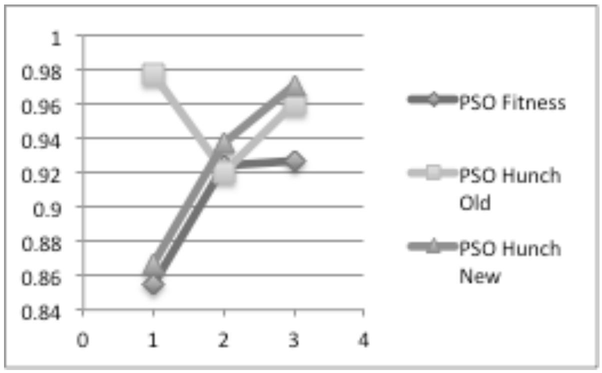

Figure 19: Hunch Applied New Information (Fitness Value vs. Particles in 100s)

Experiment #3 Results: Hunch Factor Manipulation

Experiment #3 consisted of 5 test cases of various manipulations of the hunch factor. The test cases are as follows:

- Case #1: Height and width 10% increase/decrease
- Case #2: Height 10% increase when correct, 20% decrease when incorrect, width 10% increase/decrease
- Case #3: Height and width 10% increase when correct, 20% decrease when incorrect
- Case #4: Height 20% increase when correct, 10% decrease when incorrect, width 10% increase/decrease
- Case #5: Height and width 20% increase when correct, 10% decrease when incorrect

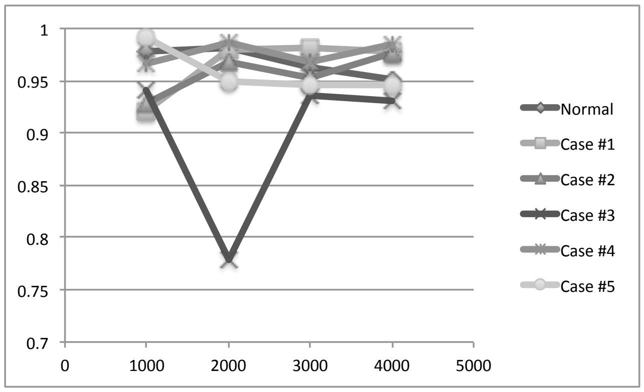

Figure 20: Hunch Test Cases (Fitness Value vs. Iterations)

Figure 20 illustrates (using the data from Figure 11) the fitness value that resulted from the various test cases. As can be observed in Figure 21, of the 5 test cases, one test case stood out: #4.

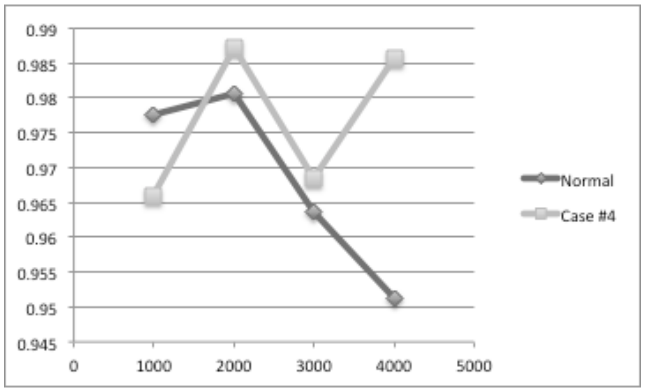

Figure 21: Hunch Test Case #4 (Fitness Value vs. Iterations)

Test case #4: Height 20% increase when correct, 10% decrease when incorrect, width 10% increase/decrease. This test case used positive reinforcement. When the particle was moving in the correct direction, the hunch was increased more. However, if it was moving in an incorrect direction, less hunch manipulation occurred (i.e., less punishment).

Discussion and Concluding Remarks

The hunch factor has been shown in the experiment results of the previous section to have promise. These are simply the preliminary and exploratory results using a very simple optimization problem. The hunch factor works extremely well with smaller number of particles and fewer iterations. The hunch factor, is thus, best suited for assisting particles in finding the general area within the search space. (The hunch factor has either no effect or slight negative effect when honing in on a solution). Thus, the hunch could possibly make an excellent addition to an algorithm designed to find "good" starting points for other optimization algorithms that require such starting points.

Additionally, it can be observed that the hunch factor should be applied to the portion of the velocity function that contains the previous information. It would be helpful to use either the full hunch factor or a fraction of the hunch factor when determining the velocity and, thus, the next possible solution. The single global hunch should be manipulated using positive reinforcement.

Future Work

The work done in this paper focuses on a specific optimization problem and provides only the beginning of exploration into the use of the hunch. Given these preliminary results, the next step would be to expand to other problem sets – with larger and more varied problems. Although the extra computation for the hunch factor is small and should not cause a hindrance when expanding to larger problems, more experimentation is needed to verify that conclusion. Additionally, it would be interesting to experiment further with various membership functions and not just a simple triangle membership function for the hunch factor representation. Another step would be to move to a completely fuzzy algorithm version of the PSO (Coffman-Wolph 2013a and Coffman-Wolph 2013b) and run these same experiments with the hunch factor. The hunch factor showed great promise when using a small numbers of particles and less iterations – something that could be further explored. In particular the application of using the hunch factor when trying to find starting points for other optimization algorithms.

Acknowledgments

The author would like to thank the reviews for their helpful suggestions.

References

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).

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

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