Ciencia Digital

ISSN: 2602-8085
Vol. 1, N°3, p. 40-52, octubre - diciembre, 2017

ESTRATEGIAS PARA ACELERAR Y REFINAR EL ALGORITMO GENETICMINER.

Ing. Asiel Díaz Vasallo.[1], MsC. Orlenys López Pintado. [2], Dr.C Yasser Vázquez Alfonso [3] & M.Sc Efraín Velasteguí López.[4]

ABSTRACT

The techniques of process mining allow the extraction of knowledge from the event records. The discovery of models is one of the stages that make up process mining. In this stage a series of algorithms are used that initially start from an event log and generate different process models The present work focuses on the Genetic Miner algorithm, which is one of the best results in the aforementioned discovery phase. Despite the accuracy of the model obtained, the algorithm has a high temporal cost. In order to reduce the response times, in this work a distributed version of the Genetic Miner is developed, making use of the remote invocation infrastructure .Net Remoting. We compare the results obtained by the original version of the algorithm, incorporated in the ProM tool, a variant developed in the framework of the CITI-UNAH project, which extends the causal matrix model with the addition of an index matrix and the alternative proposed in this document. These tests evaluated the accuracy and response times of the algorithms.

Keywords: Environmental Partner Perceptions, Siapsadt 1.0, Computer System, Earth Slips.

CÓDIGO UNESCO: Sistemas de Información 120318

RESUMEN

Las técnicas de la minería de procesos permiten la extracción de conocimientos de los registros de eventos. El descubrimiento de modelos es una de las etapas que componen la minería de procesos. En esta etapa se utilizan una serie de algoritmos que parten inicialmente de un registro de eventos y generan diferentes modelos de procesos. El presente trabajo se centra en el algoritmo *Genetic Miner*, que es uno de los que ofrecen mejores resultados en la mencionada fase de descubrimiento. A pesar de la precisión del modelo que se obtiene, el algoritmo posee elevado coste temporal. Con vistas a reducir los tiempos de respuesta, en este trabajo se desarrolla

[1] Universidad Agraria de la Habana, La Haba, Cuba, asieldv@unah.edu.cu
[2] Universidad de la Habana, Cuba, orlenyslp@gmail.com
[3] Universidad de la Habana, Cuba, yalfos1@gmail.com
[4] Universidad Técnica de Ambato, Ambato, Ecuador, le.velastegui@uta.edu.ec

ISSN: 2602-8085
Vol. 1, N°3, p. 40-52, octubre - diciembre, 2017

una versión distribuida del *Genetic Miner*, haciendo uso de la infraestructura de invocación remota *.Net Remoting*. Se comparan los resultados obtenidos por la versión original del algoritmo, incorporada en la herramienta ProM, una variante desarrollada en el marco del proyecto CITI-UNAH, que extiende el modelo de matriz causal con la adición de una matriz de índices y la alternativa propuesta en este documento. Estas pruebas evaluaron precisión y tiempos de respuesta de los algoritmos.

Palabras Claves: percepciones socio ambiental, SIAPSADT 1.0, sistema informático, Deslizamientos de tierra,

INTRODUCCIÓN

Los tiempos modernos con sus avances científicos y tecnológicos introducen nuevos retos. Extraer conocimiento procesando grandes volúmenes de datos es unodeestos.Actualmente, diversasinstitucionesgestionanlaejecucióndesus procesos de negocios mediante complejos sistemas computacionales. Muchos de estos sistemas almacenan los datos relacionados con los procesos ejecutados en grandes registros de eventos, cuyo procesamiento manual se torna una tarea imposible de realizar. En este sentido, laminería de procesos, una rama relativamente nueva en la ciencia de la computación, centra sus esfuerzos en la obtención automatizada de información almacenada en ficheros estructurados según diversos estándares. En este trabajo, se aplicará laminería de procesos para la reconstrucción de modelos a partir de un conjunto de ejecuciones reales. Con este fin, se centrará en la obtención de modelos de procesos utilizando el algoritmo *Genetic Miner*, combinado con técnicas de programación paralela y distribuida.

