Optimisation based on ‘Nearest Better Neighbor’: A Preliminary study

WK Wong
Department of Electrical and Computer Engineering
Faculty of Engineering and Science
Curtin University Malaysia, CDT 250,
98009 Miri Sarawak, Malaysia
Weikitt.w@curtin.edu.my

Abstract. A multi agent optimization was proposed and evaluated in this research report. It incorporates multi agent search by learning from the ‘nearest better solution’. Each agent learns from the nearest agent with better solution thereby ensures that the optimization explores solution spaces between the solutions. Weightage multipliers determine how much each solution segment adapts to the ‘nearest better solution’ based on the ‘nearest better solution’ fitness to the best population fitness. A small amount of random mutation is introduced in every iteration to prevent early convergence. A converging time period (50% of last iteration period) is assigned in order for the agents to converge to a specified range in which the random mutation mechanism is totally repressed. Results showed that the algorithm is able to produce comparable results as compared with other algorithm such as Genetic Algorithm (GA) and particle swarm optimization (PSO). Evaluations were performed using 11 benchmark functions based on average ranking.

I. INTRODUCTION AND RELATED WORK

Optimization algorithms are a group of algorithms that functions to extract good solutions in numerical or state space solutions. Optimization algorithms are generally categorized as derivative or non-derivative methods. Derivative methods incorporate first order or second order derivatives of the fitness surface points and change the solution candidates based on the gradients. This is specifically suitable for fitness surface that are smooth and highly convex such that the solution vectors can be adjusted based on the partial derivatives of the fitness with regards to the vector elements.

It is noteworthy that not all systems are convenient to acquire the gradients at specific points. While it is accepted that these methods are highly efficient in highly convex surfaces, these methods may get trapped in a ‘local minima’ solution for non-convex search space. These several factors justifies the preference for the other category of optimization methods which are non-derivative methods. These methods usually apply a form of stochastic method or nature inspired mechanism to intuitively adapt to find better solutions over a series of iterations. Some well-known metaheuristics optimization algorithms include Genetic algorithm (GA) [1], Particle swarm optimization (PSO) [2] and Ant Colony Optimizations (ACO) [3]. GA is loosely based on the mutation and recombining of DNA information in DNA chromosome. GA introduced the concept of ‘mutation’ and ‘crossover’, a borrowed terminology of biological DNA which enables creation of better individual.

PSO is another major category of biological inspired algorithms inspired by swarm behavior of social animals. PSO involved the concept of swarming in which the candidate solutions will converge on the best acquired solutions in each population. The main problem with PSO is the ‘early convergence’ because candidate solutions converge to best population solutions and the current best global solution at early stages. Various variants of PSO attempt to solve this problem namely the hierarchical PSO (H-PSO) [4] which introduces a hierarchy system in nearby particles in which the solutions may adapt to. These methods introduces further complexity to the
existing PSO method. It is also notable that the bare bones PSO [11] which is a simplification of the original PSO performed better on benchmark datasets. This showed that exact mimicry of the biological counterpart might not be a necessity for numerical optimization albeit providing a form of inspiration for the optimization algorithm.

ACO proposed in [12] introduced the concept of mapping in space inspired by pheromone trail in ants. Ants navigate to find and optimal (nearest) path between nest and food source by emitting pheromone to strengthen candidate paths. The same concept was adopted for numerical optimization by having multiple virtual ants to strengthen transition paths between candidate solutions. These elements can be seen in further derivative developments as discussed in [8].

Other new form of nature inspired algorithm includes Bat algorithm (BA), Cuckoo search (CS), Lion algorithm (LA). Generally, these algorithms do no guarantee a global optimal solution over the search range despite the elaborate nature inspired scheme. Some algorithms such Harmony Search (HS) [5] based on musician analogy are even claimed to be a special version of evolutionary strategy with elaborate metaphorical analogies as the source of inspiration as mentioned in [6]. It is therefore imperative to clearly state and focus the discussion on the mechanism of the algorithm rather than lengthy explanation on the inspiration behind the algorithm.

