2023 IEEE Symposium Series on Computational Intelligence (SSCI)
Mexico City, Mexico. December 5-8, 2023

Swarm Intelligence Numerical Optimization Algorithm Representing Individuals as Dynamic Graphs in the Euclidean Search Space

Kaho Hayashi
Kyushu Institute of Technology
Iizuka, Japan
hayashi.kaho465@mail.kyutech.jp

Kei Ohnishi
Kyushu Institute of Technology
Iizuka, Japan
ohnishi@csn.kyutech.ac.jp

***Abstract*—We propose a new swarm intelligence numerical optimization algorithm that represents individuals as dynamic graphs in the Euclidean search space. We call it Graph Building Optimization Algorithm or GBO. The unique point of GBO is that an individual is represented by a dynamic graph whose nodes have coordinates (search points) in the Euclidean search space. Due to this unique point, we can draw a GBO's search process as a generation-transition of a feature of a graph. It is expected that we can obtain better understandings on a given problem by comparing the generation-transition for the given problem to the baseline for the simplest unimodal problem. We assume the maximum node degree in the best individual as the feature and the generation-transition of the feature for F1 in the CEC'13 test problems as the baseline. We demonstrate that we can guess the characteristics of other 27 problems in the CEC'13 test problems by comparing their generation-transitions to the baseline. In addition, we evaluate GBO using the same problems and show that GBO is capable of finding good solutions for various problems.**

***Index Terms*—dynamic graph, graph building, swam intelligence optimization, numerical optimization**

I. Introduction

Recently, swarm intelligence optimization algorithms inspired by intelligent behaviors of biological swarms have been actively studied. The ant system [1] inspired by a colony of ants which are social insects, the particle swarm optimization algorithm [2] inspired by movement of a swarm of birds or fishes, and the artificial bee colony algorithm [3] inspired by a colony of bees which are also social insects are representatives of swarm intelligence optimization algorithms.

A common point among a variety of swarm intelligence optimization algorithms is that individuals equivalent to solution candidates move around in a search space according to given rules. Differences among swarm intelligence optimization algorithms are basically in rules for individual movement. However, there has not been any swarm intelligence optimization algorithm which introduces some structure such as a graph into an individual. The reason for that would be that under the purpose of optimization, it is not easy to discover a merit to introduce a structure into an individual.

Some spiders take food using webs and spider webs can be represented by graph structures. Two swarm intelligence optimization algorithms inspired by characteristics and behaviors of social spiders have been proposed. One is Social Spider Algorithm or SSA [4] and the other is Social Spider Optimization or SSO [5]. Both of SSA and SSO regard an entire search space as a web and also regard a position of each spider on the web (a point in the search space) as an individual. Thus, SSO and SSA do not represent individuals as graphs.

We previously proposed an optimization algorithm for binary problems which represents not an individual but an entire population of individuals as just one dynamic graph [6]. Each node represents a pair of position and value in a binary string of solution candidate. The topology of the graph keeps changing during the search. The algorithm generates new solution candidates through random walk on the graph. Therefore, if change in features of a graph representing a population can be observed, it could lead our understandings on problems to be solved.

Similar to our previous idea above, in the paper we propose a swarm intelligence numerical optimization algorithm in which individuals are represented by dynamic graphs whose nodes have coordinates, that is, search points, in the Euclidean search space, toward understandings on problems to be solved. We call the proposed algorithm Graph Building Optimization Algorithm or GBO. The objective of this paper is to demonstrate that GBO enables us to visualize the search process as change in the features of graphs of individuals and the visualization brings the understandings on problems to be solved to us. It is also the objective to demonstrate that GBO is capable of finding good solutions for various test problems.

There have not been evolutionary algorithms or swarm intelligence optimization algorithms for numerical optimization which represent individuals as dynamic graphs in a seaech space so far. However, apart from evolutionary algorithms and swarm intelligence optimization algorithms, an optimization method using a graph structure has been proposed. The optimization method is based on a dynamical model of behaviors of true slime molds and can find the shortest path for a given maze [7] [8]. When food is placed at two different points, a true slime mold can connect the two food sources with the

978-0-7381-4408-5/23/$31.00 ©2023 IEEE

shortest path. The mechanism for finding the shortest path is represented by differential equations regarding materials' flow at nodes in the graph. In the optimization method, an entire maze is represented as a true slime mold in a from of graph, and calculations of virtual materials' flow using the differential equations at each node bring the shortest path of the maze.

