Hindawi
Discrete Dynamics in Nature and Society
Volume 2017, Article ID 2013673, 15 pages
https://doi.org/10.1155/2017/2013673

Research Article

GPU-Based Parallel Particle Swarm Optimization Methods for Graph Drawing

Jianhua Qu,[1] Xiyu Liu,[1] Minghe Sun,[2] and Feng Qi[1]

[1]*College of Management Science and Engineering, Shandong Normal University, Jinan, Shandong, China*
[2]*College of Business, The University of Texas at San Antonio, San Antonio, TX, USA*

Correspondence should be addressed to Jianhua Qu; qujh1978@163.com

Received 17 March 2017; Accepted 15 June 2017; Published 30 July 2017

Academic Editor: Filippo Cacace

Particle Swarm Optimization (PSO) is a population-based stochastic search technique for solving optimization problems, which has been proven to be effective in a wide range of applications. However, the computational efficiency on large-scale problems is still unsatisfactory. A graph drawing is a pictorial representation of the vertices and edges of a graph. Two PSO heuristic procedures, one serial and the other parallel, are developed for undirected graph drawing. Each particle corresponds to a different layout of the graph. The particle fitness is defined based on the concept of the energy in the force-directed method. The serial PSO procedure is executed on a CPU and the parallel PSO procedure is executed on a GPU. Two PSO procedures have different data structures and strategies. The performance of the proposed methods is evaluated through several different graphs. The experimental results show that the two PSO procedures are both as effective as the force-directed method, and the parallel procedure is more advantageous than the serial procedure for larger graphs.

1. Introduction

Graphs are often used to depict objects and to represent the relationship between objects. In a social network, for example, vertices represent individual users and edges represent their relationships if users are acquainted. Graph drawing is a conventional tool for the visualization of graphs. The effect of graph visualization depends on whether the drawing is aesthetic. Although there are no strict criteria for aesthetics of a drawing, it is generally agreed that a drawing with the following characteristics is aesthetic [Battista et al., 2012; [1]]:

(1) Minimal edge crossings are in the graph.

(2) Vertices are evenly distributed in the drawing canvas.

(3) Connected vertices are close to each other.

(4) Symmetry may exist in the graph.

With the development of graph theory and computer science, it is possible to visualize a graph automatically. Many automatic methods for graph drawing emerged, especially for planar graphs. For nonplanar graphs, efficient or effective drawing methods are needed. Tutte [2] proposed an algorithm to draw planar graphs by fixing vertices on a face and placing the rest of the vertices at the barycentre of their neighbors. Sugiyama et al. [3] proposed a method for the hierarchical drawing of directed graphs. Eades [4] proposed a force-directed heuristic method for graph drawing. In recent years, many other methods have been proposed [5, 6]. Among these methods, the force-directed method is popular, which uses a heuristic cost or energy function to map the layout of a graph to a real number. The layout with the minimum cost or energy is aesthetic. Due to simplicity and flexibility of implementation, some variants of the force-directed method were proposed. Kamada and Kawai [7] proposed the spring embedding algorithm in which the ideal length of a spring is proportional to the distance between the vertices. Fruchterman and Reingold [8] improved the force-directed method by introducing the attractive forces and the repulsive forces. This improved force-directed method is called the F-R method in the following. All the variants

differ in the definition of the energy or in the optimization method used to find the minimum of the energy. The force-directed method is especially suitable for drawing graphs with straight-line edges.

In this study, the force-directed method is also used to draw undirected graphs with straight-line edges. In the force-directed method, the layout of a graph is obtained by finding the minimum of the cost function. Therefore, the layout problem can be converted to an optimization problem. In this study, a Particle Swarm Optimization (PSO) procedure is proposed to solve the graph drawing problem.

PSO is a stochastic global optimization approach developed by Eberhart and Kennedy [9]. As a population-based metaheuristic, PSO has the advantages of robustness, effectiveness, and simplicity compared to other swarm intelligent approaches such as genetic algorithms and ant colony optimization [10–12]. In a PSO procedure, a swarm of particles is kept and each particle in the swarm adjusts its position by keeping track of its own and the global best positions. The quality of each particle is measured by a fitness function. At the end, the search is expected to converge to the global best solution of the whole search space. In recent years, PSO has been widely applied to many complex and difficult optimization problems in practice, and many improved PSO procedures have been proposed [13–16].

Although PSO has the above advantages, it still needs a long computation time to converge for large-scale problems needing a large number of particles to search for the optimal solution. The main reason is that the serial computation of the fitness values of the particles in a swarm takes too much computation time. If the computations of the fitness values of the particles are independent of each other, the computation can be decomposed for parallel operation to improve efficiency. In recent years, many parallel PSO procedures have been proposed. Graphic Processing Units (GPUs) have high speed, parallelism, and programmable functions. Therefore, GPUs have great potential in the field of general computing. Some of the parallel PSO procedures reported in the literature are implemented on GPUs [17–20] with high-performance parallel computing capabilities using NVIDIA's Compute Unified Device Architecture (CUDA) [21] as a handy programming environment.

Schutte et al. [2003] designed the first synchronous parallel PSO procedure and successfully applied it to the biomechanical system identification problem. Venter et al. [2006] and Koh et al. [22] implemented asynchronous parallel PSO procedures to improve computational efficiency. Some methods proposed in the literature focus on the communication strategies or the neighborhood topologies [23–25]. These methods are all more efficient than serial PSO procedures and are implemented on distributed systems. The first GPU implementation of PSO was proposed by Li et al. [26]. With the convenient programming environment in NVIDIA CUDA, more methods are implemented on GPU and in CUDA [21]. The implementation of parallel PSO procedures on GPU has higher demands on information sharing than on CPU. Prasain et al. [27] designed serial and parallel procedures for the option pricing algorithm using basic principles of PSO and evaluated the performance of these procedures on a cluster of multicore machines. Solomon et al. [28] implemented a collaborative multiswarm PSO procedure on GPU using many swarms rather than just one. They applied this PSO procedure to a real-world application, the task matching problem, in a heterogeneous distributed computing environment. Roberge and Tarbouchi [29] developed parallel implementation of PSO on CUDA-GPU and applied this parallel PSO procedure to the problem of 3D pose estimation of a bomb in free fall. Souza et al. [30] proposed a cooperative evolutionary multiswarm optimization procedure based on CUDA to solve engineering problems. The procedure used the concept of master/slave swarm with the mechanism of data sharing for the acceleration of convergence. Therefore, parallel PSO procedures and their variants based on GPU and CUDA have been used in many domains.

In this study, two PSO procedures, a serial PSO procedure for graph drawing (S-PGD) and a parallel PSO procedure for graph drawing at vertex level (V-PGD), are proposed for undirected graph drawing. These two PSO procedures have similar modules but different implementation. PSO is used to optimize the graph drawing problem. The graph is initialized as a swarm of random particles. Each particle stores the position information of all vertices in the graph and corresponds to one layout of the graph. All particles automatically update their positions and velocities in the searching process for the optimal layout until the procedure terminates. Each particle has a fitness value representing the energy of the corresponding layout. The definition of the fitness function is given on the basis of the force-directed method. The performances of the two procedures are analyzed for their effectiveness, running time, and convergence through experiments on different graphs.

The remainder of this paper is organized as follows. In Section 2, introductions to the force-directed method for graph drawing and to PSO are given. Section 3 describes the S-PGD procedure in detail including the structure of the particles, the definition of the fitness function, and a pseudocode of the procedure. In Section 4, the implementation of V-PGD is explained and its time complexity is analyzed. The performances of the two procedures are examined in Section 5 through several experiments on different graphs. Conclusions and future research directions are given in Section 6.

2. Related Works

The force-directed graph drawing method is briefly discussed first. Relevant concepts in PSO are then reviewed.

2.1. Force-Directed Graph Drawing. Due to its simple implementation and good flexibility, the force-directed method is often used for drawing undirected graphs with straight-line edges. The methods in Eades [4], Fruchterman and Reingold [8], Hu [31] and Kamada, and Kawai [7] are some known examples based on the force-directed method.

The spring-embedder method proposed by Eades [4] is the earliest method for drawing general graphs. This method likens a graph to a system with electrically charged rings (the vertices) and connecting springs (the edges). Any two vertices

are pushed by a repulsive force and adjacent vertices connected by an edge are pulled together by an attractive force. The method seeks equilibrium of these conflicting forces as constraints and is very successful with small graphs. Let d_{jk} represent the Euclidean distance between the two vertices j and k. The attractive and repulsive forces between vertices j and k, represented by $f_a(d_{jk})$ and $f_r(d_{jk})$, respectively, are defined by [4]

$$f_a\left(d_{jk}\right) = K \log d_{jk},$$
$$f_r\left(d_{jk}\right) = \frac{K}{d_{jk}^2}, \tag{1}$$

where K is the radius of the circle. The center of the circle is a vertex and no other points are in the circle. In the F-R method [8], the attractive force is defined as follows:

$$f_a\left(d_{jk}\right) = \frac{d_{jk}^2}{K}, \tag{2}$$

and the repulsive force is defined as follows

$$f_r\left(d_{jk}\right) = -\frac{CK^3}{d_{jk}^2}, \tag{3}$$

where K and C are constants. The final layout has a locally minimal energy with respect to the vertex positions. It can be seen from (2) and (3) that the attractive force is proportional and the repulsive force is inversely proportional to the squared Euclidean distance.

