An approach to support generic topologies in distributed PSO algorithms in Spark

Xoán C. Pardo[0000−0001−8577−6980]1, Patricia González[0000−0003−0378−9222]1, Julio R. Banga[0000−0002−4245−0320]2, and Ramón Doallo[0000−0002−6011−3387]1

[1] CITIC, Computer Architecture Group, Universidade da Coruña, Spain
{xoan.pardo,patricia.gonzalez,ramon.doallo}@udc.es
[2] CSIC, Computational Biology Lab, Spain
j.r.banga@csic.es

Abstract. Particle Swarm Optimization (PSO) is a popular population-based search algorithm that has been applied to all kinds of complex optimization problems. Although the performance of the algorithm strongly depends on the social topology that determines the interaction between the particles during the search, current Metaheuristic Optimization Frameworks (MOFs) provide limited support for topologies. In this paper, we present an approach to support generic topologies in distributed PSO algorithms within a framework for the development and execution of population-based metaheuristics in Spark, which is currently under development.

1 Introduction

Parallel population-based metaheuristics [6] are common approaches to the optimization of large-scale problems because of their potential to obtain a satisfactory solution within a reasonable time. However, implementing scalable and efficient parallel metaheuristics requires knowledge of the underlying software platform and hardware architecture. Metaheuristic Optimization Frameworks (MOFs) provide customizable parallel implementations of the most popular metaheuristics, but there are significant differences in the level of support for parallelism and execution performance they provide [16, 23]. Furthermore, although distributed frameworks for Big Data, like Spark or Flink, have allowed to apply parallel metaheuristics at an unprecedented scale, MOFs support to these frameworks is very limited.

Particle Swarm Optimization (PSO) [14] is a population-based metaheuristic inspired by the collective behavior of flocks of birds that is very popular because it has a simple algorithmic description and has been successfully applied to a great variety of optimization problems. PSO represents the set of candidate solutions as a swarm of particles that traverse a multidimensional search space seeking the optimum. The interaction between particles during the search is determined by a social topology [15, 18] that dictates the grouping of particles into neighborhoods where information is shared. Even though topologies are fundamental in the performance of PSO, most MOFs only support the *gbest* topology.

In this paper, we present an approach that aims to overcome the aforementioned limitations by providing support for generic topologies in distributed PSO algorithms. The approach is implemented within a framework for the development and execution of population-based metaheuristics in Spark, which is currently under development. The rest of the paper is organized as follows: section 2 summarizes the related work; section 3 briefly introduces PSO and topologies; in section 4 the proposed approach is described; a preliminary experimental evaluation is presented in section 5; and section 6 concludes the paper.

2 Related work

To the best of our knowledge, there are only a few recent studies that compare MOFs and they only partially address the support for parallel and distributed computing. The comparative study in [16] shows that there are significant differences in the level of support, if any, that the frameworks provide for parallel and distributed computing. The analysis in [23] concludes that most MOFs only parallelize the evaluation of solutions, and only two provide support for both of the most common distributed models, i.e. the master-worker and the island-based models. Also, experimental evaluation found significant differences in performance under demanding configurations, both in terms of execution time and memory usage.

Furthermore, there are only a few proposals of MOFs that support Big Data frameworks. ECJ+Hadoop [4] is an enhancement to ECJ based on MapReduce that distributes the evaluation of solutions on Hadoop clusters using a *map* evaluator. jMetalSP [2] combines jMetal with different streaming engines such as Spark, Flink or Kafka. In Spark, RDDs are used to distribute the evaluation of solutions. HyperSpark [5] provides a configurable iterative workflow for the parallel execution in Spark of sequential metaheuristics as independent tasks, with or without cooperation between them. In [10] a framework to support the development of parallel evolutionary algorithms (PEAs) in Spark was proposed, and three PSO variants implemented in a unified way. With the exception of HyperSpark, all the proposals implement a master-worker model of parallelism, distributing only the evaluation of solutions.