In addition, apart from optimization, ways of spider web building are used for analyzing or modeling data. An image segmentation method in medical images based on spider web building has been proposed [9]. A community detection method in social networks based on spider web building has been proposed [10]. A clustering method for nodes in wireless sensor networks based on spider web building has been proposed [11]. These studies seek to model target data for analysis as a spider web or growth of a spider web.

The paper is organized as follows. We propose GBO in Section II. Section III demonstrates that GBO brings the understandings on problems to be solved to us. In Section IV, we evaluate GBO using test problems. Section V describes the conclusions and future work.

II. Graph Building Optimization Algorithm

A. Design

The main strategies of our proposed algorithm, GBO, are below. The strategies are simple and reasonable for meta-heuristics.

- GBO makes its graph (individual) larger when the graph obtains better fitness values and makes it smaller or moves to other place when the graph does not obtain better fitness values.
- GBO reforms its graph based not on a big picture but on local rules.

First, we regard the area of a graph as the number of nodes in a graph. We then consider that graphs with better fitness values can increase their nodes, which means that graphs with worse fitness values decrease their nodes, on the other hand. Furthermore, at every generation, the worst graph in a population of graphs is randomly initialized. This procedure is realized as steps 5) and 6) in the proposed algorithm flow described in the following section. Also, IncreaseDecreaseNodesGraph and MoveGraph functions in the sixth and the seventh lines of the pseudocode shown in Algorithm 1 match this procedure.

Second, we can consider a variety of local rules for reformation of a graph. We herein roughly devise two rules.

The first rule is that a new tentative node equivalent to a new search point is generated on an enlarged edge of the graph and if the tentative node's fitness value is better than the node's at one end of the original edge, the original node is replaced by the tentative one. This procedure does not cause change in topology of the graph, but causes change in positions of nodes in the search space. This procedure is realized as step 2) in the proposed algorithm. Also, SearchPointGeneration function in the third line of Algorithm 1 matches this procedure.

The second rule is that nodes make edges to ones with better fitness values. This procedure causes change in a topology of the graph, which results in that more tentative search points are generated around nodes with better fitness values. This procedure is realized as step 3) in the proposed algorithm. Also, TopologyReformation function in the forth line of Algorithm 1 matches this procedure.

Exploration of GBO is undertaken by the random initialization of the worst graph in the first rule and generation of new search points on enlarged edges of the graph in the second rule. Exploitation of GBO is undertaken by the increase of nodes for better graphs in the first rule and the increase of edges for better nodes in the second rule.

B. Algorithm

It is assumed in the proposed algorithm that multiple graphs coexist in a search space and are represented as positions in the space. A graph is generated in a hyper rectangle. The area of a graph is represented as the number of nodes in the graph. Reformation of a graph is conducted according to local rules. All graphs initially have a constant number of nodes. After the initialization of graphs, the areas of the graphs increase or decrease depending on their obtained returns, that is to say, fitness values.

The algorithm flow is as follows. Also, the pseudocode of the algorithm is shown in Algorithm 1.

1) **Initialization of a set of graphs**
   This step randomly generates N graphs. Each graph consists of G nodes and each node has E directed edges to other nodes. More precisely, positions in a search space are randomly generated, and then a graph is randomly generated in a length of R hyper-square centered at each position.
2) **Generation of search points and movement of graphs**
   This step repeats sub-steps a) to d) below for each of N graphs.
   a) Randomly select a node, v, to start random walk from among G nodes if the number of times of the selection has not reached G. Otherwise go to step 3).
   b) Randomly select a directed edge from among E directed edges generated by the current node and move to the node, w, to which the selected directed edge is connected (see the center of Figure 1).
   c) A line segment connecting the nodes v and w is enlarged α times by adding $\alpha/2$ times length of the line segment connecting the nodes v and w to those nodes, and then a tentative node is randomly generated on the α times enlarged line segment (see the center of Figure 1) and a fitness value of the generated node, which is a point in the search space, is calculated. If the fitness value of the tentative node is better than the node v's, then the node v is replaced by the tentative one (see the right of Figure 1). This node replacement can also be regarded as node movement.
   d) If the currently visited node, w, is the H-th node excluding the start node, then the random walk

ends and the procedure goes to step 2a). Otherwise return to step 2b).