Of all known nature inspired optimization schemes, only PSO and its variants are shown to fully converge over iterations. Convergence is a desirable trait in optimization algorithms as it allows iteration to settle to a single solution. This can be easily proven in derivative based optimization algorithm but not a trivial task when it comes to meta-heuristic algorithms.

Another important criteria is that algorithm need to be adjustable based on the estimated surface of solutions such as complexity and convexity. Hence, knowledge of how to tune these parameters needs to be stated clearly. This is further reinforced with the ‘No free lunch Theorem’ which states that all algorithms performed averagely the same over all problems [7]. Hence, an in-depth knowledge of how to tune the optimization algorithm is imperative to optimize the iterative solution finding.

With regards to swarm intelligence based optimization (PSO and its variants), various configurations have been explored. There are countless research works that discuss the various strategies to improve exploration. Glow worm swarm optimization proposed in [9] (which somewhat resembles PSO) applies a fixed radius strategy in which the learning is based on local selected neighbor within the sensor radius. Glow worm optimization (GWO) is proven to perform poorly on large dimensional solutions. GWO solution candidates does not adapt to the best solution but adapts to on random solution in a specified radius. The reason it does not have early convergence could also be the very reason it performs so badly on large dimensional problems (there is insufficient convergence to known good solution candidates.)

PSO variant called the teacher learner PSO (TL-PSO) proposed in [10] applies phases in which a solutions adapts to a neighbor with better fitness and the global best solution. This is an attempt to prevent early convergence in the original PSO. Hierarchical-PSO (H-PSO) which also claim to be designed specifically to counter the early convergence problem in [4] and [13], uses a mechanism that organizes the candidates based on the hierarchy of the fitness. Therefore the candidates do not immediately converges to the best solution but the various candidates with intermediate fitness.

These PSO variants and its related inspired works tend to answer the early convergence phenomenon which is a critical problem in PSO. There is no guarantee that these solutions will work well in all problems as ‘no free Lunch’ theorem [7] still applies. In another words, the average performance of any algorithm across all problems is the same. Continuous motivation to explore algorithms could emerge from the desire to seek and apply specific algorithms to specific problems. Hence there is no ‘one size fits all’ solution for such optimization problems.
As of this point of writing, there is no known research work that focuses on optimization algorithms based on the proposed mechanism for real value problems. This leads to the clear objective of investigating this configuration. How would this mechanism be suited to a specific type of problem? How does tuning certain parameters pertaining to the optimization algorithm lead to solving specialized problems (e.g., multimodal/unimodal). This research does not aim to understand the full benefits of such mechanism but merely as a preliminary work to test the effectiveness on several well-known bench tests.

The main distinction presented in this research is as stated i.e., to apply a “nearest better neighbor learning” for all the solutions in the population (except the population best). Each solution will adapt to the nearest neighbor with better fitness value regardless of the value difference based on “Line of sight”. It is hypothesized that it is important to explore local regions (areas between candidate solutions). Hence, the proposal of following the nearest better neighbor would make more sense than following the better neighbor as the latter could cause high possibility of missing solutions between the smaller regions.

Despite having more exploration locally on regions around the solutions in the current population, it will not solve the early convergence. In order to solve this problem, two mutation schemes were proposed to ensure that solutions do not get stuck in local minima. The first is to apply a batch mutation in which a certain percentage of solutions in the entire population are mutated by 1 element. The second is the mutation to the best solution in the population using a Gaussian mutation scheme. Given the objective of the algorithm, the research will be organized in following 3 sections. The description of the code along with its respective details will be discussed in the section 2. The experimental setup and results will be discussed in section 3 followed by conclusion remarks in section 4.

2. DESCRIPTION OF CODE

The section below will explain the working of the proposed algorithm. The global optimization is to find optimal solution of \( \bar{x}_g \in \mathbb{R}^d \) such that \( \bar{x}_g = \arg \min_{x_i} F_n(\bar{x}_i) \) where \( n = \{1,2,\ldots,11\} \) represents the various test functions shown in Table 2.2. \( \bar{x} \) represents a solution vector \( \in [lb, ub] \) where \( lb \) and \( up \) are boundaries defined in Table 2.2.

