International Journal of Applied Metaheuristic Computing
Volume 13 • Issue 1

A Parallel Particle Swarm Optimization for Community Detection in Large Attributed Graphs

Chaitanya Kanchibhotla, National Institute of Technology, Warangal, India*
Somayajulu D. V. L. N., National Institute of Technology, Warangal, India
Radha Krishna P., National Institute of Technology, Warangal, India

ABSTRACT

Social network analysis (SNA) is an active research domain that mainly deals with large social graphs and their properties. Community detection (CD) is one of the active research topics belonging to this domain. Social graphs in real-time are huge, complex, and require more computational resources to process. In this paper, the authors present a CPU-based hybrid parallelization architecture that combines both master-slave and island models. They use particle swarm optimization (PSO)-based clustering approach, which models community detection as an optimization problem and finds communities based on concepts of PSO. The proposed model is scalable, suitable for large datasets, and is tested on real-time social networking datasets with node attributes belonging to all three sizes (small, medium, and large). The model is tested on standard benchmark functions and evaluated on well-known evaluation strategies related to both community clusters and parallel systems to show its efficiency.

KEYWORDS

Community Detection, Parallel Computing, Particle Swarm Optimization, Social Networks

1. INTRODUCTION

Real-world optimization problems are usually complex in nature and are NP-hard. It can be due to large dimensions and multiple objectives. One of such real-world problems belongs to the domain of complex networks. The social networks domain belongs to complex networks in which several fundamental problems can be considered as optimization problems. Resolving complex optimization problems requires heavy CPU computation time and is challenging for a computer with standard hardware. Some of the reasons for the complexity in real-world social networks are large network topologies with numerous edges and nodes and the frequency of network growth.

The derivation of important structures from large social networks is always an interesting and noteworthy problem. In social network terminology, these structures are called *communities*. From the social network's perspective, a community is a dense substructure in which the

DOI: 10.4018/IJAMC.306913 *Corresponding Author nodes communicate more frequently with each other when compared to node communications outside of the group (Newman,2011). Analyzing social network groups allows us to expose the social features and predict important information such as discussion, entities involved in the discussion, etc. (Stanley,1994) In literature, two types of algorithms have been proposed by researchers. The first type considers community detection is considered as a clustering problem. In contrast, the second type considers community detection as an optimization problem and is solved by techniques such as optimizing modularity (Newman,2004)(Hespanha,2004)(Guimera,2011)(Blondel,2008). These works focus on graph structures that do not analyze the content of nodes in a graph. Analyzing node contents is important because most real-time social networks store helpful information in their nodes, not limited to user information and connectivity information.

In this work, we consider community detection is considered to be an optimization problem and solve it using a popular evolutionary, iterative algorithm called Particles Swarm Optimization (PSO). Most of the evolutionary algorithms are parallel by nature, and they contain individuals which improve during iterations. PSO has already proved to solve many real-world problems (Goldberg. 1989)On the other hand, applying optimization on large real-world networks can be computationally expensive due to the presence of large number of decision variables. Due to the iterative nature of evolutionary algorithms, applying optimization on large real-world networks can be time, resource-consuming and can increase the objective function evaluation time. Applying PSO to large-scale real-world optimization problems can degrade the algorithm's performance, making it unsuitable for large-scale optimizations R. Cheng and Y. Jin,(2015).To address the above problems, we present a hybrid parallel architecture and implement it on one of the previously introduced community detection algorithms based on PSO called PSO-based clustering (PSOC) (K. Chaitanya et al., 2018).

Parallelization is one option that can increase algorithm performance and be applied to complex problems to increase efficiency. The architectures on parallel constructs can quickly scale with the problem size, making them suitable for applying large complex problems. The other advantage is the concurrency, in which simultaneous executions can be done at the same time. With the availability of multi-core CPU and GPU in modern computers, parallel models can be developed on both. CPU-based parallelization strategies execute on multiple cores in one or more CPUs. GPU-based parallelization models accelerate CPUs for high compute performance. The processors in any parallel model cannot interact with one another and exchange information by themselves. They need a proper communication strategy that is usually based on network topologies such as star, ring, bus, and so on. In the paper, we design an architecture based on two popular parallelization strategies: master-slave and island models (refer to section 2.2). The proposed architecture is hybrid and has the following features:

1. The architecture is CPU-based and utilizes the capability of multi-cores in modern computers to speed up the processing.
2. The architecture can quickly scale to a required number of processes.
3. The architecture can be applied to heterogeneous systems.
4. The architecture supports message passing/ information exchange within the processors and between the processors.

The rest of the paper is organized as follows. The relevant concepts and works in each area are presented in section II. Section III introduces the design of the proposed architecture along with the evaluation criteria. Experimental results are presented in section IV, and section V concludes the paper.

2. CONCEPTS AND RELATED WORK

2.1 Particle Swarm Optimization

$$v_{ij}^{t+1} = \omega v_{ij}^{t} + c_1 r_{1j}^{t} * \left[pBest\, i^{t} - x_{ij}^{t} \right] + c_2 r_{2j}^{t} * \left[gBest\, i^{t} - x_{ij}^{t} \right] =$$
$$\omega v_{ij}^{t} + c_1 r_{1j}^{t} * \left[pBest\, i^{t} - x_{ij}^{t} \right] + c_2 r_{2j}^{t} * \left[gBest\, i^{t} - x_{ij}^{t} \right] \quad (1)$$

$$x_i = x_i + v_i \quad (2)$$

Particle swarm optimization (PSO) (J. Kennedy and R. C. Eberhart, 1995) is a class of evolutionary algorithms inspired by social animals such as birds, fishes, insects, etc. It is an optimization process in which particles move around the solution space for finding the optimal solution. Each particle posses velocity and position at every iteration by using its previous velocity, its best previous position, and the best position searched by all the particles so far. During every iteration, the velocity of each particle is updated using the following equation:

Where v_{ij}^{t+1} is the bounded velocity for a particle at time t+1. The bounded velocity is always between [v_{min}, v_{max}], ω is the inertia weight, c_1 and c_2 are cognitive and social coefficients, r_1 and r_2 are random numbers in the range of [0,1], $pBest$ is the particles best position and *gBest* is the global best position. Due to its simplicity and high search efficiency, PSO has widely used in many areas such as function optimization (Y. Shi, R.C. Eberhart,1999), artificial neural networks (J. Kennedy, R.C. Eberhart,2001), image processing (Wachowiak,2004). PSO was also applied for clustering problems in the literature. (Cui et al.,2005)applied the PSO algorithm for the document clustering problem using K-means, which performs a global search in the solution space. The procedure was used to classify over 800 documents effectively. (Van der Merwe and Engelbrecht,2003) also used K-means-based clustering to start the initial swarm and further refine clusters formed by K-means.

2.2 Parallelization Strategies

Parallel computing refers to the simultaneous computational resources used to solve a complex problem by breaking them into multiple chunks. These chunks can be executed on a single machine or group of machines. Single machine parallelization is called CPU based which refers to utilizing multi-cores and multiprocessors. Executing chunks on a group of computers which includes executing tasks on clusters, grids are referred to GPU-based parallel processing. CPU-based approaches can be developed using software packages such as Hadoop MapReduce, Matlab, R, etc. Some of the GPU-based strategies include CUDA, Open ACC (Lihua,2009)

From the development perspective, two strategies can be used to build parallel algorithms. 1) Data parallelism 2) Task parallelism. In data parallelism, the same task is executed simultaneously on multiple processors with a definitive data set. In task parallelism, different tasks are executed simultaneously across multiple processors belonging to the same data set. Any strategy out of these two can be selected based on the data set's size and the data set's complexity. Parallelization in PSO can be synchronous and asynchronous. In synchronous PSO, each particle performs function evaluation and waits for the other particles to complete. The velocity and position are updated at the end of each iteration. In asynchronous PSO, the particles move asynchronously in search space and do not wait for other particles to update the velocity and the position.