2.2. Particle Swarm Optimization. PSO is a simple but powerful heuristic optimization technique introduced by Eberhart and Kennedy [9]. It is a global optimization method using a swarm of particles with random positions searching for the best position by updating their velocities and positions. Each particle in the swarm searches the optimum of a function, termed the fitness function, by keeping track of the best position it has found and the best position found by the whole swarm of particles. The best position found by a particle is called the local best position of the particle, and the best position found by the whole swarm is called the global best position. The domain of the fitness function is called the search space. Guided by the local best position and the global best position found so far, particles move over the search space in the searching process for an optimum.

Let Pos_i^t and Vel_i^t be the position vector and the velocity vector, respectively, of particle i at time t. Let Pos_i^{pbest} represent the local best position found by particle i and let Pos^{gbest} represent the global best position found by all the particles in the whole swarm up to the current time t. The position vector and velocity vector of particle i at time $t+1$ can be computed using (4) and (5), respectively, as follows:

$$\mathrm{Vel}_i^{t+1} = \omega \mathrm{Vel}_i^t + c_1 R_1 \left(\mathrm{Pos}_i^{pbest} - \mathrm{Pos}_i^t\right) + c_2 R_2 \left(\mathrm{Pos}^{gbest} - \mathrm{Pos}_i^t\right) \tag{4}$$

$$\mathrm{Pos}_i^{t+1} = \mathrm{Pos}_i^t + \mathrm{Vel}_i^{t+1}, \tag{5}$$

where ω is the inertia weight; R_1 and R_2 are two random numbers; and c_1 and c_2 are the cognitive and social scaling parameters.

3. S-PGD: A Serial PSO Procedure for Graph Drawing on CPU

S-PGD is discussed in detail in this section. After the structure and the fitness function are described, a pseudocode is presented.

3.1. Structure of the Particles. The key of designing a good PSO procedure is to determine the structure of the particles. A good structure makes the problem simple and intuitive. In this work, designing a PSO heuristic for the graph drawing problem is a difficult task. The following method is adopted for the structure of a particle.

Let $G(V, E)$ be an undirected graph, where V is a set of vertices or nodes and E is the set of edges or links. A layout of a graph is a mapping of the vertices to the Euclidean space: $V \to R^2$. That is to say, a vertex is mapped to a position in the Euclidean space in the graph drawing procedure. Different distributions of the vertices correspond to different layouts. The following model is used to represent the structure of the particle swarm:

$$S = \left(\left(P_1^t, P_2^t, \ldots, P_i^t, \ldots, P_m^t\right), \mathrm{Pos}^{gbest}, f^{gbest}, m, n, t\right), \tag{6}$$

$$P_i^t = \left(\mathrm{Pos}_i^t, \mathrm{Vel}_i^t, \mathrm{Pos}_i^{pbest}, f_i^{pbest}\right), \quad i = 1, 2, \ldots m, \tag{7}$$

$$\mathrm{Pos}_i^t = \left(\left(p_{i1}^t, p_{i2}^t, \ldots, p_{ij}^t, \ldots, p_{in}^t\right)\right), \tag{8}$$

$$\mathrm{Vel}_i^t = \left(\left(v_{i1}^t, v_{i2}^t, \ldots, v_{ij}^t, \ldots, v_{in}^t\right)\right). \tag{9}$$

In (6), S is the structure of the particle swarm; P_i^t is the structure of particle i; Pos^{gbest} and f^{gbest}, as defined above, represent the global best position and the global best fitness value found by the whole swarm up to the current time t; m is the number of the particles in the swarm; and n is the dimension of the search space, that is, the number of vertices in V. In (7), as defined above, Pos_i^t is the position vector and Vel_i^t is the velocity vector of particle i; Pos_i^{pbest} represents the local best position and f_i^{pbest} represents the local best fitness value found by particle i up to the current time t. In (8) and (9), p_{ij}^t is the position and v_{ij}^t is the velocity of vertex j in particle i at the current time t. For vertex j, both p_{ij}^t and v_{ij}^t are two dimensional vectors. Furthermore, let $N = mn$ represent the total number of vertices in the whole particle swarm.

A particle with such a structure corresponds to one layout of the graph. The energy of the layout is converted to the fitness value of the particle. Whether a layout is aesthetic can be evaluated by the fitness value $f(\mathrm{Pos}_i^t)$. The objective of a graph drawing algorithm is to find an aesthetic layout with the best fitness value. It is evident that the particle with the best fitness value corresponds to an optimal layout of the graph. Each particle in the swarm updates its velocity and position using (4) and (5), respectively, in the searching process.

3.2. Fitness Function. The selection of a fitness function is very important for the success of the PSO procedure. An efficient fitness function is helpful for the particles in the swarm to find good solutions quickly. In the PSO procedure, a particle corresponds to one layout of graph drawing. The objective of graph drawing is to find an aesthetic visual representation of the vertices and the links between the vertices. Different types of graphs need different types of representations. However, no uniform criteria can be used to evaluate the performance of different representations. Inspired by the spring-electrical model, the ideas in the F-R method [8] and the method in Hu [31] are used to define the fitness function.

The force-directed method is an iterative procedure that repeatedly calculates the attractive and repulsive forces of each vertex and then moves the vertices along the direction of the forces for a displacement until the layout reaches a stable state. The attractive and the repulsive forces between two vertices j and k are defined in (2) and (3), respectively.

The energy contribution by the attractive and repulsive forces of the link between two vertices j and k in a layout at any given time is defined as follows:

$$f_{ec}\left(d_{jk}\right) = f_a\left(d_{jk}\right) + f_r\left(d_{jk}\right). \tag{10}$$

The fitness function of particle i at time t representing a layout based on the energy contributions of the links is defined as follows:

$$f\left(\mathrm{Pos}_i^t\right) = \sum_{jk \in E} f_{ec}\left(d_{jk}\right) \quad \text{for } i = 1, \ldots, m. \tag{11}$$

The fitness function maps the position vector Pos_i^t of particle i into a real number representing the energy of the layout $f(\mathrm{Pos}_i^t)$. The energy will be low if adjacent vertices in the original graph are close to each other in the layout and will be high otherwise. A particle with the minimum value of the fitness function is a global optimal particle. An aesthetic layout of a graph is obtained by searching for a minimum value of the fitness function. Therefore, the graph drawing problem reduces to the problem of finding a minimum value for the fitness function.

3.3. Pseudocode. S-PGD has four basic modules: initialization, fitness value computation, local and global best position and best fitness value update, and position and velocity update of the particles. In S-PGD, the fitness value computation, the most time-consuming module, is based on the definition in (11). It computes the fitness value by adding together the energy contributions of the links. Pseudocode 1 is a detailed description of S-PGD. In the pseudocode, Itr is the number of iterations.

4. V-PGD: The Parallel PSO Procedure on GPU

In this section, V-PGD, the proposed parallel PSO procedure at vertex level on CUDA-GPU, is described. Although having the same operations in the modules of initialization, local and global best position and best fitness value update, and position and velocity update for the particles as in S-PGD, V-PGD adopts parallel strategies in the module of fitness computation to reduce computation time. There are three reasons for doing this.

The first reason is that the GPU has more cores in comparison to the CPU. A CPU commonly has 4 to 8 fast and flexible cores, whereas a GPU has hundreds of relatively simple cores. Tasks that can be efficiently divided across many threads will see enormous benefits when running on a GPU.

The second reason is that the performances of two parallel methods should be compared in the same environment. It is difficult to run the parallel procedure on multicore CPU due to the large number of particles and vertices.

The last reason is that CUDA is a handy tool to develop parallel scientific codes for massively parallel computation. It is actually sufficient to install a compatible GPU and the compiler SDK [21] to develop parallel codes using a high-level computer language. The computer language C is used in this study.

In order to take full advantage of parallelization offered by CUDA-GPU, the following two specific programming guidelines are followed [21]:

(1) Minimize data transfers between CPU and GPU.

(2) Minimize the use of global memory. Shared memory is more preferred.

CUDA C extends C by allowing the programmer to define C functions, called kernels, that, when called, are executed multiple times in parallel by multiple different CUDA threads, as opposed to only once like regular C functions [21]. The kernel function does the following:

(1) Loads data from the global memory

(2) Processes data

(3) Sends results back to the global memory

4.1. V-PGD: The Vertex-Level Parallel PSO Procedure for Graph Drawing. The energy contribution of vertex j in the layout of particle i at time t is the sum of the energy contributions of all the links connected to vertex j, which is given as follows:

$$f_v\left(p_{ij}^t\right) = \sum_{\{jk\} \in E} f_{ec}\left(d_{jk}\right) \quad \text{for } j = 1, \ldots, n. \tag{12}$$

The fitness function of particle i representing a layout at time t in (11) can be written as (13) based on the energy contribution of vertices (12):

$$f\left(\mathrm{Pos}_i^t\right) = \frac{1}{2}\sum_j f_v\left(p_{ij}^t\right) \quad \text{for } i = 1, \ldots, m. \tag{13}$$

In V-PGD, a kernel function is defined to compute the energy contribution of each vertex using (12) in parallel. When it is called, the kernel function executes N times in parallel by N different CUDA threads and once for each vertex in V in the layout of a particle. The computation of the

```
% Initialization
Set the values for the parameters ω, c1, c2 and Itr;
Randomly initialize the position vector Pos_i^0 for each particle i, for i = 1, 2, ... m;
Randomly initialize the velocity vector Vel_i^0 for each particle i, for i = 1, 2, ... m;
for t = 1 to Itr
{    % Computation of fitness for each particle
     for i = 1 to m
     {    f(Pos_i^t) = 0;
          for j = 1 to n
               for k = 1 to n
               {    if (jk) ∈ E
                         compute d_jk;
                         compute f_a(d_jk) using (2);
                         compute f_r(d_jk) using (3);
                         compute f_ec(d_jk) using (10);
                         f(Pos_i^t) = f(Pos_i^t) + f_ec(d_jk);
               }
          % Update Pos_i^pbest and Pos^gbest
          If f(Pos_i^t) ≤ f_i^pbest, then f_i^pbest = f(Pos_i^t), Pos_i^pbest = Pos_i^t;
          If f(Pos_i^t) ≤ f^gbest, then f^gbest = f(Pos_i^t), Pos^gbest = Pos_i^t;
          % Update Pos_i^(t+1) and Vel_i^(t+1)
          Update Vel_i^(t+1) using (4);
          Update Pos_i^(t+1) using (5);
     }
}
Output the final best layout Pos^gbest.
```

Pseudocode 1: Pseudocode of S-PGD.

