Algoritmos de Inteligencia de Enjambres Orientados a Map Reduce

S. Molina y G. Leguizamón

Laboratorio de Investigación y Desarrollo en Inteligencia Computacional (LIDIC)
Universidad Nacional de San Luis,
Ejército de los Andes 950, (5700) San Luis, Argentina
{smolina,legui}@unsl.edu.ar

Resumen La *Inteligencia de Enjambres* involucra acciones de grupos de individuos descentralizados y auto-organizados, las cuales pueden realizarse en paralelo, particularmente, utilizando *Map-Reduce*, un modelo de programación paralela que permite con facilidad conseguir algoritmos escalables. En este trabajo se propone una breve revisión de algoritmos de *Inteligencia de Enjambres* orientadas a *Map Reduce*, observando especialmente su escalabilidad. Se revisan publicaciones de metaheurísticas clásicas como *Optimización de Colonias de Hormigas* y *Optimización de Enjambre de Partículas*, además de metaheurísticas más recientes como *Búsqueda Cuco* y *Optimización de Enjambre de Luciérnagas*.

1. Introducción

La *Inteligencia de Enjambres* (*Swarm Intelligence, SI*) estudia el comportamiento colectivo de sistemas compuestos por muchos individuos (el *swarm*) interactuando localmente y con su entorno. Ejemplos de *SI* son: *Optimización de Colonias de Hormigas* (*Ant Colony Optimization*, ACO) (Dorigo and Blum [9]); *Optimización de Enjambre de Partículas* (*Particle Swarm Optimization*, PSO) (Eberhart and Kennedy [10]); *Búsqueda Cuco* (*Cuckoo Search*, CS) (Yang et al. [24]) y la *Optimización de Enjambre de Luciérnagas* (*Glowworm Swarm Optimization*, GSO)(Krishnanand and Ghose [13]).

Varias razones hacen a los *SI* aptos para aplicar técnicas paralelas en su solución computacional, por un lado el comportamiento de cualquier *swarm* implica acciones independientes de sus individuos. Por otro lado, tienen mejor desempeño cuando se requiere mucho tiempo de procesamiento, como: cuando el *swarm* es de gran tamaño o cuando los problemas a abordar son de gran escala (por ejemplo, *BigData*) en donde se manipulan espacios de búsqueda grandes y se evalúa la calidad de soluciones de alta dimensionalidad.

Map-Reduce, es un modelo de programación paralela, diseñado originalmente por *Google* para simplificar el procesamiento de datos en paralelo sobre grandes clusters, actualmente también es utilizado para el procesamiento de *BigData* [15]. En este modelo, un usuario especifica la computación a través de dos funciones, *Map* y *Reduce*. La librería *Map-Reduce* paraleliza la computación,

maneja la distribución de datos, el balance de carga, la tolerancia a fallas y la asignación de recursos.

El objetivo de este trabajo es realizar un cuidadoso análisis del estado del arte de los algoritmos *SI*: *ACO*, *PSO*, *CS* y *GSO*. Nos focalizamos en algoritmos propuestos desde el año 2012, poniendo especial interés en cómo cada una de las partes algorítmicas de los *SI* son incluidas en las funciones *Map* y *Reduce*, de manera tal de conseguir los beneficios de este modelo, particularmente la escalabilidad.

El trabajo está organizado de la siguiente manera, en la sección 2 se describe brevemente la arquitectura *Map-Reduce* destacando sus principales características. En la sección 3 se realiza una introducción a las metaheurísticas *SI*, para luego a partir de la sección 4 dedicarnos al análisis de los algoritmos *SI* orientados a *Map-Reduce*. Finalmente en la sección 5 se presentan las conclusiones.

2. Architectura Map-Reduce

Map-Reduce es un modelo de programación paralelo (McNabb et al.[18]). La infraestructura provista por su implementación, maneja los detalles de comunicación, balance de carga, tolerancia de fallas, asignación de recursos, inicialización de tareas y distribución de archivos.

