Universidad Agraria de La Habana
Facultad de Ciencias Técnicas

DISEÑO DEL ALGORITMO GENETIC MINER DISTRIBUIDO CON MEJORA A TOLERANCIA ANTE FALLOS PARA LA PRODUCCIÓN DE LANGOSTA

Tesis presentada en opción al título académico de Máster en Biomatemática

Autor: Ing. Asiel Díaz Vasallo

Tutor: Dr.C Yasser Vázquez Alfonso

Mayo de 2019

Resumen

Las técnicas de la minería de procesos permiten la extracción de conocimientos de los registros de eventos. Estas posibilitan el descubrimiento, monitoreo y mejora de procesos en dominios de aplicaciones variadas. El descubrimiento de modelos es una de las etapas que componen la minería de procesos. En esta etapa se utilizan algoritmos que parten inicialmente de un registro de eventos y descubren modelos que reflejan el comportamiento real del proceso.

La presente investigación se apoya en dichas técnicas para facilitar el análisis y la toma de desiciones en la industria pesquera del municipio Batabanó. Se centra en extender el algoritmo *Genetic Miner*, uno de los que ofrecen mejor resultado en la mencionada fase de descubrimiento. Se propone un esquema de distribución híbrido que combina las estrategias grano grueso con maestro esclavo. Se comparan además las variantes del algoritmo con la alternativa propuesta en esta investigación. Estas pruebas fueron efectuadas en base a precisiones obtenidas y tiempos de respuesta de los algoritmos. Los experimentos fueron realizados sobre registros de eventos del proceso de producción de langosta.

Palabras Claves: minería de procesos, descubrimiento, modelo

Abstract

The techniques of process mining allow the extraction of knowledge from the event logs. These enable the discovery, monitoring and improvement of processes in domains of varied applications. The discovery of models is one of the stages that make up process mining. In this stage algorithms are used that initially start from an event log and discover models that reflect the real behavior of the process.

This research is based on these techniques to facilitate the analysis and decision making in the fishing industry of Batabanó municipality. It focuses on extending the Genetic Miner algorithm, one of those that offer the best result in the aforementioned discovery phase. A hybrid distribution scheme that combines coarse grain strategies with master slave is proposed. The variants of the algorithm are also compared with the alternative proposed in this investigation. These tests were carried out based on obtained precisions and response times of the algorithms. The experiments were performed on logs of lobster production events.

Keywords: process mining, discovery, model

Índice

Introducción

La pesca se originó hace miles de años, siendo sin lugar a dudas, una de las primeras actividades del hombre, encaminada a satisfacer sus necesidades alimenticias. En un principio la pesca se limitaba a una simple recolección, pasando a usarse posteriormente ingenios habituales de caza, tales como lanzas, el arco y las flechas, tanto en las aguas continentales como en el mar.

A la llegada de los colonizadores a Cuba, existían dos grupos de aborígenes, uno de ellos nombrados Recolectores, Cazadores, Pescadores; eran esas sus principales actividades económicas (Díaz Beltrán, 2014). La pesca tiene un mayor desarrollo al ir mejorando sus instrumentos de trabajo. Los colonizadores europeos trajeron consigo conocimientos y otras herramientas que convirtieron con el paso de los siglos a esta pequeña actividad en una fuerte industria y actividad de comercio para el país.

En la actualidad la Industria Pesquera de Cuba se basa en una estrategia de desarrollo que tiene en cuenta el uso racional de los recursos naturales y la protección del medio ambiente. Fundamentada en un programa de administración para la explotación de los recursos naturales y promover la ampliación de la producción de los cultivos acuáticos. Resulta importante por su contribución en la oferta de alimentos a la población y por sus exportaciones.

El 24 de enero de 1960 se funda la cooperativa pesquera Batabanó, luego en febrero de 1969 se crea el Combinado pesquero Batabanó, que actualmente se reconoce como Empresa Pesquera Industrial (“Creación del Combinado Pesquero de Batabanó | MINAL,” n.d.); la misma tiene como objetivo la captura, industrialización y comercialización de forma mayorista, de especies de la plataforma frescas o congeladas, siendo la langosta su principal renglón

exportable. Sus procesos productivos garantizan que los productos son de alta calidad comparables con los existentes en el mercado mundial y que satisfacen las necesidades y expectativas de los clientes, nacionales y extranjeros, de igual forma se cumple con un grupo de requisitos como son: el cumplimiento de las regulaciones pesqueras, que contribuyen a disminuir la contaminación del medioambiente en el Golfo de Batabanó, se trabaja para lograr el máximo aprovechamiento industrial; velan, además por el sistema de gestión de la calidad e inocuidad de los alimentos.

Por tal motivo, resulta de gran utilidad comprender el conjunto de actividades que se ejecutan de forma coordinada durante sus procesos de producción (fundamentalmente el de langosta), mediante la interpretación de patrones estructurales contenidos en los datos recopilados tras la ejecución de las actividades. Estas, se encuentran incorporadas en un marco organizacional y tecnológico que permite su efectiva y completa realización, siendo su administración un factor decisivo en el proceso de mejora continua de la organización; proporcionando ventajas competitivas como el aumento de la productividad, la disminución de la cantidad de recursos utilizados, entre otros factores.

El incremento constante de información a analizar en esta empresa, trae consigo la necesidad de emplear técnicas analíticas para la extracción de conocimiento en grandes volúmenes de datos, necesarias para diagnosticar problemas e identificar posibles áreas de mejora en el proceso. Su uso permitiría aplicar una estrategia de optimización de los procesos de producción de langosta como la clave para reducir costos y expandirse en su entorno. En este sentido, la minería de procesos, centra sus esfuerzos en la obtención automatizada de información almacenada en ficheros estructurados según diversos estándares; develándose así la siguiente **problemática**:

Múltiples algoritmos han sido propuestos para descubrir modelos que reflejen el comportamiento real del proceso; estas técnicas de descubrimiento parten de información previa. El *Genetic Miner* es uno de los algoritmos que arroja el resultado más prometedor (van der Aalst, 2011), sin embargo, presenta como desventaja un tiempo de respuesta elevado. Diversos autores han centrado sus investigaciones en la mejora de su costo computacional, obteniéndose variantes a lo largo de los años. Las propuestas centradas en algoritmos genéticos distribuidos son las que finalizan su ejecución en un tiempo menor y en su mayoría, no se puede acceder al código fuente.

Se cuenta con la propuesta de Díaz Vasallo et al., (2017), donde se combinan estrategias de distribución que dan lugar a una variante distribuida del *Genetic Miner*. En esta propuesta se logra disminuir de manera significativa el tiempo de repuesta, sin embargo, la arquitectura planteada no contempla la tolerancia a fallos. Se parte de un nodo central para coordinar la ejecución del algoritmo, dicho nodo es el encargado de evolucionar la población de individuos y ejecuta las etapas de selección, cruzamiento, mutación. La función de los restantes nodos es efectuar, de forma paralela, el cálculo de adaptabilidad para pequeños grupos de la población evolucionada. Al colapsar el nodo central se interrumpe la ejecución del algoritmo, perdiéndose toda la información procesada hasta el momento; siendo necesario reiniciar el procesamiento, lo cual trae consigo una pérdida innecesaria de tiempo.

A partir de la problemática descrita se plantea el **problema a resolver:** ¿Cómo mejorar la eficiencia en el algoritmo *Genetic Miner* distribuido para su aplicación en el proceso de producción de langosta?

El problema planteado conlleva a la siguiente **hipótesis**: si se redistribuye a los restantes nodos, las funciones del nodo central ante un fallo se mejora la eficiencia en el algoritmo *Genetic Miner* para su aplicación en el proceso de producción de la langosta.

El **objeto de estudio** se enmarca en los algoritmos genéticos distribuidos y su **campo de acción** los algoritmos genéticos distribuidos aplicados al proceso de producción de langosta.

El **objetivo general** es diseñar una variante del algoritmo *Genetic Miner* distribuido que permita tolerancia a fallos para su aplicación en el proceso de producción de la langosta.

En aras de cumplir este objetivo general se establecieron los siguientes **objetivos específicos:**

1. Valorar tendencias actuales del uso de los algoritmos genéticos en la minería de procesos.

2. Identificar técnicas de programación paralela y distribuida que mejoren la eficiencia en cuanto a la tolerancia a fallos del algoritmo *Genetic Miner*.

3. Diseñar el algoritmo *Genetic Miner* utilizando las técnicas de programación paralela y distribuida identificadas.

4. Validar el algoritmo *Genetic Miner* con la mejora en la tolerancia a fallos utilizando datos del proceso de producción de la langosta.

El **aporte práctico** de esta investigación es el diseño e implementación de una variante del algoritmo *Genetic Miner* que permita tolerancia ante fallos del nodo central, mejorando la eficiencia en el descubrimiento de modelos en el proceso de producción de langosta a partir de registros de eventos.

Capítulo

1 Revisión bibliográfica

Este capítulo aborda elementos relacionados con la minería de procesos, fundamentalmente en la etapa de descubrimiento de modelos. Centra su atención en el análisis del *Genetic Miner*, sus antecedentes, así como en las estrategias existentes que permiten distribuir el algoritmo con el fin de aumentar su eficiencia.

1.1. La minería de procesos y sus retos

La minería de procesos es una rama en la ciencia de la computación que posibilita el descubrimiento y mejora de procesos en dominios variados de aplicaciones. La misma está dividida fundamentalmente en 3 etapas (Aalst, 2016):

1. Descubrimiento del modelo
2. Análisis de conformidad
3. Extensión del modelo

En la primera etapa, las técnicas de descubrimiento parten de información previa, sobre la cual se efectúan análisis para la obtención de un modelo del proceso. En el análisis de conformidad, se realiza la comparación de un modelo de procesos con el registro de eventos (*log*) de un proceso real, para verificar la aproximación del modelo con respecto a dicho proceso. Cuando se comparan modelos que especifican requisitos de un proceso con tendencias en registros de eventos, se contribuye a la detección de fraude.

Por tal motivo, la comprobación de conformidad puede ser usada para descubrir, localizar y explicar inconsistencias en el proceso. Por otro lado, la tercera etapa (extensión del modelo) tiene como reto mejorar los modelos existentes utilizando información acerca de procesos actuales, almacenados en registros de eventos.

A pesar de las funcionalidades brindadas por técnicas para realizar minería de procesos, aún existen desafíos importantes asociados a limitantes. En la fase de descubrimiento, resulta difícil construir modelos consistentes a partir de *logs* incompletos y ruidosos. Las técnicas de descubrimiento emplean un lenguaje particular: modelos y notaciones de procesos del negocio (por sus siglas en inglés BPMN) o redes de Petri, entre otros, para producir un modelo. Por tal motivo, el espacio de búsqueda se ve limitado, pues los procesos que no pueden representarse por el lenguaje elegido no pueden ser descubiertos.

Las redes sobre flujos de trabajo, BPMN, cadenas de procesos conducidas por acontecimientos, entre otros, pueden representar procesos con puntos muertos o actividades que no deberían ser iniciadas. También es importante descubrir el instante en que cambian los procesos y visualizar dichos cambios, cuestión no tomada en cuenta por los enfoques de descubrimiento actuales. Además, los procesos pueden contener un conjunto de patrones estructurales, tales como, secuencias, paralelismo, ciclos, tareas invisibles, entre otros (van der Aalst, 2011).

Son varios los algoritmos diseñados e implementados que descubren modelos. La elección de cuál se ajusta mejor para procesar un registro de eventos determinado, constituye un reto para los analistas; resulta difícil conocer a priori los patrones presentes y detectar datos ruidosos. A pesar de estas dificultades, algunos algoritmos y modelos parecen reflejar de manera acertada los procesos que reconstruyen, entre estos se encuentra el algoritmo *Genetic Miner* (van der Aalst, 2011).

1.2. Algoritmos para el descubrimiento de modelos

Múltiples algoritmos diseñados para construir modelos de procesos, se basan en análisis estadísticos, métodos heurísticos y otras técnicas de la matemática aplicada. Entre los más conocidos se hallan: el *Alpha Miner* (Premchaiswadi and Porouhan, 2016), el *Heuristic Miner* (Weijters and Ribeiro, 2011), el *Fuzzy Miner* (CW and van Der Aalst, 2007) y el *Genetic Miner* (Alves De Medeiros and Weijters, 2005). Todos se encuentran implementados en la herramienta *ProM* (Claes and Poels, 2013) desarrollada por investigadores de la Universidad Tecnológica de *Eindhoven*, en Holanda, centro donde se han obtenido varios de los principales resultados en el área de minería de procesos.

El *Alpha Miner* es un algoritmo pionero en esta área, de fácil entendimiento y que se convierte en una referencia obligada para todo investigador que se inicia en esta rama de la ciencia. A partir de un registro de eventos (recibido como entrada) que contiene varias instancias de la ejecución de un proceso, se ordenan las relaciones entre las tareas que son deducidas. Mediante el orden establecido se indica la precedencia entre las tareas, su ejecución en paralelo, entre otros aspectos, que garantizan la reconstrucción de un flujo de procesos. El modelo descubierto se corresponde con una red de Petri, que representa un flujo de trabajo (van der Aalst, 2011). Este algoritmo no es robusto al ruido ni ante patrones estructurales complejos, por lo que su uso es casi exclusivamente académico.

Los algoritmos *Fuzzy Miner*, *Heuristic Miner* y *Genetic Miner* son capaces de lidiar con registros de eventos ruidosos y patrones estructurales complejos, por lo que tienen gran aplicabilidad en situaciones prácticas. En el caso del *Fuzzy Miner* se extraen modelos jerárquicos. Según criterios y métricas definidas, que determinan cuáles arcos deben ser incluidos en un grafo de dependencias, las actividades con baja frecuencia pero que se encuentran estrechamente relacionadas son agrupadas en subprocesos. De este modo, se genera un modelo *fuzzy*, ideal para representar procesos espagueti (van der Aalst, 2011).

El algoritmo *Heuristic Miner* utiliza matrices causales (A. de Medeiros et al., 2007). Establece heurísticas considerando la secuencia y frecuencia de eventos que rigen la construcción del modelo. Se centra en detectar caminos poco frecuentes que no deberían aparecer en el modelo resultante. Su tendencia de representar modelos mediante redes causales y el manejo de las frecuencias, hacen que este algoritmo sea uno de los más robustos.

El *Genetic Miner*, centro de esta investigación, utiliza un procedimiento iterativo para simular el proceso de evolución natural. A diferencia de los algoritmos *Alpha Miner*, *Fuzzy Miner* y *Heuristic Miner*, que construyen los modelos mediante técnicas directas y deterministas locales; el algoritmo *Genetic Miner* utiliza una estrategia global para procesar los registros de eventos. Gupta (2014) recomienda su uso en la extracción de registros con ruido, manejo de nombres de tareas duplicados y tareas invisibles.

1.3. Funcionamiento general del algoritmo *Genetic Miner*

El algoritmo *Genetic Miner* toma como entrada un registro de eventos y simula el proceso de evolución de los individuos como se observa en la figura 1.