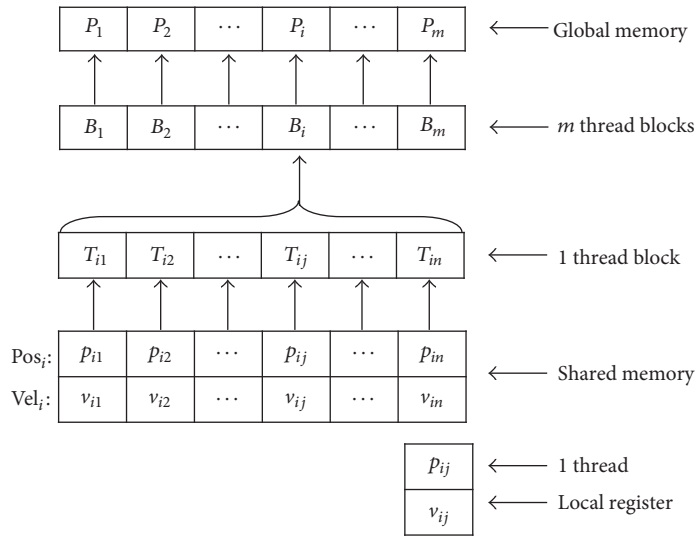

Figure 1: Data structure and parallel pattern of V-PGD.

energy contribution in (12) in V-PGD is quite different from that in S-PGD, where the fitness defined in (11) is computed in serial by one thread. Such parallel implementation greatly reduces the running time and improves the performance of the procedure. Figure 1 gives a detailed description of the data structure and the parallel pattern in V-PGD. In Figure 1, B_i represents thread block i in CUDA, and T_{ij} represents thread j in thread block i. One thread executes the kernel function to compute the energy contribution of one vertex (12) in a layout of a particle and n threads in one thread block execute the kernel function n times in parallel to compute the contributions of all n vertices in one layout at the same time.

As shown in Figure 1, the position vectors of all particles are transferred to the global memory once at the start of each iteration. Each thread reads the position vector of one particle from the global memory to the shared memory of its block

```
Initialization
for t = 1 to Itr = 1
{    Compute the fitness using the kernel function of V-PGD;
     Update Pos_i^pbest and Pos^gbest;
     Update Pos_i^{t+1} and Vel_i^{t+1};
 }
Output the final layout P^gbest.
```

Pseudocode 2: Pseudocode of V-PGD.

```
Transfer position vectors of all particles from CPU to GPU
(p_ij^t) = 0;
for k = 1 to n
{    if (j, k) ∈ E then
     compute d_jk;
     compute f_a(d_jk) using (2);
     compute f_r(d_jk) using (3);
     compute f_ec(d_jk) using (10);
     f(p_ij^t) = f(p_ij^t) + f_ec(d_jk);
}
syncthreads();
compute f(Pos_i^t) in the parallel reduction
```

Pseudocode 3: Pseudocode of the kernel function of V-PGD.

and only computes the energy contribution of one vertex which has the same index as the thread does. The pseudocode of V-PGD is shown in Pseudocode 2.

A pseudocode of the kernel function is shown in Pseudocode 3. It is for a typical vertex j in particle i. The kernel function is executed at each vertex in parallel in the n vertices of a layout of a particle within each of the m thread blocks implementing the particle swarm. Hence, it runs on N threads in parallel at the same time.

V-PGD is implemented using one CUDA kernel for one swarm, one particle in the swarm corresponds to a thread block, and one vertex of the layout in a particle corresponds to one thread. The number of thread blocks and the number of particles in a swarm are the same. Each thread only computes the contribution to the fitness of one vertex. The number of threads in each thread block is equal to the number of vertices in the graph. After each thread in the same block has computed the contribution to the fitness of one vertex, the sum of the contributions to the fitness of all vertices in the corresponding layout of the particle is computed by means of parallel reduction [Karniadakis et al., 2003]. Reduction is a parallel approach to compute the sum and can take further advantage of the power of parallel computation.

4.2. Time Complexity. In fact, fitness computation is often the most computation-intensive module in a PSO procedure. The time complexity of fitness computation is usually considered to be a significant measure when evaluating the performance of a procedure. In S-PGD, all modules are implemented serially on a single CPU. In each iteration, the time complexity of fitness computation is $O(mn^2)$. It is evident that the running time becomes longer and longer when the number of particles and the number of vertices increase.

In V-PGD, one thread block with n threads calculates the fitness value of a particle. There are m thread blocks. In each iteration, all N threads call the same kernel function simultaneously. The pseudocode of the kernel function of V-PGD shows that the time complexity of the kernel function is $O(n)$. Because all threads call the kernel function simultaneously, the time complexity of V-PGD is also $O(n)$. Therefore, the time complexity of V-PGD is reduced to $O(n)$.

4.3. Updating Strategy of the Best Positions. The serial and parallel PSO procedures have another difference in their updating strategies of the best positions in addition to their implementation of the fitness computation. The flowcharts in Figure 2 show this difference. S-PGD uses a synchronous strategy to update the best positions Pos_i^{pbest}, for $i = 1, \ldots, m$, and Pos^{gbest}. Each Pos_i^{pbest}, for $i = 1, \ldots, m$, or Pos^{gbest} is updated immediately after the fitness of a particle is computed in each iteration. V-PGD uses an asynchronous strategy to update the best positions Pos_i^{pbest}, for $i = 1, \ldots, m$, and Pos^{gbest}. Each Pos_i^{pbest}, for $i = 1, \ldots, m$, or Pos^{gbest} is updated at the end of each iteration after the fitness of all particles in a swarm has been computed. The asynchronous strategy skips some computation steps and, therefore, makes the swarm find the optimal solution more quickly.

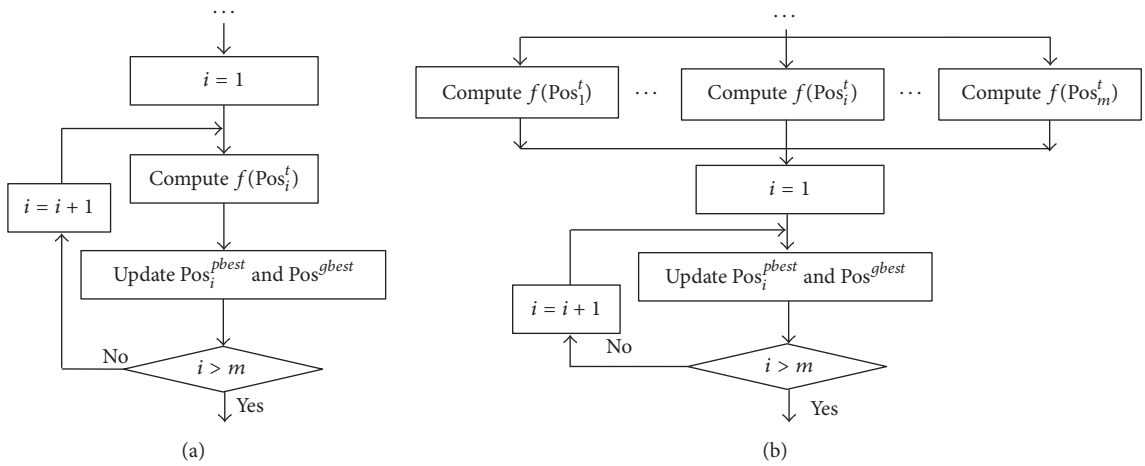

Figure 2: The updating strategies: (a) the synchronous updating strategy of S-PGD and (b) the asynchronous updating strategy of V-PDG.

5. Experiments and Performance Analysis

Computational results on some graphs are reported in this section. Some of the graphs are from the literature and others are randomly generated. Since the fitness function is based on the force-directed method, the performance of the two PSO procedures is compared with that of the F-R method [8] on the layouts of graph drawing. The performance of V-PGD is also compared with that of S-PGD. These two procedures are compared on some graphs with different numbers of vertices and edges. The parameters in the PSO procedures are set to $\omega = 0.72$ and $c_1 = c_2 = 2.02$. The computer used for the experiments in this study has an Intel Core 2 Quad 2.83 GHz CPU, 4.00 G RAM, and NVIDIA GeForce GTX 560 Ti GPU using the Windows 8 operating system. For the attractive and repulsive forces defined in (2) and (3), values of the constants K and C similar to those in the F-R method [8], that is, $K = 0.1$ and $C = 0.2$, are used. Tmp in the F-R method is the temperature parameter [8].