Un programa basado en este modelo consiste de una función *Map* y una función *Reduce* las cuales procesan entradas de la forma <*clave,valor*>.

Las operaciones *Map-Reduce* se dividen en dos etapas. En la primera etapa la función *Map* es invocada para cada una de las entradas, en cada invocación puede producir cualquier número de salidas. En la segunda etapa estas salidas son ordenadas y agrupadas según el valor *clave* y una función *Reduce* es invocada por cada *clave*. La función *Reduce* produce una lista de valores de salidas asociados a una *clave* determinada.

Todas las operaciones *Map* pueden realizarse en paralelo. La operación *Reduce* puede comenzar cuando todas las operaciones *Map* se completan. Las operaciones *Reduce* también pueden ser realizadas en paralelo.

Un programa más complejo puede consistir de múltiples etapas *Map-Reduce*, conformando un programa *Map-Reduce Iterativo*. En este caso, la salida de una función *Reduce* es la entrada de una función *Map* de la próxima iteración.

Existen varias implementaciones *Map-Reduce* (Lee et al. [15]) tales como: la original de *Google*, su contraparte *Open Source Hadoop* (Apache [7]) con *HaLoop* para *Map-Reduce Iterativo*); *Mars* (Bingsheng et al. [12]) para procesadores gráficos, Phoenix (Ranger et al. [20]) para sistemas de memoria compartida y *Disco*(Aljarah and Ludwig [3]) para el manejo de grandes vólumenes de datos y cálculos científicos.

3. Inteligencia de Enjambres

La *Inteligencia de Enjambres* estudia el comportamiento colectivo de sistemas compuestos de muchos individuos (*swarm*) interactuando localmente y con

su entorno. Los *swarms* inherentemente usan formas de control descentralizadas y auto-organización para alcanzar sus objetivos (Martens et al. [17]). En este trabajo revisamos los algoritmos orientados a *Map-Reduce* de las metaheurísticas *ACO*, *PSO*, *CS* y *GSO*. A continuación mostramos sus características:

- *ACO* (Dorigo and Blum [9])
  Se inspira en el comportamiento de una especie de hormigas en busca de comida. Éstas, inicialmente exploran aleatoriamente el área cercana al nido, luego se guían por los niveles de $feromona$ que depositan en el entorno, cuando llevan comida al nido.
  Un algoritmo *ACO* básicamente consta de tres procedimientos: *Construcción de Soluciones* (simula una colonia de hormigas, las cuales concurrente y asincrónicamente construyen una solución mediante movimientos a través del grafo de construcción, representante del espacio de búsqueda de la solución), *Modificación de Feromona* (deposita o evapora feromona, modifica la matriz de feromonas τ) y *Acciones Demonios* (son acciones centralizadas que no pueden ser realizadas por una única hormiga).
- *PSO* (Eberhart and Kennedy [10])
  El *PSO* imita la interacción social entre individuos (o *partículas*), tales como la interacción de los pájaros durante el vuelo en búsqueda de comida. En el mismo, un número de *partículas* son colocadas en el espacio de búsqueda. Cada *partícula* mantiene su mejor posición (*Local Best*) según el valor de fitness de una función objetivo dada. Luego, cada *partícula* determina su movimiento (o *velocidad*), teniendo en cuenta la historia de su posición actual y la mejor posición alcanzada por el *swarm* y otras constantes. A su vez, el *swarm* se mueve a lo largo de las iteraciones del algoritmo cerca de la *Mejor Solución Global* (*Global Best*).
- *CS* (Yang et al. [24])
  Se basa en el comportamiento parásito de algunas especies de pájaros cucos al depositar sus huevos en nidos ajenos. Si el dueño del nido descubre que el huevo no es suyo, lo tira o abandona el nido construyendo un nido en otro lugar. Si el huevo del cuco eclosiona el ciclo se repite.
  El algoritmo *CS* comienza con un conjunto de nidos con un huevo cada uno. Los mejores huevos pasarán a la próxima generación. Los peores huevos tendrán una cierta probabilidad de ser descubiertos y no sobrevivirán. Los nidos cuyos huevos han sido descubiertos se sustituirán por nuevos nidos.
- *Optimización de Enjambre de Luciérnagas* (Krishnanand and Ghose [13])
  Se basa en el comportamiento de los enjambres de luciérnagas. Cada luciérnaga tiene una posición dentro del espacio de búsqueda, un *Nivel de luminosidad* asociado con el valor objetivo de su posición y un rango de decisión local. La luciérnaga que emite más luminosidad está más cerca de una posición de interés.
  A lo largo del proceso, las luciérnagas se mueven hacia otras con mayor luminosidad dentro de su rango de decisión local. Finalmente, muchas de ellas quedarán en los picos del espacio de búsqueda.