Communication or information exchange between the processors in the parallel computation is done based on network topologies. 4 popular communication strategies in the parallel computation are:

1. Master-Slave model
2. Coarse-grained model
3. Fine-grained model
4. Hybrid model

Master-slave is the most straightforward communication strategy based on star topology. The master processor is responsible for generational slave processors, and the function evaluations are executed in parallel in slave processes, as shown in figure 1a. In the Coarse groined model, the entire population is divided into 'n' independent subpopulations referred to as islands. In this model, some individuals are exchanged among islands at certain times, called the migration strategy. This model is dependent on parameters such as total population in islands, migration strategy, the number of individuals that are migrated, etc. The model is shown in figure 1b. In the fine-grained model, all the individuals are spatially structured into subpopulations called demes. The individuals share a common neighborhood which helps for the information exchange. This model is well suited for massively parallel operations.

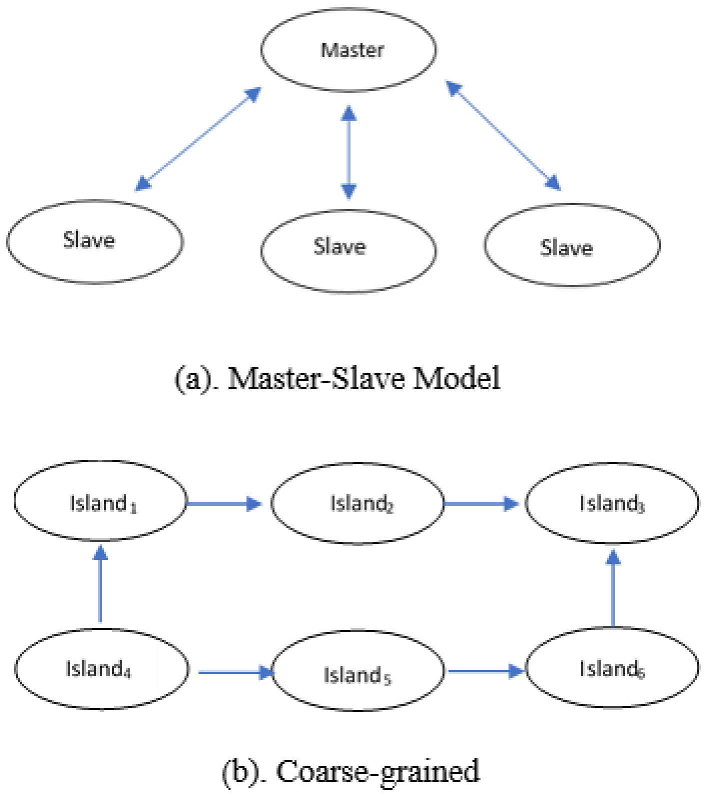
As PSO supports parallelism by nature, many works on improving its efficiency by combining it with several parallel techniques are proposed in the literature. Some of the works include the method introduced by (Gies and Yahya,2003). They have improved the algorithm by executing on the bewulf cluster. The method has increased the performance by 8 times when compared to the regular algorithm. (Cui and Weile,2005) proposed a model in which all the particles update the position synchronously by exchanging the gbest position. Its performance is further enhanced by (Venter and Sobieszczanski, 2006) by converting it to the asynchronous version. (Chusanapiputt et al.,2005) proposed Parallel Relative PSO (PRPSO) method based on master-slave methodology. In this method, all the slave nodes exchange the best position and fitness value with the master node, and the master node selects the best value and is exchanged with all the slave nodes to update the position of particles. (Lihua et al.,2009) proposed a parallelization strategy based on client architecture.

The information is exchanged using a communication strategy. (Singhal et al.,2009) combined both asynchronous PSO with MPI for information exchange. Tian. et al. presented parallel co-evaluation quantum behaved PSO, which breaks up the high dimensional problems into several low dimensional problems. Each low-dimensional problem is optimized individually by exchanging some information to improve the solution quality. (Li.et.al, 2005) proposed a PSO-based parallelization method for multi-objective PSO based on decomposition using both MPI and OpenMP based on the island model. The process proved to be 2 times faster on a normal machine. (Mostaghim et al.,2008) also proposed a PSO-based multi-objective optimization technique implemented on the master-slave model. The method is heterogeneous and combines the binary search method's search capability for better exploration with multi-objective optimization. A GPU-based Fine-grained parallel PSO(FGPSO) was proposed by (li. et al., 2007) for eliminating premature convergence. (Shenghui and Shuli, 2014) proposed a GPU thread-based optimization model called Cellular particle swarm optimization (CPSO), which reduces the search time. The method accelerates the convergence rate of the article by triggering many CPU threads to process each particle. The number of CPU threads created is equal to the number of particles in the search space. (Hussain et al.,2016) proposed standard PSO (SPSO) on a GPU, based on the CUDA architecture using video RAM, which uses threads in a wrap. The method proved 46 times faster than CPU serial implementation. (Wachowiak et al.,2017) proposed adaptive PSO (APSO) for high-dimensional problems based on parallelization using heterogeneous parallel computational hardware containing multi-core technologies containing GPU Intel Xeon Phi co-processors.

2.3 Diversity in PSO

Although PSO has been successful in many issues, premature convergence remains an open problem due to its fast convergence. The search process is stopped in premature convergence as the particles

Figure 1. Network topologies



cannot come out of local optima. In the case of multimodal functions, the chances of a premature problem are high. There are several works in the literature for addressing the problem. (Zang and Ding,2011) used the subswarms concept and used four sub-swarms to exchange information to increase diversity. A center learning strategy was used by (Niu et al., 2013)in which the particle direction is updated using historical particle experience in all sub-swarms. A dynamic multi-swarm particle swarm optimizer (DMS-PSO) is implemented (Liang et al.,2005). The swarm population is divided into many sub-swarms and grouped using a regrouping schedule and information sharing between sub-swarms. DMS-PSO also achieves great diversity in the population. A cooperative particle swarm optimizer was introduced by (Van den Bergh,2003) in which the cooperative learning behavior was used in multiple swarms by optimizing multiple solution vector components:

$$Np(t) = | x_p(t) - x_q(t) | \leq \alpha_i \tag{3}$$

In computational methods, the exchange of information among the subpopulation plays an important role. Groso proposed distributed genetic algorithm (Groso, 1985) that uses the GA framework to advance many subpopulations in parallel. The migration of populations from one population to another occurs regularly to increase the algorithm's search capability. In the approach presented by (Ray and Liew,2002), the particles interact with other particles having better fitness values. In this study, we presume that a particle's neighborhood changes dynamically during iterations. Each particle can communicate only with a subgroup of particles(called the neighborhood of particles) in each iteration. In this method, we find the neighborhood of particles as follows. Every particle *i* in the swarm posses an *area of influence* represented by α_i where $\alpha i > 0$ and can exchange information with other particles present in that area.

Here, *Np*(*t*) is particle neighborhood of particle *p* at time *t,* $x_p(t)$ is the current particle position, $x_q(t)$ is the position of particle *q,* which is the neighboring particle of p, and α_i is the area of influence. Below are our assumptions on particle neighborhood:

1. Particle neighborhood *Np*(*t*) changes in iterations.
2. Also, the area of influence α_i is the same for all the particles. i.e., if particle *p* is a neighbor of particle *q* at time t, then particle *q* is also a neighbor of particle *p* at time t.
3. Particles exchange information only in one direction, which is more suitable for decentralized, distributed, and asynchronous environments (Zhang. et al., 2011).

3. PARALLEL PSO BASED CLUSTERING

This section introduces a clustering method based on particle swarm optimization called particle swarm optimization-based clustering called PSOC proposed by us (K.Chaitanya, 2018). Then we extend the algorithm using a parallel architecture.

3.1 PSO Based Clustering (PSOC)

The basic idea of this method is to divide the swarm into *n* subswarms, and each subswarm executes the PSO algorithm in parallel by executing a fitness function. During the execution, the particles also exchange information among subswarms to maintain the population's diversity, eliminating premature convergence. This information exchanged is typically the clustering information, and the primary advantage is that the information can be reused with other particles in the sub swarm. The process is seized until it reaches the stopping criteria (usually the number of iterations or if all particles are clustered). This procedure is applied to large social graphs belonging to the social network's domain. The basic idea is to find communities out of the network based on node attributes information. Below are the important characteristics of the approach:

1. Nodes in the same community are connected.
2. A node can be present in more than one community as overlapping communities exist in real-time.
3. The attribute information of nodes belonging to the same community is likely to be similar.