5.1. Effectiveness. Although S-PGD and V-PGD can find similar solutions, the time complexity of V-PGD is much lower and, therefore, is used for the comparison with the F-R method. In V-PGD, $m = 20$ and Itr $= 300$ are used. In the F-R method, Tmp = 5 and Itr = 300 are used. The layouts obtained by the F-R method and by V-PGD on 10 graphs from Fruchterman and Reingold [8] are listed in Figure 3. Most of the drawings by V-PGD look aesthetic and symmetric. Specifically, the layout of g2 obtained by V-PGD has fewer crossing edges than that obtained by the F-R method.

Results on three real graphs from social networks obtained with the F-R method and with V-PGD are compared in Figure 4. The first, g_K, is Zachary's karate club (data and network are available at UCI Network Data Repository, http://networkdata.ics.uci.edu/data.php?id=105) [32] which has 34 vertices and 78 edges. The second, g_D, is the dolphin social network (data and network are available at UCI Network Data Repository, http://networkdata.ics.uci.edu/data.php?id=6) [33] which has 62 vertices and 105 edges. The last, g_F, is the American college football network (data and network are available at UCI Network Data Repository, https://networkdata.ics.uci.edu/data.php?id=5) [32] which has 115 vertices and 613 edges. The parameters in V-PGD are set to $m = 50$ and Itr = 2000 for g_K, $m = 60$ and Itr = 4000 for g_D, and $m = 80$ and Itr = 6000 for g_F. The parameters in the F-R method are set to Tmp = 20 and Itr = 2000 for g_K, Tmp = 30 and Itr = 3000 for g_D, and Tmp = 50 and Itr = 5000 for g_F. The layouts obtained by these two methods look aesthetic and are distributed evenly.

The graphs g3, g6, g7, g8, and t2 are also drawn by V-PGD with varying values of m to see its effects. The evolutionary drawings with varying values of m by V-PGD are shown in Figure 5. When $m = 1$, all of the five graphs are poorly drawn. The layouts become better and better with the increase in the value of m. When m increases to 10, g3, g7, and g8 have aesthetic layouts. When m increases to 20, all of them have aesthetic layouts.

Unlike in the F-R method, no restrictions are imposed on the size of the drawing canvas in either S-PGD or V-PGD, although the size of the drawing canvas determines the size of the layout of a graph. Most of the layouts obtained by V-PGD in Figures 3–5 lie in the unit square $[0, 1] \times [0, 1]$ except for g8 with $m = 1$ and t2 with $m = 1$ in Figure 5, whereas the drawing canvas of the F-R method has different sizes for different graphs.

In V-PGD, two factors determine the size of the drawing canvas. One factor is the initial layout, and the other is evolution of the vertices in the PSO optimization process. When the PSO procedure is initialized, the positions of the vertices are randomly generated in the unit square. In the PSO optimization procedure, the positions of the vertices are not limited to stay within the unit square and eventually some of the vertices may move out of the unit square. As a result, the layouts of most of the graphs obtained by V-PGD are in the unit square with a few exceptions.

The drawing canvas in the F-R method is defined from experience before the drawing starts. In the searching process, a strategy is used to adjust the positions of those vertices outside of the drawing canvas. Hence, the final layout is within the predefined drawing canvas. The results show that,

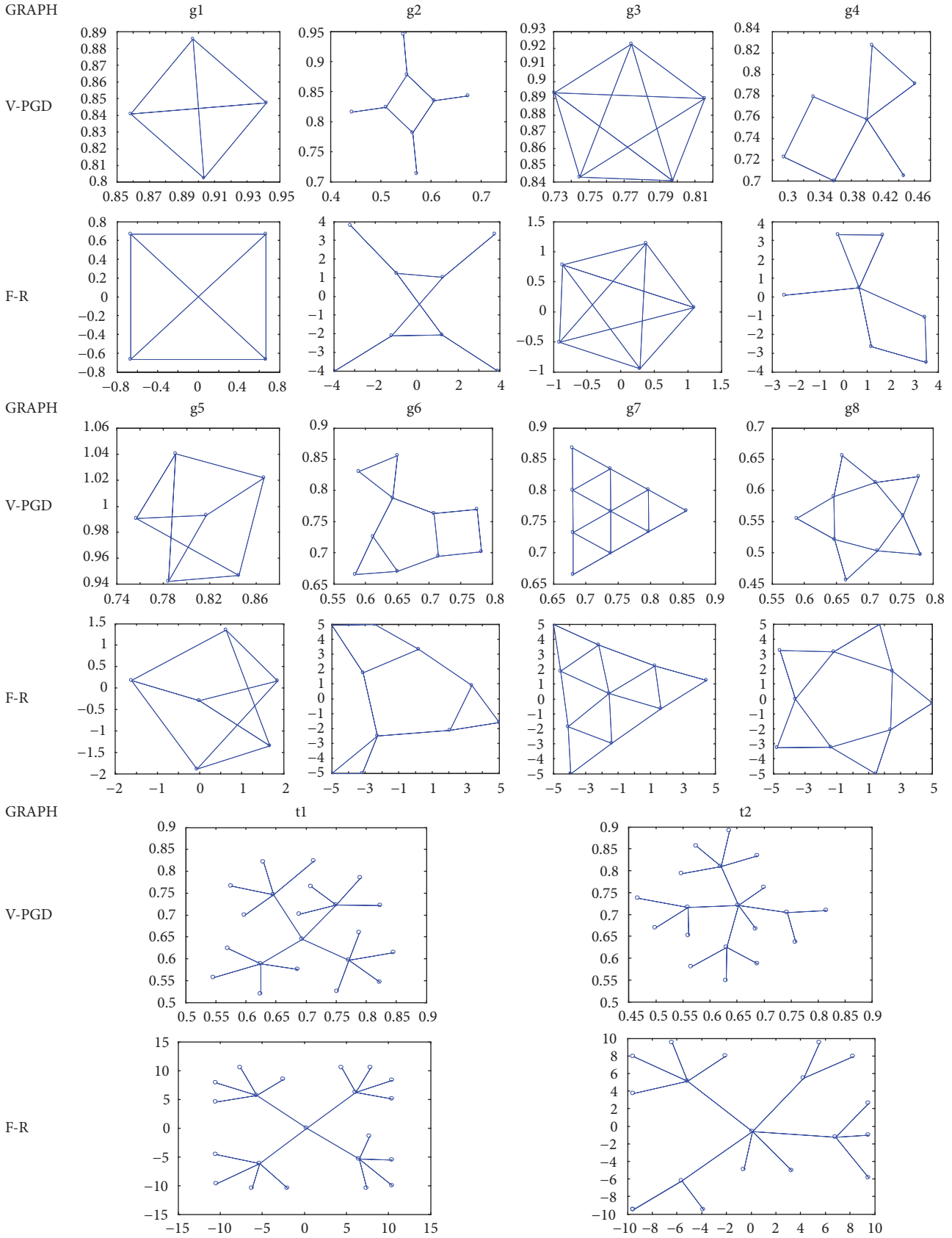

Figure 3: Drawings by V-PGD and by the F-R method on 10 graphs.

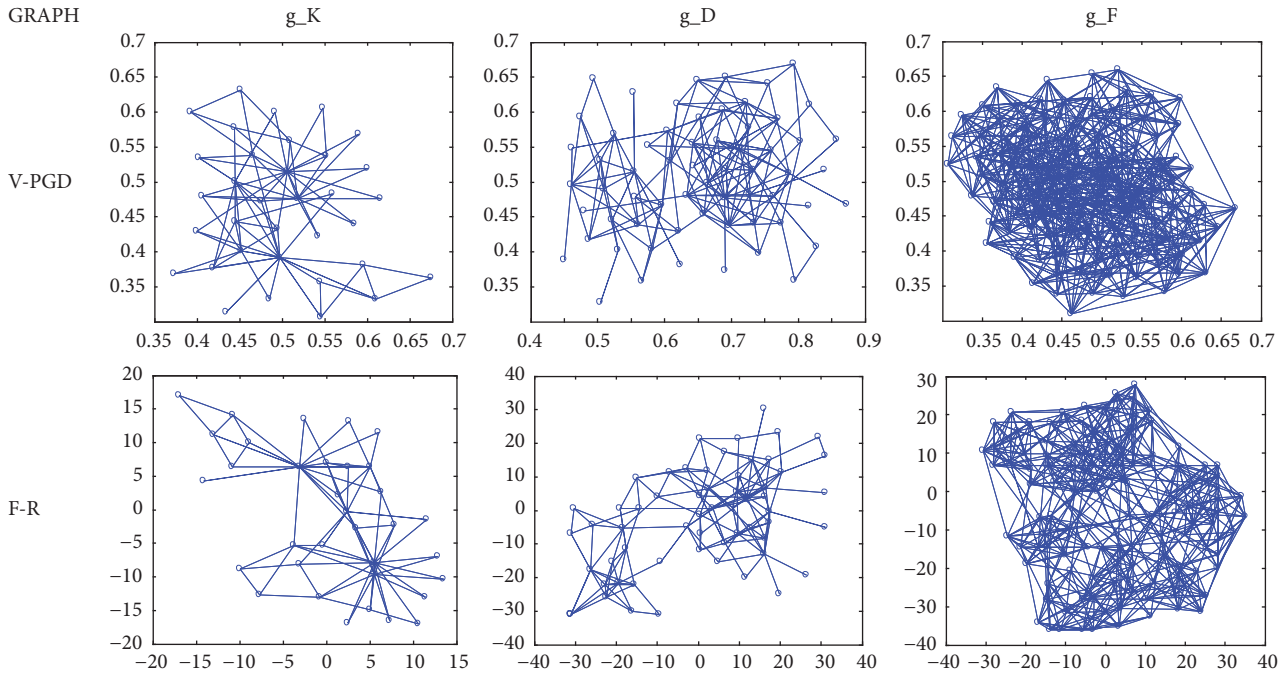

Figure 4: Drawings by V-PGD and by the F-R method on 3 real graphs.