4. Algoritmos SI orientados a Map-Reduce

La propuesta de este trabajo, surge a partir de la observación de que se han realizado varios trabajos abocados al estudio del estado del arte de metaheurísticas paralelas en general y sus tendencias (Alba et al. [2]), otros más abocados a *SI* específicas (Krömer et al. [14]) y para un tipo de arquitectura particular (Molina et al. [19] para $GPUs$), pero en ninguno de ellos hacen referencia al uso de *Map-Reduce*.

Recientemente, Gong et al. [11] han presentado un estado del arte de modelos y algoritmos evolutivos distribuidos, además de los *SI*: *ACO* y *PSO*, en donde mencionan que implementaciones *Cloud*, *Map-Reduce* y $GPUs$ son temas de interés actual para investigar.

En el caso de *Map-Reduce*, no existen trabajos con la idea que proponemos por lo que creemos que es una interesante aportación.

4.1. Revisión de Publicaciones Relacionadas a *SI* sobre *Map-Reduce*

Los algoritmos *SI* sobre *Map-Reduce* buscan mejorar su eficiencia afectada por, incluir técnicas que mejoran aspectos de éstos pero hacen que aumente su tiempo de ejecución (t_e), o para abordar problemas con espacios de búsquedas muy grandes.

Como mencionamos anteriormente, aquí ponemos especial interés en cómo las partes algorítmicas de los *SI* son incluidas en las funciones *Map* y *Reduce*, para lograr algoritmos escalables.

Distinguimos tres tipos de *Escalabilidad*: A, B y C. La escalabilidad tipo A y B son incluidas en Gong et al. [11] como *Escalabilidad de Tamaño* y *Escalabilidad de Tarea* respectivamente. La escalabilidad tipo C o *Escalabilidad de parámetros* es incluida por nosotros para agregar claridad a nuestra revisión.

La escalabilidad tipo A muestra la habilidad de los algoritmos de mejorar su desempeño en proporción al aumento del número de procesadores. La escalabilidad tipo B muestra la habilidad de los algoritmos para adaptarse a los cambios en la escala del problema, manteniendo su eficiencia cuando se incrementan las dimensiones del mismo. La escalabilidad tipo C muestra la habilidad de los algoritmos de mantener su eficiencia al aumentar la cantidad de procesamiento por un aumento en el valor de algún parámetro del mismo.

A continuación describimos cada una de las publicaciones revisadas. En la mayoría de los casos, se utiliza el framework *Hadoop*. En los casos excepcionales se hacen los comentarios pertinentes. El cuadro 1 resume las características de los trabajos revisados, su nomenclatura se describe a continuación. m_1: *Un individuo construye una solución*, m_2: *Una colonia construye una solución*, m_3: *Un individuo construye una solución parcial*, m_4: *Cálc. de centroides y distancia entre éstos y los datos*, m_5: *Definición de un subespacio de búsqueda*, m_6: *Gen. de huevos*, m_7: *Cálc. de centroides y distancia entre éstos y un conjunto parcial de datos*, r_1: *Sel. del Global Best y actualización de* τ, r_2: *Unificación de matrices* τ *parciales y ejecución de ACO*, r_3: *Cálc. de valores de fitness*, r_4: *CS*, r_5: *Sel. de huevos*, r_6: *Cálc. de centroides y distancia entre éstos y el conjunto*