2.1 Overview of code

The overall pseudo code of the proposed algorithm is shown in Figure 1. The algorithm stopping criteria is fulfilled when the number of predefined iterations is reached (iteration number). The selected iteration number is 3000 for all the benchmark functions. The initial generation of N candidates in the population is randomly generated using a fixed probability stochastic random function. The main concept of the algorithm is that each individual learns from the nearest better solution as expressed in algorithm (2) (refer to section c). The algorithm is divided into three main components: i) mutating the best solution ii) batch mutation iii) candidates learn from the nearest better neighbour (in which this algorithm receives its namesake descriptive title).

The algorithm can be divided into 2 phases which are the non-fully converging phase and fully converging phase. These phases are activated by the threshold of the iteration time (50%). In the first 50% of the iteration time (non-fully converging phase), all the elements are activated (best candidate mutation/batch mutation/learning mechanism). The second half of the iteration period excludes the batch mutation thereby allowing the learning mechanism to fully converge to the next better neighbour. Ignoring the best candidate mutation, the rest of the candidates will ultimately converge to a point. The best candidate mutation acts as continuous mutation scheme to prevent getting stuck in a local. If the best candidate mutation generates an inferior candidate solution, it will be
eliminated in the population by replacement with the Global best. The values of adjustable parameters are shown in Table 1.

To illustrate convergence, a graphical analogy is presented. Assume that all N candidate start at random position. In every iteration, the candidates are ranked from 1 to N. Over a series of iteration, the distances between two ranked candidates \( \|x_{i+1} - x_i\| \) were considered where \( i \) denotes the ranked position of the candidates. As iteration \( \to \infty \), \( \|x_{i+1} - x_i\| \approx 0 \) and thus \( \|x_1 - x_n\| \approx 0 \) (distances between the N-ranked and the first ranked candidate). This is only achieved if the converging period is sufficient for convergence.

2.2 Best candidate mutation

The best candidate in every iteration will be mutated using a Gaussian mutation scheme in which a section is randomly selected for mutation using a fixed probability. The section is added with a value acquired from a normal probability distribution with \( \mu = 0 \) and \( \sigma^2 = 0.1 \). This ensures that the global best solution is not trapped in a local minima throughout the iteration period (continuous optimization).

2.3 Batch Mutation

Batch mutations refer to the mutation to the entire population using a small random mutation. This random mutation is further explained based on the pseudo code expressed in Figure 2. Line 1 to line 5 express a mechanism in which the limits the mutation for only 50% of the iteration period. The last 50% of the iteration period is reserved for the population to converge. A longer converging period may be applied if the search space is large and a specific convergence criterion is set. A weightage denoted \( l_i \) determines the amount of step change for the batch mutation. Generally this parameter is dependent on the search space. For all the test functions, \( l_i \) is specified as 0.01.

2.4 Learning from the nearest Better Neighbour (learning mechanism)

Pseudo code in figure 3 shows the learning mechanism based on adapting to the ‘nearest better solution’. As shown, Line 9 of the pseudo code depicts the main mechanism in which the each individual adapts to the nearest better solution. The probability of a section selected for mutation is 30%. i.e. approximately 30% of the vector solution is selected stochastically for adapting to the neighbor with better fitness. The amount of adaption to the nearest better solution is weighted by the ratio of ‘fitness difference between the current individual and population best’ and ‘the difference between nearest better solution to the current individual’ as shown in figure 2.3 (line3). Intuitively, this represents the amount of change the current individual should adapt to the nearest better solution based on the fitness value, \( F_n \). The adapting mechanism applies to all the solution in the population except the best in the population.

The ‘nearest better neighbor’ for the \( i_{th} \) solution (\( x_i \)) candidate is decided using the bit wise product of distance map \( D_M \) and fitness map \( G_m \), expressed as \( K \) in equation 1. \( G_m \) is a binary matrix acquired by applying eqn 2 on difference between the \( i^{th} \) solution against the next solutions in the other solutions (\( j^{th} \)). An element of 1 in the represents a better value while an element of 0 represents an inferior fitness relative to \( i^{th} \) solution.