Table 1: Running time (s) of the three methods on 10 graphs.

	PSO		F-R
	S-PGD	V-PGD	
g1	0.049	0.148	0.014
g2	0.119	0.150	0.016
g3	0.061	0.149	0.015
g4	0.097	0.151	0.016
g5	0.077	0.149	0.015
g6	0.173	0.153	0.016
g7	0.173	0.152	0.016
g8	0.172	0.153	0.016
t1	0.663	0.160	0.023
t2	0.549	0.158	0.026

without this restriction, either S-PGD or V-PGD obtained very good results for these graphs.

5.2. Running Time. The running times of S-PGD, V-PGD, and the F-R method on the 10 small graphs in Figure 3 are shown in Table 1. The parameters of these three methods are the same as those used when the layouts in Figure 3 were drawn. From the results in Table 1, the F-R method takes the shortest running time among the three methods. The running time of V-PGD is indeed an order of magnitude higher than that of the F-R method. Although both V-PGD and the F-R method produce quality graph drawings, some graphs produced by V-PGD are more aesthetic. Such results at least show that the proposed methods are feasible. Of course, many improvements need to be made. Between the two PSO procedures, the running time of S-PGD is shorter from g1 to g5; the running time of V-PGD is shorter from g6 to t2. The running time of V-PGD is between 0.14 s and 0.16 s. These results show that the performance of S-PGD is better than that of V-PGD on small graphs with $n < 6$. In fact, the data transfers between the CPU and the GPU can produce large overhead, especially for small-scale graphs. This is inevitable in the current computing environment of the hardware and the software. One way to improve performance is to reduce the number of the data transfers in the program.

Three groups of graphs with similar structures but different numbers of vertices and edges are used in the following to analyze the relationships among the number of vertices, the number of particles, and the running time. Group 1 consists of seven randomly generated graphs with $n = 6$ to $n = 18$. Group 2 consists of 10 randomly generated graphs with $n = 10$ to $n = 100$. Group 3 consists of 6 randomly generated graphs with $n = 1000$ to $n = 6000$.

The layouts of the graphs in Group 1 are shown in Figure 6. The running times in seconds of S-PGD and V-PGD on the seven graphs in Group 1 are shown in Figure 7 with $m = 30$ and Itr $= 1500$. At $n = 6$, S-PGD is faster. As n increases, V-PGD becomes faster.

The running times of S-PGD and V-PGD on g14 with $n = 12$ are shown in Figure 8 with Itr $= 600$ and varying m. Figure 8 shows that V-PGD is faster. The running time of S-PGD increases linearly but that of V-PGD stays almost the same as m increases. Such results show that the performance of V-PGD is better than that of S-PGD for large-scale graphs.

The layouts of the 10 randomly generated graphs in Group 2 are displayed in Figure 9. The running times of S-PGD and V-PGD on the 10 randomly generated graphs in Group 2 with $m = 50$ and Itr $= 6000$ are compared in Figure 10. It is

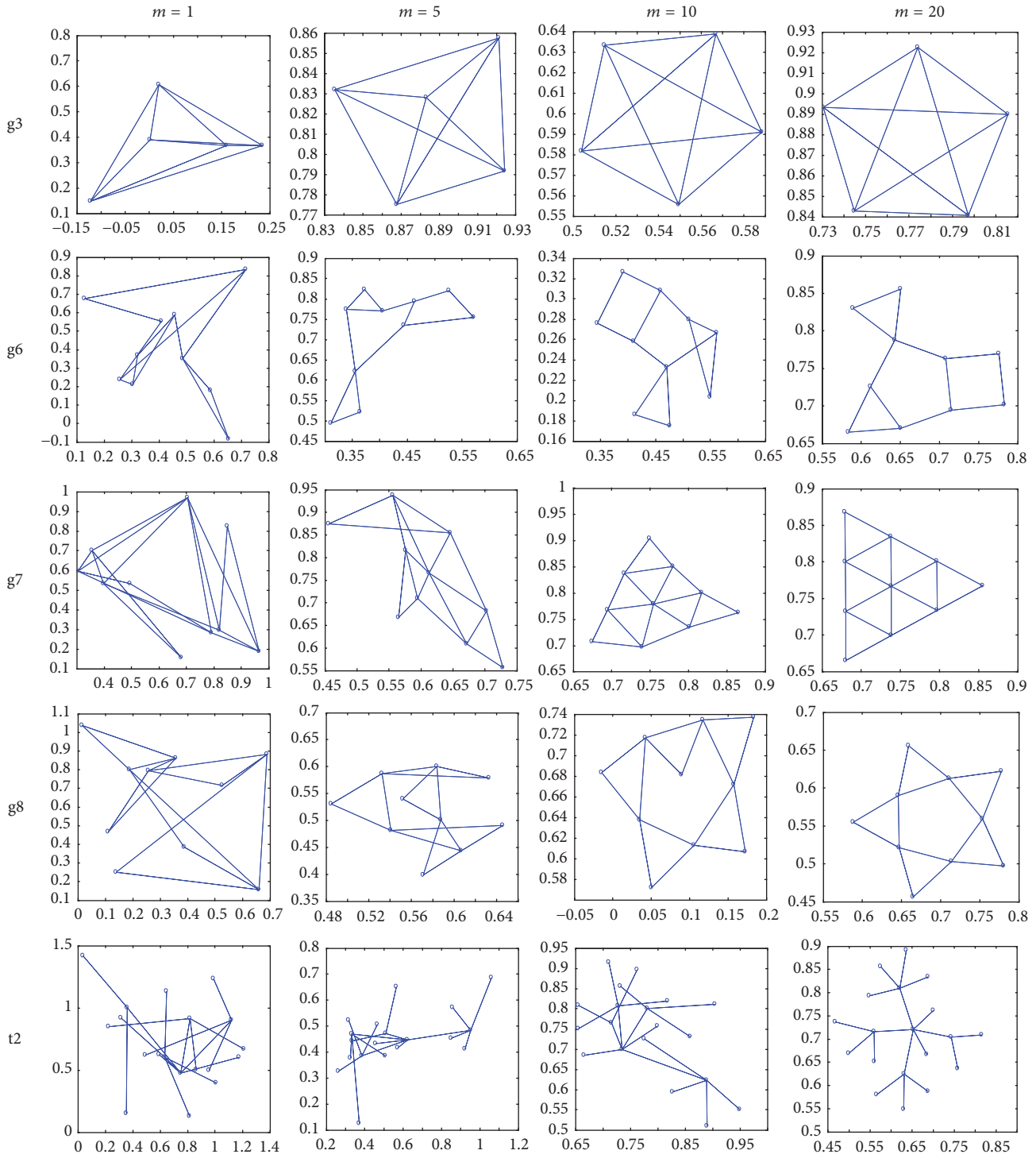

Figure 5: Evolutionary drawings with varying number of particles by V-PGD.

evident that S-PGD takes much more running time than V-PGD does, with an exception of graph rg1, when the same number of particles is used running the same number of iterations. When the numbers of vertices and edges increase, the advantage of V-PGD becomes more and more evident.

The running times of S-PGD and V-PGD on graph rg6 with Itr = 6000 and varying m are shown in Figure 11. As m increases, V-PGD becomes faster than S-PGD. The running time of S-PGD increases linearly and that of V-PGD only increases slightly. Such results show that the performance of V-PGD is better than that of S-PGD for large-scale graphs.