SI	Modelo	Algoritmo	Problema	Map	Reduce	Año	Ref.
ACO	*Estándar*	*[6]-ACO*	*TSP*	m_1	r_1	2014	[6]
	Multicolonia	*[8]-ACO*	*MST*	m_2	r_1	2014	[8]
	Isla Dinámico	*DIIMR-ACO*	*TSP*	m_1	r_1	2013	[22]
	ACO Independientes	*[21]-Indep*	*TSP, PM*	m_1	r_1	2012	[21]
	Espacio Particionado	*[21]-Part*	*TSP, PM*	m_3	r_2	2012	[21]
PSO	*Estándar*	*IDS-MRCPSO*	*Clustering*	m_4	r_3	2013	[5]
	Estándar	*MR-CPCO*	*Clustering*	m_4	r_3	2012	[3]
CS	*Espacio Particionado*	*[23]-CS*	*Benchmark*	m_5	r_4	2014	[23]
	Estándar	*MRMCS*	*Benchmark, Ingeniería*	m_6	r_5	2013	[16]
GSO	*Espacio Particionado*	*MRCGSO*	*Clustering*	m_7	r_6	2014	[1]
	Estándar	*[4]-GSO*	*Benchmark*	m_4	r_7	2013	[4]

Cuadro 1. Características de SI revisados.

completo de datos, r_7: *Cálc. de luminosidad*, MST: *Minimum Spanning Tree*, TSP: *Traveling Salesman Problem* y PM: *Problema de la Mochila*.

Metaheurística ACO sobre Map-Reduce

Se han propuesto algoritmos ACO basados en modelos: con una única colonia de hormigas y varias colonias de hormigas independientes; con varias colonias de hormigas que interactúan entre sí como el modelo *Multicolonia de Hormigas* y el modelo *Isla Dinámico* en donde los niveles de colaboración y de competencia entre colonias se gradúa a lo largo del tiempo.

Una única publicación, propone particionar el espacio de búsqueda para distribuir la carga de trabajo entre las hormigas (Wu et al. [21]).

La mayoría de las publicaciones revisadas proponen algoritmos para el TSP (Wu et al. [21], Anuraj and Remya [6] y Cheng and Xiao [22]), Wu et al. [21] también estudian el *Problema de la Mochila* y Bhavani and Sudha [8] estudian un problema de clustering de genes.

- En Anuraj and Remya [6] proponen un *ACO Map-Reduce Iterativo*. Cada Map calcula una solución, la función *Reduce* realiza la modificación de τ y calcula la mejor ruta de cada iteración. La etapa *Reduce* final retorna la mejor solución global. Para mejorar la versión paralela, múltiples hormigas se implementan en un mismo Map.
  El algoritmo es escalable. Se visualiza la escalabilidad tipo A, B y C a través de la observación del t_e. Para la escalabilidad tipo B, se observa que el algoritmo es eficiente para un número grande de ciudades. Para la escalabilidad tipo A, se prueba el algoritmo variando el número de nodos de distintos clusters. Para la escalabilidad tipo C, se evaluan los t_e variando el número de: hormigas por Map, de Map por etapas *Map-Reduce* y etapas *Map-Reduce*; en este caso el incremento del número de hormigas no afecta al t_e pero sí lo hace el incremento del número de Map y etapas *Map-Reduce*.
- Bhavani and Sudha [8] proponen un algoritmo *Multicolonia de Hormigas*. El mismo construye el árbol de expansión mínimo (MST, *Minimum Spanning*