\( D_M \) as shown in eqn 4 is the norm/Euclidean distance between \( i^{th} \) solution and other solutions in the population. Hence \( K \) is the matrix in which the smallest non-zero value can be acquired to find the better nearest neighbor.
The test functions are shown in Table 2 where \( d \) is the solution dimensions. The solution dimension, \( d \) for the tests is selected to be 30. The \( F_{min} \) is the known optimal value that can be acquired by each test functions. The search ranges were also specified for the respective test functions.

\[
K = D_m \otimes G_m
\]

\[
G_m = \begin{bmatrix}
B(F_n(x_1) - F_n(x_1)) & \cdots & B(F_n(x_N) - F_n(x_1)) \\
\vdots & \ddots & \vdots \\
B(F_n(x_N) - F_n(x_N)) & \cdots & B(F_n(x_N) - F_n(x_N))
\end{bmatrix}
\]

\[
B = \begin{cases}
0, & \text{if } (F_n(x_i) - F_n(x_j)) > 0 \\
1, & \text{if } (F_n(x_i) - F_n(x_j)) \leq 0
\end{cases}
\]

\[
D_M = \begin{bmatrix}
\|x_1 - x_1\| & \cdots & \|x_1 - x_N\| \\
\vdots & \ddots & \vdots \\
\|x_N - x_1\| & \cdots & \|x_N - x_N\|
\end{bmatrix}
\]

1. Random generate \( N \) solution candidates
2. While \( iter < \text{iter number} \)
3. Rank each individual solution \( 1:N \)
4. if (iteration < converging_period )
5. Mutated_best = Gaussian mutation (best solution)
6. if (\( F_n \) (Mutated_best) < \( F_n \) (best mutation))
7. Best mutation = Mutated_best
8. else
9. no action
10. Random mutate 50% of population using algorithm except the best solution based on Algorithm (1)
11. For solution 2: \( N \) //exclude the best
12. Select the ‘nearest better solution’
13. Computer the segment change based on Algorithm (2)
14. end for
15. mutate the best solution in the current population
16. Replace worst solution with global best
17. Rank each individual solution
18. if \( F_n \) (population_best) < \( F_n \) (global_best)
19. population_best=global_best
20. end if
**Figure 1.** Pseudo code for overall program

**Table 1.** Parameters of algorithm

| Parameter              | values |
|------------------------|--------|
| iter number            | 3000   |
| converging_period      | 50%    |
| N (Solution population)| 50     |
| $l_1$                  | 0.01   |
| $d$                    | 30     |

1. if \( \frac{(\text{iter number} - \text{iter})}{\text{iter number}} \) > converging_period
2. rate = 0.7
3. else
4. rate = 1.00
5. end
6. for \( i = 2 : N \)
7. for \( j = 1 : d \)
8. if \((\text{rand}(1)>\text{rate})\)
9. \(x_{(i,j)_{\text{iter}+1}} = x_{(i,j)_{\text{iter}}} + 2l_1 \times \text{rand}(1) - 1\)
10. end if
11. end for
12. end for

**Figure 2** Pseudo code for Algorithm (1)

1. for \( i = 2 : N \)
2. for \( j = 1 : d \)
3. \( \bar{x}_{(i,j)_{\text{iter}+1}} = \bar{x}_{(i,j)_{\text{iter}}} + \frac{f_{(i,j)_{\text{iter}}} - f_{(i,j)_{\text{iter}}}}{f_{\text{pop,best}_{\text{iter}}}} \times [x_{\text{better}(i,j)} - \bar{x}_{(i,j)_{\text{iter}}}] \)
4. end for
5. end for

// $f_{\text{pop,best}_{\text{iter}}}$ is the population best at the specific iteration
// $x_{\text{better}}$ is the identified better nearest neighbour based on eqn 1