Funcionamiento general del algoritmo *Genetic Miner*

El algoritmo *Genetic Miner* toma como entrada un registro de eventos (15; 19 y simula el proceso de evolución de los individuos (19), que son representados por una matriz causal. A partir de la entrada se genera una población inicial formada por individuos, cada uno de los cuales representa un modelo diferente del proceso que se quiere recuperar.

Para cada individuo se calcula su medida de adaptabilidad (*fitness*) que evalúa cuán bien estos reflejan el comportamiento del proceso almacenado en el registro de eventos (15; 19). Una vez obtenido el *fitness*, si se cumplen ciertas condiciones de para da, el algoritmo termina devolviendo a la matriz causal más adaptada de la población. En caso de no cumplirse ninguna de las condiciones esperadas se construirá entonces una nueva generación de modelos, con la esperanza de que algunos de estos estén mejores adaptados que sus antecesores.

Para la construcción de la siguiente generación, los mejores individuos son ascendidos directamente: proceso conocido como elitismo. A través de torneos, se selección los padres que generarán nuevos modelos al aplicárseles dos operadores genéticos (11; 12; 13): cruzamiento y mutación. Para el cruzamiento, se toman dos padres que forman dos descendientes desconocidos, como resultado de la combinación de sus características. A estos individuos se les aplica aleatoriamente el operador mutación, que agrega nuevos rasgos (no presentes en sus padres). Una vez completada la próxima población se calcula el *fitness* de cada matriz causal; repitiéndose el proceso antes descrito.
Alcanzarlas condiciones de parada establecidas para el algoritmo puede requerir un gran número de iteraciones (construcción de generaciones). Esto provoca que el algoritmo *GeneticMiner*, como todos los procesos evolutivos, presente como principal desventaja no ser eficiente computacionalmente, por necesitar de un gran volumen de cálculos para obtener el resultado final.

Solución Propuesta

Para este algoritmo, el cómputo de adaptabilidad de los individuos es el proceso más costoso en cuanto a tiempo de ejecución. Este, junto con la obtención de las nuevas generaciones, puede efectuarse de modo independiente. Dicha característica permite el uso de técnicas de programación paralela y distribuida con el propósito de disminuir el tiempo de respuesta que presenta el *GeneticMiner*.

En aras de conformar la nueva variante del algoritmo se utilizó una estrategia híbrida (Figura 2), constituida por una jerarquía de dos niveles. En el nivel superior los nodos evolucionan de manera independiente las poblaciones en sus espacios de memoria, donde utilizan técnicas de programación paralela para calcular la adaptabilidad de los individuos. Un coordinador central recibe cada cierto número de iteraciones las poblaciones construidas por cada nodo para formar una general, de esta obtiene una siguiente generación y selecciona los individuos de mayor adaptabilidad según el índice de elitismo establecido. Estos serán insertados en todas las poblaciones que radican en los nodos, reemplazando a los menos adaptados.

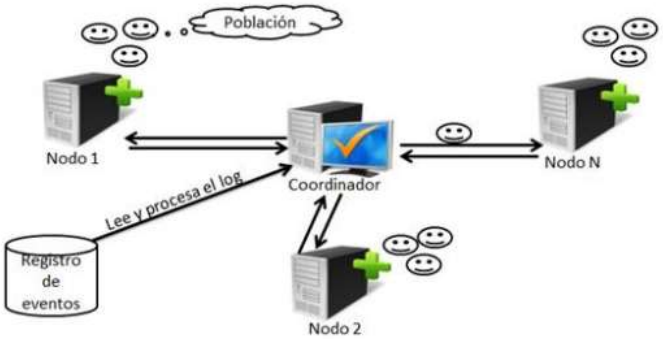


Figura 2: Estrategia de distribución seleccionada.