3) **Reformation of graph topologies**
Steps (i) and (ii) below are conducted.

(i) Cycle times to change directed edges for all nodes are determined. If the fitness value of a node is better than the average fitness value over all nodes', the cycle time of the node is randomly chosen from between 1 and C/2, where C is an even number and more than or equal to 2. If it is worse than or equal to the average, the cycle time of the node is randomly chosen from between C/2+1 and C.

(ii) At every time during the time period from 1 to T_R, the following operations are executed for all nodes. When the cycle time of a node, v, is a divisor of the current time t, the node becomes a target to change its directed edge. Then, random walk starts from the target node v as in the search point generation of step 2), the node v obtains K visited nodes. The node with the best fitness value among the K nodes is taken.
If the target node v has already made M_E or larger edges to the take node r, the node v does nothing. Otherwise, the node v finds a node with the worst fitness value, w, among nodes to which it has made E directed edges. Then, the node v deletes the edge to the node w and makes a new directed edge to the node r (see Figure 2).

4) **Assignment of fitness values to graphs**
The best fitness value among all nodes' in each graph is assigned to the fitness value of the graph.

5) **Increase of decrease of nodes**
The following operation is conducted N times. Two graphs are randomly selected from all graphs, and one graph with better fitness value increases a node and the other graph with worse fitness value decreases a node. There are the maximum number of nodes in a graph, G^+, and the minimum number, G^-. The number of nodes in a graph cannot exceed the maximum and the minimum numbers. When a graph increases a node, the new node and its directed edges are randomly generated. When a graph decreases a node, the node with the worst fitness value and its directed edges are deleted. Nodes which had made directed edges to the deleted node first delete those edges, and then, newly make new edges of the same number of deleted ones to randomly chosen nodes from among existing nodes in the graph. The number of edges from a node has to be less than or equal to M_E.

6) **Movement of the worst graph and reformation of the graph**
The graph with the worst fitness value among all graphs is randomly re-generated in terms of nodes and edges. The number of nodes is the same as before the re-generation.

7) **Judgement of end of run**

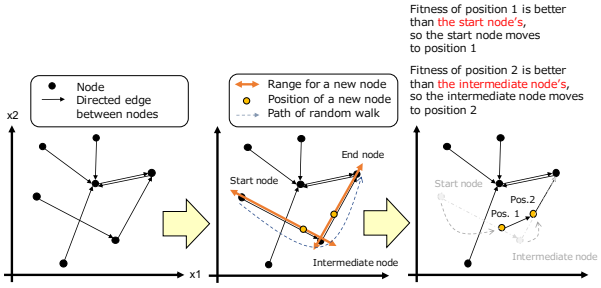

Fig. 1: Search point generation and node movement.

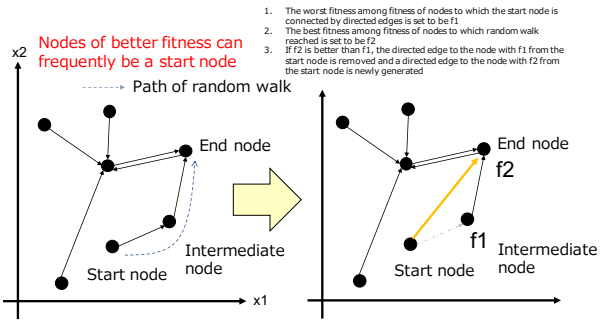

Fig. 2: Graph topology reformation.

If the number of fitness evaluations has reached the maximum number given in advance, the run ends. Otherwise, return to step 2).

The pseudocode of Graph Building Optimization Algorithm (GBO) is shown in Algorithm 1. GraphInitialization function in the first line of Algorithm 1 corresponds to step 1) in the algorithm flow described above. The second line of Algorithm 1 corresponds to step 7) in the algorithm flow. When the number of function evaluations, nfe, reaches the maximum number of function evaluations, MAX_NFE, the algorithm stops running. SearchPointGeneration function in the third line of Algorithm 1 corresponds to step 2) above. TopologyReformation function in the fourth line corresponds to step 3). GraphAssignFitness function in the fifth line corresponds to step 4). IncreaseDecreaseNodesGraph function in the sixth line corresponds to step 5). Finally, MoveGraph function in the seventh line corresponds to step 6).