Tree) desde el dato de expresión del gen, utilizando *Map-Reduce Iterativo* y luego realiza el proceso de clustering, utilizando el procedimiento *K-medias*. Cada *Map* es una colonia que construye un MST. En cada generación, cada colonia intercambia información referente a su solución. La función *Reduce* encuentra la mejor solución hasta un momento dado y actualiza τ. Finalmente, se calcula el valor de *Threshold* utilizando los pesos de los arcos del MST mínimo, los arcos que son mayores a este valor son quitados obteniendo de esta manera el clustering.
Se estudia la escalabilidad tipo A y B para diferentes tamaños de archivos de genes y números de procesadores. Utilizan hasta 4 procesadores y archivos de entrada de hasta 17.8 MB. El algorimo tiene buena escalabilidad.

- Cheng and Xiao en [22] proponen el algoritmo *DIIMR-ACO*, basado en un modelo *Isla* dinámico e iterativo. Para disminuir el overhead causado por incluir una técnica de feedback dinámico, utilizan un *Map-Reduce Iterativo* con la plataforma *HaLoop*.
  Las hormigas son divididas en varias colonias, cada hormiga elige el próximo paso según los valores de feromona de las colonias *Inter* e *Inner*.
  La función *Map* construye un tour, calcula los niveles de feromona para la matriz de la colonia *Inter* e *Inher*, calcula la probabilidad de transición y elige el próximo arco. Finalmente realiza el depósito de feromona según el tour construido. La función *Reduce* modifica los valores de feromona respecto a una hormiga de una determinada colonia y modifica la τ global.
  Estudian la escalabilidad tipo B para diferentes instancias. Para las instancias más grandes (en este caso se utilizan instancias de 198, 318, 442 y 532 ciudades), el algoritmo *DIIMR-ACO* tiene mejor desempeño que un algoritmo *ACO* secuencial y un algoritmo $MMAS$ paralelo.
- Wu et al. [21] implementan dos versiones *ACO*.
  En la primera versión, cada *Map* ejecuta un algoritmo ACO independiente que calcula una única solución. Cada *Reduce*, selecciona el valor óptimo.
  En la segunda versión, el espacio de búsqueda es particionado en el número de tareas *Map*. Para el problema de la mochila, los items y τ se particionan y se envían a cada tarea *Map*. Cada *Map* calcula las soluciones parciales y realiza las modificaciones parciales de niveles de feromonas. En las tareas *Reduce* las matrices parciales son unificadas y el algoritmo *ACO* es ejecutado sobre todos los ítems considerando la τ optimizada. Para el TSP se replican las entradas para cada *Map* y se le asignan diferentes ramas del árbol de búsqueda, la tarea *Reduce* produce la ruta óptima.
  Los algoritmos son escalables alcanzando valores cercanos al *Speedup Lineal*.
  Se estudia la escalabilidad tipo A utilizando hasta 16 nodos.

Metaheurística PSO sobre Map-Reduce

- Aljarah and Ludwig [3], proponen el algoritmo *MR-CPSO* para clustering, el cual en cada generación aplica dos tareas *Map-Reduce*. En la primera tarea, se modifican las partículas centroides del *swarm* y en la segunda tarea se evalúan sus fitness. Luego, se actualizan los valores *Local Best* y *Global Best*.

En la primera función *Map* se modifican los centroides, la función *Reduce* los ordena y los combina en un archivo de salida.
La segunda función *Map* calcula la distancia entre un registro (dato) y los centroides, luego emite a la función *Reduce* una clave compuesta armada con el identificador del centroide con distancia mínima y dicha distancia. La función *Reduce* calcula el promedio de los valores con igual clave y lo asigna como el nuevo valor de fitness de cada centroide en cada partícula. Luego, todos los valores de fitness son guardados en el *Sistema de Archivo Distribuido*.
Se estudia la escalabilidad tipo A y B, utilizan las métricas *Scaleup* y *Speedup*. Para el *Scaleup* varía el tamaño del conjunto de datos y el número de nodos. Para el *Speedup* se fija el conjunto de datos y varía el número de nodos. El algoritmo escala muy bien alcanza valores de *speedup* cercanos al *Speedup Lineal*.