With regard to PSO, some proposals implement parallel PSO variants on frameworks for Big Data. Examples using MapReduce are a constricted PSO, named MRPSO, in [19]; a cooperative PSO in [25]; or a quantum-behaved PSO in [27]. In [7] implementations in Hadoop and Spark are compared with a problem of energy optimization in buildings. Examples of implementations in Spark are a Cooperative co-Evolution PSO in [3]; a quantum-behaved PSO in [28]; or an hybridized island-based PSO in [12]. There are also applications to real problems, like the efficient utilization of water resources [17]; the training of Recurrent Neural Networks [26]; or clustering problems [1]. As far as we know, only MRPSO has support for different static or dynamic topologies, although only results for the *gbest* topology are reported in [19].

3 The PSO algorithm

The synchronous canonical PSO [13], in which all particles are updated at once, was used as a reference in this work, combined with different variants of the velocity update equation, i.e. the standard 2007 and the constricted and condensed constricted forms. Each particle stores its current position, velocity, fitness, and its best historical position and fitness. Positions and velocities are assumed to be D-dimensional vectors of real values.

3.1 Social topology

The influence of other particles is determined by the social topology, that dictates which particles are part of a neighborhood and the information shared between them. Originally, two topologies were proposed: (i) Global best *(gbest)*, all particles form a unique neighborhood; and (ii) Local best *(lbest)*, particles are influenced by its adjacent neighbors, usually two, following a *Ring* topology. In both of them, particles share their best position with their neighbors and the best of the neighborhood is used to update the velocity. As the topology highly influences the performance of the algorithm and there is no topology that is the best for all problems, different topologies have been studied [15, 18], including static (e.g. Von Newmann, Pyramidal, or Four Clusters), spacial (i.e. formed based on distance between particles), dynamic and adaptive structures, as well as other *models of influence* not based only on the best particle of the neighborhood, e.g. the *Full Informed PSO* (FIPS) [20].

We have analyzed the support for PSO topologies provided out of the box by four of the most widely used MOFs: Paradiseo (v3.0.0) [9], ECJ (v27) [24], HeuristicLab (v3.3.16) [11] and JMetal (v6.0) [8]. Our findings are summarized in the following:

- With the exception of JMetal, that does not support topologies out of the box, all the MOFs reviewed support at least the most common static topologies: Ring, Star and Random. ECJ and HeuristicLab also support dynamic topologies by randomly regenerating the topology at the end of each generation.
- Using the global and/or neighborhood best solution is the only model of influence supported.

4 An approach to support generic topologies

In this section we present a proposal to support generic topologies in distributed PSO algorithms that aims to overcome the limitations found in MOFs. An early version has been implemented within a framework for the development and execution of population-based metaheuristics in Spark, which is currently under development. In our approach, a social topology is composed by three elements:

1. A *self* boolean value, to indicate whether a particle should be included in its neighborhood.

Complete (*aka* GBest, Star, All), Directed Ring, Directed Scale Free, Generalized Petersen, G_{nm} Random, G_{np} Random Bipartite, Grid, HyperCube, KClusters, Kleinberg's Small World, Linear, Planted Partition, Pyramid, Random, Regular (with small-world shortcut [15]), Ring (*aka* LBest), Scale-Free Network, Square (*aka* VonNewman), Star (one particle in the center connected to all the others), Wheel, Windmill

Table 1: Topologies supported out of the box. With few exceptions, the topologies are parameterizable, e.g. for the Ring topology, the generalized version with k neighbors is provided.