Algorithm 1 Graph Building Optimization Algorithm (GBO)

Require: Grp: population of graphs, N: # of graphs, MAX_NFE: maximum # of fitness evaluations
Ensure: Grp: population of graphs
1: GraphInitialization(Grp)
2: **while** nfe is smaller than MAX_NFE **do**
3: SearchPointGeneration(Grp)
4: TopologyReformation(Grp)
5: GraphAssignFitness(Grp)
6: IncreaseDecreaseNodesGraph(Grp)
7: MoveGraph(Grp)
8: **end while**

In addition, the pseudocode of SearchPointGeneration func-

tion, which is step 2), is shown in Algorithm 2. Although the seventh line is not described in details, a tentative node, nnp, is generated on the α times enlarged edge between nodes i and j. The pseudocode of TopologyReformation function, which is step 2), is shown in Algorithm 2.

Algorithm 2 Function of SearchPointGeneration

```
Require: Grp: population of graphs, N: # of graphs, G(i): # of nodes, H: # of allowed hops
Ensure: Grp: population of graphs
for i = 1 to N do
    for g = 1 to G(i) do
        j ← RandomChooseNode(G(i))
        snp ← GetNode(i, j)
        for h = 1 to H do
            enp ← RandomChooseNextNode(snp)
            nnp ← GenerateNewNode(snp, enp)
            if Fitness(nnp) is better than Fitness(snp) then
                ReplaceNode(i, j, nnp)
            end if
            snp ← enp
        end for
    end for
end for
```

Algorithm 3 Function of TopologyReformation

```
Require: Grp: population of graphs, N: # of graphs, G(i): # of nodes, K: # of allowed hops
Ensure: Grp: population of graphs
for i = 1 to N do
    for j = 1 to G(i) do
        c(j) ← AssignCycle(j)
    end for
    for t = 1 to T do
        for j = 1 to G(i) do
            if t mod c(j) ≡ 0 then
                snp ← GetNode(i, j)
                for k = 1 to K do
                    enp ← RandomChooseNextNode(snp)
                    if k = 1 then
                        bn ← enp
                        bf ← Fitness(enp)
                    else k > 1 & Fitness(enp) is better than bf
                        bn ← enp
                        bf ← Fitness(enp)
                    end if
                    snp ← enp
                end for
            end if
            if WorstNodeFitness(j) is worse than bf then
                wn ← GetWorstLinkedNode(j)
                DeleteWorstLink(j, wn)
                MakeLink(j, bn)
            end if
        end for
    end for
end for
```

III. Example of Visualization for Search Process

A. Purpose and Simulation Settings

The good point of GBO is that rough regions that a population of individuals can produce new search points can be represented as structures, that is, graphs in a search space. Also, it is expected that nodes in those graphs are placed in good regions and their edges are connections between such good regions. Therefore, if we follow change in a population of graphs or change in features of a population of graphs, we might obtain better understandings on characteristics of a given problem and how good solutions for a given problem have been obtained.

TABLE I: Parameter settings of GBO for visualization of search process.

Parameter	Description	Value
N	the number of graphs	4
G	the number of nodes in the initial graph	1000
E	the number of directed edges that each node can generate	50
G^-	the minimum number of nodes in a graph	500
G^+	the maximum number of nodes in a graph	1500
R	range for graph generation	1000
α	enlargement ratio of an edge for search point generation	1.0
H	the number of nodes visited by a random walker for generating search points	5
C_Y	the maximum value of cycle time to change edges	10
K	the number of nodes visited by a random walker for edge change	4
T_R	the end time of the procedure for edge change	20
M_E	the number of edges allowed to connect to the identical node from a node	1

Therefore, we here demonstrate that change of the features of graphs representing individuals can be drawn when applying GBO to test problems with 10 variables. Concretely, we consider the maximum degree of node in the best individual (graph) at each generation to be the feature. A degree of node stands for the number of edges of the node. Nodes with better fitness values are expected to acquire more edges from other nodes in GBO. Therefore, if it is easy for many nodes to obtain better fitness values, many nodes have a chance to obtain more edges. As a result, high edge concentration to particular nodes is unlikely to occur. On the other hand, if only a few nodes can obtain better fitness values, high edge concentration to particular nodes is likely to occur.