- Aljarah and Ludwig [5] presentan el algoritmo *IDS-MRCPSO*, un *Sistema de Detección de Intrusión*. Este algoritmo, utiliza el algoritmo *MR-CPSO* descripto en el punto anterior para generar centroides óptimos para los datos de entrenamiento.
  Se evalúa la escalabilidad tipo A utilizando la métrica *Speedup* y el t_e. El algoritmo muestra una escalabilidad razonable para distintos número de nodos (en este caso desde 2 hasta 16). Para el conjunto de datos más grande se observan valores de *speedup* cercanos al *Speedup Lineal*.

Metaheurística CS sobre Map-Reduce

- Xu et al. [23] proponen un algoritmo que combina la técnica *Dividir* y *Conquistar* y *Map-Reduce* con el *CS* secuencial, para funciones benchmarks.
  Primero el dominio de búsqueda es transformado en una región de 2 dimensiones y es dividido en n^2 subrectángulos no sobrelapados. Luego, un procedimiento *Map* para cada subrectángulo, arma un par <*clave,valor*>, con el identificador del subrectángulo como *clave* y como *valor* la posición de su esquina inferior izquierda. Cada par se emite R veces a una etapa *Reduce*. La etapa *Reduce*, ejecuta R veces un algoritmo *CS* estándar para un mismo subrectángulo, evitando así caer en soluciones óptimas locales. De esta manera, el tiempo de comunicación de la red disminuye al aprovechar la localidad del dato. El mejor resultado es la salida. Finalmente, se comparan los mejores resultados para cada subrectángulo y la salida será la mejor solución global.
  Se compara un algoritmo *CS* secuencial respecto al *CS Map-Reduce* ejecutado sobre un cluster con dos nodos de 24 cores cada uno. El estudio no está focalizado en el estudio de escalabilidad, aunque se visualiza la existencia de una escalabilidad tipo A, ya que consiguen un menor t_e usando *Map-Reduce* y un mayor número de nodos. Destacan que esto se debe principalmente a que *Hadoop* propaga todas las tareas de computación a través de los cores de CPU y las reordena dinámicamente.

- Lin et al. [16] proponen el algoritmo *MRMCS* sobre un *Map-Reduce Iterativo* y se aplica a dos problemas de ingeniería y funciones benchmarks.
  Las funciones *Map* se encargan de la generación de un nuevo huevo y la función *Reduce* realiza la selección de los mejores huevos para la próxima generación.
  Estudian la escalabilidad tipo A. El t_e disminuye a medida que aumenta el número de procesadores (en este caso de 1 a 8) mostrando así su eficiencia.

Metaheurística GSO sobre Map-Reduce

- Al-Madi et al. [1] presentan el agoritmo *MRCGSO* para tareas de *clustering*, para encontrar múltiples centroides. Este algoritmo se basa en el algoritmo *CGSO*, en el mismo, cada luciérnaga compite por ser un centroide y trata de cubrir la mayor cantidad de registros de datos. El *MRCGSO* primero a cada luciérnaga le asigna un valor de posición inicial.
  El conjunto de datos es particionado según el número de tareas *Map* a utilizar. La función *Map* calcula para cada luciérnaga el número de registro de datos que son cubiertos por ella y la suma total de las distancias entre ella y cada dato cubierto. La función *Reduce* realiza las mismas operaciones con los resultados de todas las tareas *Map*.
  Se estudia la escalabilidad tipo B. Los resultados revelan que el algoritmo escala muy bien y alcanzan valores de *speedup* cercanos al *Speedup Lineal*.
- Aljarah and Ludwig [4] presentan el algoritmo *MR-GSO* aplicado a funciones multimodales de gran escala con diferentes dimensiones. El *MR-GSO* consiste de las fases: *Inicialización* y *Map-Reduce Iterativo*.
  En la fase *Inicialización* se crea un grupo de luciérnagas con su posición; para cada una se evalúa la función objetivo y se calcula el nivel de luminosidad. Luego, todo el grupo es guardado en un archivo.
  En la fase *Map-Reduce Iterativo*. Cada tarea *Map-Reduce* representa una iteración del algoritmo estándar *GSO*. En cada tarea el algoritmo se focaliza sobre las etapas que consumen tiempo: modificación del nivel de luminosidad, modificación de la posición de las luciérnagas, búsqueda del grupo de luciérnagas vecinas.
  En la tarea *Map* se realiza la búsqueda del grupo de luciérnagas vecinas con mayor luminosidad para cada luciérnaga del *swarm*. En la función *Reduce* se modifican los niveles de luminosidad.
  Estudian la escalabilidad tipo A y C. La métrica utilizada es el *Speedup Paralelo*. El algoritmo es escalable cuando son optimizadas funciones polimodales dificultosas y se utilizan *swarms* grandes.