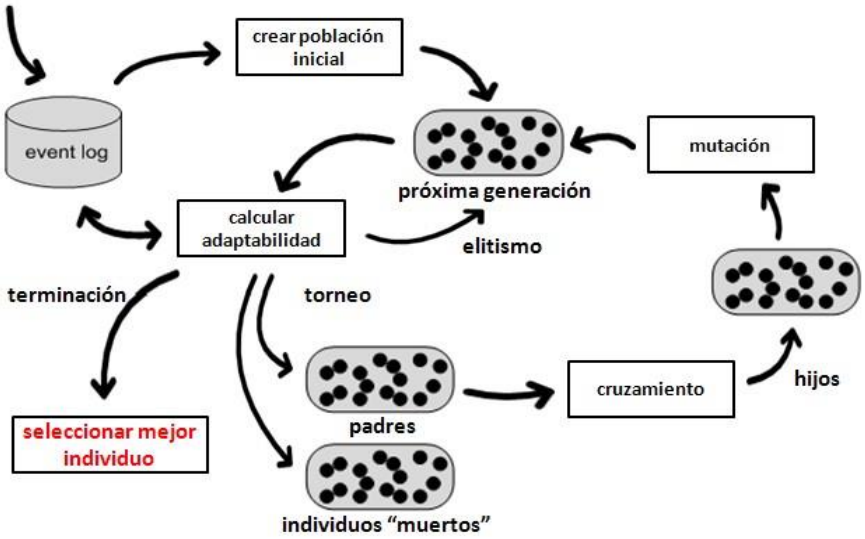


Figura 1 Funcionamiento general del algoritmo Genetic Miner Fuente: (van der Aalst, 2011)

A partir de la entrada se genera una población inicial formada por n individuos, cada uno de los cuales representa una instancia del proceso que se quiere recuperar. Para cada individuo se calcula la medida de adaptabilidad (*fitness*) que evalúa cuán bien es reflejado el comportamiento del proceso almacenado en el registro de eventos. Una vez obtenido el *fitness*, si se cumplen ciertas condiciones de parada, el algoritmo termina devolviendo al individuo más adaptado de la población. En caso de no cumplirse ninguna de las condiciones esperadas se construirá una nueva generación de individuos, con la esperanza de que algunos estén mejores adaptados que sus antecesores (Alves De Medeiros and Weijters, 2005).

Para la construcción de la siguiente generación, los mejores individuos son ascendidos directamente: proceso conocido como elitismo. A través de torneos, se seleccionan a los padres que generarán nuevos individuos al aplicárseles dos operadores genéticos (Martínez et al., 2008; O'Reilly et al., 2006): cruzamiento y mutación. Para el cruzamiento, se toman dos padres que generan dos descendientes, como resultado de la combinación de sus características. A los nuevos individuos se les aplica aleatoriamente el operador mutación, que agrega rasgos no presentes en sus padres. Una vez completada la próxima población se calcula el *fitness* de cada individuo; repitiéndose el proceso antes descrito.

Alcanzar las condiciones de parada establecidas para el algoritmo puede requerir un gran número de iteraciones (construcción de generaciones). Esto provoca que el *Genetic Miner*, como todos los procesos evolutivos, presente algunas limitaciones.

1.3.1. Antecedentes y limitaciones de los algoritmos genéticos para el descubrimiento de procesos

El algoritmo *Genetic Miner* presentado en el año 2006 por la investigadora Ana Karla Alves de Medeiros, como tesis doctoral, constituyó una estrategia flexible y robusta para la construcción de modelos de procesos. Al igual que el *Heuristic Miner*, genera una matriz causal. La utilización de técnicas heurísticas permite el manejo de registros de eventos incompletos y con ruidos. Dada la flexibilidad del algoritmo,

otros autores posteriormente han modificado algunos pasos obteniendo variantes del mismo.

Banzhaf et al., (1998), se presentaron un enfoque del algoritmo modificando fundamentalmente la representación de los individuos mediante un nuevo grafo. Esta propuesta reflejaba de manera precisa construcciones complejas, presentes en registros de eventos ruidosos con más de 20 actividades involucradas. Además, incorporaba una adaptación a los pasos fundamentales del *Genetic Miner* (aunque mantenía el hilo central del mismo) para permitir el manejo de la nueva estructura de datos.

La principal limitación que tiene la minería de procesos utilizando algoritmos genéticos, viene dada por la baja eficiencia ante registros de eventos y modelos muy grandes. Como todos los procesos iterativos, cuando el número de iteraciones es elevado, los tiempos de respuesta para encontrar individuos con una precisión aceptable aumentan considerablemente. Afortunadamente, esta situación puede ser contrarrestada realizando implementaciones paralelas. Es decir, el procesamiento de los individuos puede ser realizado a través de nodos computacionales que trabajen simultáneamente.

Bratosin, C.C. et al., (2011) ofrecen una variante del algoritmo *Genetic Miner* en un entorno distribuido conservando los pasos fundamentales del algoritmo original. Este acercamiento permitía el procesamiento de registros de eventos grandes; distribuyendo las tareas en un conjunto de nodos. La implementación de esta plataforma contribuía a disminuir los tiempos de respuesta del algoritmo, mejorando la eficiencia.

Desafortunadamente, las implementaciones obtenidas en (Banzhaf et al., 1998; Bratosin, C.C. et al., 2011) no se encuentran disponibles, por lo que solamente resultan de utilidad teórica. En la herramienta *ProM* (Claes and Poels, 2013), se brinda la implementación en *Java* de la versión original del *Genetic Miner*. Esta

variante presenta tiempos de respuesta muy elevados cuando el registro de eventos es grande, por lo que su utilización práctica es un tanto limitada.

López Pintado, (2013) presenta una modificación a las matrices causales adicionándoles matrices de índices. Esta propuesta implementada en *C Sharp* (C#) combina las representaciones de grafos sobre listas y matrices de adyacencia; explotando las fortalezas de cada una. A pesar de que las mejoras realizadas reducen significativamente los tiempos de respuesta del algoritmo, con respecto a la versión de la herramienta *ProM,* la rapidez en la obtención de los resultados continúa viéndose afectada.

Díaz Vasallo et al., (2017) combinan diversas estrategias de programación paralela y distribuida que dan lugar a una nueva versión del *Genetic Miner*. En la investigación se logra disminuir de manera significativa los tiempos de repuesta. Sin embargo, la arquitectura planteada no contempla la tolerancia a fallos a nivel del nodo central. En consecuencia, se perderían los datos alcanzados tras la interrupción de dicho nodo. Esto se debe a la total dependencia de un ordenador para coordinar la ejecución del algoritmo; siendo necesario un cambio en la técnica de paralelización asumida.

1.4 Paralelismo en algoritmos genéticos

La computación paralela es una forma de cómputo en la que varias instrucciones son ejecutadas de manera simultánea. Se basa en la posibilidad de descomponer problemas grandes en subproblemas más pequeños para resolverlos posteriormente en paralelo (Ivan , 2018). El paralelismo ha sido utilizado durante muchos años por la computación de altas prestaciones.

Las computadoras paralelas pueden ser clasificadas atendiendo al tipo de paralelismo admitido por su *hardware*. Primeramente, se encuentran los ordenadores con procesadores multinúcleo y multiprocesador, esto quiere decir que en la misma computadora contienen múltiples elementos de procesamiento.

Por otra parte, están los sistemas distribuidos, compuestos por varios ordenadores que de conjunto trabajan en una tarea común.

La programación en paralelo permite resolver problemas que requieran de recursos como almacenamiento, procesamiento y comunicación. La implementación de estos problemas puede efectuarse sobre arquitecturas distribuidas.

1.5. Sistemas Distribuidos

"Un sistema distribuido es una colección de ordenadores independientes que aparecen a los usuarios del sistema como una única computadora" (Turner, 2009). A modo general, el proceso de diseño de un sistema distribuido involucra tres actividades principales (Bratosin, C.C. et al., 2011):

1) Diseñar el mecanismo de comunicación que habilita la distribución de los recursos y objetos del sistema para el intercambio de información.
2) Definir la estructura del sistema y los servicios que habilitan múltiples computadoras fusionadas entre sí.
3) Definir las técnicas de programación para desarrollar aplicaciones paralelas y distribuidas.

Apoyándose en el proceso de diseño antes expuesto, el sistema distribuido puede ser elaborado siguiendo una estructura de tres capas (Sánchez Villalba and Rey García, 2009)

1) ***Comunicación de red, protocolos e interfaz.*** Describe los principales componentes que serán utilizados para controlar el paso de información entre los recursos del sistema distribuido. A su vez, esta puede ser dividida en tres subcapas: tipo de red, protocolos de comunicación e interfaz de red.
2) ***Arquitectura y servicios del sistema distribuido.*** Define la estructura, arquitectura y los servicios que debe soportar en aras de funcionar como un único sistema.

3) ***Paradigmas de la computación distribuida.*** Representa cómo el programador percibe el sistema distribuido. Se centra en los paradigmas de programación que pueden ser utilizados para desarrollar aplicaciones de esta índole. Estos pueden ser ampliamente caracterizados atendiendo a dos grupos.
   a) Los modelos de cálculo, que pueden ser descritos en dos paradigmas: paralelismo funcional y paralelismo de datos. En el paralelismo funcional los cálculos son divididos en distintas funciones que son asignadas a diferentes computadoras. En el paralelismo de datos todas las computadoras siguen la filosofía de ejecutar el mismo programa aplicándolo sobre cadenas de datos distintas.
   b) Los modelos de comunicación pueden ser clasificados en paso de mensajes y memoria compartida distribuida según la forma en que se comunican las tareas. En la comunicación por paso de mensajes las tareas se intercambian información, mientras la memoria compartida distribuida se basa en el acceso a espacios de memoria en direcciones comunes.

Dentro de los modelos de comunicación que se han apoyado por herramientas de paso de mensajes están: punto a punto, mensajes activos y el de cliente-servidor. Entre las clasificaciones de este último modelo se encuentran los denominados Llamada a Procedimiento Remoto (por sus siglas en inglés *RPC*). *RPC* es un protocolo capaz de ejecutar aplicaciones en otra máquina, manteniendo transparencia en las comunicaciones. Entre los entornos de RPC se pueden mencionar:

- Modelo de Objetos de Componentes Distribuidos de Microsoft (por sus siglas en inglés DCOM). A pesar de que ninguno de estos modelos es compatible entre sí, en su mayoría utilizan un lenguaje de descripción de interfaz que define los métodos exportados por el servidor (Rubtcova and Pavenkov, 2018).

- *Java Remote Method Invocation* soporta la invocación de métodos remotos bajo un paradigma orientado a objetos (Sharan, 2018).
- *.NET Remoting* es una infraestructura de invocación remota de *.NET* que asume un enfoque diferente que el de otras interfaces de programación de aplicaciones, como DCOM e Invocación de Métodos Remotos, para establecer la comunicación y el formato de mensajes. Esta infraestructura utiliza estándares bien establecidos como Protocolos Simples de Acceso a Objetos para mensajería y Protocolos de Transferencia de Hipertexto/Protocolos de Control de Transmisión para comunicación (Poniszewska Maranda and Wasilewski, 2016).

Aplicando las técnicas de comunicación establecidas, fue posible construir modelos de ordenadores interconectados. Cada uno de ellos presenta distintas características y se adapta a situaciones específicas. Entre los primeros modelos de ordenadores interconectados está el conocido como centralizado donde todo el procesamiento de la organización se llevaba a cabo en una sola computadora, normalmente un *Mainframe* [1] , mientras los usuarios empleaban sencillos ordenadores personales. Algunos de los inconvenientes que presenta este modelo son (Sánchez Villalba and Rey García, 2009):

- Cuando la carga de procesamiento aumentaba se tenía que cambiar el *hardware* del *Mainframe*, lo cual es más costoso que añadir más computadores personales, clientes o servidores, que aumenten las capacidades.
- El otro problema surgió con las modernas interfaces gráficas de usuario, las cuales podían traer consigo un gran aumento de tráfico en los medios de comunicación y por consiguiente, podían colapsar.

Dado que para dar solución al problema que inició esta investigación se busca autonomía en los nodos, el modelo antes mencionado es descartado. Otro que se

[1] *Mainframe*: computadora muy potente que puede procesar enormes cantidades de información en poco tiempo.

acerca algo más al enfoque requerido es el de Grupo de Servidores, el cual también es un tanto centralizado. Está conformado por un conjunto de ordenadores actuando como servidores, por lo general de archivos o de impresión, sobre un número de minicomputadoras que trabajan conectadas a una red de área local. Con este modelo se podrían saturar los medios de comunicación cuando se realicen grandes pedidos por varios clientes simultáneamente. Esto provocaría una disminución en la velocidad de transmisión de la información.

Utilizar un grupo de servidores tampoco satisface las necesidades del proyecto, pues se desea obtener un sistema que permita la descentralización del procesamiento y los recursos, posibilitando que los servidores estén dedicados solo a una aplicación determinada para ejecutarla eficientemente. La presencia de un medio físico de comunicación entre las máquinas constituye un elemento primordial, cuya naturaleza determina la viabilidad del sistema.

En los sistemas Cliente/Servidor los objetos distribuidos juegan un papel fundamental, estos son gestionados por un servidor y sus clientes invocan sus métodos utilizando un "procedimiento de invocación remota". El cliente invoca el método enviando un mensaje al servidor que gestiona el objeto, se ejecuta el objeto en el servidor y el resultado se devuelve al cliente mediante otro mensaje.

La implementación de los sistemas distribuidos más conocidos son los denominados clústeres computacionales, estos son fáciles de construir y pueden hacer uso de recursos no utilizados o de bajo costo.

1.6. Clúster Computacional

Un clúster computacional está conformado por un conjunto de ordenadores conectados entre sí, por medio de una red para compartir recursos, con el objetivo de realizar tareas y funciones como si fuesen un único ordenador (memoria distribuida). Según Wiley, (2004) existen fundamentalmente tres clasificaciones de clústeres, ellas son:

- **Alto rendimiento:** El objetivo es mejorar el rendimiento, de tiempo o precisión, para la solución de un problema, ya sean cálculos matemáticos, mejora de gráficos, descifrado de códigos, entre otros.
- **Alta disponibilidad**: Estos clústeres tratan de brindar la máxima disponibilidad de los servicios que ofrecen. Se caracterizan por mantener una serie de recursos compartidos y estar monitoreándose constantemente entre sí. Si falla algún nodo, es reemplazado inmediatamente.
- **Balanceo de carga:** Está caracterizado por tener uno o varios ordenadores monitoreando constantemente las peticiones realizadas para repartirlas entre los otros nodos que conforman el clúster.

Durante la investigación, se valoraron distintas estrategias que combinaran a los algoritmos genéticos con los sistemas distribuidos. A continuación, se brindan algunas topologías de distribución existentes.

1.7. Topologías de algoritmos genéticos distribuidos

La clasificación de los algoritmos genéticos paralelos está dada según la forma en que se distribuye la carga de cómputo en los procesadores, así como la topología y el diseño de estos algoritmos (Bratosin, C.C. et al., 2011).

1.7.1. Algoritmos genéticos maestro-esclavo o comunicación estrella

En esta topología las funciones de adaptabilidad son evaluadas en paralelo (figura 2). Con el propósito de generar poblaciones, el maestro aplica los operadores genéticos de elitismo, cruzamiento y mutación de modo centralizado. Al obtener la nueva población, los individuos son enviados a los esclavos. Posteriormente, estos nodos realizan el cálculo de adaptabilidad de su individuo, al finalizar envían el valor al nodo central y reciben de este un nuevo individuo. Dicho proceso se realiza hasta evaluar a toda la población.

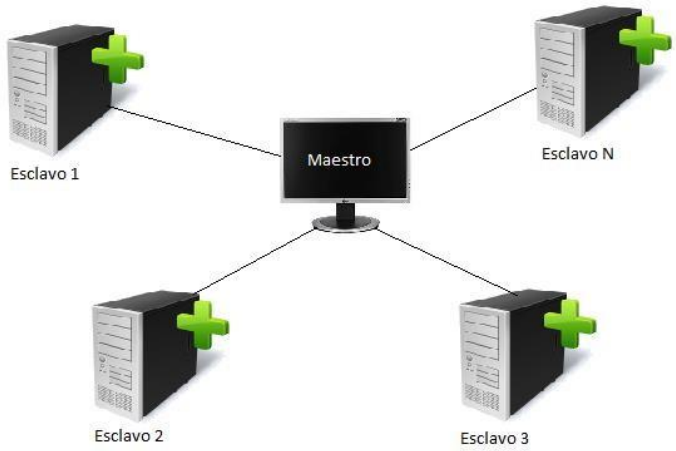


Figura 2 Estructura de distribución maestro-esclavo. Fuente: Elaboración propia

El diseño de esta estrategia permite la escalabilidad del sistema, pues es posible utilizar tantos procesadores como se desee. La implementación presenta como inconveniente el uso intensivo del canal de comunicación, pudiendo ocurrir que el tiempo de envío de individuos se iguale al tiempo consumido por el procesador del algoritmo secuencial (Helal et al., 2017).

