II CONFERENCIA INTERNACIONAL DE PROCESAMIENTO DE LA INFORMACIÓN ¨CIPI 2019¨

Revisión de las meta-heurísticas implementadas en Apache Spark.

Review of metaheuristics implemented in Apache Spark.

Ernesto Fundora Fernández[1], Ernesto Diaz López[2]

1-Ernesto R. Fundora Fernández. Universidad Central "Marta Abreu" de las Villas, Cuba. E-mail: erfundora@uclv.cu
2- Ernesto Diaz López. Universidad Central "Marta Abreu" de las Villas, Cuba. E-mail: ediaz@uclv.edu.cu

Resumen: Los problemas clásicos de optimización no se encuentran ajenos a las nuevas tendencias del Big Data y computación distribuida. Actualmente las meta-heurísticas de optimización son reconocidas como estrategias efectivas para el tratamiento de problemas complejos de optimización. La adaptación de estos algoritmos al nuevo ecosistema de Apache Spark, no constituye un procedimiento trivial. Este artículo realiza un análisis de un grupo de meta-heurísticas implementadas actualmente, concentrando las desarrolladas en ambientes distribuidos para análisis de Big Data. Atendiendo a dos clasificaciones fundamentalmente, las basadas en trayectoria y las basadas en población. Este trabajo presenta conceptos, implementaciones y estrategias utilizadas en las investigaciones presentes en el estado del arte.

Abstract: *The classic problems of optimization are not outsiders to the new trends of Big Data and distributed computing. Currently, optimization metaheuristics are recognized as effective strategies for the treatment of complex optimization problems. The adaptation of these algorithms to the new ecosystem of Apache Spark, is not a trivial procedure. This article makes an analysis of a group of metaheuristics currently implemented,*

concentrating those developed in distributed environments for Big Data analysis. Attending to both classifications fundamentally, those based on trajectory and those based on population. This paper presents concepts, implementations and strategies used in the research present in the state of the art.

Palabras Clave: Big Data, Apache Spark, problemas de optimización, meta-heurísticas.

Keywords: *Big Data, Apache Spark, optimization problems, metaheuristics.*

1. Introducción

En las últimas décadas, el indetenible crecimiento de los datos generados por el desarrollo de la ciencia, la tecnología y la sociedad, hacen imprescindible el uso de herramientas que ayuden al procesamiento de esta información. Por esta razón Apache Spark se ha popularizado como una ventajosa plataforma de código abierto, alineado al novedoso paradigma del Big Data. Fue diseñado para el análisis y procesamiento de grandes volúmenes de datos, potenciando características como la escalabilidad y velocidad. Además de atrapar la atención de los científicos, precisamente por el gran número de áreas y empresas donde pueden actualmente aplicarse. En julio de 2016, los miembros de Databricks[1] hicieron un estudio resultando como industrias claves que utilizan Apache Spark: la industria del software, consultoría, negocios financieros (bancos), publicidad, mercadotecnia, comercio electrónico, medicina, farmacia, biotecnología, telecomunicaciones, educación y hardware, son de las pioneras en adaptarse a las nuevas posibilidades y potencialidades de esta plataforma.

Los problemas clásicos de optimización no se encuentran ajenos a las nuevas tendencias del Big Data y computación distribuida. Las técnicas que dan solución a estos problemas se clasifican en dos grandes grupos de algoritmos: exactos y aproximados. Los métodos exactos por su parte permiten encontrar soluciones exactas, pero consumen mucho tiempo en el proceso de búsqueda para problemas reales. Sin embargo, las meta-heurísticas

[1] Apache Spark Survey 2016 Results Now Available - The Databricks Blog. (n.d.) https://databricks.com/blog/2016/09/27/spark-survey-2016-released.html.

dentro de los algoritmos aproximados son métodos diseñados a partir de estrategias de exploración y explotación de espacios de búsqueda, obteniendo buenas soluciones en un tiempo de cómputo razonable (Alba, Luque, & Nesmachnow, 2013).