5. Conclusiones

En este trabajo se logra una breve pero detallada revisión de algoritmos *SI* orientados a *Map-Reduce*, considerando su *escalabilidad*.

El estudio revela que, los *SI* abordados son escalables y en muchos casos alcanzan valores cercanos al *Speedup Lineal*. Inclusive, en los casos que se utiliza

Map-Reduce Iterativo, el cual ha mostrado tener menor desempeño que el *Map-Reduce* de una única iteración, según se menciona en la literatura. Aunque, con el uso de *HaLoop* se espera que los algoritmos tengan mejor desempeño por las técnicas que el mismo utiliza para el manejo de las iteraciones.

Los algoritmos muestran un mejor desempeño para instancias más grandes de problemas y con pocas comunicaciones entre sus nodos. Esto se debe a que se logra superponer las computaciones con las comunicaciones disminuyeno así el overhead causado por las mismas.

En general, las funciones *Map* realizan las acciones independientes de los *SI*, como las realizadas por lo individuos del *swarm*, las funciones *Reduce* se encargan de las acciones centralizadas.

Si bien, el desarrollador no se esfuerza en los aspectos mencionados en la sección 2, debe distribuir cuidadosamente las acciones de un *SI* en las funciones *Map* y *Reduce* para evitar el exceso de escrituras en archivos además de un incremento del overhead de comunicaciones.

Finalmente, se vislumbra la posibilidad de extender la aplicabilidad de las *SI* estudiadas a *Big Data*, a problemas con espacios de búsqueda más grandes. En esta dirección de estudio es interesante además, considerar la experimentación en plataformas *Cloud Computing* lo que mostraría la versatilidad de las *SI*. Modelos *SI* multicolonias con diferentes técnicas de particionado del espacio de búsqueda, muestran ser alternativas de estudio prometedoras.

Referencias

1. N. Al-Madi, I. Aljarah, and S.A. Ludwig. Parallel glowworm swarm optimization clustering algorithm based on mapreduce. In *Swarm Intelligence (SIS), 2014 IEEE Symposium on*, pages 1–8, Dec 2014.
2. E. Alba, G. Luque, and S. Nesmachnow. Parallel Metaheuristics: Recent Advances and New Trends. *International Transactions in Operational Research*, 20(1):1–48, 2013.
3. I. Aljarah and S.A. Ludwig. Parallel Particle Swarm Optimization Clustering Algorithm Based on MapReduce Methodology. In *Nature and Biologically Inspired Computing (NaBIC), 2012 Fourth World Congress on*, pages 104–111, Nov 2012.
4. I. Aljarah and S.A. Ludwig. A MapReduce Based Glowworm Swarm Optimization Approach for Multimodal Functions. In *Swarm Intelligence (SIS), 2013 IEEE Symposium on*, pages 22–31, April 2013.
5. I. Aljarah and S.A. Ludwig. MapReduce Intrusion Detection System Based on a Particle Swarm Optimization Clustering Algorithm. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 955–962, June 2013.
6. M. Anuraj and Remya G. Article: A Parallel Implementation of Ant Colony Optimization for TSP Based on MapReduce Framework. *International Journal of Computer Applications*, 88(8):9–12, February 2014. Published by Foundation of Computer Science, New York, USA.
7. Apache. Documentación de Hadoop de Apache, 2014.
8. R. Bhavani and S. G. Sudha. A Novel Ant Based Clustering of Gene Expression Data Using MapReduce Framework. In *International Journal on Recent and Innovation Trends in Computing and Communication*, volume 2, pages 398–402.