**Figure 3.** Pseudo code for Algorithm (2)
Table 2. Benchmark test functions

| No. | Function Common Name | Equation | Fmin | Search range | Modality |
|-----|----------------------|----------|------|--------------|----------|
| F1  | Spherical            | $\sum_{i=1}^{d} x_i^2$ | 0    | $[-100 100]^d$ | Uni      |
| F2  | Schwefel 2.22        | $\sum_{i=1}^{d} |x_i| + \left|\sum_{i=1}^{d} |x_i| \right|$ | 0    | $[-10 10]^d$ | Multi    |
| F3  | Schwefel 2.21        | $\max |x_i|, 1 \leq i \leq n$ | 0    | $[-100 100]^d$ | Uni      |
| F4  | Rosenbrock           | $\sum_{i=1}^{d} (100(x_{i+1} - x_i)^2 + (x_i - 1)^2)$ | 0    | $[-30 30]^d$ | Multi    |
| F5  | Step                 | $\sum_{i=1}^{d} |x_i + 0.5|^2$ | 0    | $[-100 100]^d$ | Uni      |
| F6  | Schwefel             | $418.9829 d - \sum_{i=1}^{d} -x_i\sin\sqrt{|x_i|}$ | 0    | $[-500 500]^d$ | Multi    |
| F7  | Rastrigin            | $10 \times d + \sum_{i=1}^{d} x_i^2 - 10\cos(2\pi x_i)$ | 0    | $[-5.12 5.12]^d$ | Multi    |
| F8  | Auckley              | $-20 \times \exp\left(\frac{1}{4}\sum_{i=1}^{d} x_i^2 - \exp\left(\sum_{i=1}^{d} \frac{x_i}{\sqrt{d}} \cos(2\pi x_i) + e\right)\right)$ | 0    | $[-32 32]^d$ | Multi    |
| F9  | Griewank             | $\sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{d}}\right) + 1$ | 0    | $[-600 600]^d$ | Multi    |
| F10 | Penalized 1          | $y = 1 + \frac{1}{\pi} \tan^{-1}\left(\tan(\pi x_1)\right) + \sum_{i=2}^{n} \left[1 + \left(1 - \tan^2\left(\pi y_i\right)\right)\right]$ | 0    | $[-50 50]^d$ | Multi    |
3. Experimental setting and benchmarking

As an initial evaluation, the results are benchmarked with PSO and GA from Matlab optimization toolbox. The performance comparisons with GA/PSO are shown in Table 2. The results are acquired over 30 runs. The results for both PSO and GA were acquired using default values in Matlab toolbox. Each function is optimized for 3000 iterations and the best fitness is accepted as the fitness value.

The average fitness values, standard deviations, Best fitness and worst fitness acquired for the optimization algorithms are shown in Table 3. The ranking \( \in [1, 2, 3] \) is assigned based on the average performance on the respective benchmark functions. The average ranking is the mean ranking of algorithm averaged across the 12 functions. The proposed algorithm average ranking is 1.72 as compared to PSO (2.36) and GA (1.90). Hence the average ranking can act as a performance indicator for the algorithms.

| Test Fun | Nearest better neighbour | Particle Swarm Optimisation (PSO) | Genetic Algorithm |
|----------|--------------------------|----------------------------------|-------------------|
| F1       | 0.002217 (1)             | 3.97E-08 (1)                     | 0.000177 (2)      |
| F2       | 0.114096 (2)             | 0.018044 (2)                     | 0.000308 (1)      |
| F3       | 0.112724 (1)             | 2.193252 (3)                     | 0.2135259 (2)     |
| F4       | 72.80092 (3)             | 24.75082 (2)                     | 14.53085 (1)      |
| F5       | 0.003182 (1)             | 3.46E-07 (3)                     | 0.000113 (2)      |
| F6       | 0.000710 (1)             | 9.54E-08 (3)                     | 26.24046 (3)      |
| F7       | 0.000761 (1)             | 5.94E-08 (3)                     | 84.70003 (2)      |
| F8       | 0.151018 (1)             | 5.439592 (3)                     | 1.899148          |
| F9       | 0.001035 (2)             | 1.340423 (1)                     | 15.92103          |
| F10      | 2.335-05 (1)             | 1.011031 (3)                     | 84.70003 (2)      |
4. Conclusion and Future Work

The proposed algorithm has several differences when compared with the canonical PSO. The first difference is that the proposed algorithm does not apply ‘velocity term’ which is a distinctive feature in the canonical PSO and its variants. This reduces memory during optimization as the ‘Velocity’ term requires memory to contain the best location for each individual solution. The second difference is in the way each solution candidate changes over each iteration. The adaptation is not towards the Global best solution acquired but on the nearest better solution. The algorithm to find the nearest better neighbor was discussed in the previous section. The reason for applying this was to solve the early convergence problem experienced by most PSO algorithms.