La adaptación de estos algoritmos al nuevo contexto del Big Data es una activa línea de investigación. De ahí, que autores como Nunez & Attoh-Okine (2014) estudie enfoques meta-heurísticos con un interés marcado en el análisis de big data, para reforsar la predicción e identificación de patrones ocultos en la ingeniería feroviaria. Cui (2014) analiza el desempeño de la metaheurística Optimización de Enjambre de Partículas (PSO, por sus siglas en Inglés) en la plataforma Spark aplicándolo a un problemas de optimización de eficiencia energética. En (Teijeiro, Pardo, González, Banga, & Doallo, 2016) se presenta una versión paralela de la metaheurística Evolución Diferencial haciendo uso del modelo islas, aplicada a un conjunto de problemas de estimación de parámetros en biología de sistemas.

El objetivo de este trabajo es realizar análisis de las meta-heurísticas presentes en la literatura implementadas usando el novedoso ecosistema Apache Spark, para enfrentar el tratamiento de problemas de optimización complejos en Big Data.

El artículo se encuentra organizado de la siguiente forma: Sección I: se describe la plataforma Apache Spark como herramienta para el análisis de grandes volúmenes de datos. Se clasifican y analizan las meta-heurísticas implementadas en Apache Spark, en la Sección II, finalmente las conclusiones son presentadas en la Sección III.

2. Apache Spark

Apache Spark comenzó como un proyecto de investigación en el AMPLab de UC Berkeley en 2009 y se abrió a principios de 2010. Luego de su lanzamiento, Spark se convirtió en una amplia comunidad de desarrolladores. Su ecosistema proporciona una API de alto nivel en Java, Scala, Python y R, y un motor optimizado que admite gráficos de ejecución generales (Meng et al., 2016).

Apache Spark está compuesto por cuatro librerías de gran utilidad, además de su núcleo, lo presentamos en la Figura 1:

1. ***Spark SQL:*** integra el procesamiento relacional con la API de programación funcional de Spark (Armbrust et al., 2015).
2. ***Spark Streaming[2]:*** es una extensión del núcleo de Spark API permitiéndole a los ingenieros de datos y científicos de datos el procesamiento de datos en tiempo real.
3. ***GraphX:*** combina las ventajas de los sistemas de datos paralelos y de gráficos paralelos al expresar de manera eficiente generación de gráficos dentro del modelo de datos paralelos de Spark (Xin, Gonzalez, Franklin, & Stoica, 2013).
4. ***MLlib:*** consta de rápidas y escalables implementaciones de algoritmos de aprendizaje tales como: clasificación, regresión, filtrado colaborativo, agrupamiento y reducción de dimensionalidad, también proporciona una variedad de estadísticas, álgebra lineal y dos algoritmos de optimización (Meng et al., 2016).

Según (Zaharia et al., 2016), Apache Spark desde el 2010, fecha en que se lanzó, ha crecido a un ritmo que lo coloca dentro de las plataformas de código abierto más activas, respaldado por la no despreciable cifra de 1000 contribuidores de más de 250 organizaciones.

Varios autores han caracterizado al ecosistema de Apache Spark (Xin et al., 2013)(Zaharia et al., 2016)(Bosagh Zadeh et al., 2016)(Teijeiro et al., 2016), luego del análisis de estos artículos se resumen las siguientes características:

- API unificada, que permite un fácil desarrollo de aplicaciones.
- Provee una interfaz de programación integrada en el lenguaje, conocido como Conjunto de Datos Resilentes Ditribuidos o *Resilient Distributed Datasets* (RDD), admitiendo cálculos eficientes en memoria y tolerancia a fallos.

[2] Spark Streaming with Apache Spark - What is Spark Streaming? (n.d.) https://databricks.com/glossary/what-is-spark-streaming

- Los RDD se puden particionar por el usuario, el motor de ejecución evalúa dichas particiones y organiza las tareas evitando el movimiento de los datos.