International Journal on Recent and Innovation Trends in Computing and Communication, 2014.
9. M. Dorigo and C. Blum. Ant colony Optimization Theory: A Survey. *Theoretical Computer Science*, 344(2–3):243–278, 2005.
10. R. Eberhart and J. Kennedy. A New Optimizer Using Particle Swarm Theory. In *Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on*, pages 39–43, Oct 1995.
11. Y. Gong, W. Chen, Z. Zhan, J. Zhang, Y. Li, Q. Zhang, and J. Li. Distributed Evolutionary Algorithms and their Models: A Survey of the State-of-the-art. *Applied Soft Computing*, 34:286–300, September 2015.
12. B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang. Mars: A MapReduce Framework on Graphics Processors. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 260–269, New York, NY, USA, 2008. ACM.
13. k. N. Krishnanand and D. Ghose. Detection of Multiple Source Locations Using a Glowworm Metaphor with Applications to Collective Robotics. In *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*, pages 84–91, June 2005.
14. P. Krömer, J. Platoš, and V. Snášel. Nature-Inspired Meta-Heuristics on Modern GPUs: State of the Art and Brief Survey of Selected Algorithms. *International Journal of Parallel Programming*, 42(5):681–709, 2014.
15. K. Lee, Y. Lee, H. Choi, Y. D. Chung, and B. Moon. Parallel Data Processing with MapReduce: A Survey. *SIGMOD Rec.*, 40(4):11–20, jan 2012.
16. C. Lin, Y. Pai, K. Tsai, C. H. P. Wen, , and L. Wang. Parallelizing Modified Cuckoo Search on MapReduce Architecture. *Journal of Electronic Science and Technology*, 11(2):115–123, June 2013.
17. D. Martens, T. Fawcett, and B. Baesens. Editorial Survey: Swarm Intelligence for Data Mining. *Machine Learning*, 82(1):1–42, January 2011.
18. A. W. McNabb, J. Lund, and K. D. Seppi. Mrs: MapReduce for Scientific Computing in Python. In *SC Companion*, pages 600–608. IEEE Computer Society, 2012.
19. S. Molina, F. Piccoli, and G. Leguizamón. Algoritmos de inteligencia de enjambres sobre gpu:una revisión exhaustiva. In *Congreso Argentino en Ciencias de la Computación*, 2014.
20. C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Evaluating MapReduce for Multi-core and Multiprocessor Systems. In *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, HPCA '07, pages 13–24, Washington, DC, USA, 2007. IEEE Computer Society.
21. B. Wu, G. Wu, and M. Yang. A MapReduce Based Ant Colony Optimization Approach to Combinatorial Optimization Problems. In *Natural Computation (ICNC), 2012 Eighth International Conference on*, pages 728–732, May 2012.
22. Ch. Xingguo and X. Nanfeng. Parallel Dynamic Island ACO Based on Iterative Map-Reduce Model. *JCIS: Journal of Communications and Information Sciences*, 3(3):139–147, 2013.
23. X. Xu, Z. Ji, F. Yuan, and X. Liu. A Novel Parallel Approach of Cuckoo Search Using MapReduce. *International Conference on Computer, Communications and Information Technology (CCIT 2014)*, pages 114–117, 2014.
24. X. Yang and S. Deb. Cuckoo Search Via Lévy Flights. In *Nature Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pages 210–214, Dec 2009.