UNIVERSIDAD DE MENDOZA

FACULTAD DE INGENIERÍA

MAESTRÍA EN TELEINFORMÁTICA

"Clasificación de patrones de tráfico de redes utilizando herramientas de aprendizaje automático en entornos de computación distribuida"

Autor: Carlos A. Catania

Director: Dr. Carlos García Garino

Codirector: Dra. Cristina Párraga

2007

A mi padres, Carlos y Delia, ya que sin su ayuda, paciencia y esfuerzo no hubiera sido capaz de llevar a cabo esta tarea. También a Lili, mi chica y futura esposa, por ser "la alegría del hogar".

Agradecimientos

Este trabajo desarrollado durante los últimos 18 meses no hubiera sido posible sin la colaboración de quienes me ayudaron a concretarlo. Por este motivo debo agradecer profundamente a todos los que me brindaron su apoyo.

A mi director de tesis, Carlos Garcia Garino, por su confianza y por todo el tiempo que le dispuso a mi tarea, por sus aportes, sugerencias y correcciones.

A Cristina Párraga, mi codirectoria de tesis, por sus sugerencias y correcciones.

A mis compañeros del LAPIC, Claudio Careglio, Julio Monetti, Osvaldo Marianeti, Sergio Salinas, Emiliano Lopez y Paula Martinez. Por las horas compartidas.

A la Agencia Nacional de Promoción Científica y Tecnológica (ANPCyT), por el apoyo económico recibido mediante el Proyecto PICTR 184, el cual me permitió dedicarme completamente a esta tarea.

Resumen

Desde hace algunos años, los sistemas de detección de intrusos (IDS) se emplean como una herramienta de apoyo a las actividades del ingeniero en seguridad de redes de datos. Estos sistemas ayudan a reconocer diferentes tipos de ataques o anomalías en la red, a partir del reconocimiento de patrones del tráfico de red, que resulta uno de los componentes fundamentales de los mismos.

Con este fin, se estudian las posibilidades de aplicación de un algoritmo genético para obtener un conjunto de reglas que permitan reconocer las instancias de tráfico normales. Luego toda nueva instancia de tráfico que no pueda ser reconocida por el conjunto de reglas se considera como una posible anomalía.

El enfoque propuesto es distinto respecto a otros trabajos del estado del arte que clasifican patrones de tráfico de las instancias que presentan anomalías. En este sentido presenta la ventaja de permitir la detección de ataques no conocidos, a la vez que aporta información sobre el tráfico de red que puede resultar valiosa para el ingeniero en seguridad de redes.

En esta tesis se presenta una arquitectura para un sistema de detección de intrusos y se discuten los módulos que componen la misma. Para el módulo de aprendizaje se propone un algoritmo genético de tipo estacionario, para lo cual se discute la representación de la población, la función de fitness, las técnicas de nicho para garantizar la convergencia hacia múltiples soluciones y los operadores genéticos utilizados.

Debido a que el proceso de entrenamiento del algoritmo genético propuesto requiere una gran cantidad de recursos computacionales, tanto para su

procesamiento como para su validación, se estudian distintas alternativas para su ejecución en ambientes distribuidos fuerte y debílmente acoplados.

Índice general

Índice de figuras

Índice de cuadros

Lista de algoritmos

1 Introducción

En este capítulo se introduce el problema a investigar y se identifican los objetivos generales y secundarios de la tesis. Además se reseña la estructura de la misma.

1.1. Descripción del problema

La seguridad de las redes de datos se ha transformado en un serio problema en los últimos años. El crecimiento vertiginoso que ha presentado la Internet ha permitido mostrar las fallas de seguridad en las implementaciones de los protocolos de red subyacentes. Situación que resulta comprensible, ya que muchos de estos protocolos originalmente fueron pensados para unir de 10 a 50 computadoras de las universidades de los Estados Unidos.

Las falencias en la seguridad de protocolos como *ARP*, *TCP*, *TELNET*, *SMTP*, *FTP* han sido la causa de ataques contra la confidencialidad, la disponibilidad y la autenticidad de los datos transportados. Si bien estos problemas han sido corregidos a lo largo de los años, continuamente se van descubriendo nuevas maneras de realizar estos ataques.

El ingeniero en seguridad de redes debe estar alerta para detectar estos ataques, informándose de las nuevas vulnerabilidades descubiertas o tipos de ataques perpetrados. Para dar soporte a esta tarea se cuenta con sistemas de detección de intrusos (IDS) basados en reglas de tráfico [3].

Se identifican dos alternativas dentro del contexto de la detección de intrusos: la detección por mal uso y la detección por anomalías [4].

En la detección por mal uso se buscan patrones en el tráfico de red que indiquen comportamiento malicioso. Con este fin generalmente se emplean reglas que mediante una combinación de sentencias del tipo *if - then* permiten expresar diversos patrones de tráfico de red. De esta manera cuando el IDS, que está permanentemente monitoreando la red, encuentra el patrón expresado en las reglas, dispara una alarma. Un ejemplo de patrón podría ser un escaneo de puertos de una computadora perteneciente a la red.

La detección por mal uso presenta el inconveniente no poder reconocer tipos de ataques que no se encuentran en el conjunto de reglas. Por lo que resulta necesario actualizar las reglas para detectar las últimas técnicas de ataque. Sin embargo en muchos casos estas actualizaciones se encuentran disponibles varias horas o incluso días después de haberse descubierto el tipo de ataque.

Por otro lado una gran cantidad de estos ataques tienen lugar antes que se conozcan siquiera las vulnerabilidades o fallas que los provoca, por lo que muchas veces el ingeniero en seguridad de redes debe hacer uso de su experiencia para identificar patrones extraños en el tráfico de la red. Una tarea que se puede describir como no trivial.

La detección de anomalías ofrece una alternativa que consiste en modelar el tráfico normal de la red, luego se asume que toda instancia de tráfico que no se adapte al modelo, puede considerarse una anomalía. La misma puede deberse a algún tipo de ataque no conocido aún.

La detección de anomalías es de gran ayuda a la tarea del ingeniero en seguridad en la detección de nuevos ataques, sin embargo en muchos casos clasifica como maliciosas a instancias de tráfico normal, lo que se conoce como falsos positivos.

Para hacer frente a los inconvenientes mencionados se han propuesto alternativas que utilizan técnicas de inteligencia artificial en general y aprendizaje de máquina en particular. Así en los últimos años se han propuesto diversas estrategias que utilizan reglas de asociación [5], redes neuronales [6, 7], redes bayesianas [8], lógica difusa [9], k-vecinos cercanos [10] con el fin de reconocer el tráfico de red normal así como el malicioso. Un recopilación más completa puede verse en [11].

1.2. Propuesta

Entre las técnicas de aprendizaje de máquina que se han aplicado a los sistemas de detección de intrusos , se destaca el empleo de algoritmos genéticos. Si bien los mismos surgen originalmente como un método de optimización, los cuales recorren un espacio de soluciones posibles en búsqueda de un única solución óptima, es posible utilizarlos para obtener un conjunto de reglas de clasificación.

Una de las ventajas del empleo de reglas de clasificación es que son capaces de reconocer los ataques e intrusiones a la red, a la vez que aportan información valiosa para el ingeniero en seguridad de redes.

Al igual que en otras técnicas de aprendizaje de máquina, cuando se utilizan algoritmos genéticos, se realiza procesamiento fuera de línea. Aunque la detección de intrusos es una una tarea con requerimientos en tiempo real, donde resulta esencial la capacidad de detectar los ataques a tiempo, se considera que el procesamiento fuera de línea ofrece un buen compromiso entre la eficiencia y el tiempo necesario para el procesamiento.

Debido a que los algoritmos genéticos demandan altos recursos computacionales, no es extraño que las aplicaciones existentes estén concentradas en la detección de trafico anormal o malicioso, por cuanto el gran supuesto es que el tráfico anormal es menos voluminoso que el normal lo que hace que el algoritmo sea más eficiente, y su ejecución sea menos costosa. Sin embargo surge el inconveniente necesitar un conjunto de instancias tráfico evaluadas a priori por un experto en seguridad de redes.

En este trabajo se presenta una propuesta para la detección de anomalías basada en un algoritmo genético, que permite obtener un conjunto de reglas que presenta las mayores coincidencias con el tráfico de la red. Luego, toda nueva instancia de tráfico que se aparte del comportamiento normal de la red se considera una anomalía.

Debido a que el algoritmo propuesto demanda altos requerimientos computacionales, a lo que se suma la necesidad de un tiempo de respuesta rápido por parte de los IDS, resulta importante implementar de manera eficiente el

algoritmo propuesto así como analizar la posibilidad de adecuarlo a entornos de computación distribuida.

1.3. Objetivos.

El objetivo de la tesis es obtener una herramienta de clasificacion de patrones en el tráfico de red basada en algoritmos genéticos.

Para lograr el objetivo de esta tesis es necesario previamente cumplir objetivos secundarios que se listan a continuación:

1. Conocer las características de los algoritmo genéticos, esencialmente sus operadores y empleo de los mismos para obtener un conjunto de reglas de clasificación.
2. Revisar los antecedentes en la literatura de algoritmos genéticos que se apliquen a los sistemas de detección de intrusos.
3. Distinguir los diferentes componentes de un sistema de detección de intrusos y la relación entre los mismos. Para luego ubicar un módulo de clasificación basado en algoritmos genético.
4. Implementar un algoritmo genético para el reconocimiento de tráfico de red normal.
5. Establecer métricas para determinar los tiempos de ejecución del algoritmo para luego establecer estrategias con el fin de disminuir los tiempos de ejecución.
6. Estudiar las posibilidades de ejecución en entornos distribuidos fuerte y débilmente acoplados.
7. Diseñar experimentos y analizar los resultados obtenidos con la herramienta propuesta.

1.4. Estructura del trabajo

En el capítulo 2 se analizan las principales características de los trabajos que constituyen el estado del arte. Se pone énfasis en los trabajos que aplican algoritmos genéticos para obtener un conjunto de reglas de clasificación, sin embargo se revisan también otros trabajos que se basan en algoritmos genéticos en el contexto de la detección de intrusos.

En el capítulo 3 se propone un sistema de detección de anomalías basado en algoritmos genéticos. Se discuten los diferentes módulos que componen la arquitectura del mismo y se trata con especial énfasis la clasificación de reglas de tráfico.

En el capítulo 4 se discute la implementación computacional de la herramienta propuesta en el capítulo 3. Se detalla la implementación del algoritmo, el código secuencial resultante y la adecuación del mismo a entornos distribuidos fuerte y débilmente acoplados. También se proponen métricas para evaluar el comportamiento de las diferentes implementaciones discutidas.

En el capítulo 5 se realizan experimentos con el fin de determinar el comportamiento del algoritmo propuesto y se presentan los resultados obtenidos. Además se discuten los tiempos de ejecución que se obtienen cuando se aplican las técnicas de paralelización del capítulo 4.

Finalmente en el capítulo 6 se presentan las conclusiones y los trabajos futuros.

2 Estado del arte

En este capítulo se analizan las principales características de los trabajos que constituyen el estado del arte. Se pone énfasis en los trabajos que aplican algoritmos genéticos para obtener un conjunto de reglas de clasificación. Los sistemas de clasificación basados en algoritmos genéticos han dado buenos resultados en diversas áreas del conocimiento y su aplicación al problema de la detección de intrusos ha sido motivo de numerosos trabajos en los últimos años.

La utilización de sistemas de clasificación de reglas basados en algoritmos genéticos no es la única manera de utilizar algoritmos genéticos en el contexto de la detección de intrusos. Existen estrategias que combinan la aplicación de algoritmos genéticos junto a otras técnicas de inteligencia artificial como lógica difusa y máquinas de vectores de soporte (support vector machine).

En las siguientes secciones se discuten las características más importantes de las diferentes estrategias.

2.1. Reglas de clasificación

La utilización de algoritmos genéticos surge como un método de optimización, los cuales recorren un espacio de soluciones posibles en búsqueda de única solución óptima. Sin embargo también es posible utilizarlos para obtener múltiples soluciones.

Dentro de en este contexto muchos trabajos citados en literatura plantean la utilización de algoritmos genéticos para obtener un conjunto de reglas de clasificación [12].

Cada regla de clasificación está conformada en su lado derecho por una conjunción de los distintos valores para cada uno de los atributos seleccionados,a lo que se denomina condición. Mientras que el lado izquierdo de la regla indica el concepto o resultado de laclasificación que debe ser asignado a los ejemplos que encuentren coincidencias con la condición de la regla.

El esquema más utilizado para evaluar estos sistemas de aprendizaje es el de procesamiento por lotes (*batch mode*), en donde un conjunto de ejemplos se divide para formar un conjunto de entrenamiento y otro de prueba. El sistema debe ser capaz de encontrar reglas que presenten coincidencias con el conjunto de entrenamiento. Posteriormente el conjunto de reglas se valida al buscar el porcentaje de clasificaciones correctas que se hayan encontrado en el conjunto de prueba.

La aplicación de sistemas de clasificación basados en algoritmos genéticos al contexto del problema de la detección de intrusos resulta una alternativa estudiada previamente en numerosos trabajos. El objetivo de estos algoritmos es partir de una población de individuos conformados por instancias de tráfico elegidas al azar y un conjunto de instancias de tráfico que conforman el conjunto de entrenamiento. Al finalizar el proceso evolutivo el algoritmo permite obtener un conjunto de reglas de clasificación que presenten coincidencias con el mayor número de instancias de tráfico normal (cuando se aplica el concepto de detección de intrusos por anomalías) o instancias de tráfico conteniendo ataques (cuando se aplica el concepto de detección de intrusos por mal uso) [4].

Uno de los primeros trabajos que aplica un sistema de clasificación basado en algoritmos genéticos con aplicaciones al problema de la detección de intrusos es el de Sinclair [1], quien propone la utilización de algoritmos genéticos para generar reglas sencillas que permitan encontrar patrones en el tráfico de la red. Posteriormente estas reglas se utilizan para alimentar un

sistema de detección de intrusos. En el trabajo de Sinclair se realizan experimentos utilizando arboles de decisión [13] con el objeto de poder comparar los resultados obtenidos con el algoritmo genético con una herramienta de aprendizaje determinística como son los arboles de decisión.

En Li [2] se discute una metodología para la aplicación de algoritmos genéticos al problema de la detección de intrusos. En el trabajo se presenta un algoritmo genético cuya codificación del problema tiene en cuenta información espacial y temporal sobre las conexiones de red. Pese a que el trabajo no presenta resultados experimentales se discuten componentes fundamentales del algoritmo que facilitan su implementación. En particular se trata una propuesta para la función de fitness que ordena los atributos de la instancia de tráfico de acuerdo a un peso asociado a los diferentes atributos seleccionados para la codificación del problema.

Gong [14] también utiliza algoritmos genéticos para la generación de reglas de clasificación en el tráfico de red. En su trabajo se presenta un esquema en donde se buscan los patrones más comunes en las instancias de tráfico conteniendo anomalías. La implementación de este esquema necesita de un conjunto de instancias de tráfico que previamente haya sido auditada por un experto en seguridad de redes, en el cual el experto haya indicado las instancias que contienen anomalías o posibles ataques. Una vez finalizado el entrenamiento, el algoritmo puede generar alertas cuando observe alguno de los patrones aprendidos.

El trabajo de Gong [14] no presenta grandes variantes respecto de los esquemas anteriores. Entre los aportes del trabajo se destaca la presentación de resultados experimentales, así como una especificación más completa de la implementación del algoritmo mediante la presentación de una arquitectura del sistema. El trabajo incluye también una descripción general del pseudocódigo del algoritmo genético propuesto junto a un diagrama de clases de alto nivel. El trabajo presenta algunos resultados experimentales, de los que se desprenden los altos requerimientos computacionales de la implementación propuesta por Gong.

Otros autores como Lu [15], Yin [16] y Crosbie [17] utilizan programación genética para obtener la reglas de clasificación. La programación genética

es una variante de la computación evolutiva, que difiere de los algoritmos genéticos en el modo de representar la población de individuos [18]. Esta variante en muchos casos permite obtener reglas de mayor complejidad.

Crosbie [17] utiliza agentes autónomos combinados con programación genética. Cada agente se encarga del monitoreo de diferentes parámetros de la red. Aquellos agentes con mejor capacidad de predicción cuentan con mayor peso en el momento de determinar si se trata de una intrusión o no. La inteligencia de estos agentes consiste en un conjunto de reglas de clasificación obtenidas mediante programación genética. Si bien se trata de un enfoque interesante e innovador, la utilización de agentes autónomos trae aparejado problemas asociados a la comunicación segura entre los distintos agentes.

Lu [15] presenta un enfoque basado en programación genética para la búsqueda de intrusiones. El trabajo presenta una variante en la forma de representación de los individuos, en vez de utilizar árboles sintácticos se emplean cadenas de caracteres de longitud variable. En el trabajo se discuten brevemente los principales componentes de algoritmos genéticos como los operadores de mutación, cruzamiento y descarte de condición y se presentan resultados experimentales.

Yin [16] se centra en la aplicación de programación genética para obtener un conjunto de reglas de clasificación. En el trabajo se presenta GP-LERAD, el cual es un algoritmo para detectar ataques por anomalías. GP-LERAD es una modificación del algoritmo LERAD propuesto por Mahoney [19] para encontrar relaciones entre los diferentes atributos de las instancias de tráfico utilizando reglas de asociación.