2. A *topology shape*, represented as a DAG (*Directed Acyclic Graph*) of particle identifiers. Particles are assumed to have unique identifiers, and an edge $p_i \rightarrow p_j$ in the graph indicates that p_i is a neighbor of p_j and could share information with it. Table 1 shows the topology shapes currently supported. Custom topologies (listing 1.1) and importing topologies from GraphViz *.dot* files are also supported.
3. A *model of neighborhood influence*, represented as a function that maps particle identifiers to neighborhood influences. To support generic models of influence, two operations were defined by means of a Scala *trait* (listing 1.2): (i) *contribution*, a template method to collect the information shared by each particle with its neighborhood; and (ii) *neighborhoodInfluence*, a generic function builder, called on every swarm move, to instantiate the function that obtains the influence the neighborhood has on a particle. Different models are instantiated by passing different *collect* and *reduce* functions to *neighborhoodInfluence*. Currently, the models in [20] (i.e. Best, FIPS, wFIPS, wdFIPS, Self, and wSelf) are implemented. The instantiated function can then be used in any velocity update equation. For example, in the condensed form of the constricted equation, it would be called to get $\vec{I}_i^t$, the neighborhood influence on particle i at time t:

$$\vec{v}_i^{t+1} = \chi(\vec{v}_i^t + c_{max}(\vec{I}_i^t - \vec{x}_i^t)) \tag{1}$$

where $\vec{x}_i^t$ and $\vec{v}_i^t$ are the position and velocity of the i-th particle at time t, χ is the constriction factor, and c_{max} is the upper limit of coefficients sum.

The proposal was designed with distributed PSO algorithms in mind. The *collect-reduce* approach shown in algorithm 1 was applied in the implementation of *neighborhoodInfluence*. It has two phases:

```
CustomRing {
  topology {
    self = true
    shape {
      name = Custom
      neighborhoods {
        0: [1, 2, 4],
        1: [2, 3, 0],
        2: [3, 4, 1],
        3: [4, 0, 2],
        4: [0, 1, 3]
      }
    }
    neighborhood_influence.name = wdFIPS
  }
}
```

Listing 1.1: Configuration of a custom *Ring* topology for a swarm of size N=5 with FIPS weighted by distance model of neighborhood influence and $Neighborhood(p_i) = \{p_i, p_{i+1 \bmod N}, p_{i+2 \bmod N}, p_{i-1 \bmod N}\}$.

```
trait NeighborhoodOps {
  // method to collect the contribution of each particle to the neighborhood
  def contribution[T](pf:ParticleContributionFunction[T]):NeighborhoodContribution[T]
  // a generic factory method to instantiate the neighborhood influence function
  def neighborhoodInfluence[T,S](self:Boolean)(t:Topology)(
  collect:ParticleContributionFunction[T])(
  reduce:NeighborhoodContributionFunction[T,S]):NeighborhoodInfluenceFunction[S]
}
```

Listing 1.2: Operations added to a swarm to support generic models of neighborhood influence.

Algorithm 1 Pseudo-code of *neighborhoodInfluence*.

Inputs: self; topology; collect(); reduce()
Output: the neighborhood influence function

1: $contrib$ = contribution(collect) ▷ the contribution of each particle is collected
2: **for** each id in topology **do**
3: $neighbors$ = topology.predecessors(id) ▷ neighbors are the predecessors in the DAG
4: **if** self **then**
5: $neighborhood = id +\!\!+ neighbors$ ▷ the particle id is included in the neighborhood
6: **else**
7: $neighborhood = neighbors$
8: **end if**
9: $influence[id]$ = reduce(contrib(id), contrib($neighborhood$)) ▷ the influence the neighborhood has on the particle is calculated and stored in a map indexed by id
10: **end for**
11: **return** $id \Rightarrow influence[id]$ ▷ function that maps particle ids to neighborhood influences

1. The *contribution* method (line 1) is called to collect the information contributed by each particle of the swarm. Different implementations are provided for the two swarm states, grouped or distributed, supported by our framework.
2. The *reduce* function (line 9) is then used to calculate the influence the neighborhood has on a particle from the contributions of the particle and its neighbors.

5 Experimental evaluation

Several experiments were performed to validate the proposal, as a preliminary evaluation focused on debugging and profiling the current implementation and verifying the genericness and correctness of the approach. All the experiments were carried out in a MacBook Pro M1 Pro with 10-core CPU and 16 GB RAM using the sequential version of the PSO algorithm implemented in our framework. Due to the space limitations, only a summary of the results is reported here. Configuration files, logs and detailed analysis of the results are available in a companion repository [22].