ISSN: 2602-8085
Vol. 1, N°3, p. 40-52, octubre - diciembre, 2017

En el nivel inferior cada nodo ejecuta un proceso principal que controla la evolución de las poblaciones. De manera simultánea se aplica paralelismo de datos sobre los individuos para obtenerla adaptabilidad de cada uno, devolviendo al hilo principal los resultados alcanzados. Este enfoque brinda la posibilidad de explorar distintas partes del espacio de búsqueda.

Paralelismo en *.Net*

Con el propósito de llevar a cabo la interacción entre las aplicaciones se empleó *.Net*, haciendo uso del protocolo de control de transmisión (TCP). La elección de este protocolo se debe a que presenta la codificación binaria como predeterminada, cuestión que lo convierte en un modo eficaz de intercambiar datos entre objetos remotos. Por otra parte, los datos binarios ocupan menos espacio que los protocolos orientados a conexión XML, transmitidos mediante un canal HTTP orientado a conexión. Con el fin de proporcionarle al sistema el uso de múltiples computadoras en interacción conjunta, se definió un modelo cliente-servidor. En esta jerarquía la comunicación fue establecida mediante un procedimiento de invocación remota (Figura 3).

Figura 3: Vista del entorno remoto.

Como se muestra en la Figura 3, los objetos de lado servidor deben ser expuestos a las peticiones de las aplicaciones clientes. Para llevar a cabo la publicación de un objeto remoto el servidor debe registrar un canal por el cual estará a la escucha de las solicitudes ejecutadas por los clientes. El servidor, por su parte, debe registrar la información de la configuración remota. Este aplica un formateo binario al objeto y devuelve una instancia del mismo. Luego el cliente descifra el objeto para disponer de sus funciones. Se concibió el sistema siguiendo un paralelismo de datos, pues la población de individuos fue seccionada en sub poblaciones para distribuirlas y practicar su evolución en los diferentes nodos. Además, se ideó un modelo de comunicación por memoria distribuida y paso de mensajes.

Para el procesamiento de aplicaciones paralelas, *Microsoft .Net Framework* 4 (15) ha sido provisto de librerías que contienen sofisticados algoritmos, los cuales distribuyen los

ISSN: 2602-8085
Vol. 1, N°3, p. 40-52, octubre - diciembre, 2017

cálculos dinámicamente en arquitecturas multinúcleo. Además, el Entorno de Desarrollo Integrado *Visual Studio* 2010 (o superior), ofrece herramientas de análisis y depuración que apoyan este modelo de programación. El *framework* de *.Net* proporciona un conjunto de tipos públicos denominado *Task.Parallel.Library* (TPL) que simplifican el proceso de transformación de las aplicaciones en paralelas. La TPL balancea el nivel de concurrencia dinámicamente y utiliza los procesadores disponibles de forma eficiente. Además, permite administrar las tareas que pueden ser ejecutadas secuencialmente controlando el estado que estas adquieren (15). En las operaciones paralelas de datos, se crean particiones de la colección de origen para que varios subprocesos puedan funcionar simultáneamente en segmentos diferentes. La TPL admite el paralelismo de datos a través de la clase *Parallel*que se encuentra dentro del espacio de nombre *System.Threading.Tasks*. Esta clase proporciona las implementaciones paralelas basadas en los bucles *for* y *foreach*(15).

El trabajo con tareas permite ganar en eficiencia y escalabilidad de recursos del sistema. Esto se debe a que la TPL contiene algoritmos que determinan y se adaptan al número de hilos que maximizan el rendimiento además de aquellos destinados a establecer un equilibrio en la carga. Debido a esto, el empleo de las tareas se torna relativamente ligero, siendo posible la creación de varios hilos para establecer un paralelismo de grano fino (15).