Figures 10 and 11 show that the running time of S-PGD increases quickly as the numbers of particles and vertices increase, but that of V-PGD increases only slightly. Such

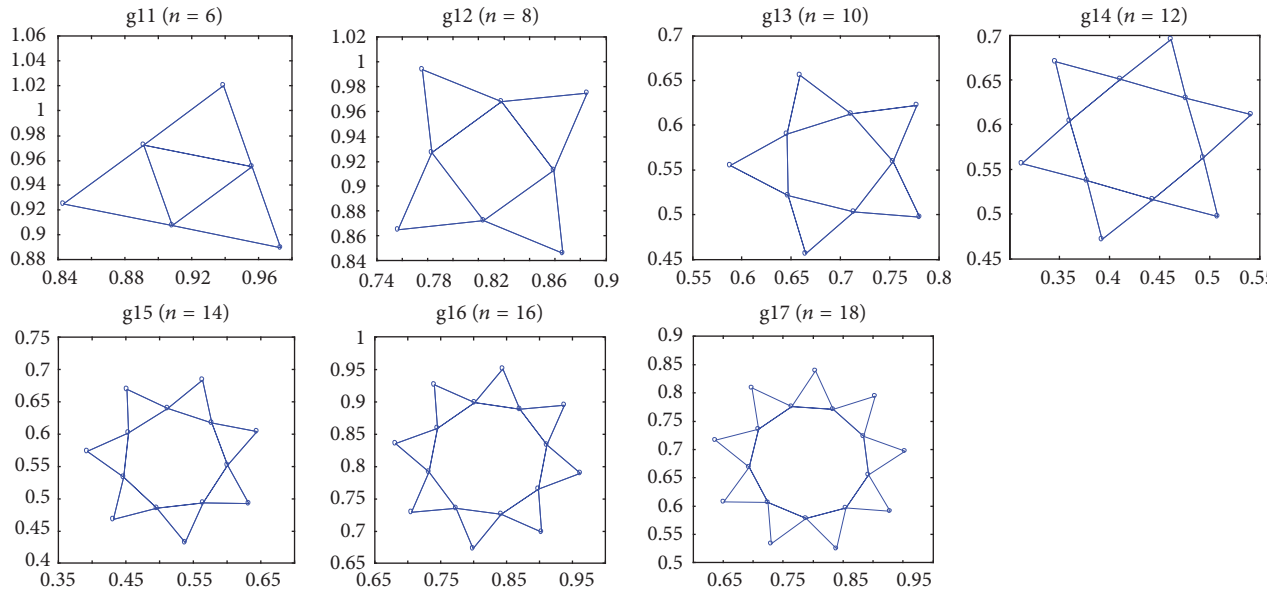

FIGURE 6: Layouts of seven graphs (g11–g17) in Group 1 with different numbers of vertices.

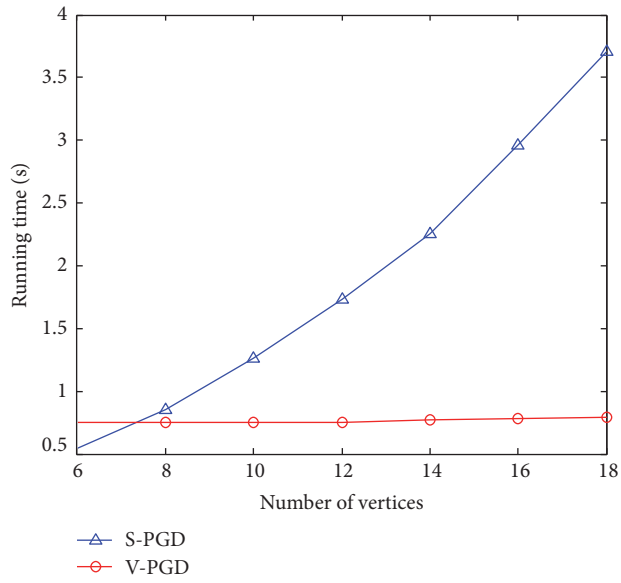

FIGURE 7: Running times of g11–g17 in Group 1 (m = 30 and Itr = 1500).

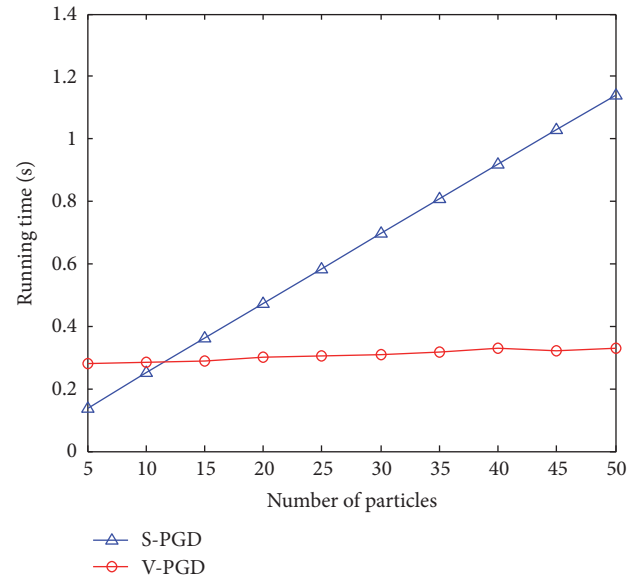

FIGURE 8: Running times of g14 (n = 30 and Itr = 600).

results can verify the time complexity of the two procedures. For each iteration, the time complexity of S-PGD is $O(mn^2)$ which is proportional to the number of particles, but that of V-PGD is $O(n)$ which is not related to the number of particles due to parallel computation. Therefore, the varying number of particles has no influence on the running time in the parallel PSO procedure. V-PGD might fail to work for very large graphs with more than 1024 vertices under the structure proposed in this study. Hence, it is a good approach for graphs with no more than 1024 vertices.

5.3. Convergence Analysis. Convergence analysis is an important aspect in evaluating the performance of PSO procedures. Figures 12–14 show the changes in the fitness values in the searching process of the PSO procedures when the iterations increase on graphs g7, g14, and g_K. These results show that the two PSO procedures can converge after a number of iterations. Different graphs need different number of iterations depending on many factors, such as the size of the graph, the number of particles, and the parameters used in the procedures. The number of iterations is only one of these factors affecting convergence.

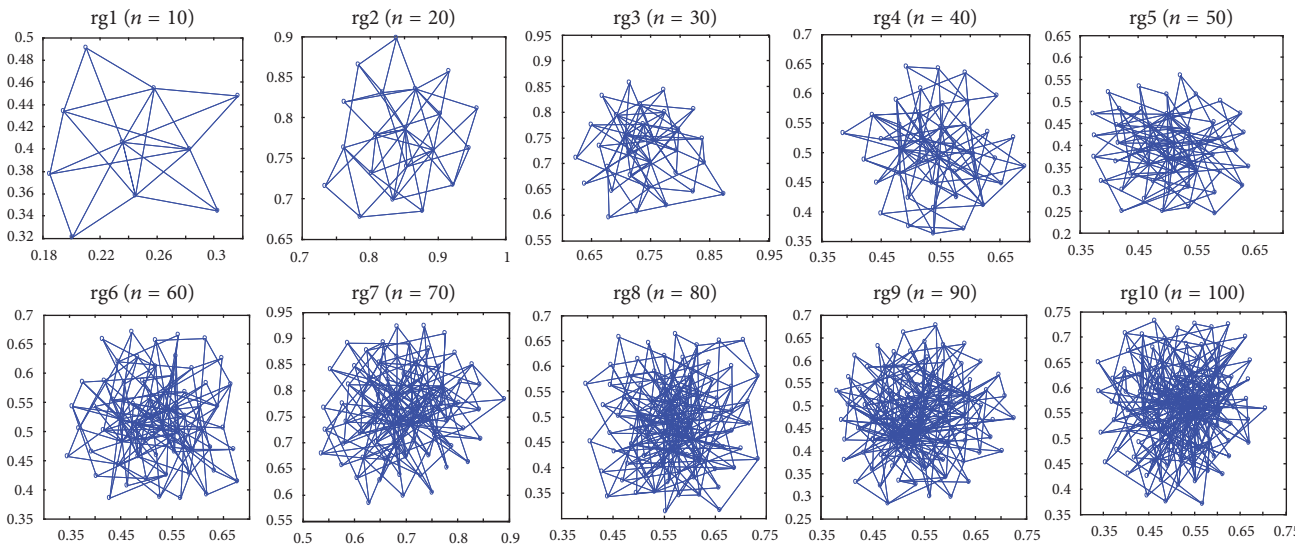

Figure 9: Drawings of 10 random graphs in Group 2.

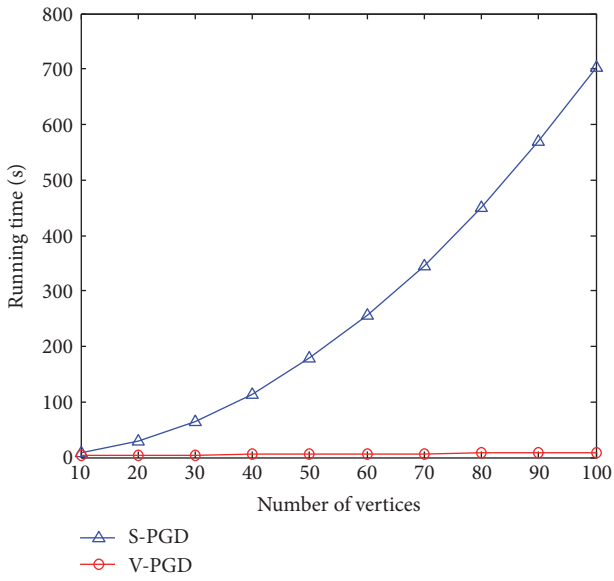

Figure 10: Running time of rg1–rg10 (m = 50 and Itr = 6000).

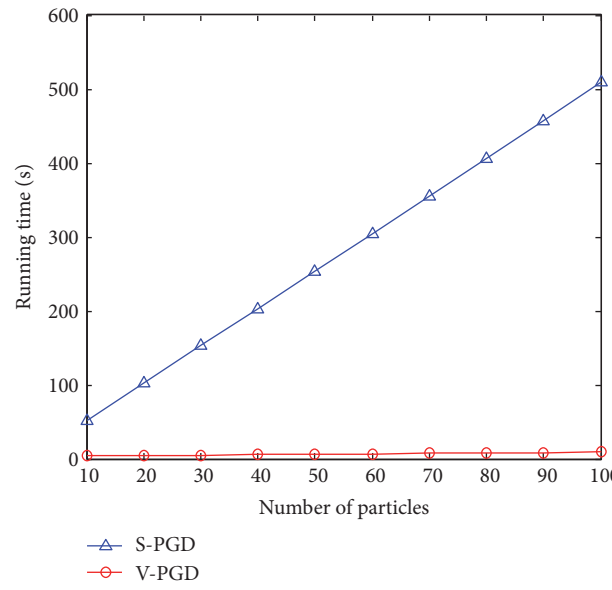

Figure 11: Running time of rg6 (n= 60 and Itr = 6000).