En las secciones siguientes se detallan los componentes principales de cada uno de los trabajos mencionados anteriormente. Se incluye la problemática de la selección de atributos, los diferentes esquemas seguidos en la representación de los individuos y las funciones de fitness más utilizadas. También se detallan brevemente las características de los principales operadores genéticos utilizados.

2.1.1. Selección de atributos

Los protocolos de comunicación constituyen la base sobre la cual se apoyan las redes de datos. Un protocolo consiste en un estándar o convención que controla la comunicación y la transferencia de datos entre dos puntos. Los distintos elementos que componen estos protocolos de comunicación se utilizan como atributos para realizar la detección de intrusión.

En los trabajos discutidos en esta sección se seleccionan los atributos en base a elementos del conjunto de protocolos basados en TCP/IP. Esta selección se debe a que si bien existen una gran variedad de protocolos, entre los más utilizados se encuentran aquellos protocolos basados en TCP/IP. Este grupo de protocolos es la base sobre la cual opera la Internet.

Debido a la gran cantidad de protocolos basados en TCP/IP, la posibilidad de contar con reglas que evalúen a todos los elementos que componen una instancia de tráfico no resulta viable, principalmente a causa de la alta demanda de recursos computacionales necesarios para su procesamiento, luego se debe seleccionar un subconjunto de estos atributos.

La selección de los atributos más representativos constituye por si mismo un problema de gran complejidad. Debido a esta dificultad es que existen propuestas que tienen en cuenta la utilización de algoritmos genéticos para optimizar el número y tipo de atributos más relevantes dentro del ámbito de la detección de intrusos. Este tema se trata con más detalle en la sección 2.2.2.

Los trabajos mencionados en la sección anterior presentan diferentes variantes en la selección de los atributos utilizados para la generación de las reglas de clasificación. En los trabajos presentados se discuten aquellos atributos de una instancia de tráfico de red que resultan más relevantes para la detección de intrusos por anomalías o por mal uso. La elección de estos atributos en muchos casos son resultado de una heurística con un fuerte contenido disciplinar.

En uno de los primeros trabajos sobre el tema [1] se consideran los atributos provenientes de campos de la cabecera del paquete IP y del segmento

TCP como: puerto origen, puerto destino, dirección IP origen y dirección IP destino y tipo de protocolo. El conjunto de atributos seleccionados es sencillo y se puede obtener con facilidad, debido a que la información se encuentra disponible en forma explícita en la trama Ethernet. Los atributos seleccionados por Sinclair, pueden brindar importante información sobre las instancias de tráfico, ya que permite identificar a cada una de estas de manera única. En consecuencia estos atributos se encuentran presentes en la mayoría de los trabajos propuestos en el área.

En trabajos como los de Li [2] y Gong [14] se tienen en cuenta además otros atributos como ser la información sobre la cantidad de bytes recibidos y transferidos y el tiempo de duración de la instancia de tráfico. En Li [2] se menciona que la inclusión del tiempo de duración, permite obtener un conjunto de reglas capaces de reconocer los ataques o anomalías que ocurran a lo largo de las semanas o incluso meses.

La utilización de atributos que provean información sobre la cantidad de bytes recibidos y transferidos ofrece una alternativa interesante, y que permiten establecer una relación entre el tipo de protocolo y sus tasas de transferencias. Aunque no es mencionado explícitamente por el autor del trabajo, esta característica permitiría detectar anomalías generadas por las técnicas de encapsulamiento de un protocolo sobre otro, comúnmente conocidas como túnel y muy utilizada en la actualidad.

La técnica de encapsulamiento de protocolos fue originalmente utilizada para el transporte seguro a través de una red no confiable pero en la actualidad también es utilizado como una técnica para traspasar las reglas impuestas por el administrador del firewall. Un ejemplo del uso de esta técnica se presenta en el trabajo de Ciarlante [20], en el cual se implementa una aplicación que permite encapsular tráfico a través de la red de mensajería instantánea. Las conexiones provenientes de la red de mensajería instantáneas suelen estar permitidas en la mayoría de las infraestructuras de red. Por lo que fácilmente puede utilizarse el encapsulamiento sobre protocolos de mensajería instantánea para saltear las restricciones de la red. Sin embargo la cantidad de bytes transferidos en una conexión típica de mensajería instantánea probablemente presente diferencias significativas respecto a los que podría transferirse a través del túnel.

La utilización de atributos con información sobre bytes recibidos y transferidos brindan un mayor grado de informacion sobre la instancia de tráfico. Sin embargo al no estar disponible esta informacion de forma explícita en las instancias de tráfico, trae aparejado un mayor costo debido a la necesidad de procesar las instancias de tráfico para obtener dicha información.

Yin [16] sigue el esquema de atributos propuesto por Mahoney [19], los cuales se listan a continuación:

- Fecha
- Hora (en segundos)
- Dirección IP origen
- 2 bytes menos significativos de la dirección IP destino
- Puerto origen y puerto destino
- Valor de indicadores TCP del primero, el anteúltimo y el ultimo paquete
- Log_2de la duración en segundos truncado a entero
- Log_2de la cantidad de bytes transferidos truncado a entero
- Primeras ocho palabras del área de datos de TCP

Entre los atributos enumerados se incluyen la dirección IP origen y destino, puerto origen y destino, duración de la conexión y cantidad de bytes transferidos y recibidos. Estos atributos no presentan variaciones interesantes y fueron discutidos oportunamente.

De los atributos utilizados por Yin se destaca la información sobre el valor de los indicadores TCP en el primer paquete de la conexión y en el anteúltimo y el último paquete. Esto permite detectar anomalías en el estado de la conexión TCP, como puede ser el caso de conexiones que no se han cerrado de manera correcta.

Otro de los atributos seleccionados en el trabajo de Yin incluye información sobre la capa de aplicación (capa 7 del modelo de referencia OSI). Para ello se utiliza la información obtenida a partir de las primeras ocho palabras del área de datos del segmento TCP. Cada palabra está delimitada

por espacios o retornos de línea y es truncada a los primeros ocho bytes. Como se menciona en el trabajo de Mahoney [19], la inclusión de las ocho primeras palabras del área de datos del protocolo de transporte permitiría detectar anomalías en protocolos de capa 7 basados en texto como ser SMTP, TELNET, FTP y HTTP entre otros.

Lu [15] utiliza un conjunto de atributos que se obtienen a partir de información de los ataques previamente conocidos, los cuales se listan a continuación:

- Tipo de protocolo
- Indicador para identificar si la conexión es desde y hasta el mismo host y puerto
- Número de fragmentos incorrectos en la conexión
- Conexión con errores en el flag SYN
- Número de condiciones comprometidas
- Porcentaje de conexiones hacia el mismo servicio
- Porcentaje de conexiones hacia servicios distintos
- Nro. de conexiones desde el mismo host origen hacia el mismo host destino
- Nro. de conexiones desde el mismo servicio origen hacia el mismo servicio destino
- Nro. de conexiones desde el mismo host destino hacia el mismo host origen
- Nro de conexiones desde el mismo servicio origen hacia el mismo servicio destino

En este caso la mayoría de los atributos seleccionados no se encuentran de manera implícita en las trazas de tráfico, sino que se obtienen mediante el procesamiento de un conjunto de instancias de tráfico obtenido previamente.

En el trabajo de Crosbie [17] se presenta un conjunto de atributos compuestos en su mayoría por métricas de tiempo. Crosbie utiliza métricas que sólo

tienen en cuenta información acerca de los *socket* de Internet. Los socket de Internet son estructuras de datos que mantienen información única sobre una conexión (Más detalles sobre las características de un socket de Internet pueden verse en [21]).

La utilización de sockets restringe los atributos a la captura de información de protocolos basados en TCP y UDP y deja de lado otros protocolos importantes como es el caso de ICMP. Una enumeración de los atributos seleccionados en el trabajo de Crosbie se presenta a continuación:

- Número total de conexiones de socket
- Tiempo promedio entre las conexiones de socket
- Tiempo mínimo entre conexiones de socket
- Tiempo máximo entre conexiones de socket
- Puerto de destino de la conexión de socket
- Puerto de origen de la conexión de socket

Los primeros cuatro atributos no se encuentran en las trazas de tráfico, por lo que que deben obtenerse mediante un preprocesamiento. Los últimos dos atributos se pueden obtener fácilmente a partir de los datos provistos por la traza de tráfico.

De los trabajos discutidos se desprende que en muchos casos dentro del proceso de selección de los atributos existe un compromiso entre el tiempo requerido para obtener un atributo y la calidad de la información que puede aportar al proceso de detección.

Muchos atributos se encuentran disponibles de manera implícita en los elementos de los protocolos que componen la instancia de tráfico y pueden ser recuperados de manera sencilla. Por ejemplo dirección IP origen, dirección IP destino, puerto origen, puerto destino, etc.

Para obtener otros atributos es necesario un preprocesamiento de una cierta complejidad, como es el caso del tiempo de duración de la conexión, la cantidad de bytes transferidos, y los tiempos promedios entre conexiones,

sólo por mencionar algunos. Estos atributos permiten ampliar las posibilidades de los sistemas de detección de intrusos a costa de un alto tiempo de preprocesamiento.

2.1.2. Esquemas de Representación

La manera de representar los individuos de la población es uno de los principales componentes de los algoritmos genéticos que necesita ser adaptado al dominio de aplicación.

En la literatura tradicional de algoritmos genéticos [22, 23, 24] se representa a cada individuo mediante un cromosoma de longitud fija. El cual a su vez está formado por genes binarios que pueden tomar valores de uno y cero. La representación de cada atributo dentro del cromosoma se determina por un número suficiente de genes binarios.

En los trabajos revisados se plantean algunas alternativas en los esquemas de representación.

Sinclair [1] utiliza para la representación un cromosoma compuesto por una lista de 29 genes. Los cuales están distribuidos de la siguiente manera: 8 genes para la dirección IP origen, 8 genes para la dirección IP destino, 6 genes para el puerto origen, 6 genes para el puerto destino y un gen para el tipo de protocolo. Para la representación de los genes de la dirección IP origen y la dirección IP destino se utilizan dígitos hexadecimales.

En el Cuadro 2.1 se presentan valores de ejemplo para los cinco atributos seleccionados en el trabajo de Sinclair.

Posteriormente la representación del cromosoma para los valores de ejemplo del Cuadro 2.1 se presentan en la Figura 2.1.

(12,0,10,8,13,10,0,1,12,10,10,8,13,11,4,6,0,0,2,1,2,1,0,0,0,0,8,0,1)

Figura 2.1: Estructura del cromosoma compuesto por 29 genes según el esquema de representación propuesto por Sinclair [1]

Cuadro 2.1: Valores de ejemplo para un individuo según el esquema de representación elegido por Sinclair

Atributos	Valor de ejemplo
IP Origen	c0.a8.da.01 (192.168.218.1)
IP Destino	c0.a8.db.46 (192.168.219.2)
Puerto Origen	002121
Puerto Destino	000080
Protocolo	1 (TCP)

Li [2] también emplea un esquema similar en el que se incluyen 57 genes para poder representar los atributos seleccionados. Se utilizan 8 genes tanto para la dirección IP origen como la dirección IP destino, 5 para el puerto origen y 5 para el puerto destino respectivamente, 8 genes para el tiempo de duración de la conexión, 10 genes para los bytes transferidos y recibidos, 2 genes para mantener información sobre la conexión y 1 gen indicar el tipo de protocolo.

El sistema de codificación elegido por Li es idéntico al propuesto por Sinclair. Se utiliza un dígito decimal para la representación de cada gen, excepto para los atributos IP origen e IP destino, en donde se utilizan dígitos hexadecimales.

La diferencia en el número de genes radica en que se han tenido en cuenta una mayor cantidad de atributos. En el Cuadro 2.2 puede observarse el esquema de representación seguido por Li.

Cuadro 2.2: Valores de ejemplo para un individuo según el esquema de representación elegido por Li

Atributos	Valor de Ejemplo
IP Origen	c0.a8.da.01 (192.168.218.1)
IP Destino	c0.a8.db.46 (192.168.219.2)
Puerto Origen	02121
Puerto Destino	00080
Duración de la conexión	20
Estado	11
Protocolo	2
Bytes Transferidos	0000007320
Bytes Recibidos	0000038891

Con los valores de ejemplo que se presentan en el Cuadro 2.2 conforma un

cromosoma como el que se muestra en la figura 2.2

(12,0,10,8,13,10,0,1,12,0,10,8,13,11,4,6,0,2,1,2,1,0,0,0,8,0,0,0,0,
0,0,0,2,0,1,1,2,0,0,0,0,0,0,7,3,2,0,0,0,0,0,0,3,8,8,9,1)

Figura 2.2: Estructura del cromosoma compuesto por 57 genes de acuerdo al esquema de representación propuesto por Li [2]

En el trabajo de Gong et. al [14] también se utiliza un cromosoma compuesto por una lista de genes para la representación de cada individuo. La diferencia consiste en que en vez de utilizar dígitos decimales y hexadecimales para representar el valor de cada gen, se utiliza un esquema mixto, con genes representados por tipo de datos que pueden variar dependiendo del atributo a representar como se muestra en la Figura 2.3.

(0,0,2,5,2121,80,192,168,218,1,192,168,219,70,1)

Figura 2.3: Cromosoma que representa una regla de tráfico.

De esta manera atributos como el puerto de origen y destino pueden estar compuestos por un único gen cuyo tipo de dato sea entero y otros atributos como la dirección IP origen y la dirección IP destino pueden estar compuestos por 4 genes cuyo tipo de dato sea un byte. Luego, un individuo como el representado en la Figura 2.3 representa una regla como la expuesta en la Figura 2.4.

if (*Tiempo*=2 **and** *PuertoSrc*=2121 **and** *PuertoDst*=80 **and** *IpSrc*=192.168.218.1 **and** *IpDst*=192.168.219.70 **) then** *intrusión*

Figura 2.4: Regla obtenida a partir del cromosoma de la figura 2.3

Un enfoque diferente para la representación de los individuos que conforman la población es utilizar árboles sintácticos en lugar de cadenas binarias de longitud fija como se propone en los trabajos de Lu [15] y Yin [16]. Este enfoque constituye una variante de los algoritmos genéticos conocida con el nombre de Programación Genética (GP) [18]. En el trabajo de Lu [15] cada regla de clasificación se representa como un árbol de derivación el cual es descripto utilizando una cadena de longitud variable. De esta manera un

árbol puede ser representado como *"AabAcdAceI".* La letra *A* se utiliza para representar el operador lógico *"and"*, las letras *a*, *b*, *c*, *d* y e representan las distintas condiciones y finalmente la letra *I* se utiliza para representar la consecuencia de la regla.

Las alternativas de representación propuestas no están basadas en cadenas de longitud fijas. Por lo que las operaciones a nivel de bits comúnmente utilizadas no pueden ser aplicadas. Debido a esto la aplicación de los distintos operadores genéticos podría aumentar considerablemente el tiempo de convergencia del algoritmo.

2.1.3. Función de optimización

En el trabajo de Li [2] se propone la utilización de pesos con el objetivo de destacar la importancia de ciertos atributos de las instancias de tráfico. La utilización de pesos responde a que por experiencia disciplinar ciertos atributos se consideran más importantes que otros en el dominio de la detección de intrusos.

Li considera que la dirección IP destino constituye el objetivo de la intrusión, mientras que la dirección IP origen, es justamente el origen probable de la intrusión, por lo que son los atributos más importantes y se les asigna un peso más alto. El puerto destino permite identificar la aplicación contra la cual se intenta realizar la intrusión (por ejemplo el puerto destino 21 corresponde al servicio de FTP). Los restantes atributos son de menor importancia, pero son útiles para la detección de cierto tipo de intrusiones. En la figura 2.5 se presenta el valor relativo escogido para los pesos de las instancias de tráfico.

Para el obtener el valor de fitness de un individuo, primero se calcula el resultado total de acuerdo a la ecuación (2.1).

$$resultado = \sum_{i=1}^{n=57} Coincidencia * peso \tag{2.1}$$

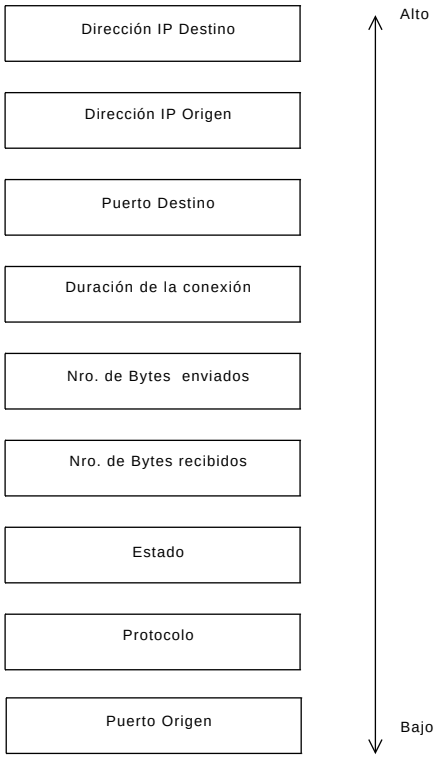

Figura 2.5: Orden de importancia en los atributos de una instancia de tráfico según Li[2].

Donde cada gen i que presente coincidencia con el correspondiente elemento de la instancia de tráfico, es multiplicado por el peso que corresponda de acuerdo al orden relativo de la figura 2.5 y posteriormente sumado al valor total del resultado. El proceso anterior se repite por cada uno de los 57 genes que componen el cromosoma del individuo. La variable $Coincidencia$ puede tomar valores 0 y 1.