Detalles de la implementación

La implementación de la nueva versión distribuida del *GeneticMiner* se llevó a cabo siguiendo el procedimiento general ilustrado en el siguiente pseudocódigo:

Entrada: log, tamaño de población, número iteraciones, radio elitismo, precisión, radio cruzamiento, radio mutación, tamaño torneo, conteo cruzamiento generaly período evolución.
Salida: Matriz causal
Leer y procesar el log
Crear conexiones remotas
Construir sub poblaciones y distribuirlas por los nodos
Iniciar la evolución de la sub población en cada nodo
Mientras no se cumplan las condiciones de parada
    Cada cierto número de iteraciones
        - El coordinador construye una población global uniendo
        La últimas sub poblaciones generadas por cada nodo
        - Obtiene una nueva generación en la población global
        Insertar mejores individuos en todas las sub poblaciones
    Actualizar mejor individuo a partir entre todas las sub poblaciones
El coordinador crea población global uniendo las últimas sub poblaciones generadas por cada nodo
Obtiene una nueva generación en la población global

ISSN: 2602-8085
Vol. 1, N°3, p. 40-52, octubre - diciembre, 2017

Devuelve el mejor individuo (Matriz Causal) de la población resultante

Para establecer la comunicación entre los nodos, se crearon dos aplicaciones basadas en una estructura cliente-servidor utilizando. *NetRemoting*(17). La primera es la encargada de distribuir las operaciones entre los ordenadores, desempeñando el rol de coordinador. Para controlar las peticiones realizadas por el nodo central se utilizó un arreglo de tareas. Estas juegan un papel importante pues permiten la ejecución simultánea de las solicitudes realizadas a los ordenadores esclavos. Mediante esta estructura es muy cómodo obtener los procesos finalizados exitosamente así como aquellas invocaciones que fallaron producto algún problema. Cada tarea se relaciona con una población y mantiene una conexión a un objeto remoto determinado. Para establecer el vínculo entre ellos se crearon estructuras de tamaño fijo tomando índices de acceso comunes entre ellas.

Al culminar una tarea el coordinador renueva su población correspondiente, verifica y actualiza el mejor fitness y crea un nuevo hilo solicitándole al nodo esclavo que continúe el proceso evolutivo. En caso de detectar alguna tarea fallida se marca el índice que responde a su posición dentro del arreglo. Luego se bloquean los envíos hacia este nodo y se monitorea constantemente, buscando si se restableció la comunicación para reanudar su ejecución. Cada cierto número de iteraciones se crea un nuevo hilo convocando al método encargado de efectuar el cruzamiento. Este método crea inicialmente una población constituida por todas las sub poblaciones existentes. Luego evoluciona la población general resultante y se ordenan los individuos según su adaptabilidad. El número de mejores individuos depende del grado de elitismo establecido. Estos son insertados en las sub poblaciones en la medida que finalicen su tarea en ejecución. El proceso se repite hasta encontrar alguna condición de parada.

Análisis de los resultados obtenidos

En aras de verificar si los resultados obtenidos por la nueva versión distribuida del algoritmo *Genetic Miner* se adaptan a las condiciones requeridas, se realizó una comparación entre la variante implementada en la herramienta *ProM*(7), la versión secuencial desarrollada en el marco del proyecto CITI-UNAH (1) y la obtenida en este trabajo. Para ello se utilizaron registros de eventos ficticios con las características mostradas en la Tabla 1.

Nombre	Número de actividades	Número de trazas
repairExample.xes	12	1 104
reviewExampleLarge.xes	20	10 000
log2_sinerror.xes	104	20 000
log1_sinerr.xes	118	20 500

Tabla 1: Datos de los registros de eventos.

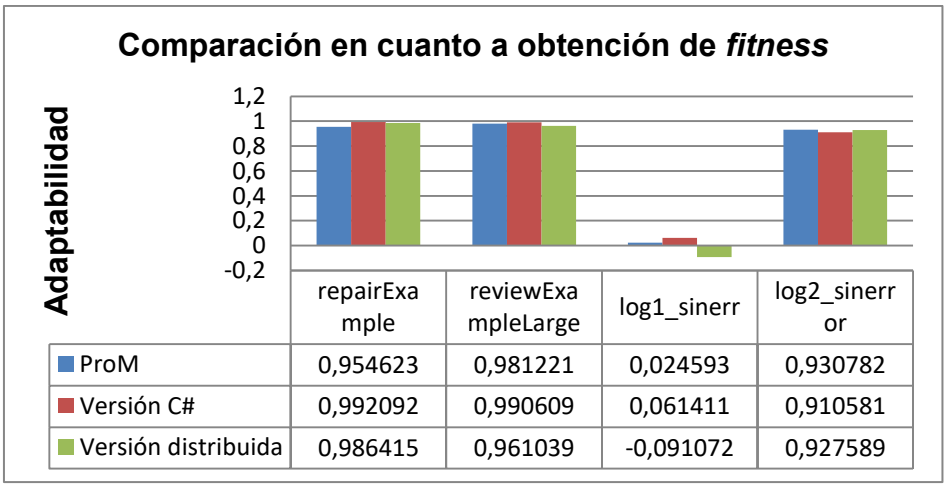
ISSN: 2602-8085
Vol. 1, N°3, p. 40-52, octubre - diciembre, 2017