1.7.2. Algoritmos genéticos de grano grueso

En la topología de grano grueso (figura 3) las poblaciones son separadas en subpoblaciones. Cada nodo realiza la ejecución independiente del algoritmo secuencial y la información resultante es intercambiada entre las subpoblaciones. Estas pueden evolucionar hacia distintas soluciones, permitiendo que aumente la oportunidad de encontrar la adecuada. La migración permite que cada subpoblación aprenda de las vecinas e introduzca nuevos individuos acelerando la convergencia del algoritmo (Hoseini Alinodehi et al., 2016).

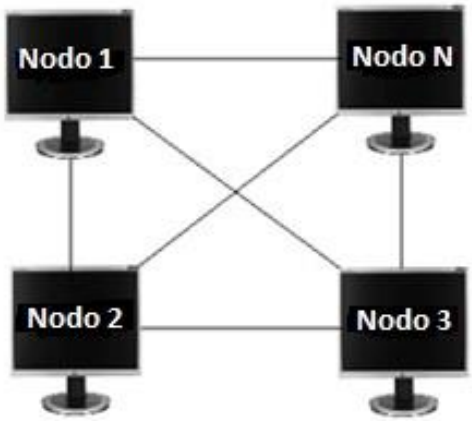


Figura 3 Modelo de grano grueso. Fuente: Elaboración propia

1.7.3. Algoritmos genéticos de grano fino

En esta estructura (figura 4) cada individuo reside en un procesador diferente que forma parte de la misma topología de red.

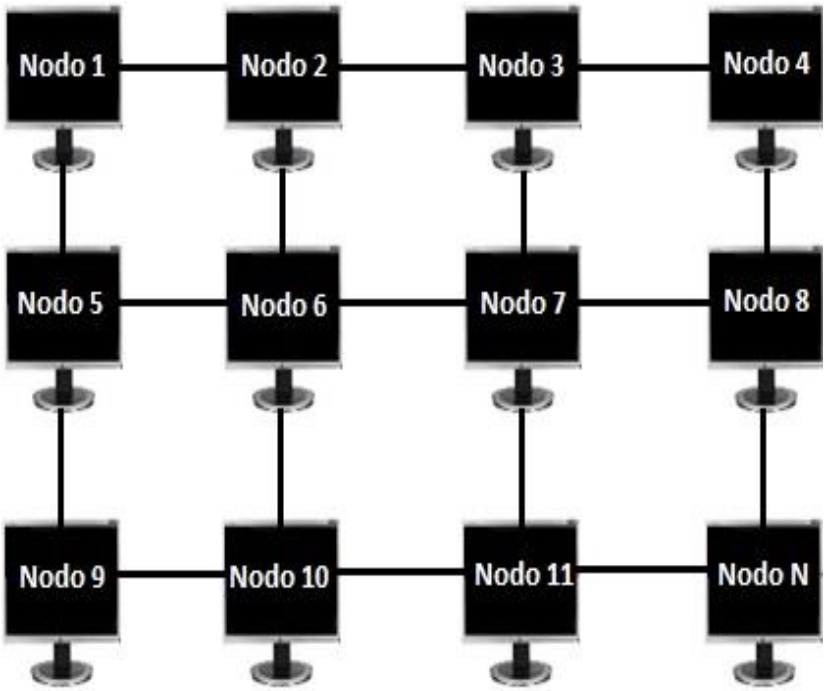


Figura 4 Modelo de grano fino. Fuente: Elaboración propia

Las operaciones genéticas sólo son realizadas entre individuos residentes en procesadores que se comunican directamente de acuerdo a la topología de red.

Este método posibilita la reducción del tiempo computacional por procesador y el empleo de la memoria cache. El algoritmo resulta no ser eficiente cuando es ejecutado con pocos procesadores (Puaut et al., 2018).

1.7.4. Algoritmos genéticos distribuidos híbridos

En estos algoritmos se establece una jerarquía de dos niveles (figura 5). En el nivel superior suele haber siempre un algoritmo de grano grueso, mientras que en el inferior puede existir una implementación de grano fino, grano grueso o maestro esclavo (Chou et al., 2017).

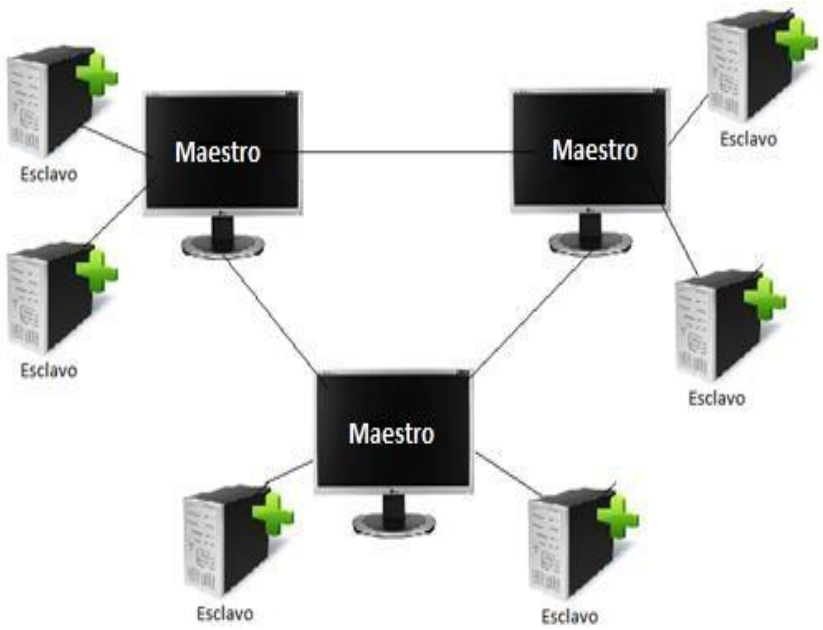


Figura 5 Modelo de implementación híbrida. Fuente: Elaboración propia

Como se puede apreciar en la figura 5 cada maestro constituye un nodo en un esquema de grano grueso. Estos nodos intercambian información con los restantes según se haya establecido el proceso de migración. Por consiguiente, los nodos maestros y sus esclavos, ejecutan el procesamiento de las diferentes subpoblaciones que después serán intercambiadas.

1.8. Conclusiones Parciales

En el marco de la minería de procesos, el algoritmo *Genetic Miner* es uno de los que ofrecen mejor resultado dentro de la fase de descubrimiento de modelos. Después de analizar las etapas que lo conforman, se verifica que alcanzar las condiciones de parada establecidas para este algoritmo, puede requerir que se realice un gran número de iteraciones, lo que implica la construcción de generaciones. Esto provoca que, al igual que la mayoría de los procesos iterativos, presente como principal desventaja un elevado tiempo de respuesta.

El cómputo de adaptabilidad de los individuos es el proceso más costoso computacionalmente. Este, junto con la obtención de las generaciones, puede efectuarse de modo independiente. Lo que permite el uso de técnicas de programación paralela y distribuida con el propósito de disminuir su tiempo de respuesta.

Resulta conveniente implementar el clúster basándose en la combinación de las clasificaciones alto rendimiento y alta disponibilidad. En este sentido se puede establecer una estrategia de distribución híbrida que combine los esquemas grano grueso en un nivel superior y en un nivel inferior asumir la estrategia maestro esclavo. De esta forma, lograr la eficiencia en las operaciones realizadas por el algoritmo además de garantizar la tolerancia a fallos.

Capítulo

2 Fundamentos Teóricos de la Investigación

El presente capítulo se estructura en cuatro epígrafes. El primero está encaminado a detallar los métodos empleados en la investigación. A su vez, el segundo se centra en caracterizar el proceso de producción de langosta estudiado, que se lleva a cabo en la industria pesquera de Batabanó. El tercer epígrafe enuncia las herramientas y tecnologías utilizadas en la implementación del algoritmo. Por último, se establecen los elementos teóricos del *Genetic Miner*.

2.1 Métodos empleados en la investigación

La investigación realizada es descriptiva, ya que permite a los diferentes resultados sustentarse en la descripción, registro, análisis e interpretación de la naturaleza actual de los procesos que integran la producción de langosta. A continuación se describe un conjunto de métodos empleados, agrupados por tipo con el propósito de ganar en organización:

1. Teóricos

- Histórico lógico: se realizó una reseña acerca de las características y utilización de la minería de procesos, así como de las principales técnicas empleadas.
- Análisis y síntesis: consiste en un estudio detallado de las fuentes de información que contienen los procesos de producción de langosta, arribando a los elementos esenciales para descubrir modelos de procesos.

- Enfoque de sistema: se empleó en el estudio de los diferentes componentes que interactúan en el proceso de producción de langosta.
- Inducción-deducción: su uso permitió descubrir modelos de procesos a partir de las fuentes de información que contienen los datos referentes al proceso de producción de langosta.

2. Empíricos

- Consultas a especialistas. Se utilizó para conocer las características del proceso de producción de langosta.
- Análisis documental: se aplicó en la recopilación de datos e información necesaria para el estudio del proceso de producción de langosta.

3. Estadístico-matemáticos

- Estadística descriptiva: se utilizó para el procesamiento de los resultados obtenidos con la aplicación del algoritmo genético, mediante tablas, gráficos y estadígrafos.
- Estadística inferencial: se aplicó la prueba de hipótesis para la comparación de muestras múltiples.

2.2 Caracterización del área de estudio

La presente investigación fue realizada en la Empresa Pesquera Industrial “Camilo Cienfuegos” situada en el municipio Batabanó, provincia Mayabeque, Cuba. Para la experimentación se utilizó la información histórica recopilada sobre el proceso de producción de langosta. En este sentido, se realizaron 20 observaciones en 5 registros de eventos diferentes, generados con los datos pertenecientes a los procesos de producción de cola de langosta cruda, congelada y pelada, así como el de cola de langosta cruda congelada en bloque. Estos procesos son representados en la siguiente figura.

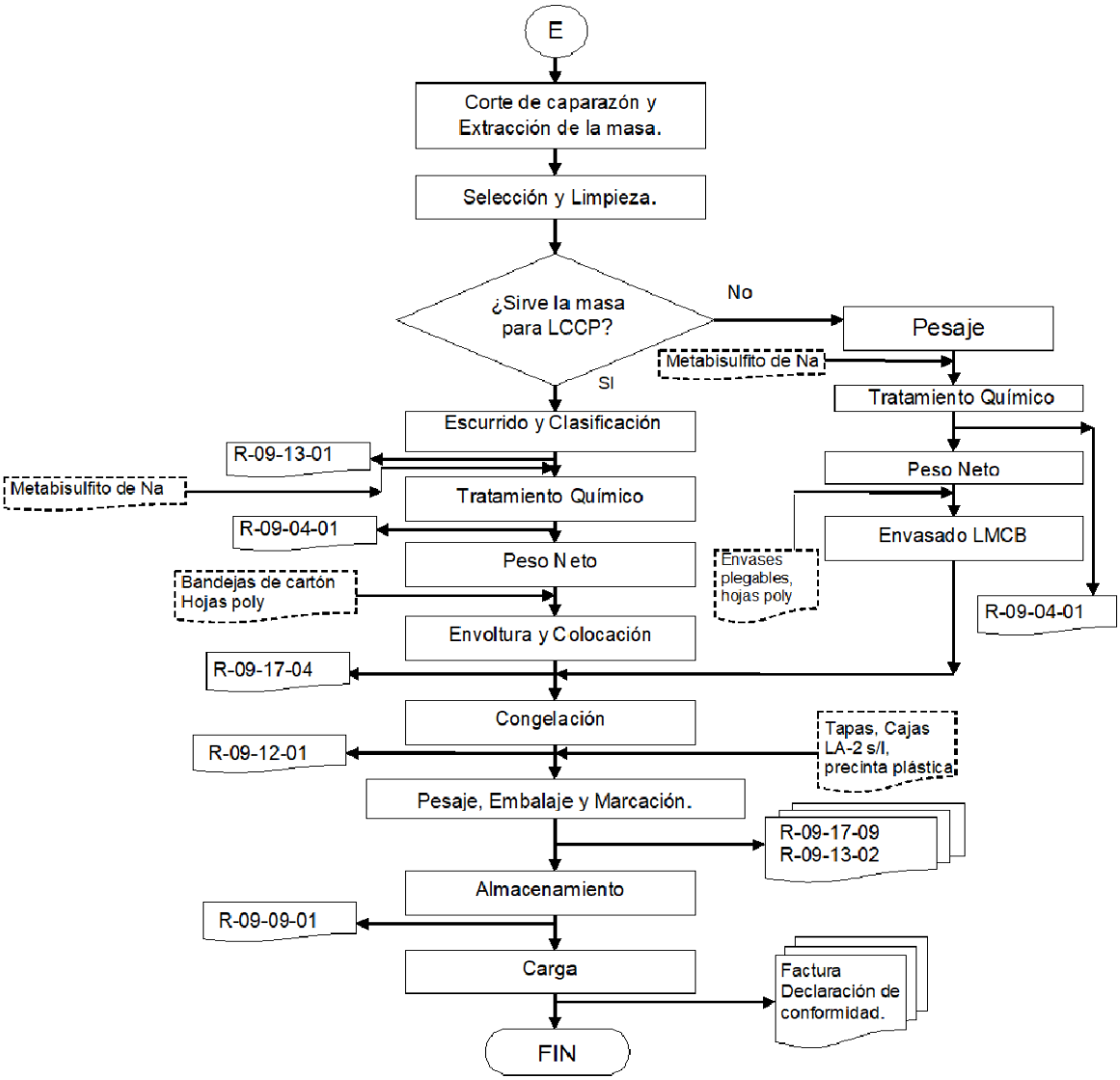


Figura 6 Producción de langosta cola cruda congelada pelada y langosta cola cruda congelada en bloque. Fuente: Elaboración propia

En un inicio, reciben como entrada las colas de langosta crudas y precocinadas que provienen del procesamiento de la langosta entera y colas crudas en los centros de acopio. Es requisito fundamental de la entidad registrar el estado del proceso en cada etapa que lo conforma. Por tal motivo, se han definido grupos de control interno que velan por la correcta ejecución de las actividades mediante modelos que recopilan los datos históricos generados en cada una de ellas (tabla

1). La información contenida en estos modelos constituye la materia prima a analizar. Durante la construcción del registro de eventos, estos datos se traducirían en los atributos que caracaterizan a cada actividad ejecutada en el proceso.

Modelos	Grupos de Control
R-09-04-01/ "Operación de tratamiento químico"	Grupo de Calidad
R-09-09-01/ "Inspección a condiciones de almacenamiento"	Grupo de Calidad
R-09-12-01/ "Control de temperatura"	Grupo de Calidad
R-09-13-01/ "Control de defectos"	Grupo de Calidad
R-09-13-02/ "Control de defectos de producto terminado"	Grupo de Calidad
R-09-17-04/ "Procesos de colas y masas Crudas"	UEB Industria
R-09-17-09/ "Reporte Empaque"	UEB Industria
Facturas y Declaración de Conformidad.	CARIBEX SA, PESCACARIBE

Tabla 1 Modelos de control. Fuente: Elaboración propia

2.3 Tecnologías y herramientas utilizadas para la implementación

El algoritmo fue desarrollado con el leguaje de programación C#, haciendo uso del Entorno de Desarrollo Integrado Visual Studio 2015. Su elección se debe a que para el procesamiento de aplicaciones paralelas, *Microsoft .Net Framework* 4 ha sido provisto de librerías que contienen sofisticados algoritmos, los cuales distribuyen los cálculos dinámicamente en arquitecturas multinúcleo y multiprocesador (TROELSEN and Japikse, 2015). Además, *Visual Studio* 2015 ofrece herramientas de análisis y depuración que apoyan este modelo de programación.