The first series of experiments performed reproduce those in [20]. The same methodology described in the paper was followed to evaluate the performance, iterations to criteria and proportion reaching criteria of six models of neighborhood influence (i.e. Best, FIPS, wFIPS, wdFIPS, Self, and wSelf) combined with five topologies (i.e. Square, Ring, 4-clusters, Pyramid and All), four of which are used in two different configurations, with and without including the target particle in the neighborhood. Only experiments with symmetrical initialization were reproduced. The results were obtained by combining the standarized individual results of five benchmark functions (i.e. Sphere, Rastrigin, Griewank -in two sizes-, Rosenbrock and Schaffer f6), that were executed 40 times each for each combination with the parameters provided in the paper.

Although all experiments run successfully, showing the genericness of the proposal, the results were quite different to those reported in [20] for all the dimensions evaluated. For example, the proportion of experiments reaching criteria, i.e. the proportion of runs that found the VTR (*Value To Reach*) within 10,000 iterations, is shown in Table 2. In general, the ratios of success are well below those reported in [20]. The best results were obtained by the wdFIPS model with a 99.6% ratio for the fully connected topologies, and the other FIPS variants with ratios higher than 91% for the UAll topology. The worst results were obtained by the Best model. The Self variants performed best in all cases, except for the fully connected topologies.

Since the values used for the PSO parameters are not given in [20] or whether any strategies were used to improve the performance of the algorithm, we decided to conduct a second series of experiments, reducing the search space of some functions and limiting the velocities and positions of the particles. Although this time the results showed more similarities, they were still worse in general.

To verify if the difference might be caused by differences in the PSO parameters, a second series of experiments were carried out to compare the proposal with the PSO implementation of ECJ using a two-sample Kolmogorov-Smirnov test. The experiments of the first series were repeated using the

	Square	Ring	4-Clusters	Pyramid	All	USquare	URing	UPyramid	UAll
Best	0.017	0.021	0.029	0.008	0.008	**0.038**	0.029	0.008	0.017
FIPS	0.167	0.167	0.217	0.283	0.663	0.167	0.167	0.167	**0.917**
wFIPS	0.167	0.167	0.167	0.167	0.625	0.167	0.167	0.167	**0.917**
wdFIPS	0.167	0.167	0.167	0.167	**0.996**	0.167	0.167	0.167	**0.996**
Self	0.417	0.546	0.479	0.475	0.375	0.571	**0.854**	0.446	0.383
wSelf	0.300	0.458	0.379	0.400	0.296	0.633	**0.804**	0.388	0.354

Table 2: Proportion of the first series of experiments reaching the VTR within 10,000 iterations. In bold are the best results for each model of neighborhood influence.

	Ring	Ring_4	All	Rand_4	URing	URing_4	UAll	URand_4
PSO-T	0.045	0.015	0.04	0.0	0.02	0.02	0.015	0.0
ECJ	0.04	0.05	0.03	0.005	0.01	0.015	0.005	0.0
Mendes	0.913		0.754		0.908		0.754	

Table 3: Ratio of success of our approach (PSO-T) and ECJ for the topologies evaluated. Results from [20] are also provided when available.

combinations supported by ECJ, i.e. the Best model of influence combined with the Ring, Random and All topologies, the first two with two different degrees, and the same benchmark functions except Schaffer f6. This time the results were very similar. The null hypothesis was rejected only in 6 out of the 160 combinations tested at significance level $\alpha = 0,043$ and 2 out of 160 at significance level $\alpha = 0,01$. This similarity can also be seen in Table 3, which shows a comparison of the ratio of success. As an anecdote, thanks to the experiments performed in this comparison, several bugs were detected in the PSO implementation of ECJ [21] and a pull request with the fix was submitted to the ECJ repository.