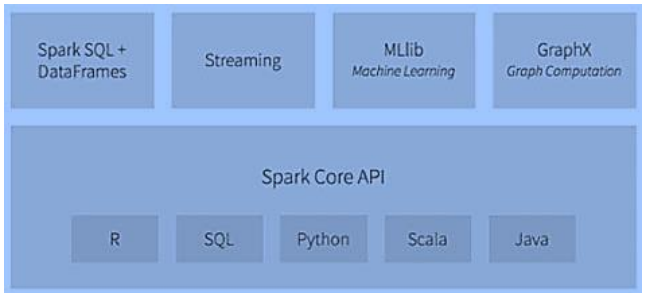

Figura 1. Ecosistema de Apache Spark

3. Meta-heurísticas implementadas en Spark

En esta sección se realiza un análisis de las principales meta-heurísticas implementadas en Apache Spark, clasificadas en dos grupos: basadas en trayectoria y basadas en población. Además, se describen sus características, ventajas y desventajas, así como las estrategias de paralelización utilizadas en la integración al ambiente distribuido de Spark.

Basadas en trayectoria

Las meta-heurísticas basadas en trayectoria, inician con una única solución y se alejan de ella, describiendo una trayectoria en el espacio de búsqueda. Los métodos de trayectoria abarcan principalmente el método de recocido simulado, la búsqueda tabú, el método GRASP, la búsqueda en vecindades variable y la búsqueda local iterada.

Búsqueda Tabú

La Búsqueda Tabú o *Tabu Search (TS)* definida en (Glover, 1989) es una meta-heurística para resolver problemas de optimización combinatoria. Además, es un procedimiento adaptativo con la capacidad de hacer uso de varios métodos diferentes, tales como algoritmos de programación lineal y heurísticas especializadas, dirigida a superar las limitaciones del optimo local.

El K-means, es uno de los algoritmos de agrupamiento más populares actualmente, por lo cual se ha integrado a las principales librerías de las plataformas de Big Data. La inapropiada selección inicial de los centroides en el K-means influye significativamente en la estructura y calidad del agrupamiento final. Además, como en otros casos de heurísticas es susceptible a quedar atrapado en un óptimo local. En (Lu, Cao, Rego, & Glover, 2018) se propone una estrategia de búsqueda tabú para abordar el problema de optimalidad local junto con una implementación paralela en la plataforma Spark haciendo posible el manejo de problemas a gran escala más eficientes. Así mismo, el modelo planteado presenta la similaridad partiendo de una medida de distancia utilizadas para el agrupamiento, enfocándose principalmente en minimizar la distancia al interior del grupo, así como maximizar la distancia entre grupos Figura 2.

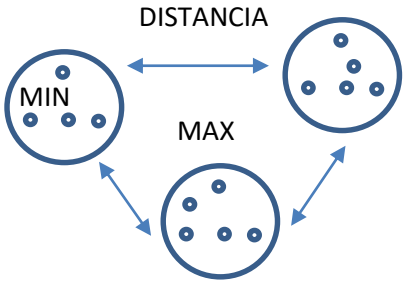

Figura 2: Modelo del agrupamiento.

El algoritmo TS se distingue de otras meta-heurísticas a partir de su enfoque en el uso de memoria adaptativa y estrategias para la explotación de esta memoria. El manejo de las fases de intensificación y diversificación en la estrategia de búsqueda es una notable ventaja del uso de la memoria.

La versión paralela del TS en Spark se basa principalmente en el hecho de que los objetos a agrupar no intervengan entre ellos en el proceso de reasignación de nuevos centroides, para crear nuevos grupos. Además, los métodos esenciales en la paralelización se definen en dos tareas:

1. Seleccionar un objeto como el centroide de un grupo.
2. Reasignar el objeto a un nuevo centroide más cercano, para crear un grupo.

Por último, destacar la importancia que los autores le brindan al diseño y construcción de la vecindad definida como un sub-espacio de conjunto inicial de soluciones; de ella depende la eficiencia del algoritmo. Una adecuada vecindad permite una promisoria exploración del espacio de soluciones, al mismo tiempo mejora la velocidad del algoritmo dejando de analizar soluciones no prometedoras.

Basadas en población

Las meta-heurísticas basadas en población son aquellas que hacen uso de una población de soluciones, en este caso, la población inicial se genera aleatoriamente y luego se mejora mediante un proceso iterativo. Estas técnicas se definen como métodos orientados a la exploración ya que su principal capacidad está dada en la diversificación en el espacio de búsqueda.