Se empleó *.Net Remoting* para distribuir el algoritmo mediante la invocación de métodos reomotos. Por otro lado, 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. Asimismo, permite administrar las tareas que pueden ser ejecutadas secuencialmente controlando el estado que estas adquieren (Poniszewska Maranda and Wasilewski, 2016).

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* (Olsson, 2018).

El trabajo con tareas permite ganar en eficiencia y escalabilidad de recursos del sistema. En este sentido, 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 muchos hilos para establecer el paralelismo.

2.4 Elementos teóricos del algoritmo *Genetic Miner*

Esta sección introduce brevemente algunas terminologías y notaciones relacionadas con los registros de eventos, así como las definiciones de red de Petri, matriz causal y red de Petri mapeada. Además, se profundiza en el algoritmo *Genetic Miner* a través de sus etapas: procesamiento del registro de eventos, construcción de la población inicial, cálculo de adaptabilidad, obtención de nuevas generaciones aplicando los operadores genéticos y chequeo de condiciones de parada.

2.4.1 Registro de eventos

En sus inicios, la minería de procesos se abastecía de sistemas de bases de datos para la extracción de información vinculada a los procesos en estudio. Por lo general, los conocimientos almacenados carecían de información relevante y al guardar los datos no se cumplía con ninguna jerarquía capaz de organizarlos. Por tal motivo, se comenzó a utilizar una nueva forma de estructuración jerárquica basada en registros de eventos.

Este tipo de registros es utilizado por algunos sistemas de información para extraer datos acerca de sus actividades y relaciones causales. Los *logs* están constituidos por secuencias de acciones (***eventos***) que son utilizadas como entrada para la minería de procesos (Kalenkova et al., 2017). Cada secuencia finita de eventos representa una ***traza*** que describe la ejecución de una instancia específica de un proceso. Los eventos pueden ser descritos por una o varias propiedades: denominadas ***atributos***, que contienen un determinado valor asociado y pueden referirse al inicio, conclusión, cancelación, etc., de una actividad para una instancia específica del proceso. A continuación, se brinda un conjunto de definiciones formales tomadas de (O'Reilly et al., 2006).

Definición 1 (Actividad). *Una actividad o tarea es una unidad atómica lógica de trabajo.*

Definición 2 (Traza). *Sea* A *un conjunto de actividades. Una traza* t *se define como una tupla* (id, s) *donde:*

- $id \in PI_{id}$[2] *es el identificador de una instancia en un proceso;*
- $s \in A^*$ *es la secuencia de actividades ejecutadas.*

[2] Asúmase que $PIid$ es el universo de todas las instancias posibles del proceso.

Definición 3 (Atributo). *Es una propiedad definida que puede describir los eventos, las trazas o los registros de eventos y contiene un determinado valor asociado.*

Definición 4 (Registro de eventos). *Un registro de eventos* $L \subseteq T_{inst}$[3] *es un conjunto de trazas, tal que si* $t_1 = (id_1, s_1)$ *y* $t_2 = (id_2, s_2)$ *con* $id_1 = id_2$ *entonces* $t_1 = t_2$*, es decir, las instancias (*id_s*) del proceso son únicas.*

Definición 5 (Modelo de procesos). *Un modelo de procesos representa las dependencias entre las actividades que son ejecutadas y las posibles alternativas del proceso. Estos son obtenidos a partir de los registros de eventos.*

2.4.2 Representación de un individuo

En el algoritmo *Genetic Miner* cada individuo constituye un modelo diferente del proceso que se quiere recuperar. Estos se agrupan en poblaciones, donde pueden evolucionar o mantenerse en la medida que avanza el algoritmo. Cada individuo intentará representar todas o la mayor cantidad de dependencias entre las actividades presentes en el *log*, o de forma equivalente, almacenar cuáles habilitan la ejecución de otras y de qué manera lo hacen.

Al finalizar la ejecución de una tarea en una traza pueden ser iniciadas una o varias actividades, cuya relación está establecida por un conector asociado, el cual varía dependiendo de la forma en que se ejecuten dichas actividades. Por ejemplo, si se ejecutan dos o más de manera simultánea, el conector asociado es conocido como *AND-SPLIT*, en caso de ejecutarse solamente una, entre todas las posibles, se está en presencia de un *OR-SPLIT*. Si una tarea precedida por otras que se ejecutan simultáneamente, tiene que esperar a que sus predecesoras concluyan para ejecutarse, el conector presente es un *AND-JOIN*. De lo contrario, si la tarea es

[3] T_{inst} es el universo de las trazas.

habilitada con la ejecución de alguna de sus predecesoras directas, se tiene un *OR-JOIN* (Banzhaf et al., 2019).

El algoritmo *Genetic Miner* durante su ejecución maneja simultáneamente dos modelos de procesos para representar a los individuos: redes de Petri y matrices causales, que se transforman convenientemente (Alves De Medeiros and Weijters, 2005).

2.4.2.1 Red de Petri

Una red de Petri es un modelo gráfico, formal y abstracto para describir y analizar un flujo de información. Permite la visualización topológica de un sistema concurrente, paralelo o distribuido de forma aceptable. Según se ofrece en la definición formal de red de Petri tomada de (Badouel et al., 2015).

Definición 6 (red de Petri). *Es una tripleta* $N = (P, T, F)$ *donde* P *es un conjunto finito de estados*[4] o lugares*,* T *es un conjunto finito de transiciones*[5] *tal que* $P \cap T = \emptyset$*, y* F *es un conjunto finito de arcos dirigidos tal que* $F \subseteq (P \times T) \cup (T \times P)$*.*

Para construir una red de Petri es necesario determinar el número de lugares en cada individuo, aspecto no esencial a los efectos del algoritmo *Genetic Miner*. Resulta más importante la representación de dependencias entre tareas, así como la semántica de los constructores asociados a las mismas. Además, la aplicación de los operadores genéticos sobre el grafo que representa una red de Petri resulta compleja al tener que manejar transiciones, lugares y sus posibles relaciones sin violar la consistencia del modelo.

En este sentido, la autora del algoritmo propone como representación la matriz causal por ser más expresiva que la red de Petri desde la perspectiva de representar dependencias entre tareas, centrándose en estas únicamente. A la matriz causal se le conoce además por red heurística.

[4] Los estados representan condiciones bajo las cuales se puede ejecutar una instrucción.
[5] Las transiciones (eventos) son las instrucciones.

2.4.2.2 Matriz causal

La matriz causal muestra cuáles actividades habilitan la ejecución de otras a partir de las funciones de entrada y salida (*Input Function* y *Output Function*) que almacenan respectivamente las actividades que preceden y suceden a una actividad dada. Las funciones de entrada (I) y salida (O) son dos conjuntos de conjuntos que contienen a las actividades del modelo. Según se ofrece en la definición formal de matriz causal tomada de (Alves De Medeiros and Weijters, 2005).

Definición 7 (Matriz causal). *Sea* LS *un conjunto de etiquetas. Una Matriz Causal es una tupla* $CM = (A, C, I, O, Label)$*, donde: -* A *es un conjunto finito de actividades,*

- $C \subseteq A \times A$ *es la relación de causalidad,*
- $I : A \to P(P(A))$ *es la función que establece las condiciones de entrada,*
- $O : A \to P(P(A))$ *es la función que establece las condiciones de salida,*
- $Label \in A \to LS$ *es una función que asocia a cada actividad* A *una etiqueta en LS.*

De modo que:

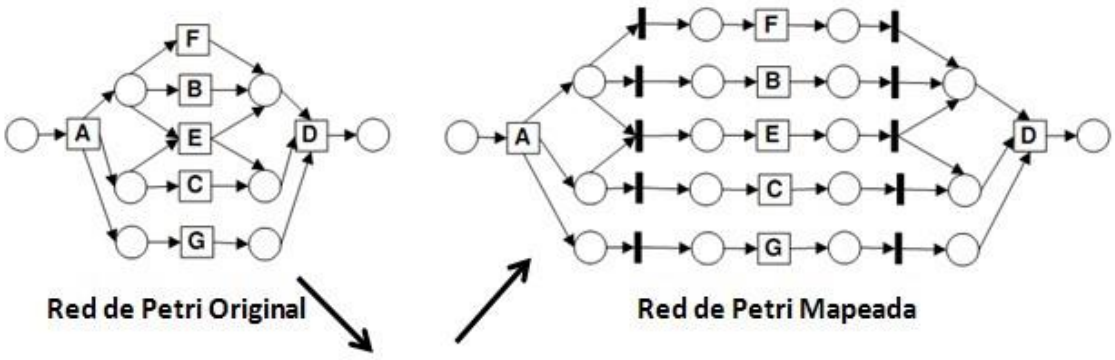
- $C = \{(a_1, a_2) \in A \times A \mid a_1 \in \bigcup I\ (a_2)\}$,
- $C = \{(a_1, a_2) \in A \times A \mid a_2 \in \bigcup O\ (a_1)\}$,
- $C \cup \{(a_0, a_i)$[6] $\in A \times A \mid I(a_0) = \emptyset \wedge O(\ a_i) = \emptyset\}$ *es un grafo fuertemente conexo,*

$Label \in A \to LS$ *es una función inyectiva, lo que quiere decir que si* $a_1, a_2 \in A$ *y* $Label(\ a_1) = Label(\ a_2)$ *entonces* $a_1 = a_2$.

Las actividades contenidas en un mismo subconjunto están relacionadas por una disyunción (*OR*) mientras que los diferentes subconjuntos establecen una relación de conjunción (*AND*). De esta manera, cada función $I(Actividad)$, $O(Actividad)$ representa una conjunción de disyunciones exclusivas. Una actividad puede aparecer en uno o más subconjuntos. Toda matriz causal debe cumplir además, que

[6] a_0, a_i representan respectivamente a actividades que inician y concluyen el proceso.

dadas dos actividades $x; y, x \in I(y)\ si\ y \in O(x)$. En la figura 7 se ilustra como la matriz causal representa las mismas dependencias entre tareas que la red de Petri original.



ACTIVITY	I(ACTIVITY)	O(ACTIVITY)
A	{}	{{F,B,E},{E,C},{G}}
B	{{A}}	{{D}}
C	{{A}}	{{D}}
D	{{F,B,E},{E,C},{G}}	{}
E	{{A}}	{{D}}
F	{{A}}	{{D}}
G	{{A}}	{{D}}

Matriz Causal o Red Heurística

Figura 7 Relación entre red de Petri y matriz causal. Fuente: (Alves De Medeiros and Weijters, 2005)

Por ejemplo, considérese la actividad D en la red de Petri original de la figura 7. La matriz causal establece que $(D) = \{\{F, B, E\}, \{E, C\}, \{G\}\}$ porque D está habilitada por una construcción *AND-JOIN* que tiene tres lugares. Nótese que $I(A) = \{\}$ y $O(D) = \{\}$ porque A comienza el proceso al no estar precedida por ninguna actividad, mientras que D no está sucedida por actividad alguna y termina el proceso.

La aplicación de los operadores genéticos sobre matrices causales, resulta un proceso mucho más simple, con relación a su modelo equivalente visualizado como red de Petri, solo hay que adicionar o eliminar elementos de conjuntos. Por otra parte, resulta complejo efectuar el cálculo de adaptabilidad de un individuo, pues se necesita reproducir la ejecución de un proceso según las secuencias de eventos que ocurren en las trazas. Esta reproducción se obtiene de manera natural colocando marcas en los lugares de una red de Petri, de ahí la necesidad del algoritmo de transformar un modelo en otro según el paso que se esté ejecutando.

2.4.2.3 Red de Petri mapeada

El proceso de transformación de una matriz causal en una red de Petri puede provocar que el modelo resultante sea inconsistente (figura 8). Esto se debe a que la matriz causal es mucho más expresiva, por lo que la información que almacena relacionada con las dependencias entre actividades causa un cambio en la lógica de la red de Petri. Por tal motivo, resulta necesario incorporar transiciones ocultas; obteniéndose una estructura denominada red de Petri mapeada.

Actividad	*Entrada*	*Salida*
A	$\{\}$	$\{\{C, D\}\}$
B	$\{\}$	$\{\{D\}\}$
C	$\{\{A\}\}$	$\{\}$
D	$\{\{A, B\}\}$	$\{\}$

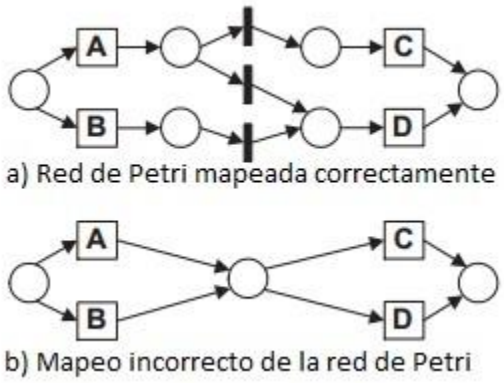


Figura 8 Mapeo en una red de Petri Fuente: (Alves De Medeiros and Weijters, 2005)

Como se evidencia en la figura 8(a), el mapeo de la red de Petri mantiene la lógica de la matriz causal. Si se eliminasen las transiciones ocultas, el modelo resultante (figura 8(b)) no concuerda con las relaciones de dependencia almacenadas en la matriz causal. Note que, al ser ejecutada B son activadas las transiciones C y D; cuestión que no tiene sentido, pues C no pertenece al conjunto de salida de B ni B pertenece al conjunto de entrada de C.

2.4.2.4 Transformación de una red de Petri a una matriz causal

A continuación, se explica brevemente cómo realizar el proceso de conversión de una red de Petri a su correspondiente matriz causal:

- Todas las transiciones de la red de Petri son agregadas al conjunto de actividades de la matriz causal.

- Se dice que existe una relación causal entre dos transiciones t_1, t_2 de la red de Petri si la intersección entre los lugares de salida de t_1 y los lugares de entrada de t_2 no es vacía.
- Para cada transición t su función de entrada se construye de la siguiente manera:
    1. Construir un subconjunto en $I(t)$ por cada lugar de entrada t.
    2. En cada subconjunto se adicionarán las transiciones de entrada del lugar p correspondiente.
- Para cada transición t, su función de salida $O(t)$ se construye de la siguiente manera:
    1. Construir un subconjunto en $O(t)$ por cada lugar de salida t.
    2. En cada subconjunto se adicionarán las transiciones de salida del lugar p correspondiente.

2.4.2.5 Mapeo de una matriz causal a una red de Petri

Conceptualmente la matriz causal se comporta como una red de Petri que contiene tareas visibles e invisibles. El proceso de conversión de una matriz causal a una red de Petri mapeada resulta complejo, pues es necesario inferir lugares para cada actividad. Más adelante se describen brevemente los pasos a seguir para obtener una red de Petri mapeada partiendo de una matriz causal (Alves De Medeiros and Weijters, 2005):

- Se crean tantas transiciones como tareas existan en la matriz causal (una transición por cada actividad).
- Se crea un lugar inicial i y un lugar final o.
- Por cada transición (actividad) cuya $I(t) \neq \emptyset$ se crea un lugar de entrada i_t por cada subconjunto en I.
- Por cada transición (actividad) cuya $O(t) \neq \emptyset$ se crea un lugar de salida o_t por cada subconjunto en O.
- Se agrega un arco entre i y todas las transiciones cuya $I(t) = \emptyset$.
- Se agrega un arco entre todas las transiciones cuya $O(t) = \emptyset$ y o.