During the implementation of the process, the particles exchange information through *shared memory*. The shared memory stores information related to the particle, such as it's assigned to cluster ID, attribute information of the nodes information between particles is exchanged when the distance between two particles is less than the influential area, which is calculated using equation (3). Asynchronous procedures update the cluster information and attribute information in the shared memory. The complete details about the method are present in (K.Chaitanya, 2018).

3.2 Similarity Index and Fitness Function

The similarity index between any two particles P_i and:

$$S_{ij} = \frac{1}{\mathrm{k}} \sum_{\mathrm{a}=1}^{\mathrm{k}} \mathrm{S}_{\mathrm{ija}} \tag{4}$$

P_j is indicated as S_{ij} where '*k*' represents the attribute set.The value of S_{ija} is 0 if a^th attributes of P_i and P_j are not matched, and the value is 1 if a^th attributes of P_i and P_j are matched. The value of S_{ij} lies between 0 and 1. If the node attribute values are numeric, values are converted into categorical variables using the label encoder method. The label encoder converts the non-numeric variables into numeric labels. Each category is assigned a unique numeric label starting from 0 to n-1 values per feature:

$$S_{ij} > \beta \tag{5}$$

Equation (5) shows the fitness function. In the equation, β denotes similarity constant (which is configurable).

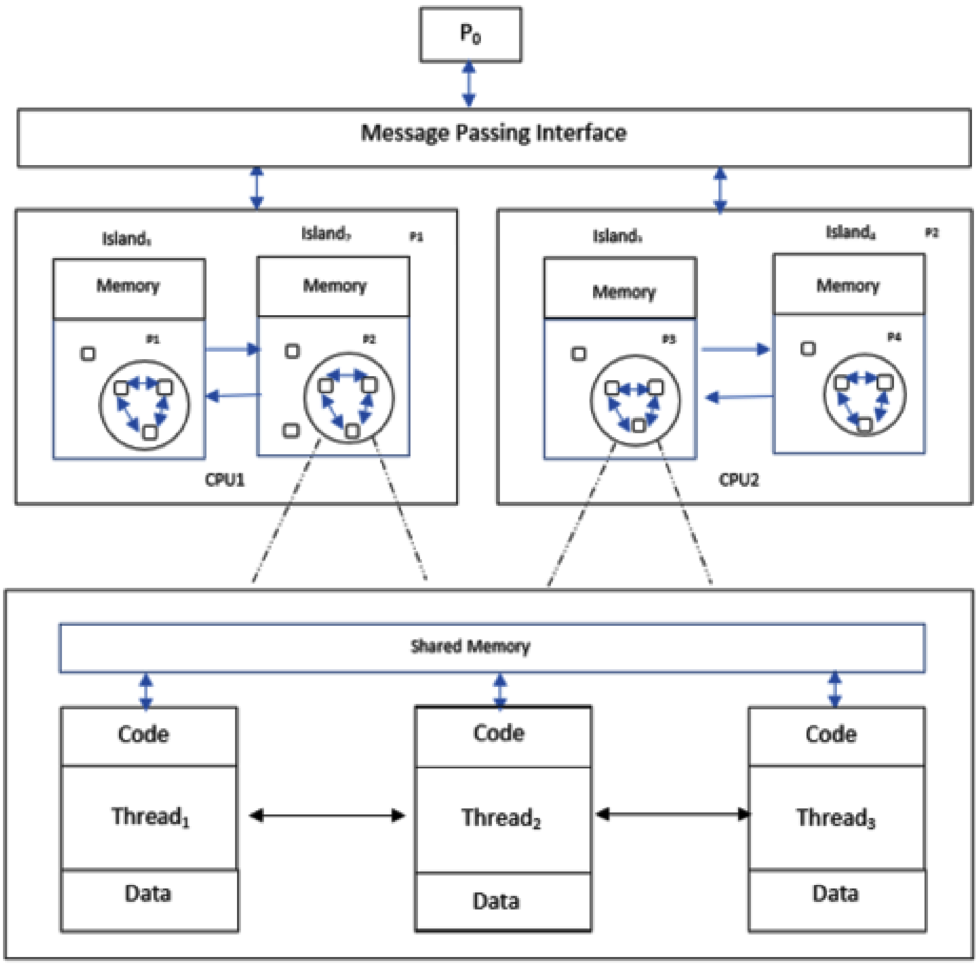
3.3 Parallel PSOC

In this subsection, we introduce a hybrid CPU-based system architecture and then execute the PSOC algorithm on it. We call the new approach a parallel PSOC(PPSOC). The algorithm is applied to a large social network dataset for community detection. The architecture is based on master-slave and island models and combines the benefits of both. In this model, we ignore the communication overhead between processors in the master-slave model and assume that all the processors are busy throughout the execution. The master-slave model is one of the simplest parallelization paradigms. This paradigm's main objective is to distribute the tasks among the processors that suit scaling the architecture to a large dataset. In a general scenario, these tasks are typically function evaluations on slave processors. The main functionality of the master processor is to gather information from all the slave processors and perform the optimization process. In the island model, the entire population is divided into *n* subpopulations which are called islands. Function evaluations take place independently on each island, and the particles are migrated from one island to another island based on the migration strategy as explained later. The basic architecture of PPSOC is shown in figure 2. The architecture consists of multiple processors in which one of the processors acts as a master processor, and the rest of the processes are slave processors.

In the figure, P_0 is the master processor, and P_1 and P_2 are slave processors. In each slave processor, islands are created by creating a new process containing threads that execute tasks. The procedure is based on the principles of PSO. In PPSOC, we initiate a new thread for every record in the data set. In PSO terminology, the thread corresponds to the particle. The number of particles in each island depends on the number of processors, processes in each processor. For example, if the model is executed on *n* CPUs and *m* processes and the number of input records is *p,* then the total number of particles created is *p/(m*n).*

All the threads in the process are assumed to have local storage(as indicated as "data" in figure 2) and share some resources such as memory, global variables, etc. In the thread data storage component, we store information such as particle id, position, attribute information, and objective function value. All the particles are initialized by storing the required information and are assigned random positions in the search space. All the custom logic, such as objective function evaluations, are executed in the "code" section of the thread. Threads communicate with each other by passing function evaluation information using variables in the global address space. After completing each task, the cluster information is stored in the global address space so that other tasks can reuse the same information, saving both computational time and cost. This information exchange is only between the particles under the influence area (calculated using equation 3). The cluster information and particle position are also shared with the master process with the help of the Message Passing Interface (MPI). The main functionalities of the master processor are:

Figure 2. PPSOC architecture



1. Collect information from slave processors.
2. Finalizing the information that should be exchanged with other processors.
3. Identifying the slave process for exchanging the information.
4. Assign/create cluster-id to the particles based on objective function evaluations.
5. Passing the required information to the slave process using MPI.
6. Trigger and implement the migration policy.

The main functionalities of the slave processor are:

1. Receive the information from the master process.
2. Execute custom logic using principles of PSO.
3. Objective function evaluation.
4. Update PSO parameters such as particle's velocity and position.

The Master processor also triggers a migration policy to maintain diversity in the islands. Doing this will not only maintain diversity but also eliminates premature convergence in the population. Out of the total particles in the island, only a subset of particles is selected for migration. The selection of particles for the migration depends on the objective function values. Since the main objective of PPSOC is to cluster the records, the method selects the particles having the least similarity values. Some of the main constraints that should be considered for the migration process are:

1. Basic criteria which trigger the migration of particles.
2. Migration rate, i.e., count of particles that are migrated.
3. Exact point to trigger the migration policy, i.e., when the migration is carried out.
4. The topology of migration, i.e., selecting the subpopulation for migration and the direction of migration.