The test results based on average ranking showed that proposed algorithm is comparable to GA and canonical PSO. The preliminary test seems to indicate that the proposed algorithm is suitable for highly non-convex surfaces due to the ability to explore regions within existing solutions in population. This is shown from the Auckley (F8), Penalized 1 (F10), Penalized 2 (F11), Schwefel (F6), Rastrigin (F7) functions (Refer Table 3.1). However, for more convex functions such as Spherical (F1), it performed relatively poor as compared to GA and PSO. This result suggests that works better with multimodal test functions. This is probably due to the nature of the learning mechanism which provides high local exploration. In each iteration, the frequency of ranking could change due to the surface of the test functions. This leads to a good exploration of local regions. However, the nature of this algorithm would lead to weaker Global exploration as the population does not converge to the Global best (as in the case of PSO). The search covers regions that are line of sight. Hence, a random batch mutation complements the algorithms and functions as stochastic Global exploration mechanism.

It is established based on no free lunch theorem [7] that the average performance of any algorithm over all the benchmark tests is same for all algorithm. A simple way of putting it is that there is no ‘ultimate algorithm’, only algorithms that works better with certain specific problems. With this philosophy in mind, the future development would revolve around exploring the proposed algorithm further and to further develop the algorithm as a specialized algorithm for specific problems.

Further research will include applying multiple chains of the learner to provide better global exploration. This technique will enable more global exploration in the search regions. However, this is expected to consume more computation time that could arise from overlapping search regions and therefore reduce the acquired fitness value. It is imperative to continually explore parameter tuning to suit specific test functions.
References

[1] Mitchell, Melanie (1996). An Introduction to Genetic Algorithms. Cambridge, MA: MIT Press. ISBN 9780585030944.

[2] Kennedy, J.; Eberhart, R. (1995). "Particle Swarm Optimization". Proceedings of IEEE International Conference on Neural Networks. IV. pp. 1942–1948.

[3] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesmen problem," in IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, pp. 53-66, Apr 1997.

[4] S. Janson and M. Middendorf (2003), "A hierarchical particle swarm optimizer," The 2003 Congress on Evolutionary Computation, 2003. CEC ’03., Canberra, ACT, Australia, 2003, pp. 770-776 Vol.2. doi: 10.1109/CEC.2003.1299745

[5] Yang XS. (2009) Harmony Search as a Metaheuristic Algorithm. In: Geem Z.W. (eds) Music-Inspired Harmony Search Algorithm. Studies in Computational Intelligence, vol 191. Springer, Berlin, Heidelberg

[6] Dennis Weyland, ‘A critical analysis of the harmony search algorithm—How not to solve Sudoku’ Operations Research Perspectives, Volume 2, 2015, pp: 97-105

[7] Wolpert, D.H., Macready, W.G. (1997), "No Free Lunch Theorems for Optimization", IEEE Transactions on Evolutionary Computation 1, 67.

[8] Chaparro I., Valdez F. (2013) Variants of Ant Colony Optimization: A Metaheuristic for Solving the Traveling Salesman Problem. In: Castillo O., Melin P., Kacprzyk J. (eds) Recent Advances on Hybrid Intelligent Systems. Studies in Computational Intelligence, vol 451. Springer, Berlin, Heidelberg

[9] Krishnanand, K.N. & Ghose, D. Swarm Intell (2009) 3: 87. https://doi.org/10.1007/s11721-008-0021-5

[10] Lim, Wei Hong & Mat Isa, Nor Ashidi. (2014). Teaching and peer-learning particle swarm optimization. Applied Soft Computing. 18. 39-58.

[11] J. Kennedy. Bare bones particle swarm, IEEE Swarm Intelligence Symposium, pp. 80-87, 2003.

[12] Ant Colony Optimization by Marco Dorigo and Thomas Stützle, MIT Press, 2004. ISBN 0-262-04219-3

[13] Chia-Chong Chen (2009) 'Hierarchical Particle Swarm Optimization for Optimization Problems’ Tamkang Journal of Science and Engineering, Vol. 12, No. 3, pp. 289298