- Se agrega un arco entre cada transición y su lugar de salida en caso de existir.
- Se agrega un arco entre cada lugar de entrada (en caso de existir) y su respectiva transición.
- Por cada relación causal presente en la matriz causal se crea una transición invisible $m_{t_1t_2}$ (las relaciones causales se detectan recorriendo la columna correspondiente a I u O).
- Si entre dos tareas t_1, t_2 existe una relación causal ($t_2 \in O(t_1)$ o $t_1 \in I(t_2)$) se agregan dos arcos, el primero de o_{t_1} a $m_{t_1t_2}$ y el segundo de $m_{t_1t_2}$ a i_{t_2} donde $m_{t_1t_2}$ es la transición invisible creada para una relación causal entre t_1 y t_2.

2.4.3 Fases del algoritmo *Genetic Miner*

De manera resumida el algoritmo genético implementado se puede visualizar en la secuencia de 5 pasos (figura 9).

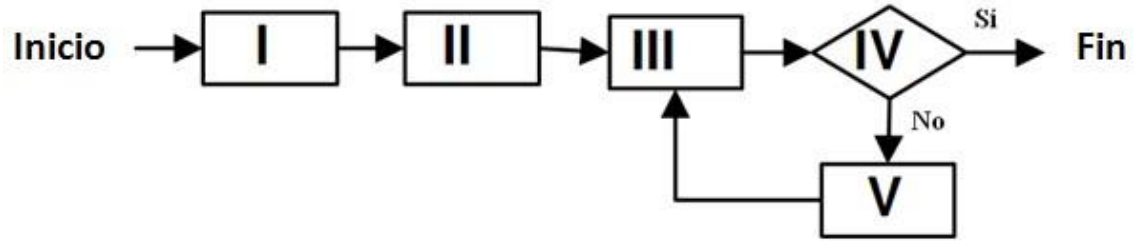


Figura 9 Pasos del algoritmo Genetic Miner. Fuente: (Alves De Medeiros and Weijters, 2005)

Donde los pasos se corresponden con las funcionalidades que se describen a continuación:

I- Lectura del registro de eventos recibido como entrada.

II- Construcción de la población inicial a través de una heurística.

III- Cálculo de la adaptabilidad (*fitness*) de cada individuo en la población.

IV- Chequeo de condiciones de parada.

V- Obtención de la siguiente generación.

2.4.3.1 Lectura del Log recibido como entrada

Al igual que en la implementación del *ProM*, los ficheros (registros de eventos) que se están procesando tienen formato *.XES*. Para garantizar el correcto funcionamiento del algoritmo, se extraen del registro de eventos recibido como entrada las actividades y su estado (comenzada, terminada, etc.), así como el orden en que aparece cada actividad en cada una de las trazas. El resto de la información almacenada en el fichero no es relevante (para el algoritmo genético) y por tanto no es almacenada en memoria.

2.4.3.2 Obtención de la población inicial a través de una heurística

Dada la información contenida en un registro de eventos, todos los individuos en una población del algoritmo genético tienen el mismo conjunto de actividades (una actividad por cada tarea presente en el *log*). Para la construcción de la población inicial se utiliza una heurística que establece la siguiente relación: mientras más frecuente ocurra que una actividad a_1 esté directamente sucedida por otra actividad a_2 (la subtraza a_1a_2 aparece en el *log*) es más probable que exista la relación causal a_1a_2 (Alves De Medeiros and Weijters, 2005).

Una vez que las relaciones causales son determinadas, las funciones I y O se construyen aleatoriamente. Para ello se establece un tamaño máximo para los

conjuntos I y O de cada actividad que no excederá el número de actividades presentes en el registro de eventos. Cada actividad a_1 que precede causalmente a una actividad a_2 es insertada aleatoriamente en uno o más subconjuntos de (a_2). Similar proceso se realiza para la función O.

La heurística construye la población inicial basada en un índice de dependencia que toma en consideración información local almacenada en el *log*. Este índice de dependencia indica cuán fuertemente una tarea depende de otra. Mientras más frecuente la tarea t_1 preceda directamente a la tarea t_2 en el registro de eventos y menos frecuente t_2 preceda a t_1, mayor es la dependencia entre t_1 y t_2, lo que quiere decir que es más probable que t_1 provoque a t_2.

Para la construcción del índice de dependencia se recorren cada una de las trazas presentes en el *log* y se construyen las funciones (Alves De Medeiros and Weijters, 2005):

1. $bucle2: TxTxL \longrightarrow \mathbb{N}$: detecta cuántas veces una tarea está involucrada en un ciclo de longitud 2. Es decir, $bucle2(t_1, t_2, L)$ devuelve la cantidad de veces que la subcadena $t_1t_2t_1$ aparece en el registro de eventos.

2. $follow: TxTxL \longrightarrow \mathbb{N}$: detecta cuántas veces una tarea está sucedida directamente por otra. Es decir $follow(t_1, t_2, L)$ devuelve cuántas veces la subcadena t_1t_2 aparece en el registro de eventos.

El índice de dependencia es una función $D: TxTxL \rightarrow \mathbb{R}$ (donde T es el conjunto de las tareas y L el registro de eventos) se calcula de la siguiente manera:

$$D(t_1, t_2, L) = \begin{cases} \dfrac{bucle2(t_1, t_2, L) + bucle2(t_2, t_1, L)}{bucle2(t_1, t_2, L) + bucle2(t_2, t_1, L) + 1} & Si\ t_1 \neq t_2\ y\ bucle2(t_1, t_2, L) > 0 \\ \dfrac{follow(t_1, t_2, L) - follow(t_2, t_1, L)}{follow(t_1, t_2, L) + follow(t_2, t_1, L) + 1} & Si\ t_1 \neq t_2\ y\ bucle2(t_1, t_2, L) = 0 \\ \dfrac{follow(t_1, t_2, L)}{follow(t_1, t_2, L) + 1} & Si\ t_1 = t_2 \end{cases}$$

Al sumar 1 en los denominadores se benefician las observaciones más frecuentes sobre las menos frecuentes.

El tamaño de las poblaciones es fijo en todas las iteraciones del algoritmo. Este número de individuos, se establece como parte de la entrada dependiendo de las condiciones establecidas por los datos.

2.4.3.3 Cálculo de la adaptabilidad de cada individuo en la población

Para determinar la adaptabilidad (*fitness*) de cada individuo se transforma la matriz causal en su respectiva red de Petri mapeada. Esta medida de adaptabilidad está basada en el parseo de trazas. Una traza es parseada correctamente por un individuo, si para una marca inicial con un solo *token* ubicado en el lugar inicial de la red de Petri mapeada, después de ser ejecutadas las tareas visibles en el orden en que aparecen en la traza, el lugar final es el único marcado y con un solo *token*. En la figura 10 se ilustra el mecanismo de disparo de una traza:

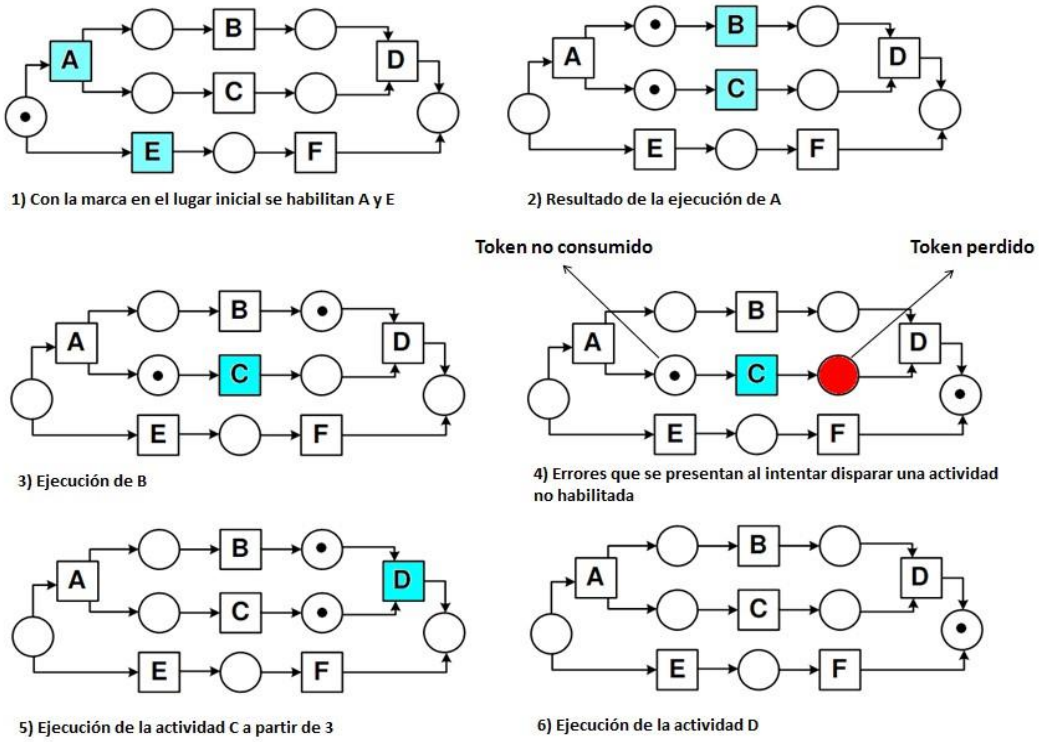


Figura 10 Mecanismo de disparo de una traza. Fuente: (Alves De Medeiros and Weijters, 2005)

El proceso de parseo visualizado en la figura 10, se corresponde con una traza formada por las actividades A, B, C y D almacenadas en ese orden. Este proceso se describe en los 6 pasos ilustrados anteriormente:

1) Inicialmente se coloca una marca (*token*) en el lugar inicial de la red de Petri habilitándose las actividades A y E. Una actividad se dice habilitada si todos sus lugares de entrada se encuentran marcados.

2) Solamente una de las dos actividades puede ejecutarse, en este caso A. Cuando una actividad se ejecuta elimina un *token* de cada uno de sus lugares de entrada y coloca un *token* en cada uno de sus lugares de salida. Nótese como una vez ejecutada A se habilitan las actividades B y C, que se ejecutan en paralelo pues se está en presencia de un *AND-SPLIT*. La actividad E deja de estar habilitada por haber sido retirado el *token* que se encontraba en el lugar inicial. Si seguidamente se disparara la actividad B se obtiene el resultado de la figura 10 3).

3) Nótese como solamente se encuentra habilitada C, aunque uno de los lugares de entrada de D tiene un *token*, para que esta última se habilite tienen que estar marcados todos sus lugares de entrada. Si se ejecutara la actividad D se produciría la situación errónea mostrada en la figura 10 4).

4) Si D fuera la última actividad en la traza que se está parseando, entonces la instancia actual se ejecutaría incorrectamente, pues se detectó un *token* perdido que era necesario para que la actividad D se ejecutara correctamente. Además, se encontró un *token* no consumido, dejado en un lugar de entrada de la actividad C. En caso de que D no fuera la última actividad en la traza a parsear, se registran los errores detectados y se continúa el parseo como si nada hubiera sucedido.

5) Asúmase ahora que D no se ejecutó, volviendo a la situación 3. Si según la traza le correspondiera disparar a C se obtendría el resultado de la figura 10 5), con el cual la actividad D quedaría habilitada.

6) Finalmente, al ejecutarse la actividad D la ejecución de la traza ABCD sobre el modelo culminaría correctamente.

Para determinar el *fitness* de cada individuo se aplicará el mecanismo de disparo antes descrito para cada una de sus trazas, registrándose los siguientes datos:

- Total de actividades en el registro de eventos.
- Número de actividades ejecutadas correctamente.
- Número de *tokens* perdidos.
- Número de *tokens* que quedaron en la red después de terminada la ejecución.
- Total de trazas en el registro de eventos.
- Número de trazas ejecutadas correctamente.
- Número de trazas donde se perdieron *tokens.*
- Número de trazas donde quedaron *tokens.*

Con estos datos se aplica la fórmula:

$$PF_{complete} = \frac{\#Actividades\ ejecutadas\ Correctamente - penalidad}{total\ de\ actividades\ en\ el\ log}$$

Donde:

$$penalidad = \frac{número\ de\ tokens\ perdidos}{total\ de\ trazas\ en\ el\ log - trazas\ donde\ se\ perdieron\ tokens + 1} + \frac{número\ de\ tokens\ no\ consumidos}{total\ de\ trazas\ en\ el\ log - trazas\ con\ tokens\ no\ consumidos + 1}$$

Esta medida de adaptabilidad es denominada *Continuous Semantics* en la herramienta *ProM*, e indica cuán completo el individuo reproduce las trazas presentes en el registro de eventos.

2.4.3.4 Condiciones de parada

En la versión actual de la implementación se están utilizando tres condiciones de parada, aunque se pueden agregar todas las que sean necesarias tomando en consideración situaciones que se puedan presentar. Estas condiciones son:

- Se realiza un número máximo de iteraciones prefijadas n.
- No ha variado el individuo más adaptado en las últimas $n/2$ iteraciones.
- El mejor individuo de la población alcanzó un valor suficientemente bueno (muy cercano a 1).

2.4.3.5 Obtención de la siguiente generación

Una vez chequeadas las condiciones de parada y si no se cumple ninguna, entonces resulta necesario que la población actual evolucione. Una parte de los individuos son llevados directamente a la siguiente generación, la élite de la población actual, que no es más que el conjunto de individuos con mayor *fitness*.

Para determinar los restantes, necesarios para completar la población se aplican dos operadores genéticos: cruzamiento y mutación los cuales se explican a continuación (Alves De Medeiros and Weijters, 2005):

Entrada: Dos individuos ($parent1$ y $parent2$), un índice de cruzamiento

Salida: Dos posibles descendientes recombinados ($offspring1$ y $offspring2$)

1. $offspring1 \longleftarrow parent1$, $offspring2 \longleftarrow parent2$
2. Con probabilidad establecida por el índice de cruzamiento hacer:
   a) Seleccionar aleatoriamente una actividad a como punto de cruzamiento de los descendientes.
   b) Seleccionar aleatoriamente un punto de intercambio sp_1 para $I_1(a)$ (para el $offspring1$). El punto de intercambio está entre 0 y $n-1$, donde n es el número de subconjuntos en la función $I_1(a)$.

c) Seleccionar aleatoriamente un punto de intercambio sp_2 para $I_2(a)$.

d) Construir el subconjunto $remainingSet_1(a)$ formado por los subconjuntos en $I_1(a)$ que se encuentran entre las posiciones $[0, sp_1)$.

e) Construir el conjunto $swapSet_1(a)$ formado por los subconjuntos en $I_1(a)$ cuyas posiciones se encuentren en el intervalo $[sp_1, n)$.

f) Repetir los pasos d) y e) para construir los conjuntos $remainingSet_2(a)$ y $swapSet_2(a)$ a partir de sp_2 e $I_2(a)$.

g) Para cada subconjunto S_2 en $swapSet_2(a)$ hacer:

   i. Ejecutar alguno de los siguientes pasos con igual probabilidad:

      A. Adicionar S_2 como nuevo subconjunto en $remainingSet_1(a)$.

      B. Unir S_2 con un subconjunto X_1 existente en $remainingSet_1(a)$.

      C. Seleccionar un subconjunto X_1 en $remainingSet_1(a)$, eliminar los elementos de X_1 que se encuentran en S_2 y adicionar S_2 a $remainingSet_1(a)$.

h) Repetir el paso g) pero utilizando S_1, $swapSet_1(a)$, $remainingSet_2(a)$ y X_2.

i) $I_1(a) \leftarrow remainingSet_1(a)$ e $I_2(a) \leftarrow remainingSet_2(a)$

j) Repetir los pasos b) hasta h) usando ahora (a).