The master process identifies the suitable particles for migration if the objective function value is less than Ω (constant). For the migration rate, we select all the particles which satisfy the migration criteria. The triggering point is the predefined iteration count. Since migration rate and triggering point cannot be generalized(please refer to section 4.1), we consider predefined constant values for both migration rate and Ω. As a migration topology, we migrate the particles only into the left and right islands. There is a chance of communication overhead if the particles are migrated to more islands. The migrated particle takes the random positions in the migrated islands. The particles exchange information using a point-to-point MPI communication strategy. In this strategy, the process *i* send its information to its immediate left and right processes, then the target processes receive the information. The procedure is executed until the termination criteria is met. The main steps in the process are explained in procedure 1.

The same procedure is explained by applying the proposed architecture to find communities on large social networks containing node attributes and the important steps below. Before the cluster procedure execution, parameters such as number of particles in each island, PSO parameters are initialized. The next step is to create corresponding objects for both master and slave processes on the CPUs based on the initial values. In the next step, the data set records are read sequentially and properties such as node id (which is a sequential number), attribute information are assigned to each particle in the island. As the particles have memory, they can store similarity information. During the execution of the procedure, the area of influence is calculated by the master processor using eq. (3). When the area of influence, i.e., the distance between the particles in subswarms, is less than the predefined constant, the similarity index eq. (4) and objective function eq. (5) are computed. A similarity string is calculated as a combination of $S_iS_{i+1}S_n$ and $X_jX_{j+1}X_{m,}$ where *n* is the length of similar attributes, S_i and S_{i+1} are similar attributes, X_j and X_{j+1} are node attributes values from the dataset. For example, the similarity string between two numerical node attributes [1,0,3,2,2] and [1,0,1,1,2] is “10XX2”. The Similarity string is stored in the particle’s memory. This same procedure is executed asynchronously in all the islands. At some predefined iteration, say 500, the similarity string and particle id are exchanged with the master node process. The master processor receives information from all the slave processors (islands) though MPI and assign cluster IDs to the particles based on a search string and store the cluster information. If the same search string is already present in the master process memory, then the particle IDs are added to the existing record; otherwise, a new entry is created. This process is executed until the stopping criteria is met.

Procedure 1. PPSOC Procedure on Social Networks

```
Input:
1) Dataset containing node attribute information
2) 2-Dimensional Search space
3) Number of processors =3, particles in each island $m$
4) PSO parameters ω, c1, c2
5) Iteration for migration, $k$
6) Migration threshold value, Ω
7) Similarity index $\alpha_i$=3, fitness constant $\beta$=0.5
8) Area of influence constant, $\mu$
9) Inter processor iteration step, $l$
Output:
Clustered Information
Begin:
                  %Step 1 Initialization phase%
1. Initialize master and slave processes based on total records
   from dataset in processors.
2. Create objects for master and slave process
3. Initialize particles in islands in slave process with size '$m$'
4. Intialize PSO particles in search space with parameter values.
               %Step 2 Island initilization phase%
5. For each record in dataset:
   a. Create particle '$p$' in island $I_i$ until island size < $m$
   b. Populate attribute information to $p$
                       %Step 3 Execution phase%
6. For each particle 'p' in each Moisland:
   a. Find particle neighborhood of 'p' by calculating dist (P,
      Pi) (where Pi is another particle in island)
   b. If dist (P, Pi) < $\mu$ then
      i.   Calculate the similarity index using eq (4)
      ii.  Calculate the fitness function using eq (5)
      iii. If fitness function is satisfied:
           1. Calculate similarity string and update particle
              memory with similarity string and particle id
           2. Update global variables in thread for information exchange
         End if
      End if
   c. If iteration count = $l$
      i.  Exchange similarity string with master processor.
      ii. Identify and trigger migration process
      End if
7. Assign cluster id in master process based on similarity string.
      End for each
   End for each
End
```

3.4 Evaluation Criteria

Here we discuss several evaluation metrics for the proposed parallel algorithm and metrics to detect clusters' quality.

3.4.1 Performance Metrics for Parallel Systems

It is essential to study the parallel algorithm's performance for determining the efficiency and benefits of parallelism. Below are the metrics that are used in the paper to measure the performance of the proposed procedure:

$$S(p) = \frac{T\left(1\right)}{T\left(p\right)}$$

1. **Speedup:** Speedup is an evaluation metric that measures performance. It is the ratio between parallel execution time and sequential execution. It is given by the below formula, where *T(1)* is the time taken for the algorithm to run on one processor and *T(p)* is the parallel runtime on *p* processors:

$$S(p) = \frac{T\left(1\right)}{p * T\left(p\right)}$$

2. **Efficiency:** Efficiency is based on *speedup* and is the ratio of speedup to the number of processors. It indicates the time for which the processor is fully utilized. It is given by the below formula, where '*p'* is the number of processor cores:

$$R(p) = \frac{N\left(p\right)}{N\left(1\right)}$$

3. **Redundancy:** Redundancy is the measure of the increase in computation that is required. It is the ratio of the number of operations performed in parallel execution by the sequential execution. It is given by the below formula, where N(p) is the number of operations performed by p processing units, and N (1) is the total number of operations on one processor.

3.4.2 Performance Metrics for Evaluating Cluster Quality

In social networks, assessing cluster efficiency is critical. Below are several evaluation criteria that are used in this paper for assessing the cluster quality:

- **Omega Index:** Omega Index (Steve, 2010) takes into account the number of value pairs precisely placed in 'n' clusters. Omega index ω_u(C1, C2) between two clusters C1, C2 is calculated as follows:

$$\omega(c_1, c_2) = \frac{\omega_u(c_1, c_2) - \omega_e(c_1, c_2)}{1 - \omega_e(c_1, c_2)}$$

$$\omega_u(c_1, c_2) = \frac{1}{M} \sum_{j=0} \left| t_j(c_i) \cap t_j(C_2) \right|$$

$$\omega_e(c_1, c_2) = \frac{1}{M} \sum_{j=0} \left| t_j(c_i) \cap t_j(C_2) \right|$$

Here ω_u(C1, C2) is the number of pairs that are present in the same community together in the same count of communities and t_j(C) is the set of pair of values present together in precisely *j* communities in cluster C. ω_e(C1, C2) is the expected value in the null model.

- **Silhouette Coefficient:** The silhouette coefficient measures the similarity for a value *i* by comparing it with all the present in its own cluster and compares it with values in other clusters:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

Here, a (i) is the average dissimilarity of value *i* compared with other values present in the same cluster; b(i) is the lowest average dissimilarity of value *i* in any other cluster in which it is not present:

$$T_{AB} \frac{c}{a + b - c}$$

- **Tanimoto Coefficient:** Tanimoto Coefficient (Segaran,2007) between two nodes, node 1 and node 2, is the similarity score calculated by considering both the common attributes and total attributes of nodes in a network. The value is 1 if the nodes are similar and is 0 if nodes are dissimilar. It is expressed by the below equation where 'a' is the attribute count of node 1, 'b' is the attribute count in node 2 and 'c' is the count of shared attributes between the two nodes.

4. RESULTS AND DISCUSSIONS

This section demonstrates the applicability of the method on both CEC benchmark functions and social network datasets. The default parameters for PSO are, Inertia weight (ω) is chosen as 0.729, and c1 and c2 as 1.49445. The results are executed on a computer running on i7 processor having 4 cores, 8 logical processors, and 16GB memory. Opensource python libraries inspyred, mpi4py, multiprocessing are used along with standard libraries such as numpy, math, and collections during the implementation of the program.

We first apply the proposed process on standard benchmark functions to analyze the efficiency and to estimate the time taken by the overall process. Table 1 shows the best, worst, median, average, and standard deviation and time taken on two segregated by each process. The values are obtained while executing the model on two slave processors having four processes each. The number of 20 particles in each island and the model is executed for 200 function evaluations. All the functions are single objective functions. It can be observed from the table that the method found optimal solutions for almost all the functions except the *Schwefel function,* which is highly multimodal. Function best values in that iteration are exchanged between the processors.