We here use 28 test functions (problems) used in the CEC 2013 Special Session and Competition on Real-Parameter Optimization [12], which are notated as F1 to F28. The number of real variables of the test functions can be varied, but we here used 10. All test functions were minimization problems. The parameter settings of GBO are shown in Table I.

B. Simulation Results and Discussion

Figure 3 shows the degree distribution of the best individual (graph) at the last generation of 96 in one run of GBO for each of the 28 test problems. Figure 4 shows the generation-transition of the maximum node degree in the best individual for each of the 28 test problems. The generation-transition is the average over 51 independent runs of GBO. In Figure 4, the generation-transition for F1, which is the simplest unimodal function, is drawn in bold line. We here consider the generation-transition for F1 to be the baseline.

As shown in Figure 3, the degree distributions of the best individuals after GBO sufficiently conducted search for all test

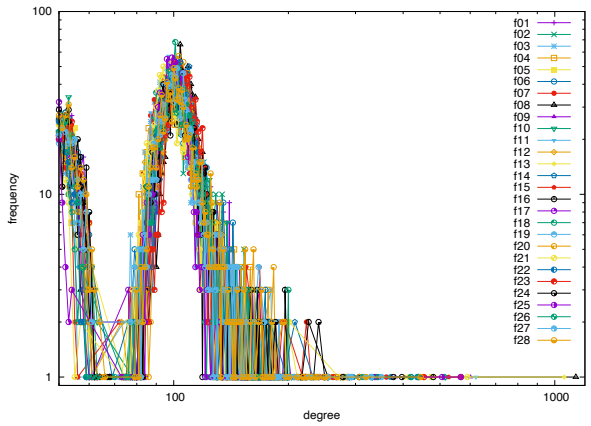

Fig. 3: Degree distribution at the last generation of 96 in one run of GBO for each test function.

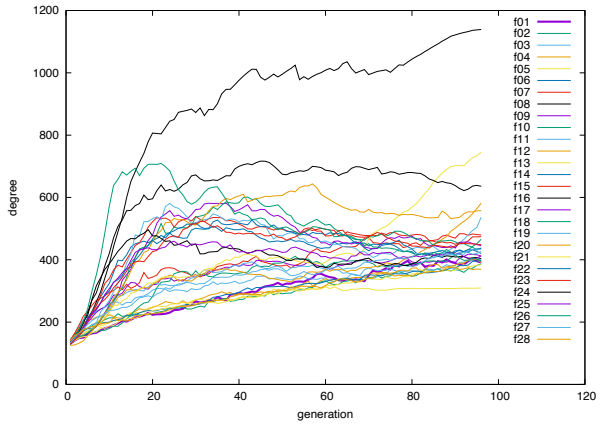

Fig. 4: Generation-transition of the maximum node degree in the best individual for each test function.

problems are roughly similar to each other. However, we can observe that the maximum node degrees are quite different among some problems. In fact, as shown in Figure 4, the generation-transition of the maximum node degree are also quite different among some problems. In the case of F1, the maximum node degree is gradually increased with generations compared to some other test problems. That would be because most nodes in a graph can easily improve their fitness values and therefore high edge concentration to particular nodes does not occur. Meanwhile, in the case of F8, the maximum node degree is rapidly increased compared to some other test problems. That would be because most nodes in a graph cannot improve their fitness values easily and only a few nodes with relatively better fitness values at the beginning keep increasing their node degrees.

Thus, we can guess that for any problems, in the phase that the maximum node degree is gradually increased as in F1, most nodes of the best individual keep improving their fitness values due to some reasons. For example, if a fitness landscape is unimodal and smooth, new search points generated on enlarged edges between two nodes would become better frequently. Meanwhile, in the phase that the maximum node degree is rapidly increased as in F8, most nodes of the best individual have stagnation of improvement of their fitness values due to some reasons. For example, if a fitness landscape is multi-modal and highly rugged or almost flat, new search points generated on enlarged edges between two nodes would not become better frequently.