Evolución Diferencial

Analizar las diversas variantes de implementaciones paralelas de la meta-heurística Evolución Diferencial o *Differential Evolution (DE)* (Storn & Price, 1997)en Spark, es la principal contribución en (Teijeiro et al., 2016). Además, se evalúa la viabilidad y desempeño de cada implementación obtenida para su ejecución en la nube. Estas alternativas responden a la necesidad de encontrar una solución a problemas reales, para los cuales la versión secuencial del DE no es aceptable por el gran número de evaluaciones de la función objetivo que realiza. Se estudiaron varios modelos y estrategias de paralelización efectivas y aplicables a este tipo de algoritmo; dentro de las escogidas por los autores para establecer comparaciones experimentales se encuentran los modelos *maestro-esclavo* y *basado en islas* (Alba et al., 2013). La estrategia de paralelización se basa en el caso del modelo *maestro-esclavo,* son paralelizados los ciclos dentro del algoritmo; en el caso del modelo *basado en islas,* es dividida en subpoblaciones la matriz que representa la población inicial y se ejecuta el algoritmo para cada división o islote.

Algoritmo 1. Evolución Diferencial (secED)

entrada: Matriz poblacional *P* con dimensión *D* x *NP*
salida: Matriz P con individuos optimizados
repetir
  Para *cada elemento x de la matriz P* **hacer**
    $\vec{a}, \vec{b}, \vec{c} \leftarrow$ diferentes individuos aleatorios de la matriz P
    Para $k \leftarrow 0$ **hasta** *D* **hacer**
      Si el punto aleatorio es menor que *CR* **entonces**
        $\overrightarrow{I_{nd}}(k) \leftarrow \vec{a}(k) + F(\vec{b}(k) - \vec{c}(k))$
      Fin
    Fin
    Si *evaluación*$(\overrightarrow{I_{nd}})$ *es mejor que evaluación*$(\overrightarrow{P_{(x)}})$
      Remplazar_individuo (P, $\overrightarrow{I_{nd}}$*)*
    Fin
  Fin
hasta

Figura 3. Algoritmo secuencial DE.

A partir de la anterior estrategia asumida en el artículo en análisis, se definen tres versiones del algoritmo para implementarse:

1. El algoritmo secuencial clásico mostrado en la Figura 3, sin utilizar Spark.
2. Tres variantes distintas del algoritmo basado en el modelo *maestro-esclavo*.
3. Una variante del agoritmo basado en el modelo *islas* mostrado en la Figura 4.

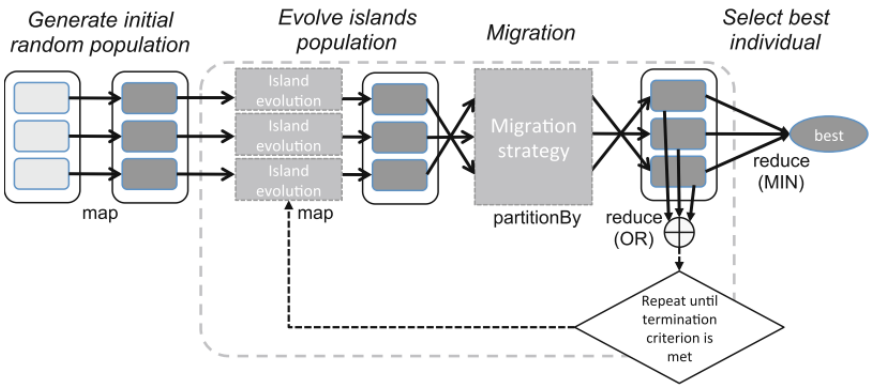

Figura 4. Algoritmo DE basado en *islas*.

Se llegó a la conclusión luego de la fase experimental, que el modelo *islas* es el más adecuado para usar en la implementación en un ambiente distribuido como Apache Spark.

Optimización basada en Enjambre de Partículas

Los procesos biológicos han inspirado a los investigadores a desarrollar algoritmos basados en comportamientos sociales presentes en la naturaleza, principalmente en animales. La meta-heurística Optimización de Enjambre de Partículas o *Particle Swarm Optimization (PSO)* (Kennedy, 2010), ha sido ampliamente estudiada y adaptada a diversos enfoques. Cui (2014), ha defendido la implementación en ambientes distribuidos del PSO paralelo con Apache Spark comparando el desempeño de este algoritmo con respecto al implementado con Hadoop's MapReduce. La versión implementada en Spark es significativamente más rápida.