Below are the real-world social networks on which we apply our procedure:

1. Github Social networks (Rozemberczki, 2019).
2. Deezer Social networks (Rozemberczki, 2019) belonging to 3 countries, Romania, Hungary, and Croatia.
3. Wikipedia article networks (Rozemberczki, 2019).
4. Twitch (Rozemberczki, 2019).
5. Facebook Ego network (Leskovec and cravel, 2014).
6. Google Plus Ego network (Leskovec and cravel, 2014).
7. Pokec (Leskovec and cravel, 2014).
8. Facebook large page-page network (Leskovec and cravel, 2014).

Table 1. Results on benchmark functions

No.	Function	Worst	Best	Median	Average	Standard Deviation	Execution Time on P1(in secs)	Execution Time on P2(in secs)
1	Rastrigin	24.897	0.380	6.869	9.547	7.909	0.41	0.43
2	Ackley	5.330	0.029	1.037	1.454	1.226	0.42	0.43
3	Griwank	1.941	0.074	1.026	0.916	0.553	0.45	0.46
4	Rosenbrock	5453.517	0.150	11.521	834.965	1628.291	0.43	0.44
5	Schwefel	1130.77	218.566	580.359	617.310	230.442	0.42	0.45
6	Sphere	0.743	0.004	0.032	0.099	0.196	0.39	0.41
7	Beale	2.999	0.497	1.643	0.592	1.654	0.21	0.26
8	Booth	8.265	0.999	2.902	3.102	1.524	0.36	0.54
9	Goldstein	90.324	-0.803	45.213	15.146	26.245	0.58	0.41
10	Easom	3.141	-0.999	1.845	2.851	1.254	0.55	0.34
11	Matyas	1.000	0.004	0.999	1.255	1.705	0.54	0.82
12	Schaffer2	1.218	0.0008	0.639	0.254	1.266	0.21	0.25

Table 2 shows the properties of the dataset, such as the number of nodes, edges, density, transitivity, and node attributes. Density in a social network refers to the ratio of the number of connections a node has to the total number of possible connections. Transitivity is a degree of relationship that refers to the two edges in which if there is an edge between node *i* and node *j* and node *j* and node *k*, then there exists a node between *i* and *k*. All the experimental datasets are categorized into 3 categories, i.e., small, medium, and big. The datasets from (1-4),(6-7),(9) are medium-sized networks, datasets (5),(8) are small-sized networks, and (10-12) are large network datasets. The description of the datasets is as below:

- **Github:** The data set is related to large undirected social networks of GitHub Developers collected from the public API in June 2019. Nodes are developers of the git repository who have starred at least 10 repositories, and edges are relationships between them. The node attributes are location, repositories starred, employer, and author's e-mail address.
- **Deezer:** The dataset refers to a friendship network that is collected from music streaming service Deezer 2017. The data relates to 3 European countries Romania, Croatia, and Hungary. Nodes represent the users, and edges are mutual relationships between them. The node attributes are the genre preference of the users. The total number of genres is 84.
- **Wikipedia:** The data is collected from English Wikipedia. The dataset relates to the 3 page -page networks related to chameleons, crocodiles, and squirrels. Nodes represent articles, and edges are mutual links between them. The node attributes are the features of articles containing nouns that appeared from Wikipedia articles' text.
- **Twitch:** The dataset represents the networks of gamers who stream in multiple languages. Nodes are the users, and the edges are friendships between them. The node features are the names of games played, liked, location, and streaming habits. The data is collected in May 2018.
- **Facebook Ego network:** This data set is collected using the Facebook app by survey participants. The dataset describes the User profiles of their circles and ego networks. The nodes represent the Facebook users, and the edges represent the connection between the user and the other users with whom the user is directly connected.

Table 2. Datasets used

S.No	Dataset	Nodes	Edges	Density	Trans itivity	Node Attributes
1	Github	37,700	289,003	0.001	0.013	15
2	Deezer **(RO)**	41,773	125,826	0.0001	0.075	11
3	Deezer **(HR)**	54,573	498,202	0.0004	0.114	9
4	Deezer **(HU)**	47,538	222,887	0.0002	0.092	42
5	Wikipedia **(Chameleon)**	2,277	31,421	0.012	0.314	36
6	Wikipedia (**Crocodile**)	11,631	170,918	0.003	0.026	70
7	Wikipedia **(Squirrel)**	5,201	198,493	0.015	0.348	28
8	Twitch (England)	7,126	35,324	0.002	0.042	25
9	Facebook Ego network	4,039	88,234	0.0002	0.1	223
10	Google Plus Ego network	107,614	13,673,453	0.0011	0.12	1318
11	Pokec	1,632,803	30,622,564	1.14	0.085	93
12	Facebook large page-page network	22,470	171,002	0.001	0.232	30

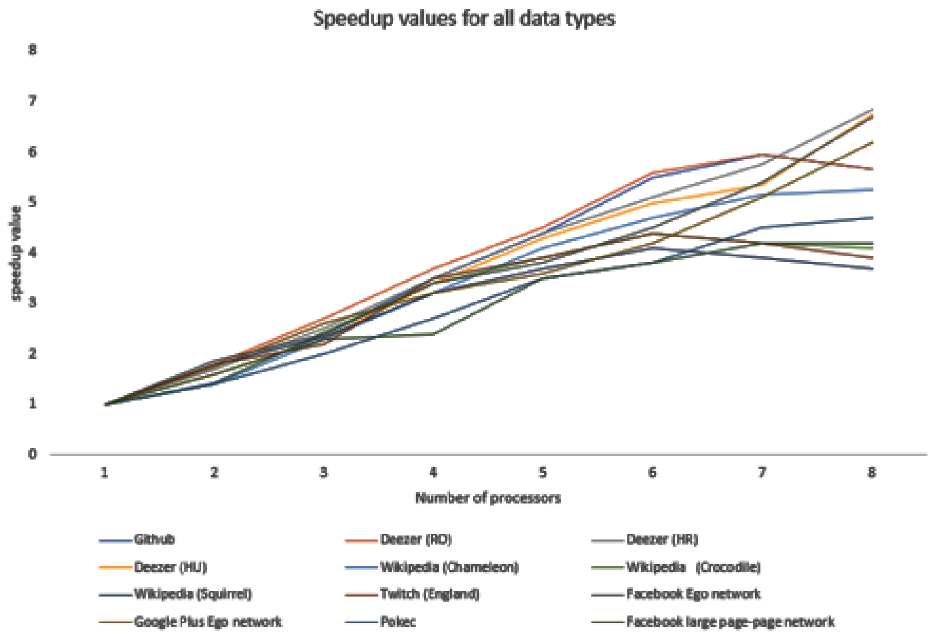
- **Google Plus Ego network:** This data set is collected from Google plus users who shared their circle data using the share circle feature. The data set consists of information related to user circles, user profile information, and ego networks.
- **Pokec:** Pokec is the most popular online social network that is widely used in Slovakia. The data set consists of information related to users and the connection between the users. Edges represent the connections. Nodes represent users having attributes such as user profile data such as age, gender, education, etc.
- **Facebook large page-page network:** This data set consists of page-page graph information from Facebook sites. The nodes in the graph are Facebook pages, and the edges are links between the pages. The node attributes information is extracted from the site descriptions, which owners fill during registration. The data is collected from the Facebook graph API. The data is related to four categories, namely politicians, government organizations, television shows, and companies.

Table 3 shows the average execution time for all the CPUs for all datasets. PPSOC procedure is executed for all the datasets for 10 iterations. The execution times are captured for all the available CPUs starting from 1 to 8. The highest execution time is observed for the pokec network on all the processors due to its large network size. The execution time is significantly reduced as the number of CPUs increases, which can also be due to reusing the existing cluster information from the master processor and the hardware capability. Figure 3 shows the speedup values for all the datasets measured on all the processors. The speedup values for all the datasets started increasing at a steady rate as the number of processors is increased. This behavior is observed until 6 processors for all the datasets.