3. Retornar $offspring1$ y $offspring2$.

El índice de cruzamiento determina la probabilidad de que dos padres experimenten este operador. Después de hacer un cruzamiento el número de relaciones causales permanece constante pero las relaciones que aparecen en los descendientes son diferentes a las de sus padres. Nótese que dos padres siempre generan dos descendientes. Una vez obtenidos estos últimos se les aplica el operador mutación.

Entrada: Un individuo, un índice de mutación

Salida: Un individuo posiblemente mutado

1. Por cada actividad en el individuo hacer:
    a) Según la probabilidad dada por el índice de mutación, realizar alguna de las siguientes operaciones para la función $I(a)$:
        i. Seleccionar un subconjunto X en $I(a)$ y adicionar una actividad a' en X, donde a' pertenece al conjunto de actividades del individuo.
        ii. Seleccionar un subconjunto X en $I(a)$ y eliminar una actividad a' de X, donde $a' \in X$. Si después de extraer a', X se queda vacío entonces se eliminará X de $I(a)$.
        iii. Redistribuir los elementos en $I(a)$:
            A. Obtener una lista l con todos los elementos en los subconjuntos de $I(a)$.
            B. Eliminar todos los subconjuntos de $I(a)$.
            C. Crear n conjuntos ($1 \leq n \leq |l|$) y distribuir aleatoriamente los elementos de l en los n conjuntos. D. Insertar los n conjuntos en $I(a)$.
    b) Repetir el paso a) para la función $O(a)$.

Como las operaciones en ambos operadores se realizan de manera aleatoria se pueden producir inconsistencias, como por ejemplo que dadas dos actividades a, b ocurra que $a \in I(y)\ y\ b \notin O(a)$ violando la definición de matriz causal y que debe ser corregida. En la tesis doctoral de Ana Karla Alves, se hace referencia a una operación que actualiza las actividades que tuvieron problemas, esto implica un recorrido de los subconjuntos de las actividades involucradas. En la implementación actual, cuando una actividad es modificada dentro de la matriz causal, las posibles inconsistencias son corregidas en el momento, con lo que se evita el recorrido antes mencionado.

2.5. Conclusiones Parciales

En este capítulo se especificó el caso de estudio relacionado con los procesos de producción de cola de langosta cruda, congelada y pelada, así como el de cola de langosta cruda congelada en bloque. Además, se definieron las herramientas y tecnologías a utilizar en la implementación del algoritmo. Por otro lado, se introdujeron las terminologías y notaciones relacionadas con los algoritmos genéticos, así como las heurísticas que rigen el cálculo de adaptabilidad de los individuos.

Capítulo

3 Resultados y Discusión

En este capítulo se describen cuestiones relacionadas con la implementación y pruebas de los resultados alcanzados en la investigación. No se profundiza en detalles técnicos del código, ni se ofrece un manual de usuario para emplear la librería obtenida, sino que se abordan aspectos considerados en la implementación que influyen directamente en los resultados. A su vez, se muestran las comprobaciones realizadas para evaluar la estrategia propuesta.

En un inicio, los experimentos se centrarán en analizar la precisión alcanzada por el algoritmo, con respecto a las obtenidas en versiones anteriores del mismo. Posteriormente, se realizarán pruebas sobre el tiempo de respuesta adquirido por la nueva versión en comparación con las soluciones anteriores. Además, se determinará el comportamiento del algoritmo tras la incrementación de nodos de cómputo. Por último, la nueva variante será sometida a entornos controlados con el fin de determinar cuán tolerante a fallos puede resultar.

3.1 Consideraciones generales de la implementación

Como resultado de este trabajo se obtuvo una librería implementada en el lenguaje de programación C#, sobre el *framework .NET* 4.0. No se ofrece una salida gráfica del modelo obtenido, por tal motivo, las pruebas fueron realizadas mediante la entrada y salida por consola. Esta librería puede incorporarse en aplicaciones que realicen minería de procesos, que serán las encargadas de visualizar el modelo descubierto o interpretarlo según el escenario en que se aplique.

Para la creación de esta nueva variante del algoritmo se utilizó una estrategia híbrida (figura 11), constituida por una jerarquía de tres niveles. En el nivel superior se manifiesta una modificación a la estructura de grano grueso, con el propósito de otorgarle autonomía propia a los nodos. De esta forma, se garantiza la no dependencia de un ordenador central para coordinar las funciones llevadas a cabo por el algoritmo. Se parte de un nodo para ejecutar la construcción de las poblaciones, así como la asignación y el control de estas hacia los nodos esclavos. A su vez, los demás velan por el buen funcionamiento del algoritmo; monitorizando constantemente la disponibilidad del actual nodo central. De detectar un fallo, se notifica a los restantantes nodos sobre la sustitución del coordinador, quien da continuidad al proceso iterativo.

La preservación de la información obtenida tras las iteraciones realizadas por el algoritmo, constituye un requisito imprescindible para garantizar la calidad y eficiencia de la arquitectura distribuida propuesta en esta investigación. En este sentido, se establecen mecanismos de replicación entre los nodos mediante la invocación de objetos remotos. Una vez efectuada la evolución de la población, el actual coordinador envía la información necesaria hacia sus semejantes, garantizando que tanto el número de iteraciones alcanzadas, el individuo más adaptado hasta el momento y la última población generada, no desaparezcan tras la interrupción del nodo central; siendo posible continuar con la evolución de las poblaciones hasta alcanzar alguna de las condiciones de parada establecidas.

En el nivel intermedio rige un modelo maestro-esclavo, donde cada nodo ejecuta un proceso principal que controla la evolución de las poblaciones. Asimismo, el ordenador central se encarga de ejecutar los operadores genéticos cruzamiento y mutación, así como la selección de la élite de modo centralizado para la obtención de nuevas poblaciones. Además, coordina la distribución de cada instancia del proceso hacia los nodos esclavos para su evaluación. De manera simultánea en un tercer nivel, se aplica paralelismo sobre los individuos para obtener la adaptabilidad de cada uno, devolviendo al hilo principal el resultado alcanzado. Este enfoque brinda la posibilidad de explorar distintas partes del espacio de búsqueda.

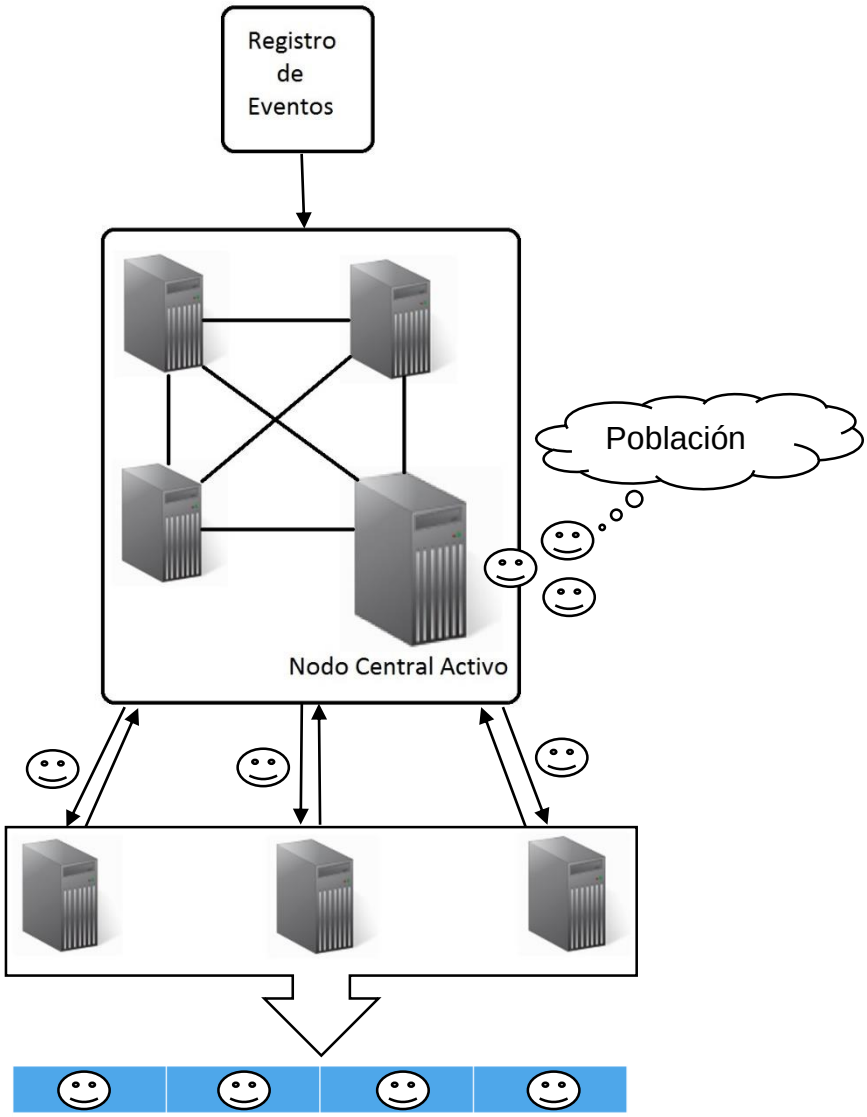


Figura 11 Estrategia de distribución seleccionada. Fuente: Elaboración propia

3.2. Paralelismo en .Net

Siguiendo el proceso de diseño de un sistema distribuido y aprovechando las características de .*Net Framework* (TROELSEN and Japikse, 2015), se implementó el algoritmo teniendo en cuenta la siguiente estructura:

1) ***Comunicación de red, protocolos e interfaz***. Para llevar a cabo la interacción entre las aplicaciones se empleó .*Net Remoting*, 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.
2) ***Arquitectura y servicios del sistema distribuido***. 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 12).

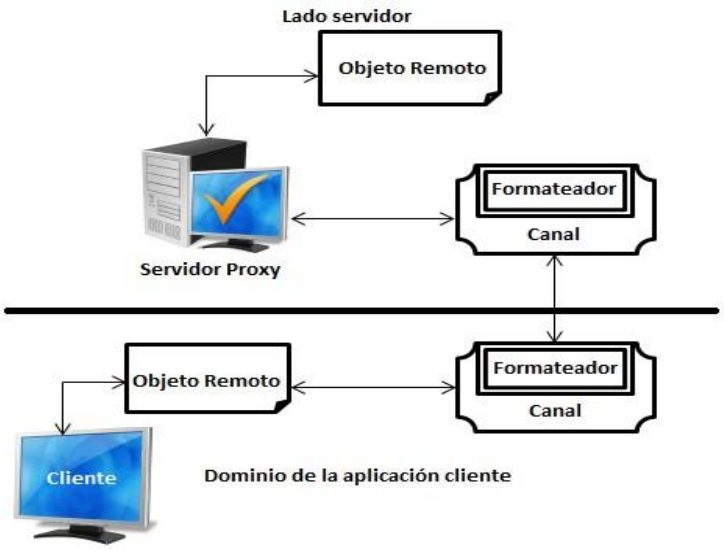


Figura 12 Vista del entorno remoto. Fuente: Elaboración propia

Como se muestra en la figura 12, 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. A su vez, debe contener 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.

3) ***Paradigmas de la computación distribuida***. Se concibió el modelo de cálculo establecido mediante la integración de los paradigmas paralelismo funcional y paralelismo de datos. En este punto, la aplicación de los operadores genéticos de elitismo, así como las funciones de cruzamiento y mutación, son efectuadas por el maestro de modo centralizado en el ordenador central, mientras que los nodos esclavos realizan el cálculo de adaptabilidad de los individuos aplicando paralelismo de datos. Además, se concibió un modelo de comunicación por memoria distribuida y paso de mensajes.

3.3. Pseudocódigo de la solución propuesta

La implementación de la nueva versión distribuida del *Genetic Miner 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 general y período evolución*

Salida: *Matriz causal*

1. *Leer y procesar el registro de eventos*
2. *Crear conexiones remotas*
3. *Si es el nodo central activo*
4. *Construir subpoblaciones y distribuirlas por los nodos*
5. *Iniciar la evolución de la subpoblación en cada nodo*
6. *Mientras no se cumplan las condiciones de parada*
7. *Construir población global uniendo las últimas subpoblaciones generadas por cada nodo*
8. *Insertar mejores individuos en todas las subpoblaciones*
9. *Obtener una nueva generación en la población global*
10. *Replicar la información hacia los restantes nodos*
11. *Devuelve el mejor individuo (Matriz Causal) de la población resultante*
12. *Si no es nodo central activo*
13. Cada cierto intervalo de tiempo
14. Verificar disponibilidad del coordinador
15. Si no está disponible
16. Cambiar nodo actual a central
17. Notificar a los restantes nodos
18. Volver al paso 5

Inicialmente se extraen los datos almacenados en el fichero con formato *.XES* recibido por parámetros y se agrupa en eventos y trazas para construir el *log* a utilizar por el algoritmo. Luego se establecen las conexines remotas entre los nodos que conforman el clúster. Concluyendo esta fase se genera una matriz causal a partir de la información contenida en el registro de eventos. Estas operaciones son realizadas comenzando el algoritmo, por lo que constituyen su inicialización.

Como fue enunciado anteriormente, el cálculo de adaptabilidad de cada instancia del proceso representa la etapa más costosa del algoritmo. De ahí la necesidad de delegar en los nodos esclavos esta tarea para establecer el procesamiento simultáneamente. Por otro lado, se combinaron técnicas para la representación de grafos al extender el modelo de matriz causal empleando matrices de índices. Esta estrategia propuesta por López Pintado, (2013) garantizó la semática del modelo y contribuyó a reducir los tiempos computacionales en operaciones realizadas sobre el mismo, explotando las ventajas de cada representación. Esta modificación garantiza que operaciones muy frecuentes, que se realizaban en orden lineal, fueran reducidas a un tiempo constante.

3.4. Análisis de los resultados obtenidos

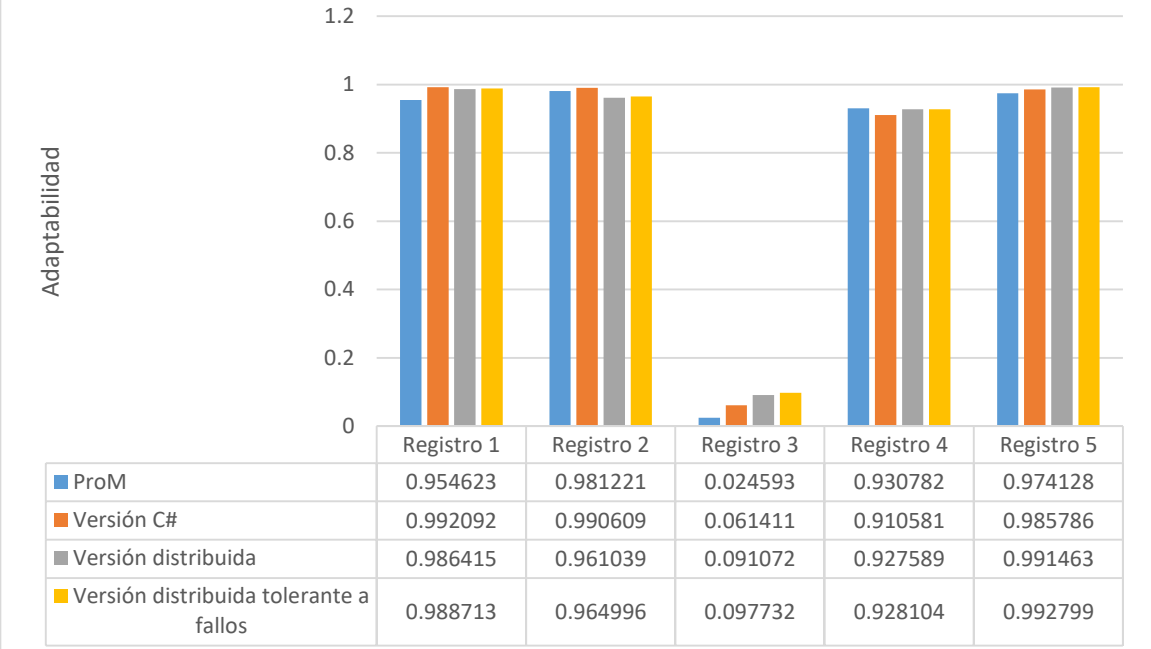
La experimentación fue desarrollada en un ambiente controlado con el propósito de ejecutar las distintas versiones del algoritmo bajo las mismas condiciones. En este sentido se mantuvo un estándar en los parámetros de entrada. En la tabla 2 se brinda la relación de los datos introducidos a las distintas 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 parseo 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 Entrada de los algoritmos

3.4.1 Análisis de adaptabilidad

La adaptabilidad de un individuo permite evaluar cuán bien este refleja el comportamieto del proceso almacenado en el registro de eventos. Su valor se encuentra comprendido entre 0 y 1. En esta escala, mientras más cercano a 1 mayor capacidad de representación tendrá el individuo. Tras llevar a cabo las ejecuciones de los algoritmos se obtuvieron los siguientes resultados (figura 13).