Posteriormente se calcula el valor absoluto de la diferencia entre el *resultado* y un umbral de sospechabilidad de acuerdo a la ecuación (2.2). Este umbral se utiliza para determinar coincidencias entre la regla evaluada y una instancia de tráfico. El valor de este umbral de sospechabilidad se determina previamente de acuerdo a información analizada a priori por expertos.

$$\triangle = |resultado - sospechabilidad| \tag{2.2}$$

Cuando una regla no presenta coincidencias con una instancia de tráfico es

penalizada. Su penalización se calcula utilizando la diferencia $\triangle$ y un valor de ranking asociado de acuerdo a la ecuación (2.3). El valor de ranking se utiliza para establecer el nivel de dificultad que presenta la detección de la intrusión. Aquellas instancias de tráfico difíciles de reconocer tienen un valor de ranking bajo, mientras que las instancias de tráfico con intrusiones fáciles de reconocer tiene un valor de ranking alto.

$$penalizacion = \frac{\triangle * ranking}{100} \tag{2.3}$$

Por ejemplo si el individuo falla en clasificar una instancia de tráfico conteniendo un anomalía muy fácil de reconocer, es fuertemente penalizado.

Finalmente para calcular la función de fitness se utiliza la penalización descripta en la ecuación (2.3) , tal cual se observa en la ecuación (2.4). El valor de la función de fitness está en el rango de 0 y 1, donde los individuos de mejor calidad tienen un valor de la función de fitness más alto.

$$fitness = 1 - penalizacion \tag{2.4}$$

La propuesta de Li resulta interesante, aunque no se discute claramente como se obtiene ni que valores tiene el umbral de sospechabilidad. La utilización de un esquema basado en pesos permite obtener una función de fitness con mayor información, lo que puede mejorar el comportamiento del algoritmo.

Por otra parte Lu [15] propone la utilización de una función de fitness basada en el *support-confidence framework* [25]. Se entiende como *Support* a una tasa con el número de instancias de tráfico clasificadas correctamente por cada regla sobre el número total de reglas. Mientras que el factor de confianza representa la precisión de cada regla, es decir la confianza de que, bajo ciertas condiciones la consecuencia sea cierta.

De esta manera si una regla se describe como *si A entonces B,* se puede determinar el fitness de dicha regla a partir de la ecuación (2.5):

$$
\begin{aligned}
support &= |A \cup B|/N \\
confidence &= |A \cup B|/|A|
\end{aligned} \tag{2.5}
$$

Donde $|A|$ es el número de instancias de tráfico que satisfacen la condición A. $|B|$ es el número de instancias de tráfico que satisfacen el consecuente B. Y $|A \cup B|$ es el número de instancias de tráfico que satisfacen la condición A y el consecuente B.

Debido a que una regla con un factor de confianza alto no necesariamente se comporta diferente del promedio, se utiliza un factor de confianza normalizado $confidence_n$. Este factor de confianza normalizado es definido como la probabilidad promedio del consecuente. $prob.$

$$
\begin{aligned}
confidence_n &= confidence * log(confidence/prob) \\
prob &= |B|/N
\end{aligned} \tag{2.6}
$$

Pueden existir reglas muy precisas con un valor de *support* bajo. Se trata de reglas con valor de $confidence$ alto, pero con una frecuencia de aparición baja en el conjunto de entrenamiento. Con el fin de evitar la pérdida de tiempo que implica la evolución de aquellas reglas con un valor de *support* bajo se define una valor mínimo para *support.* La versión final de la función de fitness se presenta en el algoritmo 1.

Si alguna regla cuenta con un valor de *support* por debajo del mínimo definido ($support_n$), el valor del factor de confianza para dicha regla no es tenido en cuenta en el valor de fitness. En caso contrario la función de fitness está compuesta por el radio de *support* y el valor del factor de confianza además de los pesos asociados w_1 y w_2. La variación en los valores de estos pesos w_1 y w_2 permite encontrar un balance entre la tasa de *support* y el factor de confianza.

Si bien en Lu [15] y Gong [14] se utiliza esta función de fitness para la

Algoritmo 1 Función de fitness basada en confidence support

if $support < support_n$ **then**
  $fitness = support$
else
  $fitness = w_1 * support + w_2 * confidence_n$
end if

detección de intrusos por mal uso, existen también antecedentes sobre la utilización de esta función de fitness para la detección de anomalías [16].

2.1.4. Otros operadores.

Para los operadores de cruzamiento y selección no se detallan implementaciones más allá de las propuestas de la literatura clásica de algoritmos genéticos [22] y programación genética [18]. En algunos trabajos [16] se descarta la utilización del operador de mutación por considerar que la mutación favorece la aparición de reglas invalidas. Sin embargo este problema se puede solucionar de manera simple, descartando a aquellas reglas que no cumplan con ciertos requisitos mínimos de validez a costa de aumentar el tiempo total de ejecución del algoritmo.

En los trabajos de Li [2], Lu [15], Sinclair [1] y Yin [16] se menciona la utilización de un operador genético conocido como descarte de condición (drop condition). El operador descarte de condición descarta de manera aleatoria algún gen del cromosoma de un individuo con el fin de hacerlo más general. Luego se obtiene un conjunto de reglas más generales que encuentren coincidencias con un mayor número de instancias de tráfico [12].

Un ejemplo de la utilidad del operador *descarte de condición* puede verse en la posibilidad de presentar rangos de direcciones IP que componen un segmento de red. De esta manera un individuo que expresa una regla como la de la figura 2.3 luego de la aplicación del operador de *descarte de condición* cambia a un individuo como el expresado en la figura 2.6.

(0,0,2,5,2121,80,192,168,218,1,10,*,*,*)

Figura 2.6: Cromosoma que representa una regla de tráfico generalizada.

2.1.5. Técnicas de nicho.

Como mencionan Miller y Shaw [26], en los algoritmos genéticos se utilizan poblaciones que con el tiempo convergen hacia una única mejor solución. Aplicado al dominio de este trabajo, el resultado estaría en aquel individuo que tenga coincidencias con la mayor cantidad instancias de tráfico. Dado que es altamente improbable que un solo individuo pueda ser un buen representante de todas las instancias de tráfico, resulta mucho más efectivo contar con un número mayor de individuos que presenten coincidencias con algunas instancias de tráfico.

En este contexto en la literatura [26, 27] se mencionan técnicas de nicho, como crowding y sharing. Las mismas se basan en el concepto de proximidad, expresado mediante una función distancia, que permite encontrar a los individuos más cercanos.

La técnica de Crowding se basa en escoger unos pocos individuos al azar y reemplazar uno de los mismos por un nuevo individuo en base a criterios de proximidad [26].

Sharing en cambio degrada el fitness de un individuo en función de la cantidad de individuos que estén próximos al mismo. Debido a la necesidad de calcular una distancia con cada individuo que compone la población, sharing resulta una opción que aumenta considerablemente la complejidad computacional del algoritmo.

En el trabajo de Li [2] se destaca la utilización de una variante de crowding aunque no se menciona cual, ni se dan detalles de su implementación. Sinclair [1] plantea la utilización de una variante de crowding utilizando distancia de Hamming para encontrar los individuos más cercanos.

Lu [15] propone una técnica diferente basada en la competición por token. Cada instancia de tráfico del conjunto de entrenamiento contiene un token. Si un individuo coincide con la instancia de tráfico adquiere el token. La

prioridad para obtener el token se determina en función de una probabilidad que tiene como base la calidad de cada individuo. Finalmente la cantidad de tokens obtenidos por cada individuo es utilizado como un parámetro de la función de fitness.

2.2. Otras aplicaciones de algoritmos genéticos en el contexto de IDS.

Además de la obtención de reglas de clasificación, los algoritmos genéticos se han aplicado exitosamente con otros objetivos dentro del contexto de IDS. Se discuten a continuación la utilización de algoritmos genéticos en la obtención de clasificadores difusos, la generación de modelos matemáticos que permiten describir el comportamiento de la red y la extracción de atributos más relevantes para ser posteriormente evaluados por algún otro algoritmo de aprendizaje automático.

2.2.1. Clasificadores difusos

En el contexto de detección de intrusos, un acceso a un alto número de puertos TCP diferentes, con la misma dirección IP origen, en un lapso corto de tiempo puede constituir una anomalía, como se expresa en la siguiente regla:

if *Nro. de puertos accedidos en los últimos 2 segundos es alto*
then *existe una anomalía*

Si se sigue el esquema de la lógica clásica para indicar el intervalo de valores que abarca la categoría *alto*, normalmente se divide el rango de posibles valores en intervalos discretos y se asigna un grado de membresía en cada uno de los intervalos (ver Fig. 2.7). Posteriormente se indica la pertenencia a una categoría asignando un grado 1 en la clase a la que pertenece y 0 en las restantes.

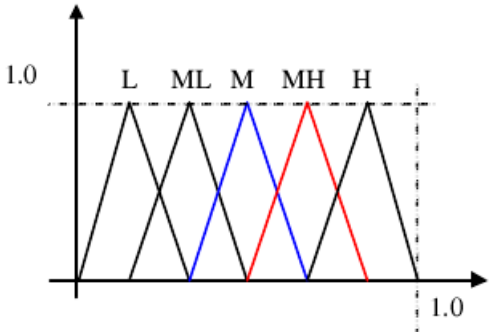

Figura 2.7: Representación de cinco conjuntos difusos

La utilización de la lógica difusa permite trabajar con conceptos que pueden pertenecer a más de una categoría. Para la lógica clásica se utiliza un intervalo para determinar cuando una instancia de tráfico es normal, así todo valor que se encuentre fuera del intervalo sera considerado anormal, sin tener en cuenta la distancia hacia dicho intervalo. Un esquema similar se sigue para determinar el nivel de anormalidad.

Como se observa en la Fig. 2.7 al utilizar lógica difusa un valor particular puede tener un valor de membresía que varia entre 0 y 1 y a su vez puede pertenecer a más de una categoría. En la terminología de lógica difusa el número de puertos accedidos constituye una variable difusa y las categorías constituyen un conjunto difuso, de esta manera la regla descripta arriba puede ser reescrita de la siguiente manera:

if *DP=alto* **then** *existe intrusion*

En Briges [28] se propone una arquitectura para un sistema de detección de intrusos inteligente basado en la aplicación de técnicas de data mining como ser reglas de asociación y episodios frecuentes, combinadas con el uso de lógica difusa. Se utilizan algoritmos genéticos para optimizar las funciones de membresía que determinan cada una de las categorías utilizadas, como así también para la selección de los atributos más relevantes para utilizar en las técnicas de minería de datos.

Gómez [29] utiliza algoritmos genéticos con el objetivos de obtener clasificadores difusos para detectar tráfico normal y anormal. El clasificador difuso está conformado en este caso por un conjunto de dos reglas, una para

la clase que incluye tráfico normal y otra para la clase que incluye tráfico anormal, donde el antecedente es definido utilizando expresiones atómicas y operadores de la lógica difusa aplicados sobre un cierto número de parámetros monitorizados y el consecuente es una expresión atómica para la clase seleccionada más un peso asociado a dicha regla. Un ejemplo sencillo, que utiliza sólo dos atributos, se presenta a continuación:

$$
\begin{aligned}
R_a &= IF\ A\ is\ HIGH\ and\ B\ is\ LOW\ THEN\ patron\ es\ anormal\ [0{,}4] \\
R_n &= IF\ A\ is\ LOW\ and\ B\ is\ MEDIUM\ THEN\ patron\ es\ normal\ [0{,}6]
\end{aligned}
$$

Como el consecuente y el peso son fijos el algoritmo genético utiliza únicamente el antecedente de la regla, con este fin se destaca la utilización de un esquema de representación de longitud variable que permite encontrar las expresiones lógicas que permitan clasificar el mayor número de instancias de tráfico.

2.2.2. Selección de atributos.

La selección de atributos constituye un tópico importante dentro de la detección de intrusos, ya que a partir del gran número de atributos que deben ser monitorizados, puede identificarse aquellos que resultan relevantes y otros que se pueden descartar por no aportar información útil. La necesidad de descartar atributos irrelevantes permite mejorar la precisión en el proceso de detección de anomalías e intrusos y disminuir el tiempo computacional requerido.

El problema de la selección de una gran cantidad de atributos debe realizarse a través de métodos empíricos, ya que un análisis completo obligaría a examinar cada una de las posibles combinaciones.

En Sung [30] se utiliza programación genética lineal, una variante de algoritmos genéticos para la extracción de los atributos más significativos. De

manera aleatoria se seleccionan subconjuntos de los atributos. La performance de cada uno de estos subconjuntos es evaluada a través de una función de fitness que incluye un conjunto de entrenamiento y un conjunto de evaluación que permiten determinar la precisión de cada subconjunto. Cada atributo es considerado como un gen binario y a su vez cada individuo está conformado por una cadena binaria de longitud fija en donde cada bit representa la eliminación o inclusión de algún atributo del conjunto de atributos evaluados. La función de fitness F de un individuo p es definida en la ecuación (2.7), la cual calculada como el error medio cuadrado (MSE) entre el resultado esperado (O_{ij}^{esp}) y el resultado deseado (O_{ij}^{des}) de todos los n ejemplos de entrenamiento y sus m resultados.

$$F(P) = \frac{1}{n.m} \sum_{i=1}^{n} \sum_{j=1}^{m} (O_{ij}^{esp} - O_{ij}^{des}) + \frac{w}{n} CE \tag{2.7}$$

Se agrega a la función de fitness el error de clasificación CE, el cual es calculado como el número de clasificaciones incorrectas más un peso absoluto asociado.

En Shon et al[31] se utilizan algoritmos genéticos para realizar la elección de los atributos más apropiados de una instancia de tráfico para mejorar la taza de detección. Posteriormente estos atributos se utilizan en una variante de support vector machine para realizar el proceso de clasificación. Se propone un esquema de representación similar al discutido anteriormente en donde cada atributo es codificado como un gen binario indicando presencia con el valor uno y ausencia con el valor cero. La población original está conformada por individuos de una longitud de 24 bits, en donde 13 bits indican atributos del paquete IP y 11 bits hacen referencia al segmento TCP.

Para realizar la extracción de los atributos más relevantes se propone una función de fitness $f(x)$ la cual asigna diferentes pesos a los atributos, en función la frecuencia de aparición de los mismos en las anomalías del tráfico de red. Para esto se utilizó el conjunto de datos de DARPA [32]. Además también se asigna un peso a cada atributo de acuerdo a la importancia que este tenga en la comunicación. Se clasifican en: Estáticos, que no va-

rían durante la conexión, Dependientes, cuyo valor depende de la conexión y Dinámicos, aquellos que pueden cambiar dinámicamente. La función de fitness consiste en ecuación polinomial que tiene en cuenta las consideraciones mencionadas arriba como coeficientes.

$$f(X) \quad = \quad A(X) + N(X) \tag{2.8}$$

Donde $A(X)$ una función polinomial que asigna pesos en función de la aparición de un atributo en una anomalía de tráfico y $N(X)$ es una función polinomial que asigna pesos en función del rol del atributo en la comunicación. La función $A(X)$ se define como :

$$A(X) = (a_1 x_1 + .. + a_2 x_2 + a_j x_i) - u_A \tag{2.9}$$

donde $x_i$es el atributo i del individuo X y a_i son los coeficientes que indican el peso de ese campo en la detección de anomalías. El valor u_A es un término de sesgo que se utiliza para evitar el sobreajuste de la función de fitness. Por otra parte la función $N(X)$ es definida en la ecuación (2.10)como:

$$N(X) = \alpha(x_\alpha) + \beta(x_\beta) + \gamma(x_\gamma) - u_N \tag{2.10}$$

Los coeficientes α, β y γ hacen referencia a la importancia del atributo dentro de la comunicación y x_α,x_β y x_γ son los conjuntos asociados. Al finalizar el proceso evolutivo se obtienen los individuos cuya combinación de atributos maximicen la función de fitness propuesta.

En Dimitris et al [33] se utiliza un algoritmo genético para extraer los atributos de una instancia de tráfico para luego ser procesado por un detector basado en redes neuronales. El algoritmo encuentra el subconjunto óptimo sobre un conjunto de 44 atributos. Al igual que en los trabajos anteriores cada individuo está conformado por un número determinado de bits que indican la presencia o ausencia de algún atributo. Cada individuo es posteriormente evaluado mediante el detector basado en redes neuronales, en

donde sólo aquellos atributos que se encuentren presentes activarán las entradas del detector basado en redes neuronales. La función de fitness utiliza el error cuadrado medio entre la salida del detector neuronal y los resultados de un conjunto de prueba.

2.2.3. Evolución de modelos matemáticos

Chittur [34] utiliza un algoritmo basado en la técnica de programación genética aplicada a la regresión simbólica, en donde cada individuo representa un posible modelo matemático. Chittur destaca la utilización de un tipo especial de terminales dentro del árbol sintáctico para la generación coeficientes basados en Constantes Aleatorias Efímeras [18] (ERC de sus siglas en inglés). Se trata de ciertos valores generados aleatoriamente para favorecer el desarrollo de constantes durante el proceso de evolución de los modelos matemáticos.