Por otra parte, Sherar y Zulkernine (2017), presentan los resultados obtenidos de analizar cuan efectivo puede ser la meta-heurística PSO mostrado en la Figura 5, para tareas de agrupamiento tanto en pequeños como grandes conjuntos de datos.

Inicializar una población con un lista de partículas con posiciones y velocidades aleatorias con N_d dimensiones en el espacio de búsqueda.
Mientras *no se cumpla el criterio de parada* **hacer**
  Para cada *partícula en el enjambre* **hacer**
    Evaluar la función de optimización para N_d variables
  Fin
  Si *Actual evaluación es mejor que pbest* **entonces**
    Actualizar la posición del ***pbest***
  Fin
  Buscar la partícula con mejor evaluación en el enjambre, y asignarselo a la variable ***gbest***
  Actualizar la velocidad y la posición de la partícula de acuerdo con las ecuaciones (1) y (2)
Fin

Figura 5. Implementación del algoritmo PSO.

La estrategia utilizada en (Sherar & Zulkernine, 2017) en la implementación del algoritmo híbrido K-means y PSO (KMPSO) se basa en que cada partícula en el enjambre se convierte en el centro de un grupo, representado además como un vector

multidimensional. La evaluación de la calidad de cada partícula, se calcula como la distancia mínima entre la partícula y el grupo más cercano en el enjambre.

Luego en la implementación en Spark mostrada en la Figura 6, los datos a agrupar se almacenan en RDD, el enjambre se inicializa con una muestra de datos seleccionadas aleatoriamente, configurándolos como los centroides de los grupos. El mayor aporte que le imprime Spark al algoritmo de agrupamiento es la velocidad al determinar la calidad de cada partícula en paralelo.

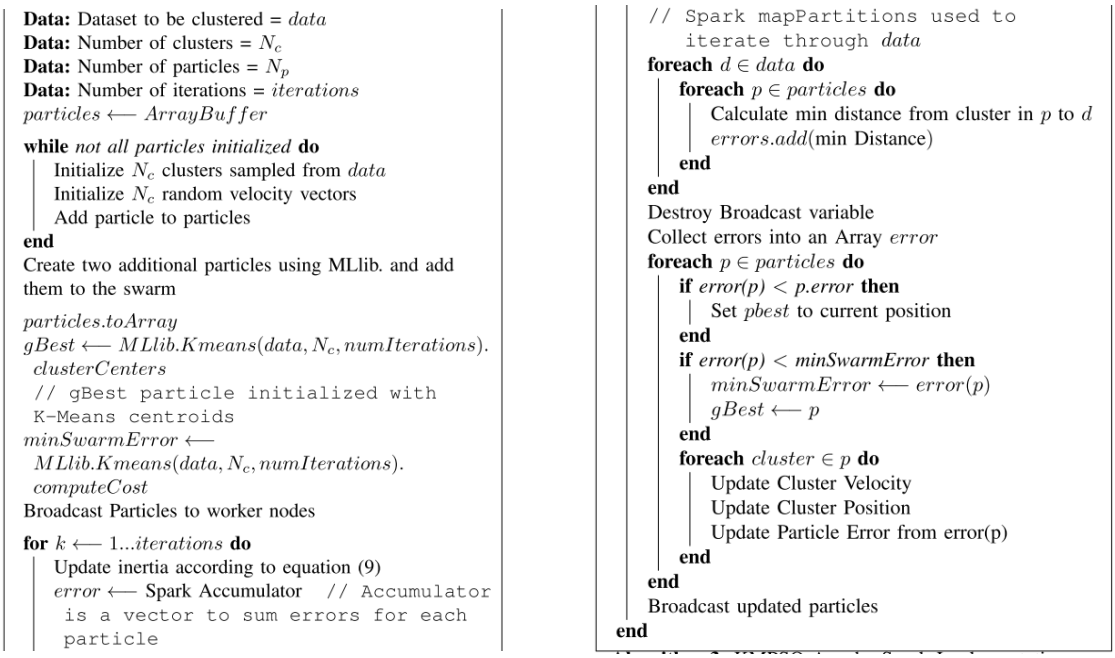

Figura 6. Implementación del algoritmo KMPSO con Apache Spark.