6 Conclusions

In this paper, an approach to support generic topologies in distributed PSO algorithms implemented within a framework for the development and execution of population-based metaheuristics in Spark has been presented. A preliminary experimental validation of the genericness and correctness of the approach was carried out using a sequential PSO implementation and different combinations of topologies and models of neighborhhood influence. To the best of our knowledge, no other MOF provides this level of genericness for PSO topologies. As future work, it is planned to perform an evaluation of the parallel performance of the approach and adding support for dynamic topologies in the near term.

Acknowledgments. This work was supported by the Ministry of Science and Innovation of Spain (PID2019-104184RB-I00, AEI, 10.13039/501100011033) and by Xunta de Galicia (Consolidation Program of Competitive Reference Groups, ref. ED431C 2021/30).

References

1. Al-Sawwa, J., Ludwig, S.A.: Parallel particle swarm optimization classification algorithm variant implemented with apache spark. Concurrency and Computation: Practice and Experience **32**(2), e5451 (2020). https://doi.org/https://doi.org/10.1002/cpe.5451
2. Barba-González, C., Nebro, A.J., Benítez-Hidalgo, A., García-Nieto, J., Aldana-Montes, J.F.: On the design of a framework integrating an optimization engine with streaming technologies. Future Generation Computer Systems **107**, 538–550 (2020). https://doi.org/https://doi.org/10.1016/j.future.2020.02.020
3. Cao, B., Li, W., Zhao, J., Yang, S., Kang, X., Ling, Y., Lv, Z.: Spark-based parallel cooperative co-evolution particle swarm optimization algorithm. In: 2016 IEEE International Conference on Web Services (ICWS). pp. 570–577 (2016). https://doi.org/10.1109/ICWS.2016.79
4. Chávez, F., de Vega, F.F., Lanza, D., Benavides, C., Villegas, J., Trujillo, L., Olague, G., Román, G.: Deploying massive runs of evolutionary algorithms with ecj and hadoop: Reducing interest points required for face recognition. The International Journal of High Performance Computing Applications **32**(5), 706–720 (2018). https://doi.org/10.1177/1094342016678302