Son estos coeficientes los que después se utilizan para determinar el grado de certeza C. La fórmula para determinar la certeza de un modelo matemático se define en la ecuación (2.11).

$$C_i(\chi) = \sum_{j=1}^{n} (\Re_{i,j}\chi_j) \tag{2.11}$$

En donde un instancia de tráfico χ se clasifica como un ataque por un modelo i, a su vez $\Re$ indica el coeficiente basado en el ERC y n es el número de atributos de la instancia de tráfico. Si el valor de C para un instancia de tráfico χ supera un valor t elegido arbitrariamente, se considera que esa instancia de tráfico constituye un ataque.

Por su parte la función utilizada para calcular el fitness de un individuo δ se define como :

$$F(\delta_i) = \frac{\alpha}{A} - \frac{\beta}{B} \tag{2.12}$$

Donde α es el número de ataques correctamente detectados y A es el número total de ataques, mientras que β representa el número de falsos positivos y B el número total de instancias de tráficos normales. De esta manera una alta taza en la detección correcta de los ataques y una taza baja en la ocurrencia de falsos positivos dan como resultados una puntuación alta en el fitness del individuo evaluado. Mientras que una taza de detección baja y un valor alto en la taza de falsos positivos ocurridos traen como consecuencia un valor en la función de fitness bajo.

2.3. Conclusiones

En este capitulo se han revisado diferentes alternativas para la detección de intrusos basadas en algoritmos genéticos. En la mayoría de los trabajos revisados se propone la utilización de algoritmos genéticos para la obtención de reglas de clasificación de tráfico. Se observa también el empleo de estos algoritmos para la selección de los atributos más relevantes para la detección de intrusos, la generación de modelos matemáticos que permitan describir el comportamiento del tráfico de red y el ajuste de las funciones de membresía de clasificadores difusos.

Se considera que los trabajos que plantean el uso de algoritmos genéticos para obtener un conjunto de reglas de clasificación son de mayor utilidad ya que son capaces de reconocer los ataques e intrusiones a la red a la vez que aportan información valiosa para el administrador de redes. El conjunto de reglas obtenido no sólo sirve para encontrar coincidencias con instancias de tráfico que contienen anomalías o ataques. Además permite conocer en detalle en que consisten dichas anomalías o ataques. Lo que no ocurre con otras técnicas como es el caso de los modelos matemáticos propuestos por Chittur [34].

En los trabajos revisados se hace referencia a sistemas de clasificación basados en reglas que utilizan instancias de tráfico que contienen anomalías. Muchos de estos sistemas presentan el inconveniente de no ser capaces de reconocer nuevos tipos de ataques. Otro inconveniente es que para alimentar al sistema se necesita de un conjunto de instancia de tráfico que

contengan anomalías o ataques, las cuales deben previamente ser identificadas por un experto en seguridad de redes.

Con el fin de superar los inconvenientes citados, en este trabajo se escoge una alternativa que consiste en obtener un conjunto de reglas en base a coincidencias en el tráfico normal y toda nueva instancia de tráfico que se aparte del tráfico normal puede ser considerada una anomalía. Si bien en la mayoría de los trabajos revisados se hace referencia a la detección de intrusos a partir de instancias de tráfico que contienen anomalías, muchas de las técnicas utilizadas se pueden emplear en el contexto del presente trabajo.

Resultan interesantes algunas de las propuestas mencionadas en Gong [14]. Especialmente en lo referente a los individuos de la población. Se considera que los atributos seleccionados ofrecen un buen balance entre la cantidad de información que se puede obtener y los requerimientos computacionales necesarios para su procesamiento.

Otros esquemas de representación de la población, como los basados en árboles sintácticos [17, 16], no se consideran en este trabajo por tratarse de un esquema de representación más complejo en muchos casos, podría llegar a aumentar los requerimientos computacionales a niveles prohibitivos.

La función de fitness propuesta en Li [2] presenta características interesantes. Si bien no se presentan resultados experimentales. Luego resulta de interés obtener resultados con el objetivo de evaluar el comportamiento de la misma. Sin embargo debido a que existen elementos como el umbral de sospechabilidad y el ranking, cuyos valores necesitan ser determinados a priori por un experto, para este trabajo sólo se considera el uso de pesos para favorecer a algunos atributos más relevantes.

En los trabajos revisados se observa una gran demanda de recursos computacionales necesarios para la implementación de las distintas propuestas. Por lo que se desprende la necesidad de estudiar algunas alternativas para disminuir los tiempos de procesamiento y consecuentemente disminuir los requerimientos computacionales de los algoritmos propuestos.

3 Detección de Anomalías Basada en Algoritmos Genéticos

En este capítulo se propone un sistema de detección de anomalías basado en algoritmos genéticos. El algoritmo genético propuesto obtiene un conjunto de reglas de clasificación que se utilizan para encontrar coincidencias con las instancias de tráfico normal de la red. Así luego toda instancia de tráfico que no presente coincidencias con el conjunto de reglas obtenido se considera una anomalía en el tráfico de red.

Este enfoque es distinto al propuesto en algunos de los trabajos mencionados en los antecedentes [1, 2, 15, 16, 17], en los cuales se busca obtener los patrones de tráfico de las instancias que presenten anomalías. Sin embargo muchas de las técnicas utilizadas en el estado del arte se pueden emplear en el contexto del presente trabajo.

En la sección 3.1 se presenta una arquitectura para un sistema de detección de intrusos (IDS) en el cual se identifican los módulos de captura de tráfico y preprocesamiento, que se tratan en las secciones 3.2 y 3.3 respectivamente. El módulo de aprendizaje que constituye el núcleo de la arquitectura se discute en la sección 3.4. Finalmente el módulo de evaluación se trata en la sección 3.5

3.1. Sistema de Detección de Anomalías.

La arquitectura general propuesta para un sistema de detección de anomalías en el tráfico de red se muestra en la figura 3.1, en la cual se reconocen cuatro módulos diferentes.

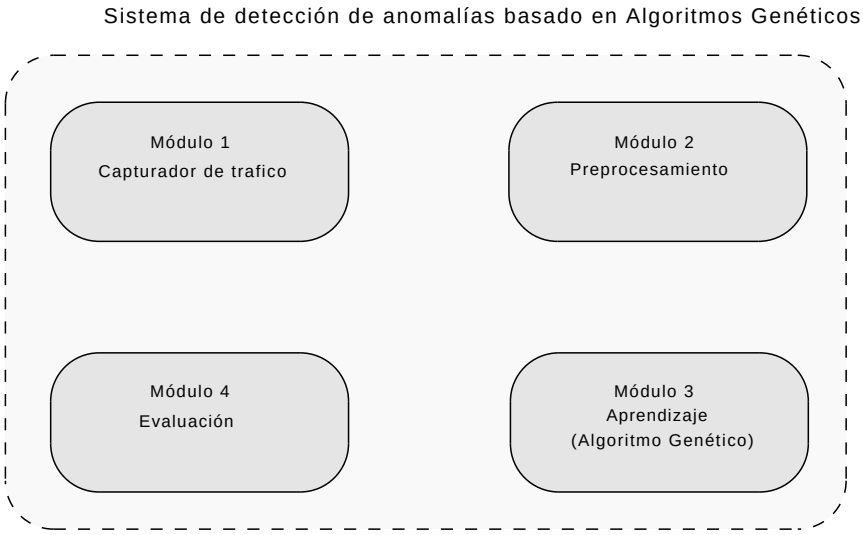

Figura 3.1: Arquitectura del sistema de detección de anomalías en el tráfico de red.

Los módulos que componen el sistema son:

1. Módulo capturador de datos, el cual se encarga de la captura en crudo del tráfico de la red.
2. Módulo de preprocesamiento, encargado de tomar los datos en crudo y realizar las modificaciones necesarias para luego ser procesados por el módulo de aprendizaje.
3. Módulo de aprendizaje, el cual se encarga de la obtención de reglas de clasificación con el objetivo de detectar instancias de tráfico normales. Para obtener dichas reglas de clasificación, el módulo de aprendizaje está basado en algoritmos genéticos. El módulo de aprendizaje constituye la parte central de este trabajo.
4. Módulo de evaluación de las instancias de tráfico, encargado del reconocimiento de anomalías en el tráfico de red.

En el sistema de detección de anomalías se observan dos etapas. La primera, denominada aprendizaje, tiene como objetivo a partir de un conjunto compuesto por instancias de tráfico, obtener un conjunto de reglas que sean capaces de encontrar coincidencias con el mayor número posible de instancias de tráfico.

En la segunda etapa, denominada evaluación, se utilizan las reglas aprendidas durante la etapa anterior y se evalúan nuevas instancias de tráfico con el objetivo de determinar cuales constituyen una anomalía.

No todos los módulos participan en las dos etapas. En la Figura 3.2 se muestra la relación entre los módulos que intervienen durante la etapa de entrenamiento. En el módulo 1 se captura un conjunto de trazas de tráfico en crudo las cuales se preprocesan por el módulo 2, lo que da origen al conjunto de entrenamiento. Luego dicho conjunto se utiliza en el módulo 3 para, mediante un algoritmo genético, obtener un conjunto de reglas. Al finalizar el aprendizaje de las reglas, en el módulo 4 se evalúa el conjunto de entrenamiento con el objetivo de determinar el error de clasificación del mismo.

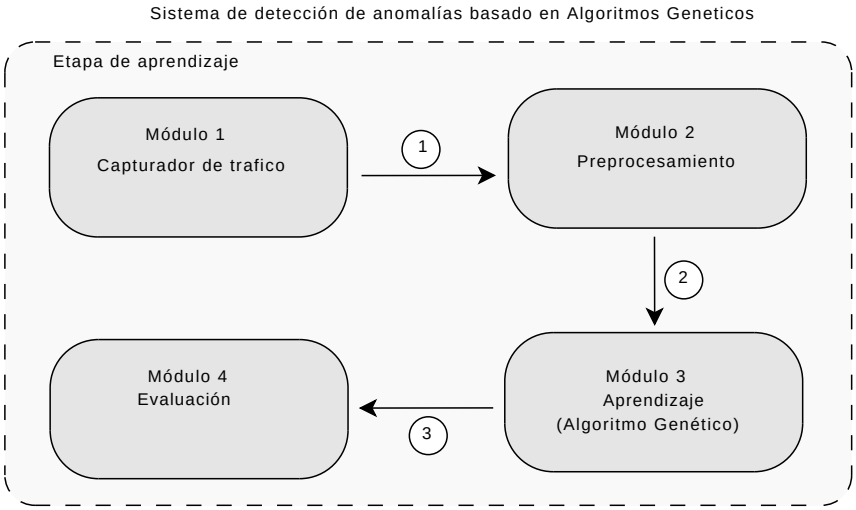

Figura 3.2: Arquitectura del sistema. Módulos que participan en la etapa de aprendizaje.

La Figura 3.3 muestra la relación entre los módulos que intervienen durante la etapa de evaluación. Durante la misma se captura en el módulo 1 un

conjunto de las trazas de tráfico, las cuales se preprocesan en el módulo 2 y finalmente se evalúan en busca de anomalías en el módulo 4.

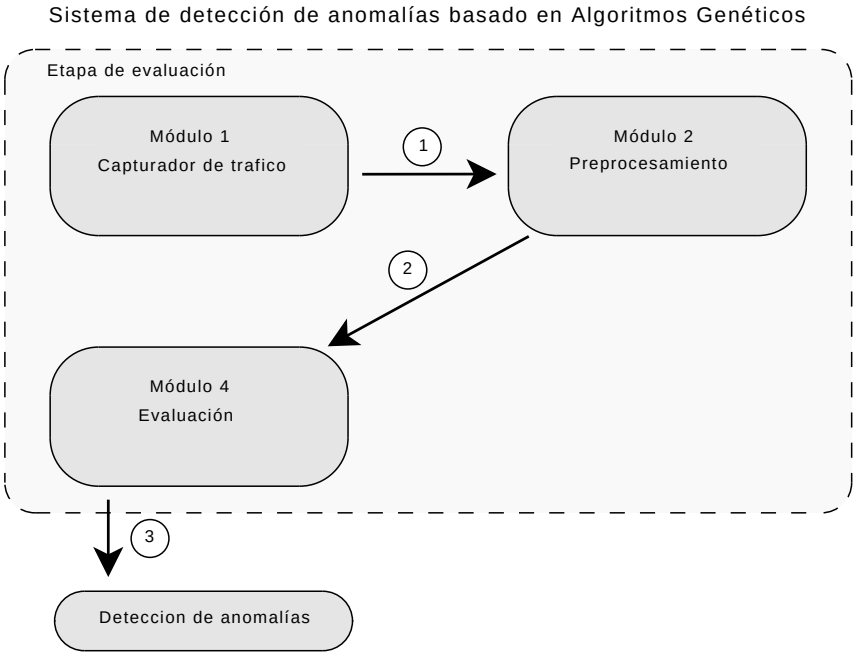

Figura 3.3: Arquitectura del sistema . Módulos que participan en la etapa evaluación.

En las siguientes subsecciones se discuten cada uno de los módulos que conforman la arquitectura propuesta, con especial énfasis el módulo de aprendizaje, uno de los objetivos centrales de este trabajo.

3.2. Módulo de Captura de Tráfico

La función principal del módulo capturador de tráfico es la recolección de trazas de tráfico de red en crudo, las cuales se utilizan posteriormente por el módulo de preprocesamiento. Las trazas de tráfico obtenidas por este módulo se utilizan para formar el conjunto de entrenamiento durante la primera etapa. Como así también es función del módulo capturador de tráfico recolectar las trazas de tráfico de red en crudo para su ser utilizadas durante la etapa de evaluación.

Muchas de las funciones del módulo capturador de tráfico son llevadas a cabo por herramientas específicas para la captura y análisis de tráfico de red, como es el caso de *tcpdump* [35].

3.3. Módulo de Preprocesamiento

El módulo de preprocesamiento se encarga de obtener el conjunto de atributos utilizados por los módulos de aprendizaje y clasificación a partir de los campos más relevantes de la traza de tráfico en crudo.

La función de este módulo no sólo se limita a obtener información de los campos de la traza de tráfico, sino que en muchos casos puede operar sobre éstos con el objeto de obtener algún atributo que resulte de interés, pero que no se muestre de manera explícita en los conjunto de datos obtenidos por el módulo de captura de tráfico. La inclusión de este módulo es necesaria ya que en la práctica resulta muy difícil trabajar con todos los atributos de la instancia de tráfico debido al alto costo computacional. Luego se debe realizar una selección de los mismos.

Para la selección de atributos se sigue a Gong [14], con la diferencia que en el presente trabajo se propone encontrar coincidencias con las instancias normales del tráfico de red. Luego se seleccionan en base a criterios disciplinares 6 atributos de una instancia de tráfico: tiempo de duración de la conexión, tipo de protocolo, puerto origen, puerto destino, dirección IP origen y dirección IP destino. La elección de los últimos 5 atributos permiten identificar unívocamente a cada instancia de tráfico. Mientras que la elección de un atributo que considere el tiempo de duración de la instancia de tráfico, permite detectar algunas anomalías que se producen a lo largo de grandes períodos de tiempo, las cuales normalmente pasan desapercibidas.

La dirección IP origen y destino y el puerto origen y destino se extraen de manera directa de la trama Ethernet obtenida por el módulo de captura. La figura 3.4 presenta gráficamente la ubicación de los campos seleccionados dentro de una trama Ethernet. En este caso el paquete IP, ubicado

al comienzo del área de datos, contiene en su área de datos un segmento TCP. De esta manera las direcciones IP origen y destino se obtienen de la cabecera del paquete IP. Los puertos de origen y destino por su parte se obtienen de la cabecera del segmento TCP .

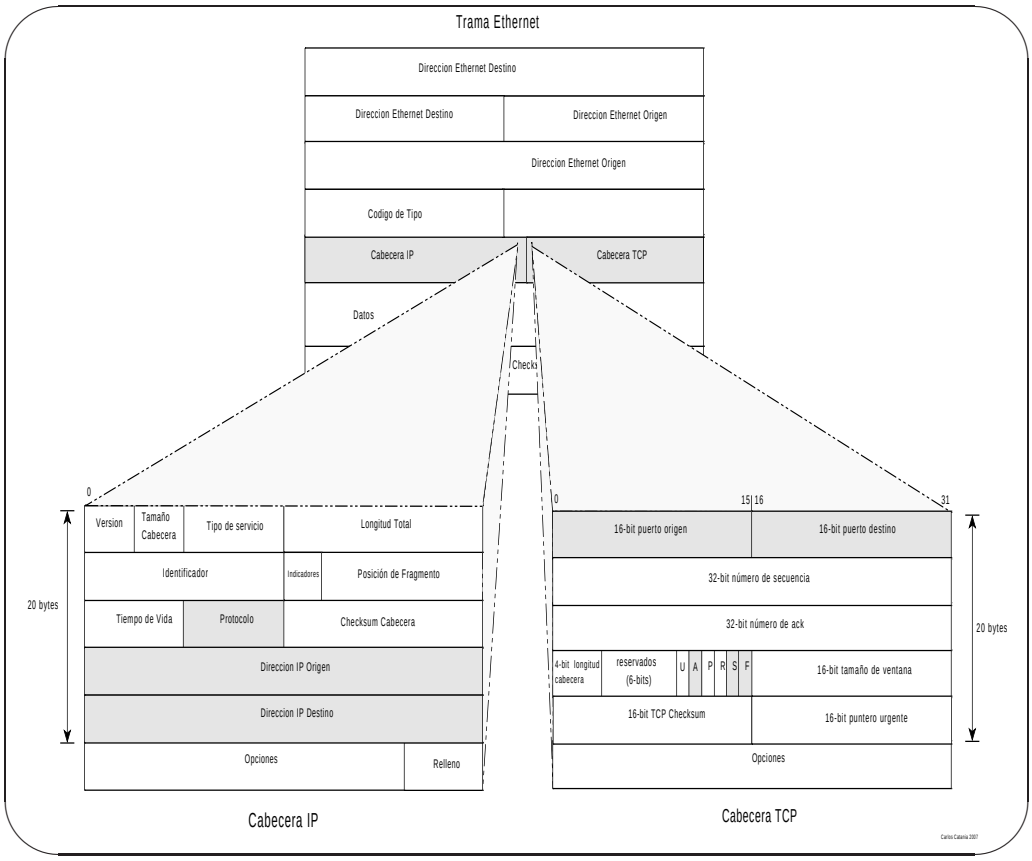

Figura 3.4: Campos de los protocolos IP y TCP utilizados para la conformación de los atributos.