	Registro 1	Registro 2	Registro 3	Registro 4	Registro 5
ProM	0.954623	0.981221	0.024593	0.930782	0.974128
Versión C#	0.992092	0.990609	0.061411	0.910581	0.985786
Versión distribuida	0.986415	0.961039	0.091072	0.927589	0.991463
Versión distribuida tolerante a fallos	0.988713	0.964996	0.097732	0.928104	0.992799

Figura 13 Precisión obtenida por las versiones del algoritmo Genetic Miner. Fuente: Elaboración propia

Se realizó un análisis de varianza simple con el propósito de encontrar diferencias significativas entre los valores de adaptabilidad alcanzados por las diferentes versiones de los algoritmos. Tras aplicar la prueba de hipótesis se obtuvo un P-Valor de 0.9998, por lo que se estimó con un 95% de confianza que no existen diferencias significativas entre las precisiones obtenidas por las versiones del *Genetic Miner*.

3.4.2. Comparación en cuanto a tiempos de respuesta

Al 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* y la secuencial implementada en *C#* se utilizó una computadora Intel Core 2 Quad Q9300 a 2.5 GHz con 4 GB de RAM.

En la ejecución de las versiones distribuidas se tomaron ordenadores con las siguientes características:

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
5. Intel Core i7-8550 U a 1.8 GHz con 16 GB de RAM
6. Intel Core 2 Quad Q9300 a 2.5 GHz con 4 GB de RAM

En la tabla 3 se brindan los tiempos obtenidos en las ejecuciones realizadas:

	ProM			Versión C#			Versión distribuida			Versión distribuida tolerante a fallos		
	Tiempo transcurrido para 1000 iteraciones											
Registros de eventos	h	min	seg	h	min	seg	h	min	seg	h	min	seg
Registro 1	00	21	36	00	01	54	00	00	10	00	00	11
Registro 2	> 7 horas			00	44	07	00	04	05	00	04	29
Registro 3	> 7 horas			04	46	26	01	12	12	01	11	51
Registro 4	> 7 horas			04	36	02	01	03	53	01	04	20
Registro 5	> 7 horas			05	10	23	01	28	44	01	28	08

Tabla 3 Tiempos de respuesta de los algoritmos. Fuente: Elaboración propia

Como se puede apreciar hay una disminución significativa de los tiempos de respuesta entre las distintas variantes del algoritmo. Las versiones distribuidas del *Genetic Miner* resultaron ser las que menos demoraron en obtener los resultados de sus ejecuciones. Es preciso señalar que existe un ligero aumento en el tiempo obtenido tras ejecutar el algoritmo propuesto en esta investigación respecto a la versión distribuida desarrollada por (Díaz Vasallo et al., 2017). Esto se debe a que para esta prueba se configuraron los ordenadores para desempeñar ambas funciones (maestro y esclavo). Por tal motivo, la sobrecarga en los microprocesadores aumenta considerablemente. Sin embargo, en comparación con las otras variantes existe una diferencia importante en los tiempos de respuesta. (figura 14)

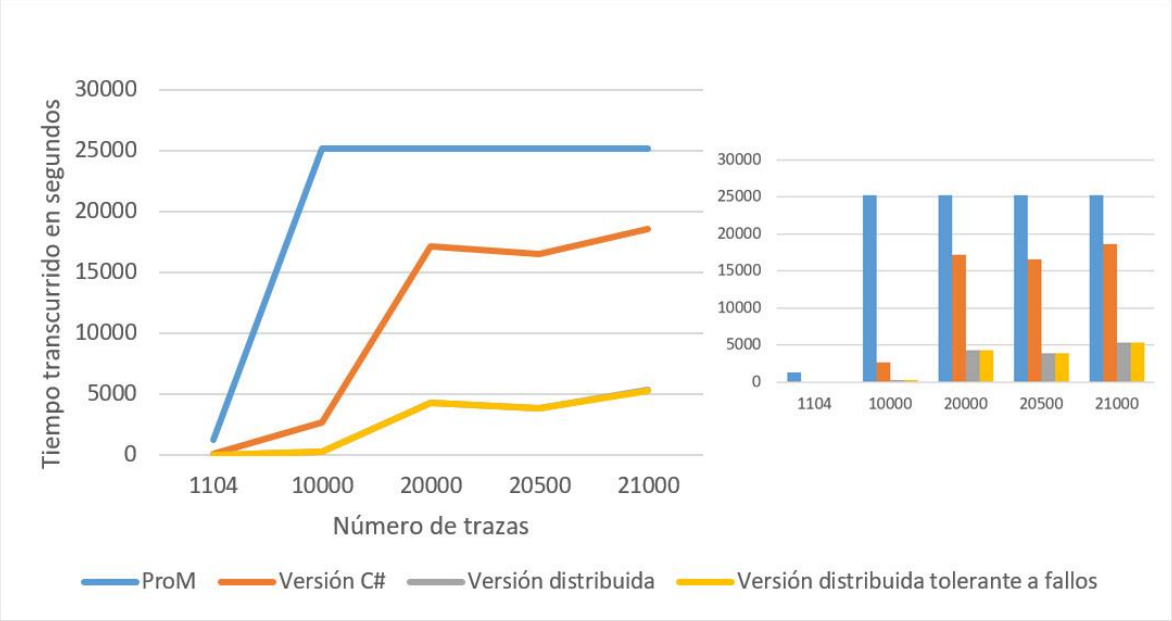


Figura 14 Gráfica de comportamiento de los tiempos de respuesta. Fuente: Elaboración propia

3.4.3. Pruebas realizadas al clúster

En estas pruebas se evaluó la escalabilidad, el rendimiento y la tolerancia a fallos del sistema distribuido. Para ello se midieron los tiempos de procesamiento del algoritmo variándole el número de recursos disponibles. Mediante el experimento se valoró el comportamiento del clúster al aumentar la cantidad de nodos esclavos.

3.4.3.1 Comprobación de escalabilidad

La comprobación se dividió en 2 etapas:

1) Durante la primera etapa se ejecutaron los algoritmos sobre varios clústeres modificando la cantidad de nodos esclavos que los integraban.
2) En la segunda etapa se muestran los resultados comparativos arribando a conclusiones.

Para la ejecución inicial del algoritmo se utilizaron dos ordenadores con las siguientes características:

- Nodo Maesto: Intel Core 2 Quad Q9300 a 2.5 GHz con 4 GB de RAM
- Nodo Maestro: Intel Core i7-8550 U a 1.8 GHz con 16 GB de RAM
- Nodo Esclavo: Intel Core i7-4700 MQ a 2.4 GHz con 8 GB de RAM

La tabla 4 muestra los resultados obtenidos.

	2 Maestros, 1 esclavo
Registros de eventos	Tiempo transcurrido
Registro 1	0h 0min 23seg
Registro 2	0h 10min 27seg

Tabla 4 Tiempos de procesamiento en clúster con 3 nodos. Fuente: Elaboración propia

Posteriormente se le adicionó a los existentes un ordenador AMD A10-4600 M a 2.3 GHz con 6 GB de RAM como esclavo y se repitió el proceso de ejecución. El tiempo transcurrido en devolver los resultados se puede apreciar en la (tabla 5), como se evidencia, hubo una disminución con respecto a los obtenidos con el clúster compuesto por 3 nodos (tabla 4), pues la diferencia establecida entre estos es de 4 y 131 segundos respectivamente.

	2 Maestros, 2 esclavos
Registros de eventos	Tiempo transcurrido
Registro 1	0h 0min 19seg
Registro 2	0h 8 min 16 seg

Tabla 5 Tiempos de procesamiento en un clúster con 4 nodos. Fuente: Elaboración propia

La incorporación de otra computadora Intel Core i3-2120 a 3.30 GHz con 2 GB de RAM contribuyó a disminuir aún más el tiempo de respuesta. En la tabla 6 se muestra que al procesar los registros de eventos se estableció una diferencia de 5 y 175 segundos respectivamente en comparación con los tiempos de la tabla 5.

	2 Maestros, 3 esclavos
Registros de eventos	Tiempo transcurrido
Registro 1	0h 0min 14 seg
Registro 2	0h 5min 21seg

Tabla 6 Tiempos de procesamiento en clúster con 5 nodos. Fuente: Elaboración propia

Por último, un sexto nodo fue adicionado. Durante la ejecución del algoritmo en el clúster conformado por 6 nodos fueron obtenidos los mejores tiempos (tabla 7).

	2 Maestros, 4 esclavos
Registros de eventos	Tiempo transcurrido
Registro 1	0h 0min 11seg
Registro 2	0h 4min 29seg

Tabla 7 Tiempos de procesamiento en clúster con 6 nodos. Fuente: Elaboración propia

Tras efectuar las pruebas se comprobó que el tiempo de respuesta de la versión propuesta del algoritmo *Genetic Miner* fue disminuyendo en la medida que eran incorporados los nodos esclavos al clúster (figura 15).

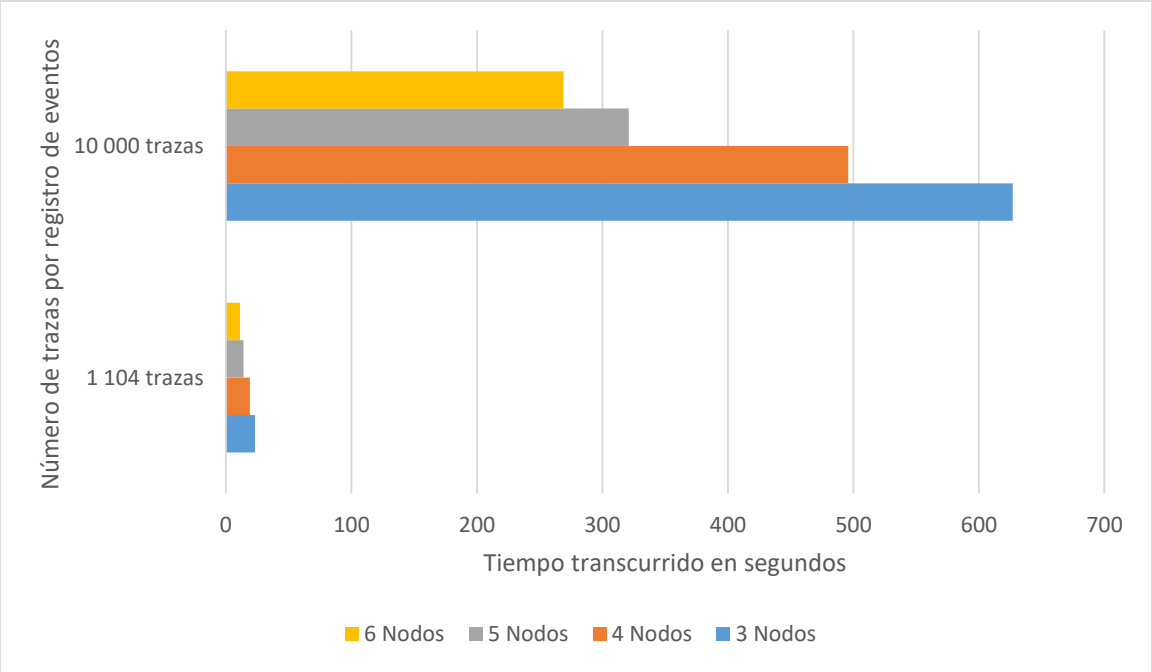


Figura 15 Comportamiento del clúster por cantidad de nodos. Fuente: Elaboración propia

3.4.3.2 Comprobación de tolerancia ante fallos

Ante la ejecución del algoritmo distribuido podían ocurrir dos posibles fallas, mediante el colapso de un ordenador esclavo y la desconexión del nodo central. Con el objetivo de verificar si la versión propuesta es capaz de adaptarse a un imprevisto de esta índole se establecieron entornos controlados que propiciaban este tipo de interrupción en los nodos.

Tras el fallo de un esclavo el nodo central fue capaz de reconocer el ordenador del clúster que presentaba el conflicto. Acto seguido estableció un monitoreo para reasignar las tareas a ese nodo en el instante en que se restauró su conexión. Por otro lado, luego de aislar el nodo central activo, uno de los ordenadores en escucha retomó su lugar. A su vez, notificó a los restantes y dio continuidad a la ejecución del algoritmo.

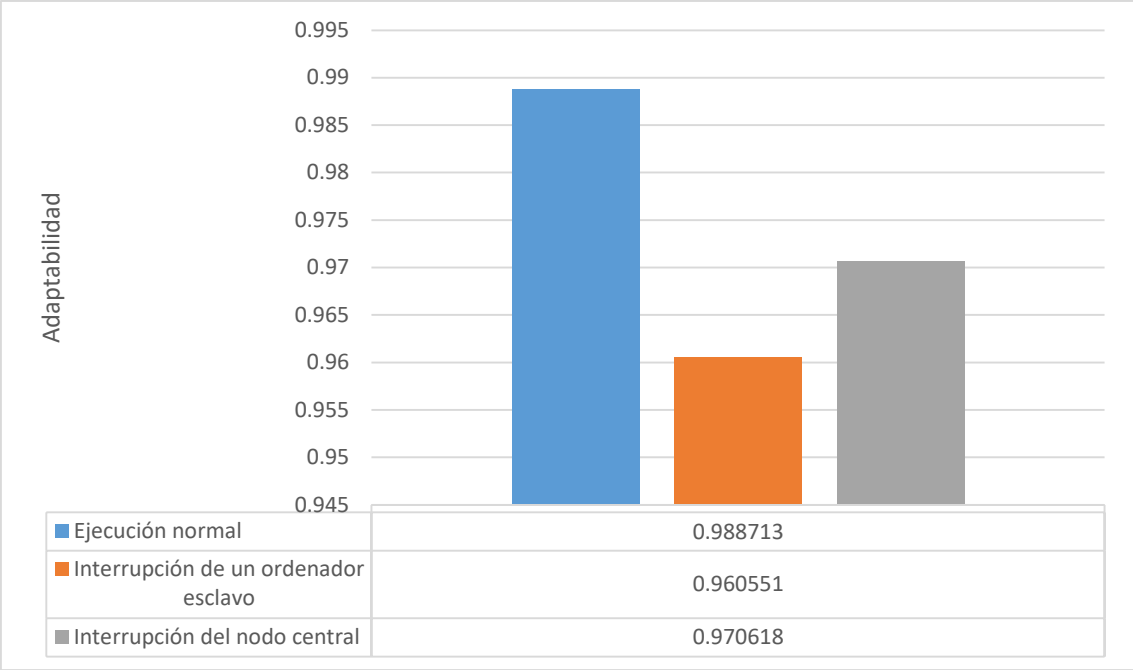


Figura 16 Adaptabilidades obtenidas en diferentes entornos controlados. Fuente: Elaboración propia

Como se puede apreciar, la adaptabilidad obtenida tras ocasionar interrupciones tanto en el ordenador central como en un nodo esclavo, no se vio afectada de manera significativa respecto a la alcanzada al someter el algoritmo a una ejecución en un estado ideal. El peor de los casos tuvo lugar al desconectar el nodo esclavo, existiendo una diferencia de 0.028162 unidades. Asimismo, la adaptabilidad obtenida en cada prueba se mantuvo por encima de 0.96 unidades, devolviendo un individuo lo suficientemente apto para reflejar el comportamiento del proceso analizado.