5. Ciavotta, M., Krstić, S., Tamburri, D.A., Van Den Heuvel, W.J.: Hyperspark: A data-intensive programming environment for parallel metaheuristics. In: 2019 IEEE International Congress on Big Data (BigDataCongress). pp. 85–92 (2019). https://doi.org/10.1109/BigDataCongress.2019.00024
6. Crainic, T.: Parallel Metaheuristics and Cooperative Search, pp. 419–451. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-319-91086-4_13
7. Cui, L.: Parallel PSO in Spark. Master's thesis, Faculty of Science and Technology. University of Stavanger (2014)
8. Doblas, D., Nebro, A.J., López-Ibáñez, M., García-Nieto, J., Coello Coello, C.A.: Automatic design of multi-objective particle swarm optimizers. In: Dorigo, M., Hamann, H., López-Ibáñez, M., García-Nieto, J., Engelbrecht, A., Pinciroli, C., Strobel, V., Camacho-Villalón, C. (eds.) Swarm Intelligence. pp. 28–40. Springer International Publishing, Cham (2022)
9. Dreo, J., Liefooghe, A., Verel, S., Schoenauer, M., Merelo, J.J., Quemy, A., Bouvier, B., Gmys, J.: Paradiseo: From a modular framework for evolutionary computation to the automated design of metaheuristics: 22 years of paradiseo. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 1522–1530. Association for Computing Machinery, New York, NY, USA (2021), https://doi.org/10.1145/3449726.3463276
10. Duan, Q., Sun, L., Shi, Y.: Spark clustering computing platform based parallel particle swarm optimizers for computationally expensive global optimization. In: Auger, A., Fonseca, C.M., Lourenço, N., Machado, P., Paquete, L., Whitley, D. (eds.) Parallel Problem Solving from Nature – PPSN XV. pp. 424–435. Springer International Publishing, Cham (2018)
11. Elyasaf, A., Sipper, M.: Software review: the heuristiclab framework. Genetic Programming and Evolvable Machines **15**(2), 215–218 (2014). https://doi.org/10.1007/s10710-014-9214-4
12. Fan, D., Lee, J.: A hybrid mechanism of particle swarm optimization and differential evolution algorithms based on spark. KSII Transactions on Internet and Information Systems **13**(12), 5972–5989 (2019)
13. Freitas, D., Lopes, L.G., Morgado-Dias, F.: Particle swarm optimisation: A historical review up to the current developments. Entropy **22**(3) (2020). https://doi.org/10.3390/e22030362
14. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of ICNN'95-international conference on neural networks. vol. 4, pp. 1942–1948. IEEE (1995)
15. Liu, Q., Wei, W., Yuan, H., Zhan, Z.H., Li, Y.: Topology selection for particle swarm optimization. Information Sciences **363**, 154–173 (2016). https://doi.org/https://doi.org/10.1016/j.ins.2016.04.050
16. Lopes Silva, M.A., de Souza, S.R., Freitas Souza, M.J., de França Filho, M.F.: Hybrid metaheuristics and multi-agent systems for solving optimization problems: A review of frameworks and a comparative analysis. Applied Soft Computing **71**, 433–459 (2018). https://doi.org/https://doi.org/10.1016/j.asoc.2018.06.050
17. Ma, Y., an Zhong, P., Xu, B., Zhu, F., Lu, Q., Wang, H.: Spark-based parallel dynamic programming and particle swarm optimization via cloud computing for a large-scale reservoir system. Journal of Hydrology **598**, 126444 (2021). https://doi.org/https://doi.org/10.1016/j.jhydrol.2021.126444
18. McNabb, A., Gardner, M., Seppi, K.: An exploration of topologies and communication in large particle swarms. Faculty Publications. 869. (2009). https://doi.org/10.1109/CEC.2009.4983015
19. McNabb, A.W., Monson, C.K., Seppi, K.D.: Parallel pso using mapreduce. In: 2007 IEEE Congress on Evolutionary Computation. pp. 7–14 (2007). https://doi.org/10.1109/CEC.2007.4424448
20. Mendes, R., Kennedy, J., Neves, J.: The fully informed particle swarm: simpler, maybe better. IEEE Transactions on Evolutionary Computation **8**(3), 204–210 (2004). https://doi.org/10.1109/TEVC.2004.826074
21. Pardo, X.C.: Ecj issue 82: bugs in the pso implementation, https://github.com/GMUEClab/ecj/issues/82
22. Pardo, X.C.: Experiments to validate the proposal of a generic topology for distributed PSO algorithms (Apr 2023). https://doi.org/10.5281/zenodo.7823049
23. Ramírez, A., Barbudo, R., Romero, J.R.: An experimental comparison of metaheuristic frameworks for multi-objective optimization. Expert Systems **n/a**(n/a), e12672 (2021). https://doi.org/https://doi.org/10.1111/exsy.12672
24. Scott, E.O., Luke, S.: Ecj at 20: Toward a general metaheuristics toolkit. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 1391–1398. GECCO '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3319619.3326865
25. Wang, Y., Li, Y., Chen, Z., Xue, Y.: Cooperative particle swarm optimization using mapreduce. Soft Computing **21**(22), 6593–6603 (2017). https://doi.org/10.1007/s00500-016-2390-9
26. Wu, K., Zhu, Y., Li, Q., Han, G.: Algorithm and implementation of distributed esn using spark framework and parallel pso. Applied Sciences **7**(4) (2017). https://doi.org/10.3390/app7040353
27. Zhang, G., Li, Y., Chen, Z., Wang, Y., Jiao, L., Xue, Y.: A novel distributed quantum-behaved particle swarm optimization. Journal of Optimization **2017**, 4685923 (2017). https://doi.org/10.1155/2017/4685923
28. Zhang, Z., Wang, W., Gao, N., Zhao, Y.: Spark-based distributed quantum-behaved particle swarm optimization algorithm. In: Luo, Y. (ed.) Cooperative Design, Visualization, and Engineering. pp. 295–298. Springer International Publishing, Cham (2018)