El tiempo de duración de la conexión no se puede extraer directamente de la trama Ethernet, ni de ninguno de los campos de las cabeceras de los protocolos TCP y UDP. En este caso el módulo de preprocesamiento debe inferir el valor del tiempo transcurrido a partir de los datos que estén disponibles.

En el caso de una conexión TCP, cuando se observa el último mensaje de aceptación de inicio de la conexión (ACK) se considera que la conexión ha comenzado y se utiliza este momento para comenzar a medir el tiempo de duración. En la figura 3.5 (a) se representa gráficamente la secuencia de mensajes que da origen a una conexión entre los nodos A y B.

Al finalizar el último mensaje de aceptación de fin de la conexión (ACK) se considera que la conexión ha finalizado y en este momento se deja de contar el tiempo de duración de la conexión TCP. La figura 3.5 (b) secuencia de mensajes que pone fin a una conexión entre los nodos A y B.

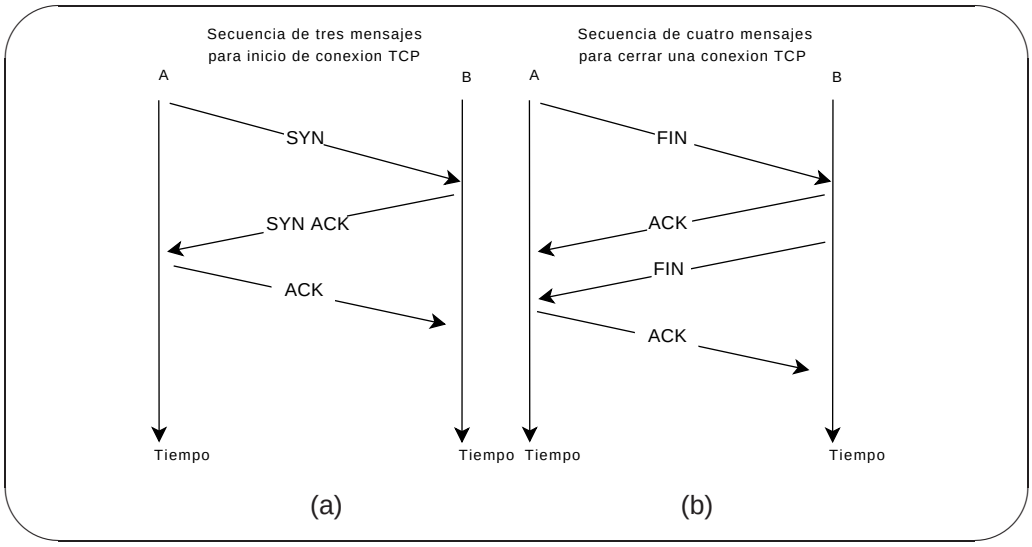

Figura 3.5: Secuencia de mensajes para el inicio y finalización de la conexión TCP. Utilizadas para calcular el tiempo de duración de una instancia de tráfico.

En el caso de otros protocolos no orientados a la conexión no se puede determinar el tiempo de duración de la conexión de la misma forma que TCP, entonces se plantea un estrategia diferente. Para el caso de instancias de tráfico UDP se identifica a las mismas por las direcciones IP origen y destino y los puertos origen y destino [36]. La duración de la instancia de tráfico UDP se toma desde la primera vez que se observa una instancia de tráfico UDP identificada unívocamente hasta que transcurren 10 segundos desde la última vez que se observa la misma. El mismo esquema se sigue en el caso particular de ICMP sólo que se tienen en cuenta el tiempo de inicio de la instancia de tráfico y las direcciones IP origen y destino [37, 38].

El identificador de los protocolos de las capas 3 y 4 del modelo de referencia OSI, se puede obtener directamente de la trama Ethernet. Sin embargo en algunos casos resulta útil identificar protocolos de capas superiores, como es el caso de los protocolos HTTP o FTP. En el caso de tratarse de puertos conocidos es posible relacionar el puerto de destino de una conexión con el protocolo utilizado. Con este fin el módulo de preprocesamiento cuenta

con una tabla que posee la información necesaria para establecer dicha relación.

Se observa que mediante la selección de estos atributos se pueden representar los elementos más importantes de los protocolos basados en TCP y UDP. Sin embargo no sucede lo mismo con otros protocolos como es el caso de ICMP. Dado que TCP y UDP son dos de los protocolos más utilizados en la actualidad, los atributos escogidos son suficientes para los objetivos de este trabajo.

3.4. Módulo de Aprendizaje

El módulo de aprendizaje constituye el núcleo del sistema de detección de anomalías en el tráfico de red. El mismo está compuesto por un algoritmo genético cuyo objetivo es el aprendizaje de un conjunto de reglas para la búsqueda de patrones que permitan reconocer las instancias normales en el tráfico de red [39, 40]. Posteriormente toda nueva instancia de tráfico que no sea reconocido por el conjunto de reglas, puede considerarse como una posible anomalía.

En la sección 3.4.1 se presenta el funcionamiento general del algoritmo. Luego se introducen los componentes principales del mismo, entre los que se destaca el esquema que se utiliza para la representación de la población, que se discute en la sección 3.4.2; la función de optimización (3.4.3) y las alternativas para selección en la sección 3.4.4. En la sección 3.4.5 se discute el operador de cruzamiento y en la sección 3.4.6 las técnicas para mantener un conjunto de soluciones posibles. Los operadores de descarte de condición y de mutación se discuten en las secciones 3.4.7 y 3.4.8, respectivamente. Finalmente en la sección 3.4.9 se propone una heurística para obtener un subconjunto con los mejores individuos a partir de la población obtenida al finalizar el proceso de aprendizaje.

3.4.1. Esquema General del Algoritmo

A partir de los antecedentes discutidos en la sección 2.1 del capítulo 2 y empleando como base un algoritmo genético general, se propone un algoritmo para la búsqueda de patrones que permitan reconocer las instancias normales en el tráfico de red.

Se muestra en la figura del Algoritmo 2 el pseudocódigo del algoritmo genético propuesto. Como se observa en el mismo primero se genera la población inicial P a partir de instancias de tráfico escogidas al azar. Luego en la línea 2 se calcula el fitness de cada uno de los individuos que componen la población. A continuación se ingresa al bucle principal del algoritmo genético, en donde la condición de salida está fijada por un número máximo de generaciones que el algoritmo debe alcanzar (línea 3).

Algoritmo 2 Algoritmo genético propuesto

```
1.  initializarPoblacion(P)
2.  calcularFitness(P)
3.  while NroMaxGeneracionNoalcanzado do
4.     for i = 0 to xfactor do
5.        p1 = seleccionar(P)
6.        p2 = seleccionar(P)
7.        (o1, o2) = cruzamiento(p1, p2)
8.        crowding(p1, p2, o1, o2)
9.     end for
10.    for i = 0 to dfactor do
11.       p = seleccionar(P)
12.       descarte(p)
13.    end for
14.    for i = 0 to mfactor do
15.       p = seleccionar(P)
16.       mutacion(p)
17.    end for
18.    calcFitness(P)
19. end while
20. selecionarIndividos(P)
```

Dentro de bucle, se selecciona probabilísticamente a dos individuos (p_1 y p_2), a los cuales se les aplica el operador de cruzamiento. Estos individuos

junto a los dos hijos recién obtenidos (o_1 y o_2), se evalúan mediante un operador de crowding. En el operador de crowding se aplica un criterio para determinar si alguno de los hijos recientemente obtenidos, será agregado a la población P. Este procedimiento se repite un número de veces determinado por el factor de cruzamiento $xfactor$.

El algoritmo sigue un esquema de tipo estacionario [41]. Luego los dos individuos (o_1 y o_2) obtenidos, pueden ser agregados de forma inmediata a la población P. De esta manera el individuo que recién ingresa a la población, puede ser seleccionado para cruzamiento sin necesidad de que el algoritmo pase a la siguiente generación. Esta variante de algoritmos genéticos provee un mayor control sobre la población generada y permite un uso más eficiente de memoria.

Posteriormente en la línea 12, se selecciona probabilísticamente un nuevo individuo p el cual es utilizado por el operador de descarte de condición (dropping condition). Al igual que en el caso del operador de cruzamiento este proceso se repite el número de veces indicado por el factor de descarte de condición $dfactor$. Un esquema similar se sigue con el operador de mutación. Luego se recalcula el fitness de aquellos individuos de la población que hayan sido agregados o modificados por los operadores mencionados con anterioridad.

Finalmente en la línea 20 se seleccionan los mejores individuos de la población P para formar el conjunto de reglas que representan las instancias normales en el tráfico de red.

3.4.2. Representación de la Población.

Como se mencionó en la sección 3.3 de este capítulo, se sigue el criterio de Gong [14], que escoge en base a criterios disciplinares 6 atributos de una instancia de tráfico. Los mismos se representan como una lista compuesta de 14 genes, de acuerdo a la estructura indicada en el Cuadro 3.1.

Este esquema de representación es diferente al propuesto por Gong. La diferencia consiste en que en vez de utilizar los tipos de datos byte, ente-

Cuadro 3.1: Estructura del cromosoma de un individuo.

Atributo	Nro. de genes	Valor máximo
HH:MM:SS	3	60
Puerto origen	1	65535
Tipo Protocolo	1	1024
Puerto destino	1	65535
Dirección origen	4	255
Dirección destino	4	255

ro y coma flotante para la representar el valor de cada gen, se utiliza un esquema con valores enteros para genes, cuyo valor máximo puede variar dependiendo del atributo a representar. De esta manera algunos atributos como el puerto de origen y destino están compuestos por un único gen cuyo valor máximo es 65535 y otros atributos como la dirección IP origen y la dirección IP destino están compuestos por 4 genes cuyo valor máximo es 255.

En la Figura 3.6 se muestra un ejemplo de un cromosoma que representa a un individuo. El ejemplo representa una conexión http (la cual se identifica con el número 5) de una duración de 2 segundos con puerto origen 2114, puerto destino 80, dirección IP destino 192.168.1.1 y dirección IP origen 192.168.1.2.

(0,0,2,5,2114,80,192,168,1,1,192.168,1,2)

Figura 3.6: Cromosoma de un individuo

Los individuos de la población admiten la posibilidad de generalizar alguno de sus genes, asignándose al mismo el valor -1. De esta manera se puede obtener individuos que presenten coincidencias con un mayor número de instancias de tráfico. Luego el cromosoma redefinido de acuerdo a esta generalización se muestra en la Figura 3.7

(0,0,2,5,2114,80,192,168,1,-1,192.168,-1,-1)

Figura 3.7: Cromosoma de un individuo generalizado

Este individuo puede encontrar coincidencia con las instancias de tráfico

dirigidas u originadas en los nodos de la red comprendidos entre las direcciones IP 192.168.1.0 y 192.168.255.255.

La población inicial es generada a partir de instancias de tráfico del conjunto de entrenamiento, seleccionadas aleatoriamente en base a una función de probabilidad uniforme.

3.4.3. Función de Optimización.

Se propone una función de optimización con el fin de obtener un conjunto de individuos que encuentren coincidencias con la mayor cantidad posible de instancias de tráfico.

La función propuesta sigue parcialmente el trabajo de Li [2] discutido en la sección 2.1.3 del capítulo 2. En el mismo se propone utilizar pesos para favorecer a determinados atributos de la instancia de tráfico. A diferencia de la propuesta de Li, en este trabajo, no se consideran el umbral de sospechabilidad ni el ranking, debido a que se trata de valores que requieren la validación de un experto.

La función propuesta se define según la ecuación (3.1):

$$f(r) = \frac{\sum_{j=1}^{m} \prod_{i=1}^{14} \alpha(r_i, d_{ji})}{|D|} \tag{3.1}$$

Para calcular el valor de fitness un individuo r, se comparan los genes del mismo con los correspondientes d_{ji} de cada una de la instancias de tráfico perteneciente al conjunto de entrenamiento D. Para ello se emplea una función α definida como:

$$\alpha(r_i, d_{ji}) = \begin{array}{l} w_i \text{ si } r_i = d_{ji} \\ w_i \text{ si } r_i = -1 \\ 0 \text{ si } r_i \neq d_{ji} \end{array} \tag{3.2}$$

Donde cada gen tiene un peso asociado w_i con el objeto de favorecer a aquellos atributos que por experiencia disciplinar resultan más relevantes. El peso asociado w_i se considera cuando el gen r_i presenta coincidencias con la posición correspondiente en la instancia de tráfico d_j. El mismo valor se asigna a los genes generalizados (indicados con el valor -1).

En la práctica como se puede observar en el capítulo 5, la asignación de w_i a los genes generalizados ha dado buen resultado. Sin embargo en trabajos futuros puede ser de interés investigar el valor de w_i , tanto en α como en la variante α' que se discute más adelante.

En caso de no presentar coincidencias en un gen, la función α asigna cero. Luego, como resultado de la productoria, la función de fitness penaliza a los individuos que no presenten coincidencias en todos sus genes asignándoles también un valor cero.

Con el fin de favorecer a los genes que presentan una frecuencia de aparición relativamente alta, se introduce una variante α' de la función de peso. La misma se define en la ecuación (3.3), la cual en lugar de asignar cero cuando algún gen de un individuo no presente coincidencia, le asigna un peso w'_i, que se ajusta en la práctica al 25 % del valor de w_i.

$$\alpha'(r_i, d_{ji}) = \begin{array}{c} w_i \text{ si } r_i = d_{ji} \\ w_i \text{ si } r_i = -1 \\ w'_i \text{ si } r_i \neq d_{ji} \end{array} \tag{3.3}$$

De esta manera la función de fitness, mediante α' permite obtener individuos que en futuras generaciones, podrían originar nuevas reglas que coincidan con un significativo número de instancias de tráfico.

Para la asignación del valor de los pesos en la función de fitness propuesta, se utiliza un esquema diferente al propuesto por Li. En este trabajo se le asigna un valor alto a los genes que se consideran menos relevantes, con el fin de favorecer la generalización de los mismos.

El ordenamiento de los valores para los pesos asociados se presenta en la Figura 3.8.

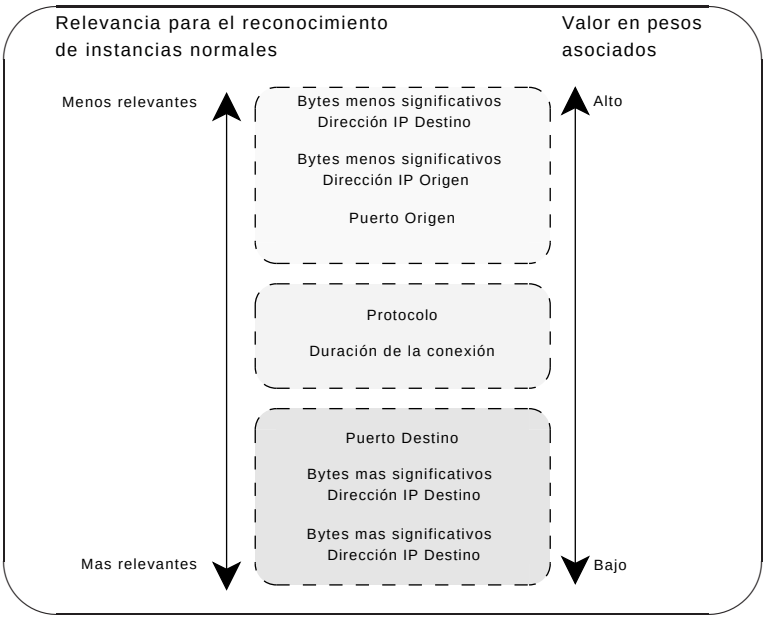

Figura 3.8: Orden de importancia en los atributos de una instancia de tráfico

De acuerdo a la experiencia disciplinar, se consideran genes de poca relevancia para encontrar coincidencias con un número de instancias de tráfico a los bytes menos significativos de la dirección IP origen y destino junto al puerto de origen. En consecuencia a los pesos asociados a los atributos se les asigna un valor alto con el objetivo de que en futuras generaciones pudieran ser generalizados.

En el caso de el tipo de protocolo, la duración de la conexión, los bytes más significativos de la dirección IP origen y destino junto al puerto destino, se consideran de mayor relevancia y se les asigna un peso más bajo.

3.4.4. Operador Selección

Para el algoritmo propuesto se utiliza la selección proporcional [22, 24], uno de los de los esquemas que más se utilizan en los algoritmos genéticos.