Con el objetivo de igualar las condiciones de ejecución entre los algoritmos se mantuvo un estándar en los parámetros de entrada. En la Tabla 2 se brinda la relación de los datos introducidos a las distintas versiones para su procesamiento. Al colocar la precisión en 1 se fuerza a que los algoritmos realicen las 1000 iteraciones pues en ningún caso se logra un paseo perfecto de todas las trazas.

Tamaño de población	10
Número máximo de iteraciones	1000
Índice de elitismo	0,2
Índice de cruzamiento	0,7
Índice de mutación	0,2
Precisión deseada	1
Tamaño de torneo	5

Tabla 2: Entradas de los algoritmos.

Tras llevar a cabo las ejecuciones de los algoritmos se obtuvieron los siguientes resultados (Figura 4).



	repairExample	reviewExampleLarge	log1_sinerr	log2_sinerror
ProM	0,954623	0,981221	0,024593	0,930782
Versión C#	0,992092	0,990609	0,061411	0,910581
Versión distribuida	0,986415	0,961039	-0,091072	0,927589

Figura 4: Precisión obtenida por las versiones del algoritmo *Genetic Miner*.

Como se puede apreciar en la Figura 4, las precisiones obtenidas por las tres versiones del algoritmo son muy similares. El peor de los casos se evidencia al procesar el fichero log1_sinerr.xes donde el *fitness* alcanzado por la variante marca una diferencia insignificante de 0,029661 unidades.

4.2. Comparación en cuanto a tiempos de respuesta

Para realizar las comparaciones entre los algoritmos en cuanto a tiempos de respuesta se utilizaron ordenadores con las siguientes características:

Para ejecutar los algoritmos de las versiones *ProM*(7) y la secuencial re-implementada en *C#*(1)se utilizó una computadora Intel Core 2 Quad Q9300 a 2.5 GHz con 4 GB de RAM.

Mientras que en la ejecución de la versión distribuida se utilizó como nodo maestro una PC Intel Core 2 Quad Q9300 a 2.5 GHz con 4 GB de RAM. Como esclavos se tomaron ordenadores con las siguientes peculiaridades:

1. Intel Core i3-2120 a 3.30 GHz con 2 GB de RAM
2. Intel Core i7-4700 MQ a 2.4 GHz con 8 GB de RAM
3. AMD A10-4600 M a 2.3 GHz con 6 GB de RAM
4. Intel Core i5 a 2.4 GHz con 4 GB de RAM

Como se puede apreciar en la tabla 3, hay una disminución significativa de los tiempos de respuesta entre las distintas variantes del algoritmo (Figura 5). La versión distribuida del *Genetic Miner* resultó ser la que menos demoró en obtener los resultados de sus ejecuciones.

	ProM			Versión C#			Versión distribuida		
	Tiempo transcurrido para 1000 iteraciones								
Registro de eventos	h	min	seg	h	min	seg	h	min	seg
repairExample	00	21	36	00	01	54	00	00	10
reviewExampleLarge	> 7 horas			00	44	07	00	04	05
log1_sinerr	> 7 horas			04	46	26	01	12	12
log2_sinerror	> 7 horas			04	36	02	01	03	53

Tabla 3: Tiempos de respuesta de los algoritmos.