Optimización Colonia de Hormigas

Optimización Colonia de Hormigas o *Ant Colony Optimization (ACO)* (Dorigo & Birattari, 2010) es una meta-heurística basada en población, para dar solución a problemas complejos de optimización combinatoria. Ortiz (2016) centra su investigación en implemantar el algoritmo ACO para tareas de agrupamiento en Spark. La primera estrategia seguida por los autores en el algoritmo se basa en paralelizar las hormigas en cada iteración, siendo esta seleccionada como la mejor variante; mientras que la segunda

paraleliza el proceso de selección de cluster para cada objeto de cada hormiga. En la experimentación del trabajo se define utilizar el algoritmo implementado en Spark K-means para compararlo con la desarrollada, obteniendo mejores resultados la optimizada en la librería MLlib.

En (Gaifang, Xueliang, Honghui, & Pengfei, 2017) los autores presentan el análisis de aplicar el algoritmo ACO como solución al problema clásico de optimización "Viajero Vendedor" en Big Data, basándose en las plataformas MapReduce y en Apache Spark, con el objetivo de mejorar significativamente el tiempo de cómputo en la búsqueda de soluciones. Así mismo incluyen una estrategia de selección del vecino más cercano para elegir la siguiente ciudad y lo combina con el algoritmo genético para mejorar la calidad de la solución. El diagrama de flujo de la implementación de Spark-ACO se muestra en la Figura 7.

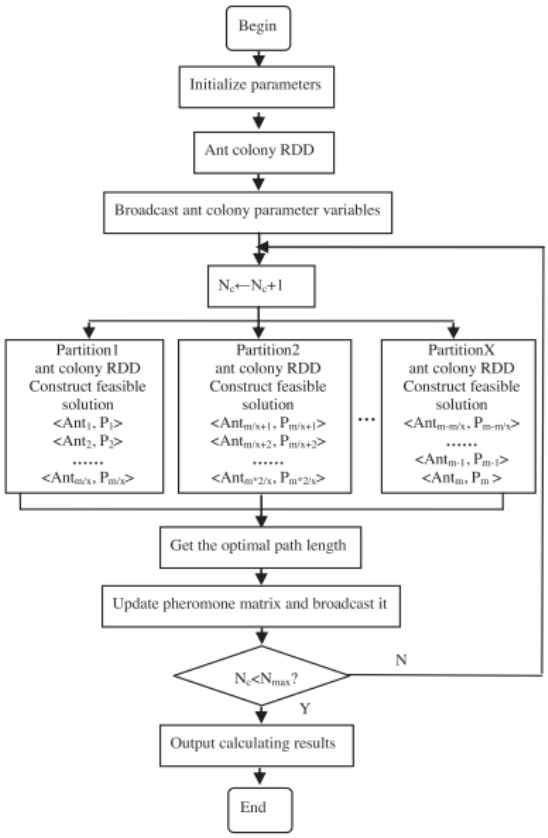

Figura 7. Flujo de Spark-ACO

4. Conclusiones

Este trabajo analizó varias implementaciones de algoritmos meta-heurísticos enfocados en el tratamiento de problemas a gran escala o Big Data, relevantes en la literatura. Además, se adoptó la clasificación de métodos basados en trayectoria y los basados en población. A pesar de la popularidad de las meta-heurísticas aplicadas a problemas complejos de optimización, no se encontraron por parte de los autores de este artículo muchos casos de algoritmos clásicos desarrollados para usarse en plataformas de Big Data. Sin embargo, el estudio mostró como el procedimiento para incorporar este tipo de algoritmos a los ecosistemas de computación distribuida, cada vez son más semejantes. La definición de tereas con mayor costo computacional, así como definir diversas estrategias de paralización evaluando el desempeño del algoritmo en cada caso, resumen varios de los enfoques presentados en las investigaciones. Así como la tendencia a hibridar algoritmos meta-heurísticos y tomar ventaja para encontrar mejores soluciones.