The relationships between the fitness values and the number of particles using V-PGD for graphs g7, g14, and g_K are shown in Figures 15–17. In Figures 15–17, $|p|$ is the number of the particles. That is to say, $|p| = m$. As shown in these figures, the number of particles does not noticeably influence convergence. Hence, using a large number of particles is not always necessary. It can be seen from the evolutionary drawings of graphs g3 and g7 in Figure 5 that there are no obvious improvements in the drawings when the number of particles increased from 10 to 20. Therefore, simply increasing the number of particles may not be able to improve the effectiveness of the procedures for some graphs. However, exceptions may still exist, especially in some real graphs.

6. Conclusions and Future Work

Two PSO procedures, S-PGD and V-PGD, are developed for the graph drawing problem. As a population-based meta-heuristic, PSO has the advantages of robustness, effectiveness, and simplicity and is suited to optimizing graph drawing. The graph drawing problem is transformed to a problem of positioning the vertices by using the force-directed method. One particle corresponds to one layout of the graph in the two procedures. The two procedures are both effective although with different implementation and time complexities. Experiments on different graphs are conducted to compare the performances of the two procedures and of the F-R method

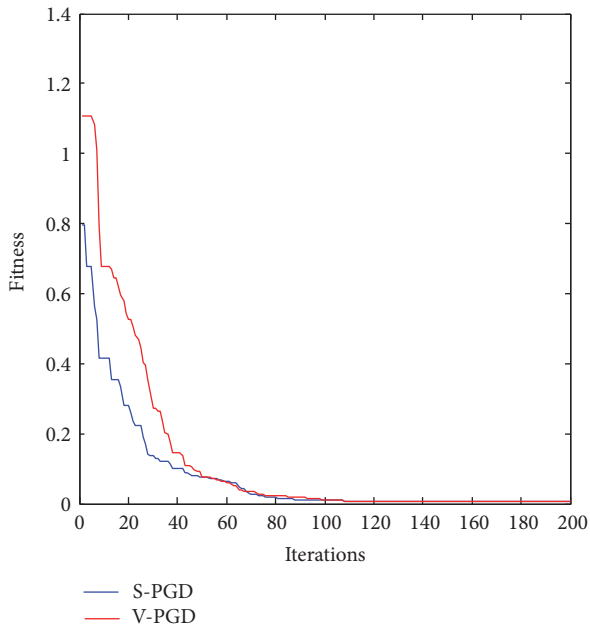

Figure 12: Convergence of g7 ($m = 20$).

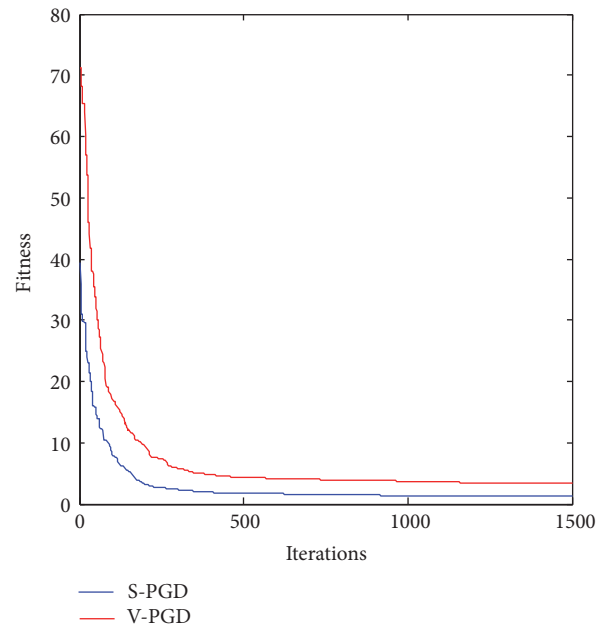

Figure 14: Convergence of g_K ($m = 50$).

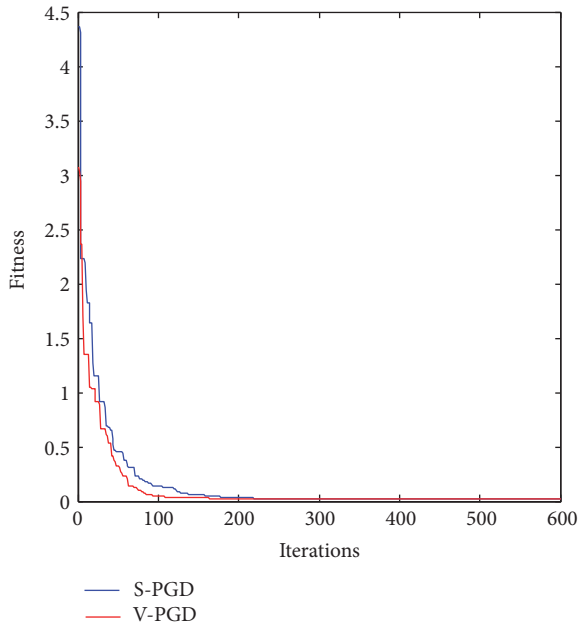

Figure 13: Convergence of g14 ($m = 40$).

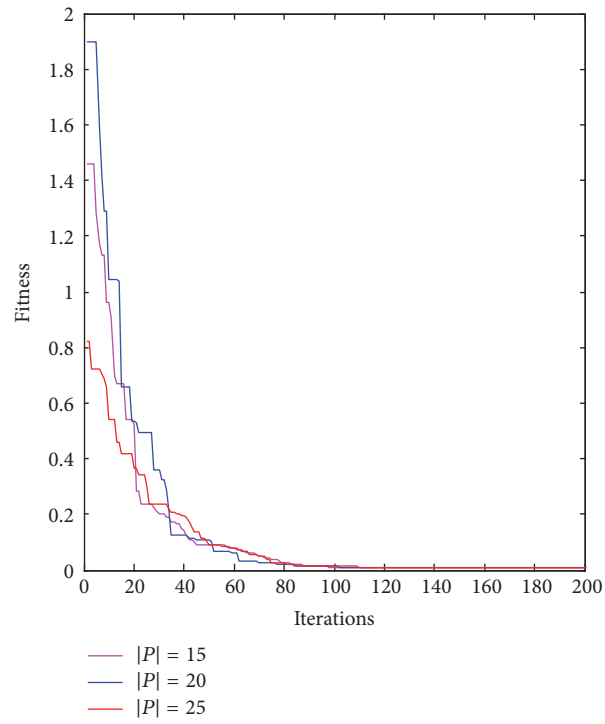

Figure 15: Fitness of g7 (V-PGD).

in terms of effectiveness, running time, and convergence. The following conclusions can be drawn:

(i) Compared to the F-R method, the two PSO procedures are effective and can obtain better results on some graphs.

(ii) The two PSO procedures converge to the best solution with the evolution of particles.

(iii) S-PGD uses less running time on small graphs but V-PGD uses less running time on large graphs.

(iv) The running time of S-PGD increases linearly but that of V-PGD does not change much as the number of particles increases. The larger the number of particles is, the relatively faster V-PGD runs as compared to S-PGD.

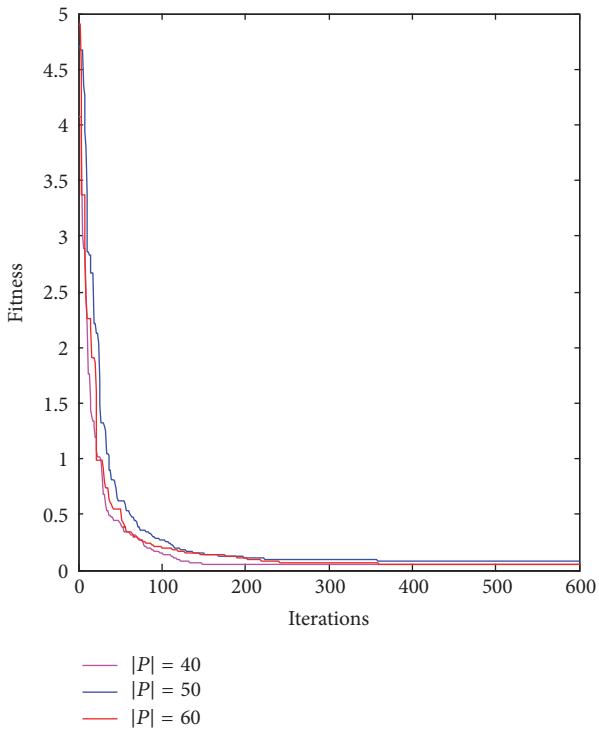

Figure 16: Fitness of g14 (V-PGD).