Furthermore, it is suggested from the results here that problems with similar generation-transition of the maximum degree have similar characteristics. For instance, F2, F3, F4, and F5 are unimodal functions as F1. Then, their generation-transitions are indeed alike. We can utilize this kind of insights in the following way. When we have to solve a unknown black-box problem, we first obtain the generation-transition of the maximum node degree for the black-box problem by applying GBO to it. Then, we find a similar generation-transition among those of the CEC'13 test problems or those of known test problems and then know its similar test problem. Once we know its similar problem, we can choose an optimization algorithm suitable for it.

IV. Simulations for Evaluation

A. Purpose

Compared with point-population based search methods, GBO might be hard to flexibly decide generation regions for search points. In addition, since GBO allows all nodes equally to be a start node for random walk to generate new search points regardless of their fitness values, the search might be redundant. Therefore, we investigate the search performance of the present form of GBO. For this purpose, we use well-known test problems in the evolutionary computation field and compare GBO with a conventional method that has good search performance to the test problems

B. Simulation Settings

We here use the same 28 test functions [12] mentioned in Section III. The number of variables is 10. The test functions are notated as F1 to F28. All test functions are minimization problems.

As a method for comparison with GBO, Success-History based Adaptive Differential Evolution (SHADE) [13] is used. SHADE was proposed in 2013 and is one of competent variants of differential evolution (DE) algorithms. SHADE has been improved [14] [15] since the proposal. In the paper on evaluation of SHADE [16], SHADE was applied to the same test functions with 10 variables mentioned above, in which the maximum number of fitness evaluations for each run was 10^5 and the number of times of runs for each function was 51 and the best fitness value among the 51 runs was reported for each function. So, we here also apply GBO to each function 51 times and report the best fitness value among them. However, we used not only 10^5 but 2×10^6 as the maximum number

of function evaluations and reported the best fitness values at 10^5, 10^6, and 2×10^6 function evaluations for each function.

The parameter settings of GBO are the same as in Table I above. For every test function, we use the same values for the number of graphs and the number of nodes in the initial graph, which were 4 and 1000, respectively. The parameter values including these are not guaranteed to be really appropriate, but we here use the parameter values that yielded better results among limited number of combinations of values in our preliminary simulations. However, since the main purpose here is to confirm if GBO indeed works as an optimization method, these settings are enough at this juncture. We need an exhaustive investigation to know appropriate parameter settings and also an investigation on the scalability of GBO regarding the problem size as our future work.

C. Simulation Results and Discussion

The best fitness values of GBO and SHADE among the 51 runs for each test function are shown in Table II. The results of SHADE are the same as reported in [16].

We can observe from Table II that for F21, F25, and F28 of composite functions, GBO obtained better fitness values at 10^5 function evaluations than SHADE and also that for other 25 functions, SHADE obtained better fitness values than GBO. However, the number of test functions for which GBO had equal to or better than SHADE at 2×10^6 function evaluations is more than that at 10^5 function evaluations. GBO is equal to SHADE for F1, F2, F4, F5, F6, F8, and F27 and is better than SHADE for F7, F9, F10, F11, F12, F13, F16, F18, F20, F21, F22, F24, F25, and F28.

Thus, GBO requires more function evaluations to obtain good solutions for most test functions compared to SHADE, which is a competent variant of DE algorithms. However, since GBO can reach as good solutions for most test functions as SHADE with more function evaluations, we can say that GBO basically works as a optimization method.

The reason for that GBO needs more function evaluations to obtain good solutions is thought to be that the exploration procedure of GBO has redundancy. The redundant parts of GBO would be, as mentioned in the purpose of the simulations above, the step that the worst graph in a population of graphs is randomly initialized at every generation and also the step that all nodes become a start node for random walk to generate new search points no matter how good or bad their fitness values are. The degree of this redundancy is expected to become higher as the number of graphs or nodes increases. So, for larger-scale problems which should require more graphs and nodes to reliably find their good solutions, the degree of the redundancy becomes higher and much more function evaluations would be necessary.

Meanwhile, as mentioned above, GBO had better fitness values at 10^5 function evaluations than SHADE for F21, F25, and F28. The eight functions from F21 are functions that composite multiple functions with different characteristics, so are basically complicated. It is suggested that for some such problems, GBO's exploration ability is effective.