Table 3. Execution time for all the datasets for all CPUs

Dataset	Execution Time for PPSOC (in seconds)							
	(1 CPU)	(2 CPU)	(3 CPU)	(4 CPU)	(5 CPU)	(6 CPU)	(7 CPU)	(8 CPU)
Github	244	170	101	69.7	55.4	44.3	41	43
Deezer (RO)	501	278	185.55	135.4	111.3	89.4	84.2	88.3
Deezer (HR)	685	428	297	285	183	163	140	125
Deezer (HU)	631	394.3	274.3	185.5	146.7	126.2	117.94	93.48
Wikipedia (Chameleon)	296	211.4	128.6	92.5	72.9	62.9	57.4	56.3
Wikipedia (Crocodile)	769	549.2	313.8	226.1	197.1	174	183	187
Wikipedia (Squirrel)	433	427.22	327	240	207	187	197	207
Twitch (England)	457	261.1	203.1	147.4	130.5	108.8	103.8	117.1
Facebook Ego network	496	268.1	206	145.8	130.5	110.2	91.8	74.02
Google Plus Ego network	713	283	190	155	137	118	97	80
Pokec	1882	1344	941	697	537	495	418	400
Facebook large page-page network	345	215	150	143.7	98.5	90.7	82.1	81

Figure 3. Chart showing speedup values for all the datasets



(except for the Facebook ego network). For the Facebook ego network, the rate decreased for 7 processors due to more clustering operations and more edges(connections) for each node(as it is ego network) compared to other networks used in the evaluation. Figure 4 shows the efficiency curves for all the datasets calculated on all the processors. The y-axis values determine the efficiency values, and x-axis values represent the number of processors. It can be observed that maximum efficiency is observed at 6 processors for all medium and large datasets except facebook ego networks. The maximum efficiency value is observed for 7 processors (0.77) than the value at 6 processors (0.75). The reason might be due to the presence of more connections for each node than other datasets. It is also observed that for the networks categorized as small, the maximum efficiency is observed for 4 processors. The efficiency started to decrease because of the completion of more than 80 percent of assigned tasks post which fewer tasks are assigned to the rest of the processors. For medium-sized networks such as Github, maximum efficiency is observed for 6 processors, after which the efficiency decreases. For large networks such as pokec, google plus, the maximum efficiency is observed at 7 processors. The difference in efficiency values for 7 and 8 processors is not significantly more, showing that all the processors are busy executing the assigned tasks. Figure 5 shows the redundancy values for 3 datasets chosen randomly, one from each small, medium, and large dataset on all the processors. For calculating redundancy, we calculated the number of operations performed in both the PSOC and PPSOC techniques.

Table 4 shows the total number of operations for all the datasets for 4 processors. 4 processors are chosen for evaluation as maximum efficiency for small category dataset when the number of processors is 4 (see figure 4). It is observed that the operations in PSOC are significantly more in number compared to the number of operations in PPSOC. For example, the total number of operations for PSOC for the Github dataset is 339303 for all records in the dataset. The total number of operations in the PPSOC algorithm is 38476. The operations are more in PSOC because each loop in step 3 and step 4 calculates the similarity index for all the particles in each subswarm.

Moreover, cluster id is assigned in a separate procedure called *cluster_async ()*, which runs asynchronously. In PPSOC, cluster-id is assigned in the master process, which significantly reduces the time and operations. From figure 5, it can also be observed that the rate of increase in redundancy

Figure 4. Chart showing efficiency values for all the datasets

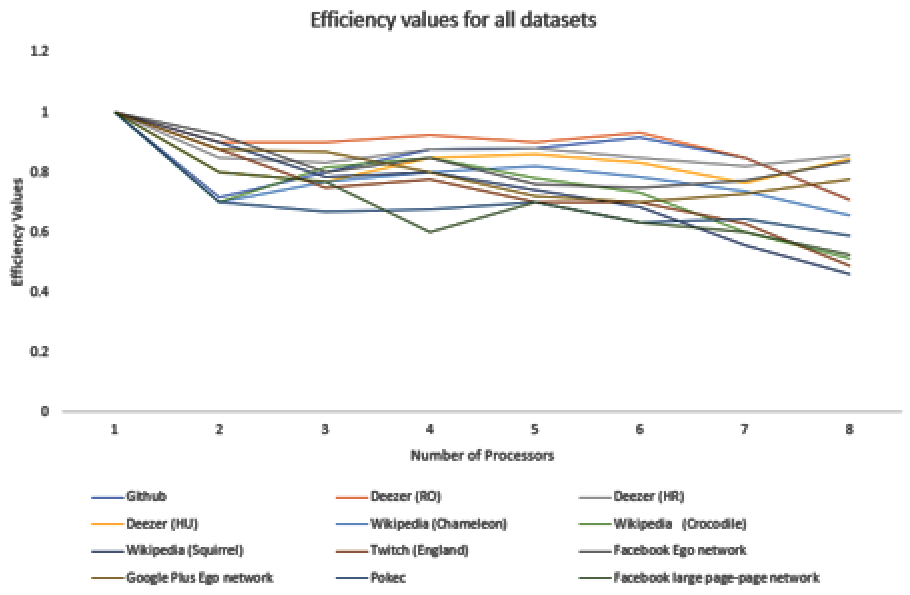


Figure 5. Chart showing redundancy values for some datasets

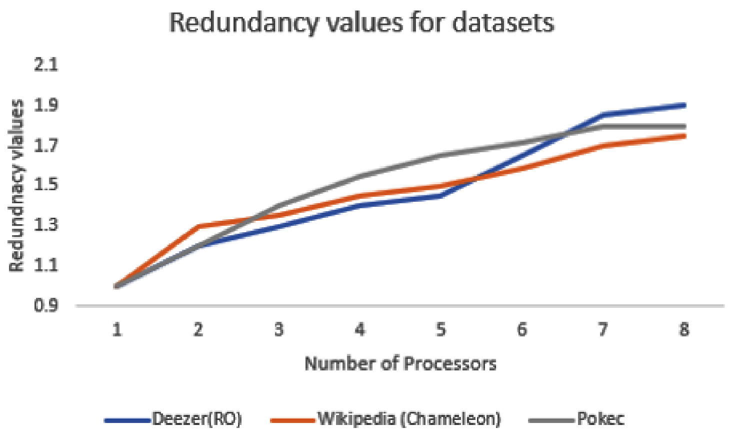


Table 4. Number of operations for PPSOC and PSOC for 4 processors

Dataset	Number of Operations	
	PSOC	PPSOC
Github	339303	38476
Deezer (RO)	375960	42202
Deezer (HR)	491160	60042
Deezer (HU)	427845	52303
Wikipedia	20496	2516
(Chameleon)		
Wikipedia	104682	12806
(Crocodile)		
Wikipedia	46812	5733
(Squirrel)		
Twitch	64137	7850
(England)		
Facebook Ego network		
Google Plus Ego network		
Pokec	3027800	209880
Facebook large page-page network	65789	7657

is more till 4 processors for small datasets, 6 for the medium-sized dataset, and 7 for large datasets, which supports our conclusion (on the maximum number of processors required for maximum efficiency, (see figure 4)) which is drawn while calculating efficiency.

Table 5 shows the evaluation values for measuring the clustering efficiency using the proposed method on social network datasets. It can be observed that the method returned satisfactory values for all the measured metrics. The omega index value is the maximum for the Hungary Deezer dataset. Table 6 shows the total number of overlapping clusters identified. It can be observed that the average

Table 5. Evaluation parameter values-cluster quality

Dataset	Cluster Evaluation Parameter Values		
	Omega	**Silhoutte**	**Tanimoto**
Github	0.61	0.67	0.7
Deezer **(RO)**	0.68	0.72	0.8
Deezer **(HR)**	0.65	0.85	0.8
Deezer **(HU)**	0.85	0.84	0.69
Wikipedia **(Chameleon)**	0.57	0.67	0.71
Wikipedia (**Crocodile**)	0.65	0.71	0.59
Wikipedia **(Squirrel)**	0.72	0.69	0.83
Twitch (England**)**	0.68	0.51	0.83
Facebook Ego network	0.63	0.61	0.76
Google Plus Ego network	0.74	0.83	0.76
Pokec	0.73	0.79	0.83
Facebook large page-page network	0.53	0.64	0.74

execution time of PSOC is 120% more than PPSOC execution time. For PSOC, the algorithm is initialized with default PSO parameters(c1 andc2 are 1.49445, ω=0.729), and the number of subswarms is taken as 4, $\alpha_{i=3,}$ β=0.5.