Al procesar el fichero log2_sinerror.xes, el objeto remoto publicado en el servidor *server2* a través del puerto 1202, presentó problemas de conexión. Esta ruta respondía a la dirección IP de la PC Intel Core i7-4700 MQ a 2.4 GHz con 8 GB de RAM. La causa de este imprevisto resultó ser un problema en el conector RJ45 del cable de red. A pesar de este inconveniente, el algoritmo continuó su ejecución. Media hora después, fue reemplazado el cable de red de la PC y se reincorporó al clúster. Inmediatamente el nodo maestro detectó la reactivación de la computadora y comenzó a realizarle peticiones. Al finalizar la ejecución del algoritmo se demostró que el mismo mantuvo integridad en los datos.

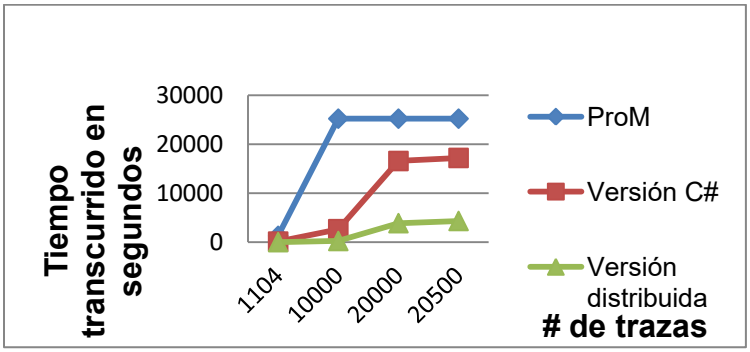


Figura 5: Gráfica de comportamiento de los tiempos de respuesta.

4.3. Pruebas realizadas al clúster

En estos experimentos se evaluó la escalabilidad y el rendimiento del sistema distribuido. Para ello se midieron los tiempos de procesamiento del algoritmo variándole el número de recursos disponibles. Mediante la comprobación se valoró el comportamiento del clúster al aumentar la cantidad de nodos esclavos. Tras efectuar las pruebas se comprobó que el tiempo de respuesta de la versión distribuida del algoritmo *Genetic Miner* fue disminuyendo en la medida que eran incorporados los nodos esclavos al clúster (Figura 6).

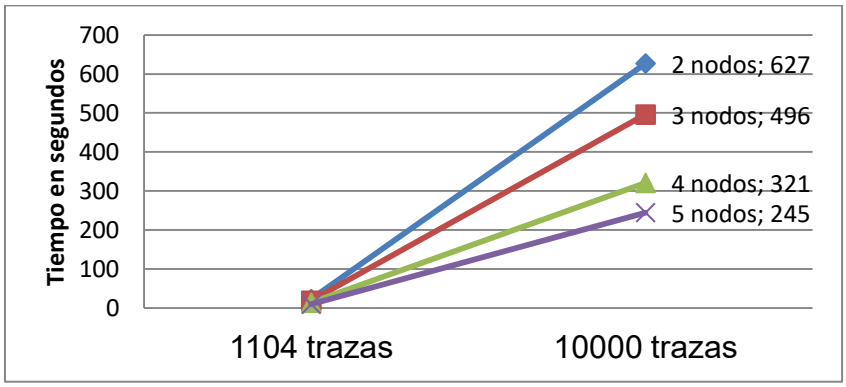


Figura 6: Comportamiento del clúster por cantidad de nodos.

Durante esta etapa de pruebas, la precisión obtenida por el algoritmo alcanzó resultados muy similares a las implementaciones anteriores. En el peor de los casos se obtuvo solo una diferencia de 0,029661 unidades. Por otra parte, los tiempos de respuesta disminuyeron considerablemente. Además, los experimentos realizados al clúster demostraron que su eficiencia aumentaba en la medida que se le iban incorporando nodos esclavos.