3.4.5. Operador de Cruzamiento

Se utiliza un operador de cruzamiento en dos puntos. Para ello se seleccionan, mediante una función de probabilidad uniforme dos posiciones dentro de la lista de genes que representa la instancia de tráfico. De las cuales surgen tres secciones.

Posteriormente se generan dos nuevos individuos utilizando una combinación de estas secciones, como se observa en la Figura 3.9

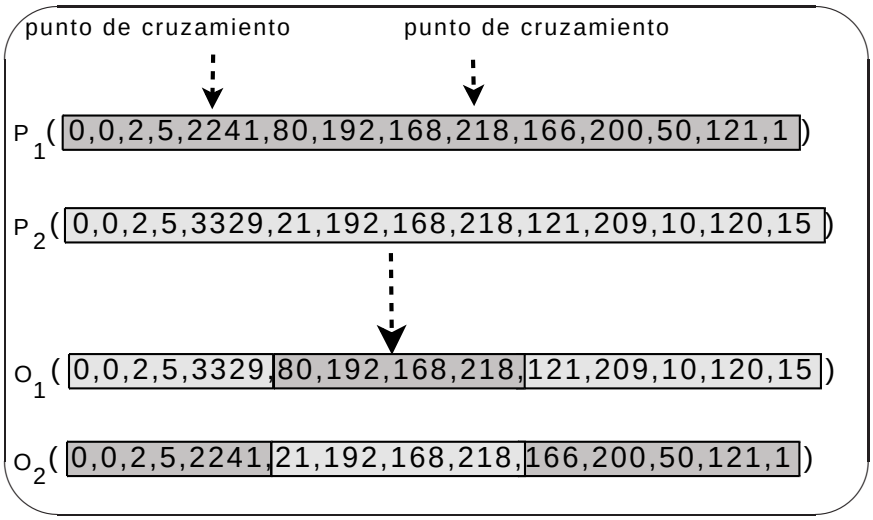

Figura 3.9: Cruzamiento en 2 puntos (DPX)

Como el operador no posee ningún tipo de conocimiento sobre la información representada en la estructura del individuo pueden darse casos en que los hijos resulten en individuos inválidos. Un ejemplo seria el de individuos con direcciones IP con valores mayores a 255. En este caso se espera que el algoritmo descarte a estos individuos al no observar ninguna mejora en el valor de fitness respecto a sus padres.

3.4.6. Crowding Determinístico

En este trabajo se utiliza la técnica conocida como crowding determinístico propuesta originalmente por Mahfoud [27] que introduce la competencia entre padres e hijos por la misma área del espacio de soluciones posibles.

Esta elección se basa en el hecho que crowding determinístico presenta buenos resultados sin aumentar la complejidad computacional del algoritmo como se menciona en Sareni et. al [42].

Las técnicas como sharing discutidas por Goldberg y otros [22, 26, 43], no resultan viables en este caso, ya que el proceso de búsqueda de los individuos de la población que se encuentran en el mismo nicho, resulta demasiado costoso en términos de tiempo computacional. A este inconveniente se agrega la necesidad de recalcular el fitness de todos los individuos en cada generación, lo que eleva en varios órdenes de magnitud el tiempo total de ejecución.

El esquema de crowding determinístico utilizado en este trabajo se debe a León et. al. [44] en donde luego de la operación de cruzamiento, cada hijo reemplaza a su padre más próximo, si y sólo si, posee un valor de fitness más alto.

El algoritmo 3 presenta el pseudocódigo de la implementación de crowding determinístico. Primero se determina cual es el padre más cercano de cada uno de los hijos. Con este objetivo se comparan en la línea 3 la suma de las distancias de cada padre con las sus diferentes hijos. Una vez determinado los padres e hijos más próximos, se calcula el valor de fitness de cada uno de ellos para agregarlos a la población si corresponde.

Como se menciona en los antecedentes la noción de proximidad está dada por una función de distancia. Mahfoud [27] observa que dicha función debe poseer la mayor cantidad de información posible sobre el dominio de aplicación del problema. Luego se propone la utilización de una función de distancia euclídea definida en la ecuación (3.4). La misma, al igual que la función de optimización, favorece a través de pesos a ciertos atributos seleccionados en base a la experiencia disciplinaria.

$$d = \sqrt{(p_i - p_j)^2 w^1 + (s_i - s_j)^2 w^2 + (s'_i - s'_j)^2 w^3 + (d_i - d_j)^2 w^4 + (d'_i - d'_j)^2 w^5} \tag{3.4}$$

donde p significa el puerto origen, mientras que s, s' y d, d' son los 2 bytes

Algoritmo 3 Crowding determinístico

1. P= Poblacion
2. o_1, o_2=Cruzamiento(p_1,p_2)
3. **if** $d(p_1, o_1) + d(p_2, o_2) \leq d(p_1, o_2) + d(p_2, o_1)$ **then**
4. **if** $f(p_1) < f(o_1)$ **then**
5. $P = P \cup o_1$
6. **end if**
7. **if** $f(p_2) < f(o_2)$ **then**
8. $P = P \cup o_2$
9. **end if**
10. **else**
11. **if** $f(p_1) < f(o_2)$ **then**
12. $P = P \cup o_2$
13. **end if**
14. **if** $f(p_2) < f(o_1)$ **then**
15. $P = P \cup o_1$
16. **end if**
17. **end if**

más significativos de las direcciones IP de origen y destino respectivamente. Los w^k indican los pesos asignados a los términos de la ecuación (3.4)

3.4.7. Operador de Descarte de Condición

En este trabajo se utiliza un operador de descarte de condición que selecciona mediante una función de probabilidad uniforme algún gen del individuo. El gen seleccionado se reemplaza por el valor -1. Como se menciona en la sección 3.4.2 los genes con un valor de -1 no se consideran durante las operaciones de evaluación.

El operador de *descarte de condición* cuenta con 2 parámetros. En primer lugar existe un factor ($dfactor$) que indica un porcentaje de individuos que se verán afectados por el mencionado operador.

El segundo parámetro del operador establece un límite en la cantidad de genes que pueden ser descartados. De esta manera se evita la aparición de individuos demasiado generales los cuales pueden encontrar coincidencias con muchas instancias de tráfico, pero en realidad proveen muy poca información.

3.4.8. Operador de Mutación

El operador de mutación utilizado selecciona a un individuo de la población de acuerdo a una función de probabilidad uniforme y efectúa una copia de dicho individuo. Posteriormente se selecciona un gen, el cual es trabajado como una cadena de bits. Sobre este gen se aplica el operador XOR sobre un bit elegido al azar.

Esta operación se efectúa siempre y cuando el gen seleccionado no haya sido previamente generalizado por el operador descarte de condición.

El operador mutación que se utiliza en este trabajo opera sobre una copia del individuo seleccionado, ya que a veces el operador de mutación puede empeorar el valor de fitness del individuo seleccionado y entonces no se lo incluye.

En el momento de decidir que hacer con el individuo obtenido de la mutación se pueden aplicar 2 políticas de reemplazo. En la primera se reemplaza al individuo con el valor de fitness más bajo de la población y en la segunda el reemplazo sólo se efectúa cuando el individuo que ha sufrido la mutación tiene un valor de fitness más alto que el peor individuo de la población.

3.4.9. Obtención de los mejores individuos

Debido a la aplicación de técnicas de crowding, al finalizar el proceso evolutivo del algoritmo se han formado dentro de la población grupos o clústeres de individuos que presentan soluciones muy similares. Con el objeto de obtener los mejores representantes de cada uno de estos grupos, se utiliza una heurística cuyo pseudocódigo se presenta en el algoritmo 4.

El individuo p con mayor fitness se extrae de la población P. Luego se verifica que no exista previamente en el conjunto de reglas R un individuo r con genes de idéntico valor a p. Con este fin se compara a p con cada uno de los individuos r del conjunto de reglas R. Durante esta comparación solamente se utilizan los genes que no hayan sido modificados por el operador

Algoritmo 4 Heurística para la selección de mejores individuos

```
P=Población
R=conjunto de reglas
N=Número máximo de reglas
for p ∈ P hasta N do
  if p ≠ algún r ∈ R then
    R = R ∪ p
  else
    R = R∪ IndividuoMasGeneralizado(r,p)
  end if
end for
```

de descarte de condición. Si ambos individuos presentan coincidencias en sus genes, se prefiere a aquel que presente el mayor número de genes generalizados y se procede a descartar al otro. Este procedimiento se repite hasta que el conjunto de reglas R esté integrado por un número máximo de N reglas.

El algoritmo expuesto puede resumirse de la siguiente manera: de un grupo de individuos que presenten similitudes en sus genes, se prefiere a aquel que tenga el mayor número de genes generalizados.

3.5. Módulo de Evaluación

El módulo de evaluación se utiliza durante la segunda etapa de acuerdo a lo presentado previamente en la Figura 3.1. En este módulo se utilizan las reglas obtenidas por el módulo de aprendizaje para evaluar un conjunto de nuevas instancias de tráfico. Para determinar cuando una nueva instancia de tráfico constituye una anomalía se utiliza una heurística descripta en el pseudocódigo del algoritmo 5.

El conjunto T contiene las instancias de tráfico a ser evaluadas y el conjunto R contiene las reglas obtenidas por el módulo de aprendizaje. Para que una instancia de tráfico t que pertenece al conjunto T sea considerada una anomalía es condición suficiente que ésta no presente coincidencias con ninguna regla de clasificación $r \in R$.

Algoritmo 5 Heurística utilizada para la detección de anomalías.

```
1. T=Instancias de trafico
2. R=Conjunto de Reglas
3. for t ∈ T hasta MaxT do
4.    for r ∈ R hasta MaxR do
5.       if t = r  then
6.          No es una anomalia
7.          break
8.       end if
9.    end for
10. end for
```

La heurística descripta no presenta gran complejidad. Su sencillez responde a que el módulo de clasificación no constituye el foco de este trabajo. Existe una gran variedad en las heurísticas para clasificar anomalías a partir del conjunto de reglas de aprendidas. Las cuales posiblemente permitan mejorar el comportamiento del módulo de clasificación. El estudio del comportamiento del sistema propuesto utilizando alguna de estas variantes se deja para trabajos futuros.

3.6. Conclusiones

En este capítulo se ha presentado una arquitectura para un sistema de detección de intrusos. Los diferentes módulos que componen la arquitectura propuesta, contienen los elementos mínimos y necesarios para la aplicación de un sistema de detección de intrusos basado en algoritmos genéticos.

El módulo de aprendizaje, núcleo principal de esta tesis, se basa en un algoritmo genético cuyo objetivo es obtener un conjunto de reglas que representen las instancias normales de tráfico.

El algoritmo genético propuesto es de tipo estacionario. Los resultados preliminares al ejecutar un algoritmo genético generacional en una máquina secuencial mostraron una alta demanda de los recursos computacionales

por lo que no se lo consideró para este trabajo. Sin embargo no se descarta un estudio de mayor profundidad en futuros trabajos.

De la revisión del estado estado del arte, como se ha indicado, surge que existen pocas propuestas de clasificación de instancias de tráfico normal, una de las características más destacable del presente trabajo.

El trabajo parte de algunas propuestas del estado del arte que se adecuan convenientemente a la detección de tráfico normal. Para la selección de atributos se sigue a Gong [14] y se utiliza, con algunas variantes, el esquema de representación de los individuos de la población, los cuales se escogen en base a criterios disciplinares.

Para la función de fitness se sigue la propuesta de Li [2], pero en este caso se utilizan pesos para favorecer algunos atributos que por experiencia disciplinar presentan mayor relevancia para la detección de intrusos. Sin embargo debido a que en el presente trabajo se clasifican instancias de tráfico normal, el orden de los atributos propuesto por Li no se aplica de la misma manera al algoritmo propuesto.

Se profundiza en las alternativas para la implementación de las técnicas de nicho, las cuales no se discuten en detalle en los trabajos revisados. Luego se propone una variante de crowding determinístico basado en una función de distancia euclídea, la cual considera los atributos más relevantes de acuerdo a experiencia disciplinar.

En síntesis el módulo propone un algoritmo genético para clasificar instancias de trafico y presenta diversas innovaciones discutidas a partir del estado del arte, cuyos resultados se presentan en el capítulo 5.

En un futuro la arquitectura propuesta permitiría experimentar con variantes en cada uno de los módulos. Entre otras posibilidades se destacan la utilización de módulos de captura distribuidos en diferentes puntos de la red, la utilización de diferentes algoritmos (Naive Bayes, redes neuronales) en el módulo de aprendizaje y la experimentación con distintas heurísticas en el módulo de evaluación y reconocimiento de anomalías.

Se observa que la función de fitness que se utiliza es costosa en términos de recursos computacionales, por lo que surge la posibilidad de investigar el comportamiento del algoritmo propuesto en ambientes distribuidos.

4 Implementación Computacional

En este capítulo se discute la implementación computacional del algoritmo genético propuesto en el capítulo 3. Es importante discutir con detalle la implementación del algoritmo y el código resultante, ya que la ejecución de algoritmos genéticos demanda una gran cantidad de recursos computacionales.

En la sección 4.1 se discute la implementación secuencial del algoritmo genético. Luego en la sección 4.2 se realiza un análisis de concurrencia con el fin de discutir la paralelización del código. Finalmente en la sección 4.3, se trata la adaptación del algoritmo para su ejecución en ambientes distribuidos.

4.1. Implementación Secuencial.

Para la implementación computacional del algoritmo genético se desarrolla un entorno de trabajo basado en el lenguaje Python [45], la elección responde a que Python es un lenguaje con características como tipos de datos dinámicos, introspección, interoperabilidad con otros lenguajes, etc., las cuales permiten el desarrollo de prototipos funcionales en un corto plazo.

En Python el código fuente se traduce a una presentación intermedia conocida como *bytecode*. Resulta importante destacar que no se trata de código para una arquitectura en particular, sino que el mismo se puede trasladar

a varias arquitecturas. Con este fin se utiliza una máquina virtual que interpreta el *bytecode* y se encarga de su ejecución. La máquina virtual se debe implementar en cada una de las arquitecturas donde se requiera ejecutar la aplicación.

Una ventaja importante del lenguaje Python es que la máquina virtual del lenguaje ha sido implementada en numerosos sistemas operativos, con lo que se asegura su portabilidad.

La implementación del entorno de trabajo hace uso de las características orientadas a objetos de Python. En la Figura 4.1 se presenta un diagrama simplificado de las clases utilizadas en el entorno de trabajo desarrollado. Las principales clases del entorno de trabajo son:

Cromosome: contiene información sobre la forma de codificación de los individuos y los genes que lo componen.

Population: contiene información sobre los individuos que componen la población del algoritmo genético. La clase posee entre sus atributos una lista que contiene un número determinado de individuos de acuerdo a las reglas de codificación impuestas por la clase *Crosomome.*

Ga: Es la clase donde se implementa el algoritmo genético. Se utiliza una población de acuerdo a las especificaciones de la clase *Population.* Además la clase se relaciona con diferentes clases que implementan los distintos operadores genéticos utilizados en al algoritmo discutido en el capítulo anterior.

Cada una de las clases citadas se ha implementado en un archivo diferente, característica que en Python se conoce como *módulo*, aspecto que se debe tener en consideración para su posterior distribución.

La elección de Python como lenguaje de codificación trae considerables ventajas. Sin embargo al ser un lenguaje que se ejecuta sobre una máquina virtual, los tiempos finales de ejecución son superiores a los que se obtienen con lenguajes con código compilado para la arquitectura nativa.

Como una solución parcial al problema de la pérdida de performance de la máquina virtual de Python se emplea un compilador en tiempo de ejecución (JIT), que permite obtener una mejora en los tiempos de ejecución del

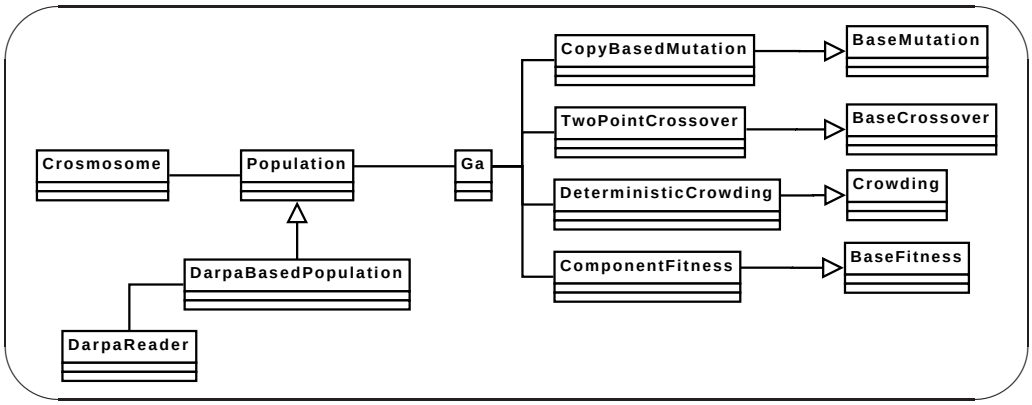

Figura 4.1: Diagrama simplificado de las clases que se utilizan para el entorno de trabajo basado en el lenguaje Python

bytecode. En el momento en que se ejecuta el bytecode, el compilador JIT traduce algunas o todas la porciones a código máquina nativo con el fin de obtener una mejor performance.