TABLE II: The best fitness values of GBO and SHADE for each test function when applying GBO and SHADE to the function with 10 variables 51 times.

	GBO			SHADE
Function	NFE=10^5	NFE=10^6	NFE=2×10^6	NFE=10^5
F1	6.3277×10	0.0000	0.0000	0.0000
F2	2.5652×10^5	0.0000	0.0000	0.0000
F3	3.7889×10^8	1.9847	1.3689×10^{-2}	0.0000
F4	3.3560×10^2	0.0000	0.0000	0.0000
F5	2.0284×10	0.0000	0.0000	0.0000
F6	1.5165×10	0.0000	0.0000	0.0000
F7	2.3300×10	3.3255×10^{-2}	5.9520×10^{-5}	9.5226×10^{-5}
F8	2.0205×10	2.0144×10	2.0120×10	2.0120×10
F9	6.0472	3.5677×10^{-1}	1.2087×10^{-3}	1.1600
F10	6.9781	7.3963×10^{-3}	7.3960×10^{-3}	0.0000
F11	2.6421×10	9.9496×10^{-1}	9.9495×10^{-1}	0.0000
F12	2.9576×10	9.9540×10^{-1}	9.9495×10^{-1}	1.2161
F13	2.9689×10	9.9496×10^{-1}	9.9495×10^{-1}	1.0962
F14	1.0286×10^3	1.8828×10^2	1.8582×10	0.0000
F15	4.1661×10^2	6.3101×10	5.6739×10	1.9595×10^2
F16	7.9135×10^{-1}	1.0814×10^{-1}	9.6707×10^{-2}	3.6202×10^{-1}
F17	3.4829×10	1.3465×10	1.1143×10	1.0122×10
F18	4.1880×10	1.3958×10	1.1065×10	1.2911×10
F19	3.0982	2.8426×10^{-1}	2.5625×10^{-1}	2.4592×10^{-1}
F20	3.3093	1.0625	8.2267×10^{-1}	1.4155
F21	3.1412×10^2	1.0007×10^2	1.0000×10^2	4.0019×10^2
F22	1.3235×10^3	4.5995×10^2	2.1781×10^2	2.4467×10^{-6}
F23	1.0756×10^3	3.0069×10^2	5.3334×10^2	1.1922×10^2
F24	1.7219×10^2	1.0435×10^2	1.0425×10^2	1.0576×10^2
F25	1.6581×10^2	1.0570×10^2	1.0499×10^2	2.0000×10^2
F26	1.3932×10^2	1.0268×10^2	1.0198×10^2	1.0125×10^2
F27	4.3923×10^2	3.0028×10^2	3.0000×10^2	3.0000×10^2
F28	2.7327×10^2	1.0000×10^2	1.0000×10^2	3.0000×10^2

Therefore, we will consider the future direction of improving GBO to be that the characteristics of a given problem is first captured as a form of graphs during search, and then an appropriate degree of exploration required for the problem is estimated based on the features of graphs, and the degree of exploration is dynamically adjusted according to the estimation.

V. Concluding Remarks and Future Work

In the paper we proposed a new swarm intelligence numerical optimization algorithm that represents individuals as dynamic graphs in the Euclidean search space. We called it Graph Building Optimization Algorithm or GBO. The unique point of GBO is that an individual is represented by a dynamic graph whose nodes have coordinates (search points) in the Euclidean search space. Due to this unique point, we can draw a GBO's search process as a generation-transition of a feature of a graph. Then, it is expected that we can obtain better understandings on a given problem by comparing the generation-transition for the given problem to the baseline for the simplest unimodal problem. We assumed the maximum node degree in the best individual as the feature and the generation-transition of the feature for F1 in the CEC'13 test problems as the baseline. We demonstrated that we could guess the characteristics of other 27 problems in the CEC'13 test problems by comparing their generation-transitions to the baseline.

We also evaluated GBO using the CEC'13 test problems with 10 variables. We used SHADE for comparison, which is one of competent variants of differential evolution algorithms. The results showed that GBO obtained better solutions for a few composite problems and that GBO did not obtain as good solutions as SHADE with the same number of function

evaluations for most problems, but could obtain good solutions for those problems with more function evaluations. Therefore, we concluded that GBO is basically capable of finding good solutions for various test problems.