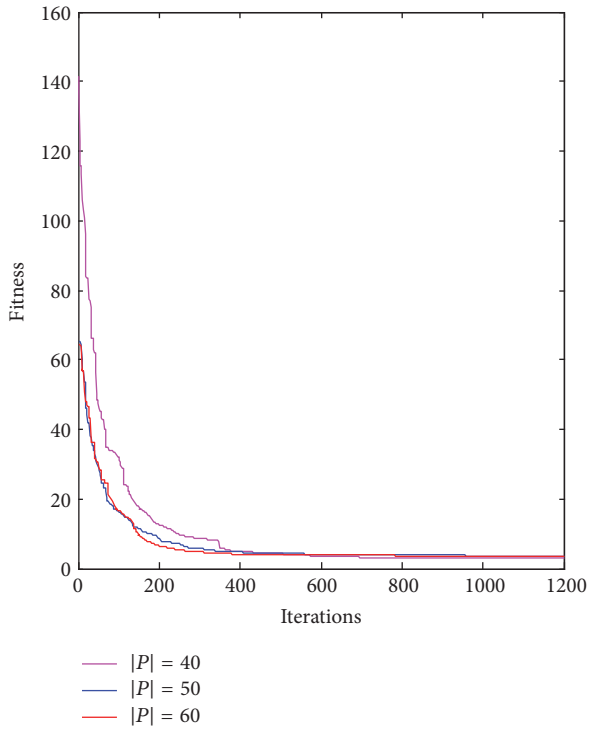

Figure 17: Fitness of g_K (V-PGD).

(v) The running times of the two PSO procedures both increase linearly as the number of vertices increases. However, the running time of S-PGD increases much faster than that of V-PGD.

(vi) As the number of particles increases, the effectiveness of the procedures improves. However, the final solution does not improve further after the number of particles reaches a certain value.

Future works in this area are outlined as follows:

(i) Improving the performance of V-PGD and apply it to large graphs by exploiting the parallel partition methods of the graphs

(ii) Comparing the performance of V-PGD with other parallel architectures such as multicore CPU, MPI, and OpenMP

(iii) Implementing the parallel PSO procedure for other application domains

(iv) Developing parallel procedures for graph drawing using other metaheuristics, such as genetic algorithms and ant colony optimization

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this manuscript.

Acknowledgments

This research project is partly supported by the National Natural Science Foundation of China (Grants nos. 61472231 and 61502283).

References

[1] E. Kruja, J. Marks, A. Blair, and R. Waters, "A short note on the history of graph drawing," in *Proceedings of the 9th Intl. Symp. Graph Drawing (GD '01)*, vol. 2265, pp. 272–286, Springer-Verlag, London, UK.

[2] W. T. Tutte, "How to draw a graph," *Proceedings of the London Mathematical Society. Third Series*, vol. 13, pp. 743–767, 1963.

[3] K. Sugiyama, S. Tagawa, and M. Toda, "Methods for visual understanding of hierarchical system structures," *Institute of Electrical and Electronic Engineers. Transactions on Systems, Man, and Cybernetics*, vol. 11, no. 2, pp. 109–125, 1981.

[4] P. Eades, "A heuristic for graph drawing," *Congressus Numerantium*, vol. 42, pp. 149–160, 1984.

[5] M. Chimani and C. Gutwenger, "Algorithms for the hypergraph and the minor crossing number problems," *Journal of Graph Algorithms and Applications*, vol. 19, no. 1, pp. 191–222, 2015.

[6] M. A. Bekos, M. Kaufmann, S. G. Kobourov, and A. Symvonis, "Smooth orthogonal layouts," *Journal of Graph Algorithms and Applications*, vol. 17, no. 5, pp. 575–595, 2013.

[7] T. Kamada and S. Kawai, "An algorithm for drawing general undirected graphs," *Information Processing Letters*, vol. 31, no. 1, pp. 7–15, 1989.

[8] T. M. J. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software—Practice and Experience*, vol. 21, no. 11, pp. 1129–1164, 1991.

[9] R. C. Eberhart and J. Kennedy, "New optimizer using particle swarm theory," in *Proceedings of the Proc. 6th Intl. Symp. on Micro Machine and Human Science*, pp. 39–43, Nagoya, Japan, 1995.

[10] R. C. Eberhart, J. Kennedy, and Y. Shi, *Swarm Intelligence*, Morgan Kaufmann Publishers, 2001.

[11] J. Park and K.-Y. Kim, "Instance variant nearest neighbor using particle swarm optimization for function approximation," *Applied Soft Computing Journal*, vol. 40, pp. 331–341, 2016.

[12] F. Valdez, P. Melin, and O. Castillo, "Modular Neural Networks architecture optimization with a new nature inspired method using a fuzzy combination of Particle Swarm Optimization and Genetic Algorithms," *Information Sciences*, vol. 270, pp. 143–153, 2014.

[13] M. R. Bonyadi, Z. Michalewicz, and X. Li, "An analysis of the velocity updating rule of the particle swarm optimization algorithm," *Journal of Heuristics*, vol. 20, no. 4, pp. 417–452, 2014.

[14] D. Chen, J. Chen, H. Jiang, F. Zou, and T. Liu, "An improved PSO algorithm based on particle exploration for function optimization and the modeling of chaotic systems," *Soft Computing*, vol. 19, no. 11, pp. 3071–3081, 2014.

[15] C. W. Cleghorn and A. P. Engelbrecht, "Particle swarm variants: standardized convergence analysis," *Swarm Intelligence*, vol. 9, no. 2-3, pp. 177–203, 2015.

[16] N. B. Yahia, N. Bellamine, and H. B. Ghésala, "Combined use of community detection and particle swarm optimization to support decision making," *Journal of Computing*, vol. 4, no. 5, pp. 157–163, 2012.

[17] L. Mussi, F. Daolio, and S. Cagnoni, "Evaluation of parallel particle swarm optimization algorithms within the CUDA™ architecture," *Information Sciences*, vol. 181, no. 20, pp. 4642–4657, 2011.

[18] L. De P. Veronese and R. A. Krohling, "Swarm's flight: Accelerating the particles using C-CUDA," in *Proceedings of the 2009 IEEE Congress on Evolutionary Computation, CEC 2009*, pp. 3264–3270, nor, May 2009.

[19] W. Wang, Y. Hong, and T. Kou, "Performance gains in parallel particle swarm optimization via NVIDIA GPU," in *Proceedings of the Workshop on Computational Mathematics and Mechanics*, 2009.

[20] Y. Zhou and Y. Tan, "GPU-based parallel particle swarm optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '09)*, pp. 1493–1500, May 2009.

[21] NVIDIA, CUDA programming guide v. 8.0, NVIDIA Corporation, https://docs.nvidia.com/cuda/cuda-c-programming-guide/, 2016.

[22] B.-I. Koh, A. D. George, R. T. Haftka, and B. J. Fregly, "Parallel asynchronous particle swarm optimization," *International Journal for Numerical Methods in Engineering*, vol. 67, no. 4, pp. 578–595, 2006.

[23] J. F. Chang, S. C. Chu, J. F. Roddick, and J. S. Pan, "A parallel particle swarm optimization algorithm with communication strategies," *Journal of Information Science and Engineering*, vol. 21, no. 4, pp. 809–818, 2005.

[24] M. Waintraub, R. Schirru, and C. M. N. A. Pereira, "Multiprocessor modeling of parallel Particle Swarm Optimization applied to nuclear engineering problems," *Progress in Nuclear Energy*, vol. 51, no. 6, pp. 680–688, 2009.

[25] Y. Zhang, D. Gallipoli, and C. Augarde, "Parallel Hybrid Particle Swarm Optimization and Applications in Geotechnical Engineering," in *Proceedings of the 4th Intl. Symp. Advances in Computation and Intelligence (ISICA'09)*, vol. 5821, pp. 466–475, Springer Berlin Heidelberg, Berlin, Germany.

[26] J.-M. Li, X.-J. Wang, R.-S. He, and Z.-X. Chi, "An efficient fine-grained parallel genetic algorithm based on GPU-accelerated," in *Proceedings of the 2007 IFIP International Conference on Network and Parallel Computing Workshops, NPC 2007*, pp. 855–862, chn, September 2007.

[27] H. Prasain, G. K. Jha, P. Thulasiraman, and R. Thulasiram, *A Parallel Particle Swarm Optimization Algorithm for Option Pricing. Doctoral dissertation*, The University of Manitoba Winnipeg, Manitoba, Canada, 2010.

[28] S. Solomon, P. Thulasiraman, and R. K. Thulasiram, "Collaborative multi-swarm PSO for task matching using graphics processing units," in *Proceedings of the 13th Annual Genetic and Evolutionary Computation Conference (GECCO '11)*, pp. 1563–1570, July 2011.

[29] V. Roberge and M. Tarbouchi, "Parallel particle swarm optimization on graphical processing unit for pose estimation," *WSEAS Transactions on Computers*, vol. 11, no. 6, pp. 170–179, 2012.

[30] D. L. Souza, O. N. Teixeira, D. C. Monteiro, and R. C. L. de Oliveira, "A new cooperative evolutionary multi-swarm optimizer algorithm based on CUDA architecture applied to engineering optimization," in *Proceedings of the Combinations of Intelligent Methods and Applications*, pp. 95–115, Springer, Berlin, Germany, 2013.

[31] Y. Hu, "Efficient, high-quality force-directed graph drawing," *The Mathematica Journal*, vol. 10, no. 1, pp. 37–71, 2011.

[32] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, no. 12, pp. 7821–7826, 2002.

[33] D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson, "The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations: can geographic isolation explain this unique trait?" *Behavioral Ecology and Sociobiology*, vol. 54, no. 4, pp. 396–405, 2003.