4.1 Discussion on Migration Policy

Identifying the number of particles to be migrated among islands and identifying the correct iteration in which migration should be triggered plays a vital role in this process. As explained in section 3, the particles whose objective function value is less than Ω are eligible for migration. In this process, we take the value of Ω as 0.3. In a scenario where the many particles in the island have a value less than Ω, too many particles are exchanged among the islands, resulting in too much information sharing between islands, leading to expensive communication costs. On the other hand, if too few particles are exchanged, then there is a chance that the particle's position in the islands does not get updated, leading to a premature convergence problem. In addition to this, the total program execution time and time taken to find the optimal solutions can be more.

We suggest the below method to choose the number of particles to be migrated among islands. We explain the method by optimizing the rastrigin function, one of the complex benchmark functions that can be easily trapped in local optima, resulting in premature convergence. Figure 6 represents the chart showing execution time in seconds for the rastrigin function. The value of Ω is taken as 0.3. The PPSOC procedure is executed for 1000 iterations which is the stopping criteria. The number of particles for which have an objective function value less than Ω is turned out to be 11. The X-axis corresponds to the total particle count. The Y-axis corresponds to the time taken to execute the objective function for finding the optimal solution after exchanging the *n* particles for 100 iterations.

Table 6. Total number of clusters identified

Dataset	Total number of overlapping clusters	
	PPSOC	PSOC
Github	2,702	2,152
Deezer **(RO)**	3,772	4,432
Deezer **(HR)**	4,753	5543
Deezer **(HU)**	12,358	11875
Wikipedia **(Chameleon)**	566	663
Wikipedia (**Crocodile**)	1,152	1124
Wikipedia **(Squirrel)**	101	208
Twitch (England)	1,126	1300
Facebook Ego network	1334	1231
Google Plus Ego network	18344	12445
Pokec	56113	45123
Facebook large page-page network	38787	25223

Figure 6. Chart showing execution time of Rastrigin function for 10 particles

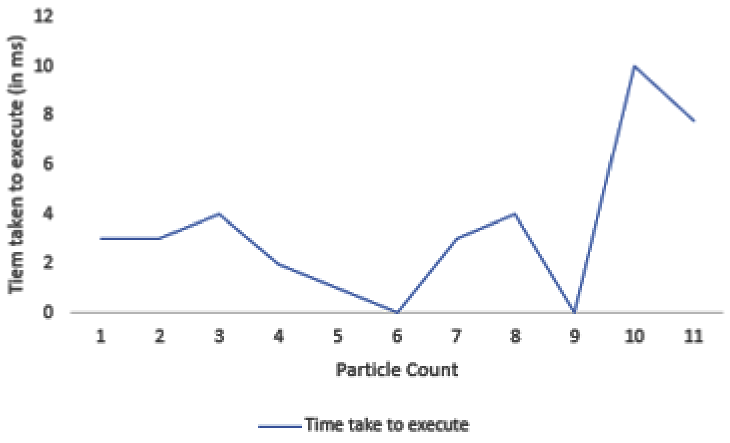
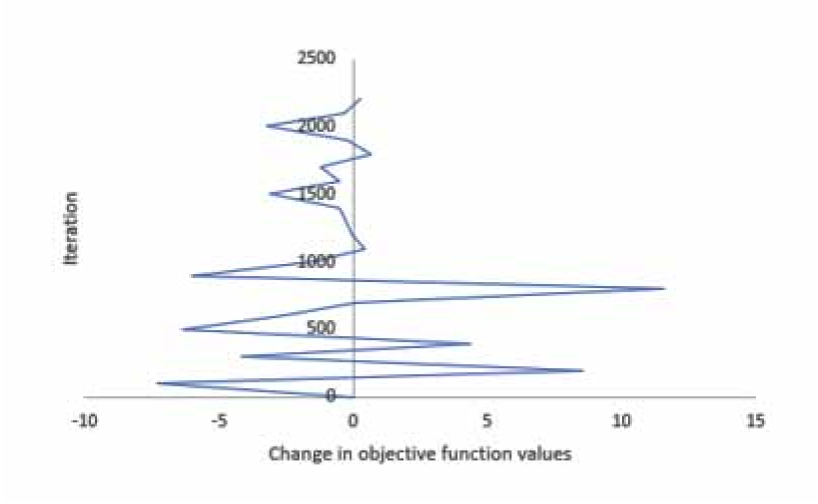


Figure 7. Chart showing velocity of a particle for Rastrigin function



We choose the number of particles for which the time taken to execute is minimum. It can be observed from the chart in fig. 3 that the time taken to execute is less if either 5 or 9 particles are exchanged. We suggest exchanging 5 particles to 9 particles as exchanging 9 particles out of 11 particles can lead to too much information exchange between islands, increasing the overall execution time.

As an attempt to find the number of iterations after which migration can be started, we execute the same Rastrigin function for a particle and compute the velocity of a particle in 1 dimension. We examine the iteration in which the velocity of the particle is degraded. It can be observed from figure 7 that the particle's velocity has started degrading after 1200 iteration. Hence, in this case, the iteration count where the migration should be started can be taken as 1200. Both of these values may not be generalized. Some of the parameters on which the variance depends are the type of objective function, the number of dimensions, execution methodology, category of the dataset, etc.

5. CONCLUSION

In this paper, we proposed a CPU-based parallel architecture based on Master-slave and island models. We applied that to an existing clustering algorithm called PSOC to find out communities in large social networks with node attributes. The model supports information exchange among CPUs with the help of the message passing interface(MPI). The model is scalable based on the computer architecture and can fully utilize modern computers' processing power. The model executes PSO principles and decomposes the problem into multiple subproblems executed in islands separately. The particles in each island are distributed randomly, and the threads are created for each particle with local memory. The model is tested on real-time datasets from the social networks domain and evaluated the performance using evaluation metrics related to parallel systems and social networks. The model has returned significantly good values on all the measured metrics. From the results, it can be observed that the model had achieved the below benefits over PSOC:

1. The proposed model takes 40% less execution time when compared to PSOC.
2. The proposed model is suitable for large dimensions.
3. The proposed model is suitable for large datasets.
4. There is an overall increase of 110 percent in terms of efficiency.

REFERENCES

Batton, D., & Kennedy, J. (2007). Defining a standard for particle swarm optimization. *IEEE Swarm Intelligence Symposium*, 120–127.

Ben Niu, H. H., Tan, L., & Liang, J. J. (2013). Multi-swarm Particle Swarm Optimization with a Center Learning Strategy. *International Conference in Swarm Intelligence: Advances in Swarm Intelligence*, 72-78.

benedekrozemberczki. (2020). https://github.com/benedekrozemberczki/datasets

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics*, (10), 10008.

Chaitanya, K., Somayajulu, D., & Krishna, P. R. (2018). A PSO Based Community Detection in Social Networks with Node Attributes. *2018 IEEE Congress on Evolutionary Computation (CEC)*, 1-6. doi:10.1109/CEC.2018.8477659H

Cheng, R., & Jin, Y. (2015, February). A Competitive Swarm Optimizer for Large Scale Optimization. *IEEE Transactions on Cybernetics*, *45*(2), 191–204. doi:10.1109/TCYB.2014.2322602

Chusanapiputt, S., Nualhong, D., & Jantarang, S. (2005). Relative velocity updating in parallel particle swarm optimization based lagrangian relaxation for large-scale unit commitment problem. IEEE Region 10 Conference, 1–6.