The results obtained in the paper suggested that problems with similar generation-transition of the maximum degree have similar characteristics. In the future work, we will confirm it with more evidences. If surely confirmed, when we have to solve a unknown black-box problem, we can find similar well-known test problems to the unknown one by comparing the generation-transition of the maximum node degree among the unknown and well-known ones. Once we know its similar problem, we can choose an optimization algorithm suitable for it. In addition, we will improve GBO using the features of graphs. In the improved GBO, the characteristics of a given problem is first captured as a form of graphs of individuals during search, and then an appropriate degree of exploration required for the problem is estimated based on the features of graphs, and the degree of exploration is dynamically adjusted according to the estimation.

ACKNOWLEDGEMENT

This work is supported by the Japan Society for the Promotion of Science through a Grant-in-Aid for Transformative Research Areas (A) (Publicly Offered Research) (21A402).

REFERENCES

[1] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: optimization by a colony of cooperating agents," *IEEE Trans. on System, Man, and Cybernetics-Part B*, vol. 26, no. 2, pp. 29–41, 1996.

[2] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, 1995, pp. 1942–1948.

[3] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," in *Technical Report-TR06*, 2006, pp. 1–10.

[4] J. J. Yu and V. O. Li, "A social spider algorithm for global optimization," in *Applied Soft Computing 30(2015)*, 2015, pp. 614–627.

[5] A. Luque-Chang, E. Cuevas, F. Fausto, D. Zaldívar, and M. Pérez, "Social spider optimization algorithm: Modifications, applications, and perspectives," in *Mathematical Problems in Engineering*, 2018, pp. 1–13.

[6] T. Sato and K. Ohnishi, "A metaheuristic relying on random walk on a graph for binary optimization problems," in *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2018, pp. 739–744.

[7] T. Nakagaki, H. Yamada, and A. Toth, "Maze-solving by an amoeboid organism," *Nature*, vol. 407, no. 6803, p. 470, 2000.

[8] A. Tero, R. Kobayashi, and T. Nakagaki, "A mathematical model for adaptive transport network in path finding by true slime mold," *Journal of Theoretical Biology*, vol. 244, no. 4, pp. 553–564, 2007.

[9] V. Muneeswaran and M. P. Rajasekaran, "Automatic segmentation of gallbladder using bio-inspired algorithm based on a spider web construction model," *Supercomputing*, vol. 75, pp. 3158–3183, 2019.

[10] H. Yang, J. Cheng, X. Su, W. Zhang, S. Zhao, and X. Chen, "A spiderweb model for community detection in dynamic networks," *Applied Intelligence*, vol. 51, pp. 5157–5188, 2021.

[11] J. Wang, Y. Zhang, X. Wang, P. Mao, and B. Liu, "A multi-objective parameter optimization approach to maximize lifetime of wireless sensor networks inspired by spider web," *Supercomputing*, vol. 79, pp. 1263–1288, 2023.

[12] P. N. S. J. J. Liang, B. Y. Qu and A. G. Hernández-Díaz, "Problem definitions and evaluation criteria for the cec 2013 special session on real-parameter optimization," in *Technical Report, Nanyang Technological University, Singapore*, 2013, pp. 1–39.

[13] R. Tanabe and A. Fukunaga, "Success-history based parameter adaptation for differential evolution," in *2013 IEEE Congress on Evolutionary Computation*, 2013, pp. 71–78.

[14] S. Chakraborty, S. Sharma, A. K. Saha, and S. Chakraborty, "Shadewoa: A metaheuristic algorithm for global optimization," *Applied Soft Computing*, vol. 113, p. 107866, 2021.

[15] Y. Li, T. Han, H. Zhou, S. Tang, and H. Zhao, "A novel adaptive l-shade algorithm and its application in uav swarm resource configuration problem," *Information Sciences*, vol. 606, pp. 350–367, 2022.

[16] R. Tanabe and A. Fukunaga, "Evaluating the performance of shade on cec 2013 benchmark problems," in *2013 IEEE Congress on Evolutionary Computation*, 2013, pp. 1952–1959.