Conclusiones

En este trabajo se investigó el algoritmo *Genetic Miner* dentro de la fase de descubrimiento de modelos en la minería de procesos. Después de analizadas cada una de las etapas del algoritmo, se determinó que el cálculo de la adaptabilidad dadas sus características constituía un punto crítico que afectaba la eficiencia del mismo. El *Genetic Miner* se desarrolla a través de un proceso iterativo evolucionando un conjunto de poblaciones que podían ser procesadas de manera independiente, lo que hizo viable su implementación paralela.

Se desarrolló un esquema híbrido que combina la estructura de grano grueso con maestro esclavo sobre un clúster de computadoras, explotando las posibilidades de los microprocesadores en cada una de ellas. Los tiempos de respuesta obtenidos se redujeron significativamente con relación a sus versiones similares implementadas en la herramienta *ProM* (7) y la de *C#* implementada en la UNAH (1).Además, la precisión obtenida por el algoritmo se correspondía con lo esperado, demostrando la efectividad de distribuir el mismo.

ISSN: 2602-8085
Vol. 1, N°3, p. 40-52, octubre - diciembre, 2017

RECEIVED: April, 2017
REVISED: August, 2017

Referencias bibliográficas

- López Pintado, Lic. Orlenys .Estrategia para acelerar el algoritmo Genetic Miner utilizando matrices de índices. San José, Mayabeque: s.n., 2013.
- Vander Aalst, WilM.P. Process Mining. Discovery, Conformanceand Enhancementof Business Processes. London New York : Springer Heidelberg Dordrecht, 2011 van der Aalst W.M.P., Weijters A. J.M.M., and L. Maruster. Work flowmining: Discovering process models frome ventlogs. IEEE Transaction son Knowledge and Data Engineering. 2004.
- Weijters, A.J.M.M., vander Aalst,W.M.P. and Alvesde Medeiros,A.K. Process Mining with Heuristics Miner Algorithm. BETA Working Paper Series, WP 166 Eindhoven University of Technology. Eindhoven :s.n., 2006.
- Günther, C.W. andvander Aalst, W.M.P. Fuzzy Mining: Adaptive Process Simplification Based on Multi-Perspective Metrics. 2007.
- Alves de Medeiros, Ana Karla. Genetic Process Mining. Eindhoven: University Press Facilities, 2006.
- Van Dongen, B.F. TheProM Framework: A New Erain Process MiningTool Support. 2005.
- Vander Aalst, WilM.P. Process Mining. Discovery, Conformanceand Enhan cement of Business Processes.LondonNewYork :Springer Heidelberg Dordrecht, 2011.
- Alves de Medeiros, Ana Karla, Weijters, A.J.M.M. and van der Aalst, W.M.P. Genetic Process Mining: An Experimental Evaluation. Data Mining and Knowledge Discovery.s.l. : Department of Technology Management, Eindhoven University of Technology, 2007.
- Bratosin, Carmen. Grid Architecture for Distributed Process Mining. Eindhoven: University Press Facilities, 2011.
- Banzhaf, Wolfgang, Nordin, Peterand Keller, RobertE.. Genetic Programming. AnIntroduction. San Francisco, California : Morgan Kaufmann Publishers, Inc., 1998.
- Koza, John R. Genetic Programming. Theoryand PracticeII. Boston: Springer Science + Business Media, Inc., 2005.
- Menon, Anil.Genetic Algorithmsand Evolutionary Computation. NewYork: s.n., 2004.
- Turner, Christopher James. A Genetic Programming Based Business Process Mining Approach. 2009.

- Palomino Vila, César. Herramientas para el descubrimiento de modelos de procesos. 2013.
- Wiley, John. Tools and Environments for Parallel and Distributed Computing. New Jersey :s.n., 2004.
- Tanenbaum, Andrew. Introdution to Distributed Systems.
- Microsoft.MSDNLibrary.[Enlínea]3de4de2014.http://msdn.microsoft.com/es-es/library/ms123401.aspx.
- Vasallo Díaz, Asiel. Algoritmo genético distribuido para el descubrimiento de procesos. 2014.

ISSN: 2602-8085
Vol. 1, N°3, p. 40-52, octubre - diciembre, 2017