3.5. Conclusiones Parciales

La versión distribuida del *Genetic Miner* fue desarrollada siguiendo la estructura de comunicación de red, protocolos e interfaz, arquitectura y servicios del sistema distribuido y paradigmas de la computación distribuida. Su implementación utilizó técnicas de programación paralela disponibles en la clase *Parallel* provista de librerías que permiten el paralelismo de datos.

Para distribuir las poblaciones en un clúster de alto rendimiento se empleó la infraestructura de invocación remota *.Net Remoting*, que contribuyó a dividir la población en subpoblaciones para repartirlas entre los nodos del clúster. Esto permitió realizar la evolución de los individuos en una menor cantidad de procesamiento por ordenador, disminuyendo la carga de trabajo de cada uno.

Durante la etapa de pruebas, la precisión obtenida por el algoritmo alcanzó resultados muy similares a las implementaciones anteriores. Por otra parte, los tiempos de respuesta se mantuvieron aceptables tras las mejoras efectuadas. Además, los experimentos realizados al clúster demostraron que su eficiencia aumentaba en la medida que se le iban incorporando nodos esclavos. A su vez, la topología de distribución híbrida empleada en la contrucción del algoritmo, le dio la capacidad al sistema de sobreponerse a posibles fallos, tras garantizar la no dependencia de un ordenador central.

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 la implementación paralela del mismo. Este algoritmo fue creado utilizando técnicas de programación paralela y distribuida soportadas por .*Net Remoting*.

Se implementó un esquema híbrido que combinaba 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 secuenciales implementadas en la herramienta *ProM* y la implementada en C#, manteniendo valores cercanos a los alcanzados por la propuesta de Díaz Vasallo et al., (2017). Además, la precisión obtenida por el algoritmo se correspondía con lo esperado, demostrando la efectividad del mismo. Las pruebas realizadas al clúster para verificar la tolerancia a fallos del algoritmo arrojaron resultados satisfactorios.

Recomendaciones

El aumento del número de nodos en la conformación del clúster puede traer consigo una mayor sobrecarga e incurrir en un incremento significativo del tiempo de respuesta del algoritmo. En este sentido, se recomienda determinar la cantidad óptima de ordenadores para la obtención de los resultados. De esta forma se podría reprogramar el grafo de asignación del clúster para que automáticamente redistribuya los nodos en subgrupos (islas) interconectados. Lo que segmentaría el consumo de recursos tras analizar subpoblaciones en cada isla.

Otro punto a tener en cuenta es el procesamiento del registro de eventos. Esta operación ocurre en múltiples ocasiones, pues para determinar la adaptabilidad de los individuos es necesario parsear el registro con el fin de comprobar cuan bien cada individuo reproduce las trazas presentes en el *log*. En la representación de las trazas dentro del parser se podrían utilizar estructuras de datos como los autómatas de prefijos, capaces de compactar el registro de eventos mediante la agrupación de comportamientos similares en las trazas. Lo que traería consigo una reducción considerable del tiempo de ejecución.

Referencias Bibliográficas

Process Mining A Comparative Study [WWW Document], n.d. . ResearchGate. URL https://www.researchgate.net/publication/274248416_Process_Mining_A_Comparative_Study/figures?lo=1&utm_source=google&utm_medium=organic (accessed 4.24.19).

A. de Medeiros, A.K., Weijters, A.J.M.M., van der Aalst, W.M.P., 2007. Genetic process mining: an experimental evaluation. Data Min Knowl Disc 14, 245–304. https://doi.org/10.1007/s10618-006-0061-7

Aalst, W. van der, 2016. Process Mining: Data Science in Action, 2nd ed. Springer-Verlag, Berlin Heidelberg.

Alves De Medeiros, A.K., Weijters, A.J.M.M., 2005. CiteSeerX — Genetic Process Mining [WWW Document]. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.76.4916 (accessed 4.24.19).

Amodeo, L., Talbi, E.-G., Yalaoui, F., 2017. Recent Developments in Metaheuristics. Springer.

Badouel, E., Bernardinello, L., Darondeau, P., 2015. Petri Net Synthesis. Springer.

Banzhaf, W., Keller, P., Robert, E., 1998. Genetic Programming. An Introduction. San Francisco, California. Morgan Kaufmann Publishers, Inc., .

Banzhaf, W., Spector, L., Sheneman, L., 2019. Genetic Programming Theory and Practice XVI | SpringerLink [WWW Document]. URL https://link.springer.com/book/10.1007%2F978-3-030-04735-1 (accessed 4.24.19).

Bratosin, C., Sidorova, N., Aalst, W. van der, 2010. Distributed genetic process mining, in: IEEE Congress on Evolutionary Computation. Presented at the IEEE Congress on Evolutionary Computation, pp. 1–8. https://doi.org/10.1109/CEC.2010.5586250

Bratosin, C.C., van der Aalst, Wil M.P., Sidorova, Natalia, Information Systems W&I, Architecture of Information Systems - SIKS, 2011. Grid architecture for distributed process mining. Technische Universiteit Eindhoven.

Burattin, A., 2015. Process Mining Techniques in Business Environments: Theoretical Aspects, Algorithms, Techniques and Open Challenges in Process Mining. Springer.

Chakraborty, R.S., Mathew, J., Vasilakos, A.V., 2018. Security and Fault Tolerance in Internet of Things. Springer.

Chopard, B., Tomassini, M., 2018. An Introduction to Metaheuristics for Optimization. Springer.

Claes, J., Poels, G., 2013. Process Mining and the ProM Framework: An Exploratory Survey, in: La Rosa, M., Soffer, P. (Eds.), Business Process Management Workshops, Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, pp. 187–198.

Creación del Combinado Pesquero de Batabanó | MINAL [WWW Document], n.d. URL http://www.minal.gob.cu/es/efemerides/creacion-del-combinado-pesquero-de-batabano (accessed 4.25.19).

Cuevas, E., Zaldívar, D., Pérez-Cisneros, M., 2018. Advances in Metaheuristics Algorithms: Methods and Applications. Springer.

CW, G., Van Der Aalst, W., 2007. Fuzzy Mining – Adaptive Process Simplification Based on Multi-perspective Metrics | SpringerLink [WWW Document]. URL https://link.springer.com/chapter/10.1007/978-3-540-75183-0_24 (accessed 4.24.19).

Díaz Beltrán, MSc.I., 2014. Comunidad Primitiva - Recolectores- Cazadores- Pescadores [WWW Document]. URL http://historia.cubaeduca.cu/media/historia.cubaeduca.cu/medias/interactividades/comunidadprimitiva/co/modulo_raiz_8.html (accessed 4.25.19).

Díaz Vasallo, A., López Pintado, O., Vázquez Alfonso, Y., Velasteguí López, E., 2017. ESTRATEGIAS PARA ACELERAR Y REFINAR EL ALGORITMO GENETICMINER. | Ciencia Digital [WWW Document]. URL

http://www.cienciadigital.org/revistascienciadigital/index.php/CienciaDigital/article/view/66?articlesBySameAuthorPage=2 (accessed 4.24.19).

Fusion of Neural Networks, Fuzzy Systems and Genetic Algorithms: Industrial Applications [WWW Document], n.d. . CRC Press. URL https://www.crcpress.com/Fusion-of-Neural-Networks-Fuzzy-Systems-and-Genetic-Algorithms-Industrial/Jain-Martin/p/book/9780849398049 (accessed 2.27.19).

Gendreau, M., Potvin, J.-Y., 2018. Handbook of Metaheuristics. Springer.

Genetic Algorithms and Evolutionary Computation | Engineering Optimization: Applications, Methods, and Analysis | Ebooks | ASME DC [WWW Document], n.d. URL http://ebooks.asmedigitalcollection.asme.org/content.aspx?bookid=2426&sectionid=188250119&resultClick=1 (accessed 4.24.19).

Gonzalez, T.F., 2018. Handbook of Approximation Algorithms and Metaheuristics, Second Edition: Two-Volume Set. CRC Press LLC.

Gupta, E., 2014. (12) (PDF) PROCESS MINING ALGORITHMS [WWW Document]. ResearchGate. URL https://www.researchgate.net/publication/274248099_PROCESS_MINING_ALGORITHMS (accessed 2.27.19).

Hariri, S., Parashar, M., 2004. Tools and Environments for Parallel and Distributed Computing | Wiley Online Books [WWW Document]. URL https://onlinelibrary.wiley.com/doi/book/10.1002/0471474835 (accessed 4.24.19).

Helal, M.H.S., Fan, C., Liu, D., Yuan, S., 2017. Peer-to-Peer Based Parallel Genetic Algorithm, in: 2017 International Conference on Information, Communication and Engineering (ICICE). Presented at the 2017 International Conference on Information, Communication and Engineering (ICICE), pp. 535–538. https://doi.org/10.1109/ICICE.2017.8478917

Hoseini Alinodehi, S.P., Moshfe, S., Saber Zaeimian, M., Khoei, A., Hadidi, K., 2016. High-Speed General Purpose Genetic Algorithm Processor. IEEE

Transactions on Cybernetics 46, 1551–1565. https://doi.org/10.1109/TCYB.2015.2451595

HsunChou, C., Chen Hsieh, S., Jie Qiu, C., 2017. Hybrid genetic algorithm and fuzzy clustering for bankruptcy prediction - ScienceDirect [WWW Document]. URL https://www.sciencedirect.com/science/article/pii/S1568494617301370 (accessed 4.24.19).

Introduction to Parallel Computing, Second Edition [Book] [WWW Document], n.d. URL https://www.oreilly.com/library/view/introduction-to-parallel/0201648652/ (accessed 2.27.19).

Ivan , 2018. Parallel and distributed genetic algorithms [WWW Document]. Towards Data Science. URL https://towardsdatascience.com/parallel-and-distributed-genetic-algorithms-1ed2e76866e3 (accessed 2.27.19).

Kalenkova, A.A., van der Aalst, W.M.P., Lomazova, I.A., Rubin, V.A., 2017. Process mining using BPMN: relating event logs and process models | SpringerLink [WWW Document]. URL https://link.springer.com/article/10.1007/s10270-015-0502-0 (accessed 4.25.19).

López Pintado, O., 2013. Estrategia para acelerar el algoritmo Genetic Miner utilizando matrices de índices. Compumat.

Mans, R.S., Aalst, W. van der, Vanwersch, R.J.B., 2015. Process Mining in Healthcare: Evaluating and Exploiting Operational Healthcare Processes, SpringerBriefs in Business Process Management. Springer International Publishing.

Martínez, C.G., Lozano, M., Herrera, F., Molina, D., Sánchez, A.M., 2008. Global and local real-coded genetic algorithms based on parent-centric crossover operators - ScienceDirect [WWW Document]. URL https://www.sciencedirect.com/science/article/abs/pii/S0377221706006308 (accessed 4.24.19).

Maulik, U., Bandyopadhyay, S., 2000. Genetic algorithm-based clustering technique. Pattern Recognition 33, 1455–1465. https://doi.org/10.1016/S0031-3203(99)00137-5

McDonnell, J.R., Reynolds, R.G., Fogel, D.B., 1995. Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming. MIT Press.

Munoz-Gama, J., 2016. Conformance Checking and Diagnosis in Process Mining: Comparing Observed and Modeled Processes. Springer.

Nowostawski, M., Poli, R., 1999. Parallel genetic algorithm taxonomy, in: 1999 Third International Conference on Knowledge-Based Intelligent Information Engineering Systems. Proceedings (Cat. No.99TH8410). Presented at the 1999 Third International Conference on Knowledge-Based Intelligent Information Engineering Systems. Proceedings (Cat. No.99TH8410), pp. 88–92. https://doi.org/10.1109/KES.1999.820127

Olsson, M., 2018. Asynchronous Methods | SpringerLink [WWW Document]. URL https://link.springer.com/chapter/10.1007/978-1-4842-3817-2_30 (accessed 4.24.19).

O'Reilly, U.-M., Yu, T., Riolo, R., Worzel, B., 2006. Genetic Programming Theory and Practice II. Springer Science & Business Media.

Paoli, F.D., Schulte, S., Johnsen, E.B., 2017. Service-Oriented and Cloud Computing: 6th IFIP WG 2.14 European Conference, ESOCC 2017, Oslo, Norway, September 27-29, 2017, Proceedings. Springer.

Petrowski, A., Ben-Hamida, S., 2017. Evolutionary Algorithms. John Wiley & Sons.

Poniszewska Maranda, A., Wasilewski, P., 2016. Communication Approach in Distributed Systems on .NET Platform | SpringerLink [WWW Document]. URL https://link.springer.com/chapter/10.1007/978-3-319-43982-2_34 (accessed 4.24.19).

Premchaiswadi, W., Porouhan, P., 2016. Process simulation and pattern discovery through alpha and heuristic algorithms - IEEE Conference Publication

[WWW Document]. URL https://ieeexplore.ieee.org/abstract/document/7368472 (accessed 4.24.19).

Puaut, I., Dardaillon, M., Cullmann, C., Gebhard, G., Derrien, S., 2018. Fine-Grain Iterative Compilation for WCET Estimation. Presented at the WCET 2018 - 18th International Workshop on Worst-Case Execution Time Analysis, pp. 1–12. https://doi.org/10.4230/OASIcs.WCET.2018.9

Rubtcova, M., Pavenkov, O., 2018. Enterprise Distributed Component Object Model's Architecture of the Information System by Mariia Rubtcova, Oleg Pavenkov :: SSRN [WWW Document]. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3167570 (accessed 4.24.19).

Sánchez Villalba, Á., Rey García, V., 2009. Implementación del algoritmos genéticos en un sistema distribuido como herramienta para la generación automática de redes neuronales utilizando datos de crudos y fracciones.

Sharan, K., 2018. Java Remote Method Invocation | SpringerLink [WWW Document]. URL https://link.springer.com/chapter/10.1007/978-1-4842-3546-1_6 (accessed 4.24.19).

TROELSEN, A., Japikse, P., 2015. C# 6.0 and the .NET 4.6 Framework. Apress.

Troelsen, A., Japikse, P., 2015. ADO.NET Part III: Entity Framework. C# 6.0 and the .NET 4.6 Framework 929–999. https://doi.org/10.1007/978-1-4842-1332-2_23

Turner, C.J., 2009. A genetic programming based business process mining approach [WWW Document]. URL https://dspace.lib.cranfield.ac.uk/handle/1826/4471 (accessed 4.24.19).

van der Aalst, W., 2011. Process Mining - Discovery, Conformance and Enhancement of Business Processes | Wil van der Aalst | Springer [WWW Document]. URL https://www.springer.com/us/book/9783642193453 (accessed 4.24.19).

Weijters, A.J.M.M., Ribeiro, J.T.S., 2011. Flexible Heuristics Miner (FHM) - IEEE Conference Publication [WWW Document]. URL https://ieeexplore.ieee.org/abstract/document/5949453 (accessed 4.24.19).

Wiley, J., 2004. Tools and Environments for Parallel and Distributed Computing - Tools and Environments for Parallel and Distributed Computing - Wiley Online Library [WWW Document]. URL https://onlinelibrary.wiley.com/doi/pdf/10.1002/0471474835 (accessed 4.24.19).