En este trabajo se utiliza el compilador JIT para Python, *psyco* [46] desarrollado por Rigo, el cual en muchos casos es capaz de acelerar los tiempos de ejecución. En el trabajo de Rigo se sostiene que en la mayoría de los casos basta con sólo incluir el módulo de psyco en el código de la aplicación para lograr mejoras en el tiempo total de ejecución sin realizar ninguna modificación sustancial al código fuente.

En el Cuadro 4.1 se compara el tiempo total de ejecución de la implementación, para un población de 100 individuos, cuando se emplea la máquina virtual de Python estándar y cuando se utiliza el compilador en tiempo de ejecución *psyco,* para 100, 200 y 500 generaciones.

Cuadro 4.1: Tiempos totales de ejecución cuando se incluye psyco y cuando se utiliza la maquina virtual de Python estándar para 100, 200 y 500 generaciones

	G=100	G=200	G=500
Psyco	2.5 min	4.3 min	9.2 min
Python	4.3 min	7.75 min	15.7 min

Como surge del Cuadro 4.1 la utilización de psyco permite disminuir el tiempo total de ejecución a valores cercanos a la mitad.

4.2. Análisis de Concurrencia

Un algoritmo genético clásico como el originalmente propuesto por Holland [24] posee diferentes etapas, las cuales se indican en el pseudocódigo del algoritmo 6. Entre las mismas se distinguen: selección, cruzamiento, mutación y finalmente el cálculo de la función de fitness de los nuevos individuos. Este procedimiento se repite a lo largo de un número determinado de generaciones.

Algoritmo 6 Algoritmo genético canónico

```
1: t=0
2: inicializar(P_t)
3: calcularFitness(P_t)
4: while ! condicion de salida satisfecha do
5:    S_t = seleccion(P_t)
6:    P'_t = operadorCrossover(S_t)
7:    operadorMutacion(P'_t)
8:    calcularFitness(P'_t)
9:    P_{t+1} = P'_t
10:   t = t + 1
11: end while
```

Una estrategia que se utiliza con el fin de disminuir el tiempo total de ejecución consiste en procesar alguna de las etapas del algoritmo genético de manera concurrente. Con este fin, mediante un análisis de concurrencia, se debe determinar cuales pueden ejecutarse en paralelo.

Con el objeto de obtener indicadores de los beneficios que pueden obtenerse cuando se ejecutan aplicaciones en paralelo se utilizan algunas métricas como *speed up,* que hace referencia al cociente del tiempo total de ejecución de un algoritmo secuencial y el tiempo total de ejecución de un algoritmo paralelo como se muestra en la ecuación (4.1).

$$S = \frac{T_s}{T_p} \tag{4.1}$$

donde T_s hace referencia al tiempo total de ejecución secuencial y T_p es el tiempo total de ejecución del algoritmo paralelo.

Otra métrica que se utiliza es la eficiencia, la cual se mide como la aceleración o *speed up* que se obtiene en relación al número de procesadores sobre los cuales se ejecuta el algoritmo paralelo, como se observa en la ecuación (4.2)

$$E = \frac{S}{p} \tag{4.2}$$

En las siguientes secciones se analiza cuales son las etapas que se pueden ejecutar concurrentemente. En la sección 4.2.1 se analiza primero las posibilidades de concurrencia de un algoritmo genético clásico y el análisis para el algoritmo genético propuesto se trata en la sección 4.2.2.

4.2.1. Algoritmo genético clásico

En general el cálculo del fitness es la etapa más costosa de un algoritmo genético. Luego conviene establecer una métrica para estimar el tiempo total de ejecución secuencial (T_{s1}) que considere el tiempo t_f de cálculo del valor de fitness de cada individuo de P a lo largo del número de generaciones (G) del algoritmo. La fórmula para calcular dicha métrica se presenta en la ecuación (4.3)

$$T_{s1} = P\, t_f\, G \tag{4.3}$$

Donde el tiempo total de ejecución se calcula como el producto del tiempo de cálculo de la función de fitness t_f de cada uno de los individuos de la población P a lo largo de las G generaciones.

Es importante destacar que los algoritmos genéticos cuentan con muchas operaciones no determinísticas, como la conformación de la población inicial y la aplicación de los diferentes operadores genéticos. Luego, resulta necesaria una validación del algoritmo repitiendo la ejecución del mismo un número suficiente de veces, con el fin de comprobar si se obtienen resultados similares.

La ecuación (4.3) se ajusta mediante un factor v que mide la cantidad de validaciones, por lo que el tiempo secuencial resulta:

$$T_s = T_{s1}\, v = P\, t_f\, G\, v \tag{4.4}$$

En las siguientes secciones se discute cuales de los factores de la ecuación (4.4) se pueden ejecutar de forma concurrente.

4.2.1.1. Paralelización de las generaciones

Como surge del pseudocódigo expuesto en el algoritmo 6 para una generación g existe una dependencia con la generación $g-1$. Luego el cálculo de las distintas generaciones no resulta ser un buen candidato para ser ejecutado en forma paralela.

4.2.1.2. Paralelización de la función de fitness

El cálculo de la función de fitness se puede realizar de manera independiente para cada uno de los individuos que compone la población y consecuentemente, es un buen candidato para realizar su paralelización.

Esta técnica conocida como paralelización global distribuye la carga de trabajo entre los distintos procesadores disponibles con el fin de disminuir el tiempo de ejecución. En la paralelización de esta etapa en general se puede seguir un esquema maestro esclavo como el que se muestra en la Figura 4.2 [47].

Cabe mencionar que este enfoque no altera el número de evaluaciones que el algoritmo efectúa para alcanzar la condición de salida respecto del procesamiento secuencial. Existen otras técnicas de paralelización que se denominadan algoritmos genéticos distribuidos, en las cuales se modifica el algoritmo genético secuencial con el fin de disminuir el número de evaluaciones. Un detalle más completo de estas técnicas puede verse en Cantú Paz [48] y Alba [49].

Para la paralelización del algoritmo genético bajo un esquema maestro esclavo debe distribuirse la población P y el conjunto de entrenamiento D entre los procesadores disponibles. Esta distribución debe garantizar un buen balance de carga y un tiempo de comunicación aceptable.

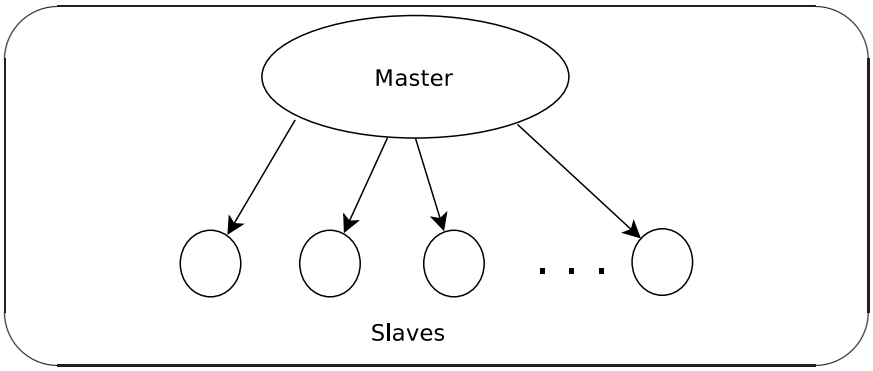

Figura 4.2: Paralelización global con arquitectura maestro esclavo

En general conviene particionar y distribuir entre los nodos esclavos al conjunto de mayor tamaño ya que de este modo se garantiza la ocupación de los procesadores disponibles. Sin embargo dado que en este caso ambos conjuntos, P y D poseen un tamaño mayor que el número de procesadores disponibles y considerando que D permanece fijo y P varía en cada generación, entonces una solución aceptable es copiar D en todos los nodos esclavos y enviar a los mismos subconjuntos de P.

En el algoritmo 7 se muestra el pseudocódigo de la paralelización de la función de fitness cuando se emplea un esquema maestro esclavo. Como se observa en el mismo, el nodo maestro (por única vez), envía a cada uno de los procesadores esclavos el archivo conteniendo el conjunto de entrenamiento D, en la línea 5. La población P se la divide en grupos, de acuerdo al número N de procesadores disponibles. Posteriormente estos grupos se envían a los procesadores esclavos en la línea 6. A su vez en la línea 9 cada procesador esclavo calcula el valor de la función de fitness para cada uno de los individuos de su correspondiente población y devuelve los resultados al nodo maestro, el cual recupera los resultados en la línea 7.

En la Figura 4.3 se puede observar la distribución de la población entre los distintos procesadores, los cuales ya han recibido el conjunto de entrena-

Algoritmo 7 Paralelización de la función de fitness

```
1: N= Numero de procesadores disponibles
2: P= Poblacion
3: D= Conjunto de entrenamiento
4: if Maestro then
5:    Enviar(D)
6:    Enviar(P/N)
7:    RecuperarFitness()
8: else
9:    fitness=Calcularfitness(P,N)
10:   Devolver(fitness)
11: end if
```

miento D, necesario para calcular el valor de fitness de cada individuo. Es importante destacar que en este tipo de algoritmos genéticos es necesario calcular el valor de fitness de todos los individuos de población en cada iteración del mismo. En la Figura 4.3 se indican con color gris oscuro los individuos que necesitan ser recalculados.

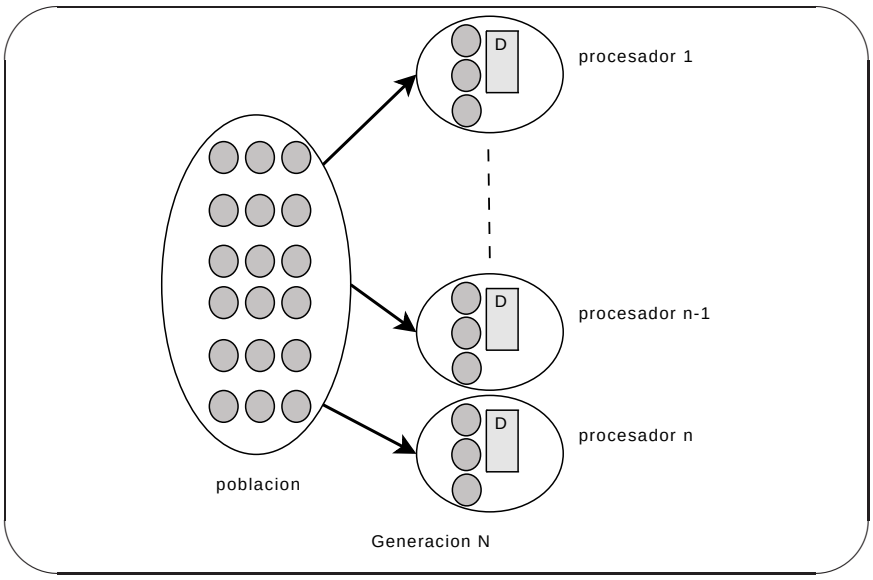

Figura 4.3: Distribución de la población en un algoritmo genético generacional. Cada procesador calcula el valor de fitness del conjunto de individuos que le corresponde.

Se debe tener en cuenta que implementar esta estrategia de paralelización obliga a modificar el código del algoritmo genético, con el objeto de realizar las transferencias de los individuos a cada uno de los procesado-

res disponibles y su posterior recuperación. En la sección 4.3.1 se discuten las modificaciones necesarias para la ejecución en ambientes fuertemente acoplados mediante la biblioteca MPI [50].

Se puede estimar el tiempo total de ejecución en paralelo para los algoritmos genéticos de tipo generacionales mediante la fórmula definida en la ecuación (4.5)

$$T_p = ((Pt_f)/N)\, G\, k\, v \tag{4.5}$$

Donde el tiempo total de ejecución en un ambiente paralelo T_p es igual al tiempo necesario para evaluar a todos los individuos de la población P sobre la cantidad de procesadores disponibles N.

La relación entre los tiempos de ejecución secuencial y paralelo se mide mediante el *speed up* correspondiente, que se muestra en la ecuación 4.6 a partir de los tiempos T_p y T_s definidos en las ecuaciones (4.5) y (4.4) respectivamente.

$$S = \frac{T_s}{T_p} = \frac{P\, t_f\, G\, v}{((P\, t_f/N)\, G\, V} = N \tag{4.6}$$

La eficiencia para este caso resulta:

$$E \;=\; \frac{S}{N} = 1 \tag{4.7}$$

De las dos ecuaciones presentadas, se desprende que cuando se dispone de un número de procesadores igual a P, la disminución del tiempo será a un valor cercano al tiempo necesario t_f para calcular el fitness de un único individuo.

El comportamiento descripto se puede observar en en la Figura 4.4, donde se presentan gráficamente el *speed up* y la eficiencia estimados para una población P compuesta por 10 individuos. En dicha figura se representa

en ordenadas el *speed up* S y la eficiencia E, en función del número de procesadores N que se representa en el eje de abscisas.

Se observa una eficiencia cuasi-lineal hasta alcanzar un número de procesadores igual al tamaño de P.

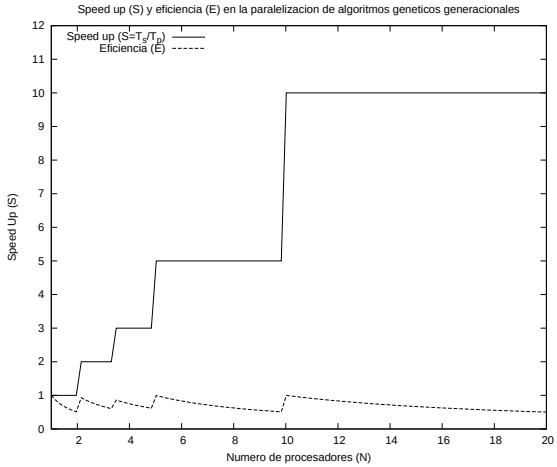

Figura 4.4: Comportamiento estimado del Speed up y eficiencia de un algoritmo genético generacional

Las ecuaciones para el cálculo de los tiempos de paralelización definidas en el presente trabajo no tienen en cuenta el tiempo de comunicación. Para una discusión detallada de las posibilidades que ofrece la paralelización global aplicada a algoritmos genéticos de tipo generacional puede verse el trabajo de Cantú Paz [47].

4.2.1.3. Paralelización de las pruebas validatorias

De la ecuación (4.4) surge que no existe dependencia entre las diferentes ejecuciones del algoritmo genético necesarias para su validación, con lo cual las mismas pueden procesarse concurrentemente.

El tiempo total de ejecución cuando se paralelizan las pruebas validatorias se puede estimar de acuerdo a la ecuación (4.8)

$$T_{pv} = P\, t_f\, G\,(\frac{v}{N}) \tag{4.8}$$

Alumno: Carlos A. Catania

Mientras que el speed up que se puede alcanzar es de:

$$S_v = \frac{T_s}{T_{pv}} = N \tag{4.9}$$

En la Figura 4.5 se puede observar el comportamiento del speed up sobre el eje de ordenadas a medida que se incremente en numero de N procesadores en donde se ejecutan las $v = 40$ pruebas validatorias del algoritmo.

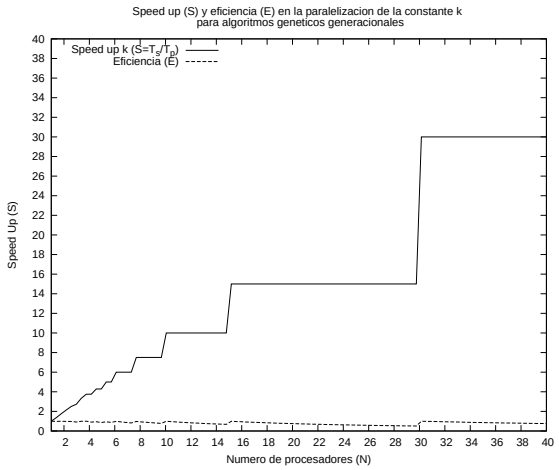

Figura 4.5: Comportamiento estimado del Speed up y eficiencia de la constante v=40 para los algoritmos genéticos generacionales

Si se cuenta con un número de N procesadores igual a v, la paralelización de las validaciones del algoritmo permitiría disminuir T_s al valor del tiempo total de ejecución T_{s1}. Con esta estrategia de paralelización se logra una eficiencia cuasi-lineal hasta alcanzar un número de procesadores igual a v, sin necesidad de realizar ninguna modificación al código fuente del algoritmo.

En este caso los procesos concurrente pueden ejecutarse sobre entornos fuerte o débilmente acoplados ya que la falla de los nodos posee un bajo impacto en el resultado final. El proceso que falla puede ejecutarse nuevamente o incluso descartarse de las validaciones.

4.2.2. Algoritmo genético propuesto

El algoritmo genético propuesto en el capítulo anterior presenta diferencias importantes con respecto a un algoritmo genético clásico. Luego se debe repetir el análisis de concurrencia y se deben proponer métricas para este nuevo algoritmo, cuyo pseudocódigo se muestra en el algoritmo 8.

Algoritmo 8 Algoritmo genético propuesto

```
1.  initializarPoblacion(P)
2.  calcularFitness(P)
3.  while NroMaxGeneracionNoalcanzado do
4.     for i = 0 to xfactor do
5.        p1 = seleccionar(P)
6.        p2 = seleccionar(P)
7.        (o1, o2) = cruzamiento(p1, p2)
8.        crowding(p1, p2, o1, o2)
9.     end for
10.    for i = 0 to dfactor do
11.       p = seleccionar(P)
12.       descarte(p)
13.    end for
14.    for i = 0 to mfactor do
15.       p = seleccionar(P)
16.       mutacion(p)
17.    end for
18.    calcFitness(P)
19. end while
20. selecionarIndividos(P)
```

El algoritmo genético propuesto no responde al tipo generacional, sino que se trata de un algoritmo de tipo estacionario (steady-state), como se mencionó en la sección 3.4.1. Por otra parte emplea técnicas de nicho como crowding y crowding determinístico, las cuales no se aplican en el algoritmo genético clásico.