Esta es una línea activa en la comunidad de investigadores, caracterizada por el auge de problemas con grandes dimensiones que necesitan ser analizados. Desarrollar algoritmos clásicos y novedosos constituye una fuente para futuros trabajos.

5. Referencias bibliográficas

Alba, E., Luque, G., & Nesmachnow, S. (2013). Parallel metaheuristics: Recent advances and new trends. *International Transactions in Operational Research*, *20*(1), 1–48. https://doi.org/10.1111/j.1475-3995.2012.00862.x

Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., … Zaharia, M. (2015). Spark SQL : Relational Data Processing in Spark. In S. A. S. I. G. on M. of Data (Ed.), *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (pp. 1383–1394). ACM New York, NY, USA ©2015. https://doi.org/http://dx.doi.org/10.1145/2723372.2742797

Bosagh Zadeh, R., Meng, X., Ulanov, A., Yavuz, B., Pu, L., Venkataraman, S., … Zaharia, M. (2016). Matrix Computations and Optimization in Apache Spark. *ACM*. https://doi.org/http://dx.doi.org/10.1145/2939672.2939675

Cui, L. (2014). *Parallel PSO in Spark*. University of Stavanger.

Dorigo, M., & Birattari, M. (2010). Ant Colony Optimization. In C. Sammut & G. I. Webb (Eds.), *Encyclopedia of Machine Learning* (pp. 36–39). Boston, MA: Springer US.

https://doi.org/10.1007/978-0-387-30164-8_22

Gaifang, D., Xueliang, F., Honghui, L., & Pengfei, X. (2017). Cooperative ant colony-genetic algorithm based on spark. *Computers and Electrical Engineering*, *60*, 66–75. https://doi.org/10.1016/j.compeleceng.2016.09.035

Glover, F. (1989). Tabu Search -Part I. *ORSA Journal on Computing*, *1*(December 2018), 190–206, 4–32,. https://doi.org/10.1287/ijoc.2.1.4

Kennedy, J. (2010). Particle Swarm Optimization. In C. Sammut & G. I. Webb (Eds.), *Encyclopedia of Machine Learning* (pp. 760–766). Boston, MA: Springer US. https://doi.org/10.1007/978-0-387-30164-8_630

Lu, Y., Cao, B., Rego, C., & Glover, F. (2018). A Tabu search based clustering algorithm and its parallel implementation on Spark. *Applied Soft Computing Journal*, *63*, 97–109. https://doi.org/10.1016/j.asoc.2017.11.038

Meng, X., Bradley, J., Street, S., Francisco, S., Sparks, E., Berkeley, U. C., … Talwalkar, A. (2016). MLlib : Machine Learning in Apache Spark. *Journal of Machine Learning Research*, *17*, 1–7.

Nunez, S. G., & Attoh-Okine, N. (2014). Metaheuristics in Big Data : An Approach to Railway Engineering. In *IEEE International Conference on Big Data Metaheuristics*. https://doi.org/978-1-4799-5666-1

Ortiz Martin, A. (2016). *Desarrollo de un algoritmo ant colony optimization para tareas de clustering en apache spark*.

Sherar, M., & Zulkernine, F. (2017). Particle Swarm Optimization for Large-Scale Clustering on Apache Spark. *IEEE*.

Storn, R., & Price, K. (1997). Differential Evolution- A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 1–12. https://doi.org/10.1.1.1.9696

Teijeiro, D., Pardo, X. C., González, P., Banga, J. R., & Doallo, R. (2016). Implementing Parallel Differential Evolution on Spark. *Springer International Publishing Switzerland*, *1*, 75–90. https://doi.org/10.1007/978-3-319-31153-1

Xin, R. S., Gonzalez, J. E., Franklin, M. J., & Stoica, I. (2013). GraphX : A Resilient Distributed Graph System on Spark. In *First International Workshop on Graph Data Management Experiences and Systems (GRADES 2013)*. New York: ACM New York, NY, USA ©2013. https://doi.org/10.1145/2484425.2484427

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., … Stoica, I. (2016, November). Apache Spark : A Unified Engine for Big Data Processing. *Communications of the ACM*, *59*(11), 56–65. https://doi.org/10.1145/2934664