Cui, , Potok, & Palathingal. (2005). Document clustering using particle swarm optimization. *Swarm Intelligence Symposium.*

Cui, S., & Weile, D. S. (2005). Application of a parallel particle swarm optimization scheme to the design of electromagnetic absorbers. *IEEE Transactions on Antennas and Propagation*, *53*(11), 3616–3624. doi:10.1109/TAP.2005.858866

Gies, D., & Rahmat-Samii, Y. (2003). Reconfigurable array design using parallel particle swarm optimization. *IEEE Int. Symp. Antennas Propag. Soc., 1*, 177.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, Inc.

Gregory, S. (2010). Fuzzy overlapping communities in networks. *Journal of Statistical Mechanics-Theory and Experiment, 2*. .10.1088/1742-5468/2011/02/P02017

Grosso, P. (1985). *Computer simulation of genetic adaptations: parallel subcomponent interaction in a multilocus mode*. University of Michigan.

Guimera, R., Sales-Pardo, M., & Anaral, L. A. N. (2004). Modularity from fluctuations in random graphs and complex networks. *Physical Review. E*, *70*, 025101.

Hespanha, J. P. (2004). *An Efficient Matlab Algorithm for Graph Partitioning*. University of California.

Hussain, M., & Hattori, H., & Fujimoto, N. (2016). A CUDA implementation of the standard particle swarm optimization. *18th IEEE International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, 219–226. doi:10.1109/SYNASC.2016.043

Hussain, M., Hattori, H., & Fujimoto, N. (2016). A CUDA implementation of the standard particle swarm optimization. *18th IEEE International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, 219–226. doi:10.1109/SYNASC.2016.043

Kennedy, J., & Eberhart, R. C. (1995). Particle swarm optimization. *IEEE International Conference on Neural Networks*, 1942–1948.

Kennedy, J., & Eberhart, R. C. (2001). *Swarm Intelligence*. Morgan Kaufmann Publishers.

Lalwani, S., Sharma, H., & Satapathy, S. C. (2019). A survey on parallel particle swarm optimization algorithms. *Arabian Journal for Science and Engineering*, 44.

Leskovec & Krevl. (2014). *SNAP Datasets: Stanford Large Network Dataset Collection.* http://snap.stanford.edu/data

Li, J., Chen, W., Zhang, J., & Zhan, Z. (2015). A Parallel Implementation of Multi-objective Particle Swarm Optimization Algorithm Based on Decomposition. *2015 IEEE Symposium Series on Computational Intelligence*, 1310-1317. doi:10.1109/SSCI.2015.187

Li, J., Wan, D., Chi, Z., & Hu, X. (2007); An efficient fine-grained parallel particle swarm optimization method based on GPU acceleration. International Journal of Innovative Computing, Information, & Control, 3(6), 1707–1714.

Liang, P. S. (2005). Dynamic multi-swarm particle swarm optimizer. *Proceedings of IEEE Swarm Intelligence Symposium*, 124–129.

Lihua, C., Yadong, M., & Na, Y. (2009). Parallel particle swarm optimization algorithm and its application in the optimal operation of cascade reservoirs in Yalong river. *Second IEEE International Conference on Intelligent Computation Technology and Automation*, 1, 279–282.

Lipkus. (1999). A proof of the triangle inequality for the Tanimoto distance. *Journal of Mathematical Chemistry, 26*(1-3), 263-265.

Mostaghim, S., Branke, J., Lewis, A., & Schmeck, H. (2008). Parallel multi-objective optimization using Master-Slave model on heterogeneous resources. *IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence),* 1981-1987. doi:10.1109/CEC.2008.4631060

10.1109/CEC.2008.4631060

Mostaghim, S., & Teich, J. (2003). Strategies for Finding Good Local Guides in Multi-objective Particle Swarm Optimization (MOPSO). In *2003 IEEE Swarm Intelligence Symposium Proceedings* (pp. 26–33). IEEE. doi:10.1109/SIS.2003.1202243

Newman, M. (2011). Communities, modules, and large-scale structure in networks. *Nature Physics*, *8*(1), 25–31. doi:10.1038/nphys2162

Newman, M.E., & Girvan, M. (2004). Finding and evaluating community structure in networks. Physical Review E: Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics, 69, 026113.

Ray, T., & Liew, K. (2002). A Swarm Metaphor for Multiobjective Design Optimization. *Engineering Optimization*, *34*, 141–153. doi:10.1080/03052150210915

Rozemberczki, B., Allen, C., & Sarkar, R. (2019). *Multi-scale Attributed Node Embedding.* arXiv:1909.13021

Rozemberczki, B., Davies, R., Sarkar, R., & Sutton, C. (2019). GEMSEC: Graph Embedding with Self Clustering. *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 65-72.

Rozemberczki, Allen, & Sarkar. (2019). *Multi-scale Attributed Node Embedding*. arXiv, 1909.13021

Segaran, T. (2007). Programming collective intelligence. O'Reilly.

Shenghui, L., & Shuli, Z. (2014). Research on FJSP based on CUDA parallel cellular particle swarm optimization algorithm. *International IET Conference on Software Intelligence Technologies and Applications*, 325–329. doi:10.1049/cp.2014.1583

Shenghui, L., & Shuli, Z. (2014). Research on FJSP based on CUDA parallel cellular particle swarm optimization algorithm. *International IET Conference on Software Intelligence Technologies and Applications*, 325–329. doi:10.1049/cp.2014.1583

Shi, Y., & Eberhart, R. C. (1999). Empirical study of particle swarm optimization. *Proceedings of the IEEE Congress on Evolutionary Computation*, 1945–1950.

Singhal, G., Jain, A., & Patnaik, A. (2009). Parallelization of particle swarm optimization using message passing interfaces (MPIs). *IEEE World Congress on Nature*, 67-71. doi:10.1109/NABIC.2009.5393602

Vander Merwe, D. W., & Engelbrecht, A. P. (2003). Data Clustering using Particle Swarm Optimization. *Proceedings of the IEEE Congress on Evolutionary Computation*, 8 – 12. doi:10.1109/CEC.2003.1299577

Venter, G., & Sobieszczanski-Sobieski, J. (2006). Parallel particle swarm optimization algorithm accelerated by asynchronous evaluations. *Journal of Aerospace Computing, Information, and Communication*, *3*(3), 123–137. doi:10.2514/1.17873

Wachowiak, M. P., Smoliková, R., Zheng, Y. F., Zurada, J. M., & Elmaghraby, A. S. (2004). An approach to multimodal biomedical image registration utilizing particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, *8*(3), 289–301.

Wachowiak, M. P., Timson, M. C., & DuVal, D. J. (2017). Adaptive particle swarm optimization with heterogeneous multi-core parallelism and GPU acceleration. *IEEE Transactions on Parallel and Distributed Systems*, *28*(10), 2784–2793. doi:10.1109/TPDS.2017.2687461

Wachowiak, M. P., Timson, M. C., & DuVal, D. J. (2017). Adaptive particle swarm optimization with heterogeneous multi-core parallelism and GPU acceleration. *IEEE Transactions on Parallel and Distributed Systems*, *28*(10), 2784–2793. doi:10.1109/TPDS.2017.2687461

Waintraub, M., Pereira, C., & Schirru, R. (2009). *Parallel particle swarm optimization algorithm in nuclear problems*. https://www.iaea.org/inis/collection/NCLCollectionStore

Wasserman, S. (1994). *Social Network Analysis Methods and Applications*. Cambridge University Press.

Zhang, J., & Ding, X. (2011). A multi-swarm self-adaptive and cooperative particle swarm optimization. *Engineering Applications of Artificial Intelligence*, *24*(6), 958–967. doi:10.1016/j.engappai.2011.05.010

Radha Krishna P. is currently a Professor at NIT Warangal, Telangana, India. Prior to this, he was a Principal Research Scientist at Infosys Labs, Infosys Limited, Hyderabad, India, a faculty at the Institute for Development and Research in Banking Technology (IDRBT) and a Scientist at the National Informatics Centre, Govt. of India. His research interests include data mining, machine learning, big data, and electronic contracts and services.