A partir de las características mencionadas en el párrafo anterior, el tiempo total de ejecución secuencial del algoritmo propuesto se expresa mediante la ecuación (4.10).

$$T_s \;=\; \{P\,t_f + [t_f\,(k_m + k_c)\,(G-1)]\}\,v \tag{4.10}$$

En este caso el tiempo T_s se obtiene como la suma de los términos $P\,t_f$ (para la primera generación) y $t_f(k_m+k_c)(G-1)$ para las generaciones restantes. Donde k_c hace referencia a los individuos recientemente obtenidos a partir del operador de cruzamiento y k_m indica a los nuevos individuos que se modifican debido al operador mutación y descarte. El valor en la práctica de $(k_m + k_c)$ es cuatro.

Se observa que para los valores utilizados en la práctica $P << (k_m+k_c)(G-1)$, luego T_s puede aproximarse mediante la ecuación (4.11).

$$T_s \approx [t_f\,(k_m + k_c)\,(G-1)]\,v \tag{4.11}$$

Finalmente dado que G es del orden de 10^3 T_s puede escribirse como:

$$T_s \approx t_f\,(k_m + k_c)\,G\,v \tag{4.12}$$

De la comparación de las ecuaciones (4.4) y (4.12) surge que el algoritmo propuesto es menos costoso que el algoritmo clásico para un mismo número de generaciones. Parece importante realizar experiencias de ambos algoritmos en el contexto de IDS.

4.2.2.1. Paralelización de las Generaciones y pruebas validatorias

De la ecuación (4.12) se desprende que la paralelización de las generaciones y pruebas validatorias presentan un comportamiento similar a los ya mencionados en la sección 4.2.1.1 y en la sección 4.2.1.3 respectivamente.

Al igual que el caso del algoritmo genético clásico no resulta posible ejecutar concurrentemente las diferentes generaciones debido a que existe una dependencia de g respecto a $(g-1)$.

Para el caso de la paralelización de las pruebas validatorias, las mismas pueden ejecutarse de manera concurrente sin efectuar mayores modificaciones al código fuente de la aplicación según el mismo esquema que para el algoritmo genético clásico.

4.2.2.2. Paralelización de la función de fitness.

Las diferencias entre el algoritmo genético clásico y el propuesto impone la necesidad de adecuar la paralelización del fitness. En este aspecto la paralelización global del algoritmo genético propuesto, solo presenta una mejora significativa en el momento de calcular el valor de fitness de la población inicial (línea número 2 del pseudocódigo presentado en el algoritmo 8).

En consecuencia parece importante discutir como se particiona y distribuyen los conjuntos D y P de acuerdo con las consideraciones realizadas en la sección 4.2.1.2. Con este fin se discuten las ventajas e inconvenientes de distribuir P y D, respectivamente.

Una primera opción consiste en distribuir P en los nodos esclavos y se envía a cada uno de los mismos un archivo que contiene el conjunto de entrenamiento D, con el fin de calcular el valor de fitness en los procesadores esclavos. Hasta este momento el algoritmo se comporta igual a un algoritmo generacional como se observa en la Figura 4.6 (a). Sin embargo esto no ocurre en las sucesivas iteraciones del algoritmo en donde sólo es necesario recalcular el valor de la función de fitness (línea número 18 del pseudocódigo presentado en el algoritmo 8) de dos individuos como máximo (aquellos que se obtienen como resultado de los operadores de mutación y descarte de condición). Los individuos restantes conservan su valor de fitness por lo que no es necesario recalcularlos.

La Figura 4.6 (b) representa gráficamente la idea que muestra la distribución de la población entre los N procesadores. Se indica con color gris oscuro a los individuos que necesitan ser recalculados. Además se presentan también en color gris claro los individuos que no han variado, dado que su valor de fitness ya ha sido calculado en las anteriores iteraciones del algoritmo.

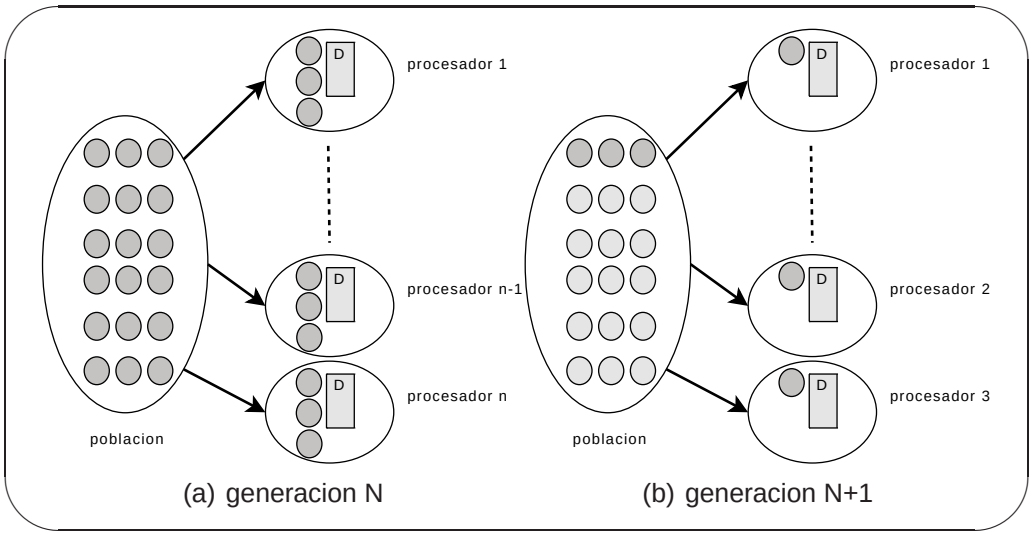

Figura 4.6: Distribución de la población en un algoritmo genético steady-state. Cada procesador evalúa el conjunto de individuos correspondiente. Lo que resulta ineficiente.

Por otra parte la utilización de crowding determinístico (línea número 8 del pseudocódigo presentado en el algoritmo 8) obliga a calcular el valor de la función de fitness de los dos individuos recientemente obtenidos mediante el operador de cruzamiento, con el fin de establecer si su valor de fitness resulta suficiente para ingresar o no a la población. El tiempo de cálculo del valor de fitness de los dos nuevos individuos recién obtenidos constituye una parte significativa del tiempo total de ejecución del algoritmo propuesto.

El tiempo total de ejecución en paralelo cuando se distribuye la Población P se puede estimar de acuerdo a la fórmula definida en la ecuación (4.13)

$$T_p = [\frac{(Pt_f)}{N} + (\frac{t_f k_m}{N} + \frac{t_f k_c}{N})(G-1)]v \tag{4.13}$$

El tiempo T_p se obtiene como la suma de los términos $\frac{Pt_f}{N}$ (para la primera generación) y $(\frac{t_f k_m}{N} + \frac{t_f k_c}{N})(G-1)$ para las generaciones restantes. Donde k_c hace referencia a los individuos recientemente obtenidos a partir del operador de cruzamiento y k_m indica a los nuevos individuos que se modifican debido al operador mutación y descarte.

Se observa que para los valores utilizados en la práctica $P << (k_m+k_c)(G-1)$, luego T_p puede aproximarse mediante la ecuación (4.11).

$$T_p \approx [(\frac{t_f k_m}{N} + \frac{t_f k_c}{N})(G-1)]\, v$$

Finalmente dado que G es del orden de 10^3 T_p puede escribirse como:

$$T_p \approx [(\frac{t_f k_m}{N} + \frac{t_f k_c}{N})\, G]\, v$$

El speed up que se alcanza cuando se utiliza esta estrategia se define según la ecuación (4.14)

$$S \;=\; \frac{T_s}{T_p} = \frac{(k_m + k_c)\, G}{(\frac{k_m}{N} + \frac{k_c}{N})\, G} = N \,/\, N \;<\; 2 \qquad (4.14)$$

Sin embargo debe destacarse que al tener k_m y k_c un valor constante y no poder ser calculados concurrentemente sólo se obtiene un speed-up igual al número de procesadores N, mientras *N* sea menor a 2, valor bajo que desaconseja el empleo de este esquema en la práctica, excepto para equipos con doble núcleo o similares.

Bajo este esquema se desprende que no se puede obtener una eficiencia cuasi-lineal como sucede con un algoritmo genético generacional, como se muestra gráficamente en la Figura 4.7 donde se presentan los resultados obtenidos para el speed-up y la eficiencia del algoritmo genético propuesto. Se observa que para una población P de 10 individuos, sólo se obtiene speed up lineal mientras no se alcance un número de procesadores igual a cuatro.

En el trabajo de Rasheed [51] se discute la posibilidad de lograr una aceleración en los tiempos totales de ejecución de algoritmos genéticos paralelos de tipo steady-state. Se propone una variante en el algoritmo que permite la selección de múltiples pares de individuos de la población y efectuando las

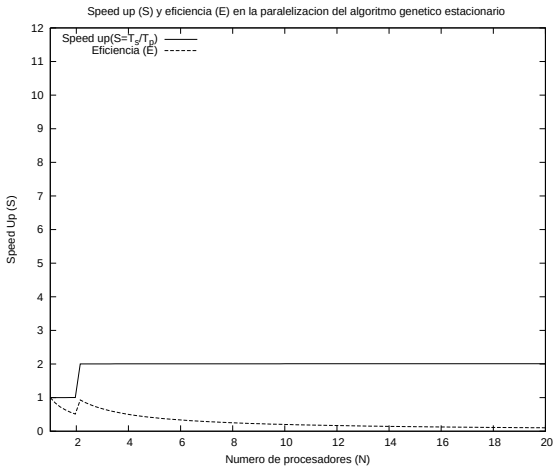

Figura 4.7: Comportamiento estimado del Speed up y eficiencia de la constante k=30 para el algoritmo genético propuesto

operaciones de cruzamiento y evaluación de cada procesador de manera asincrónica. Sin embargo el algoritmo genético presentando por Rasheed intenta solucionar un problema de optimización y no de aprendizaje de reglas, como el caso del presente trabajo. Consecuentemente el problema de la utilización de técnicas de nicho que necesiten calcular el valor de fitness no se discute de manera expresa. Además la paralelización propuesta en el trabajo de Rasheed constituye una modificación significativa en el esquema del algoritmo, lo que provoca que el mismo no obtenga exactamente los mismos resultados. Por estas razones en este trabajo esta variante no se tiene en cuenta.

A raíz de la limitación del *speed up* y la eficiencia es necesario estudiar las ventajas que pueden obtenerse de distribuir el conjunto de instancias de tráfico D.

Con este fin se divide el conjunto D y se envían cada uno de los subconjuntos a los procesadores esclavos. De esta manera cada procesador calcula de acuerdo a la ecuación (3.1) el número de coincidencias que el individuo r posee con el subconjunto de D correspondiente y al finalizar envía el resultado al nodo maestro. Este a su vez realiza la sumatoria de los valores enviados por cada uno de los procesadores esclavos para obtener el valor final de la función de fitness del individuo r.

En la Figura 4.8 se presentan gráficamente los dos esquemas de distribución discutidos anteriormente. La Figura 4.8 (a) muestra la distribución de los individuos *r, s, t* cuyo fitness se debe calcular. Cada uno de estos individuos utiliza un procesador que contiene la totalidad del archivo de entrenamiento D.

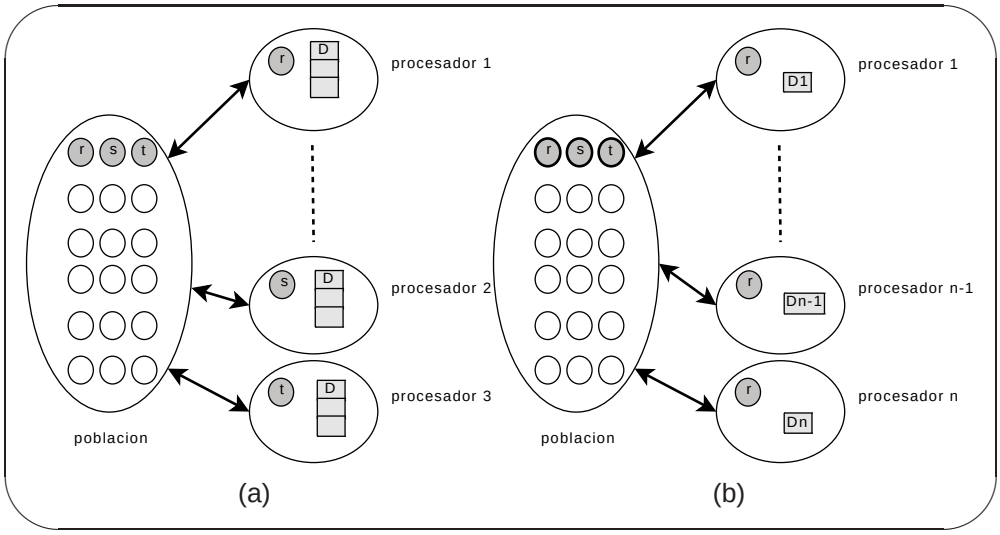

Figura 4.8: Diferencias en los esquemas de paralelización

En la Figura 4.8 (b) se muestra el esquema en que se distribuye *D*. Aquí el mismo individuo r es enviado a todos los procesadores esclavos disponibles, los cuales a su vez contienen un sección del archivo de entrenamiento D_i. De esta manera cada procesador calcula las coincidencias que presenta individuo con respecto a su sección del conjunto de entrenamiento y una vez finalizado devuelve el resultado al nodo maestro. Luego se repite dicha tarea para los individuos *r* y *t.* Es posible en la práctica calcular el fitness *r, s y t* conjuntamente.

El tiempo total de ejecución del algoritmo propuesto al distribuir las reglas *D*, se puede definir mediante la fórmula expresada en la ecuación (4.15)

$$\begin{aligned} T_p &= [P\frac{t_f}{N} + \frac{t_f}{N}(k_m + k_c)(\,G-1)]\,v \\ T_p &= \frac{t_f}{N}[P + [(k_m + k_c)\,(G-1)]]\,v \end{aligned} \tag{4.15}$$

donde $\frac{t_f}{N}$ indica que en este caso el fitness de un individuo se calcula en un

Alumno: Carlos A. Catania

conjunto de $\frac{D}{N}$ instancias de tráfico.

Como $P << (k_m + k_c)\,(G-1)$ y dado que G es del orden de 10^3 , la métrica se puede aproximar como:

$$T_p \approx \frac{t_f}{N}(k_m + k_c)\,G\,v \tag{4.16}$$

En la ecuación (4.17) se define el speed-up que se puede obtener cuando se paraleliza la función de fitness.

$$\begin{aligned} S &= \frac{T_s}{T_p} \\ S &= \frac{t_f(k_m + k_c)\,G\,v}{\frac{t_f}{N}(k_m + k_c)\,G\,v} = N \end{aligned} \tag{4.17}$$

En este caso el ahorro de tiempo obtenido se debe a la distribución de las D reglas en los N procesadores.

4.3. Procesamiento en ambientes distribuidos

Si bien existen una amplia variedad de arquitecturas de sistemas distribuidos, en este trabajo se estudian las posibilidades que brindan las arquitecturas de memoria distribuida fundamentalmente cluster de tipo Beowulf [52] y middlewares como Globus Toolkit [53] o Condor [54].

El término cluster se utiliza para representar a un grupo de computadoras independientes combinadas bajo un sistema unificado a través de software y una infraestructura de red. Los clusters se utilizan para alta disponibilidad (HA) o para computación de alta performance (HPC) con el objeto de proveer mayor poder computacional del que una única computadora puede proveer.

Los cluster de tipo Beowulf son cluster de alta performance basados en hardware de bajo costo y fácilmente accesible. Entre sus ventajas está la posibilidad de mejorar la performance del cluster simplemente aumentando la cantidad de computadoras que participan del mismo. Este tipo de cluster está basado en software de código abierto y libre como Linux.

Normalmente se trata de un conjunto de máquinas dedicadas, unidas por una red de alta velocidad como ser Gigabit Ethernet. En los últimos años han surgido algunas tecnologías de red de baja latencia como ser Myrinet [55] o Infiband [56] para mejorar el rendimiento en este tipo de cluster.

En general se puede considerar a un cluster de tipo Beowulf como un sistema fuertemente acoplado, con una baja latencia en las comunicaciones y una baja tolerancia a fallas. Su mayor atractivo reside en que se puede obtener un alto poder de cálculo con elementos de uso común.

Condor es un sistema distribuido desarrollado por la universidad de Wisconsin-Madison, diseñado para Computación de Alta Disponibilidad (High Throughput Computing HTC) y empleo de recursos ociosos [54]. Este permite de manera transparente y simultánea explotar las capacidades de estaciones de trabajo que pueden estar distribuidas en el mundo y pertenecer a distintos individuos, grupos, departamentos e instituciones. La idea principal de Condor es hacer posible que cálculos con alto costo de gestión puedan ser realizados de forma ágil y eficiente.

Globus Toolkit es una colección de componentes software que ofrecen la infraestructura básica necesaria para la creación y ejecución de aplicaciones distribuidas, así como para la construcción de Grid.

El término Grid se refiere a una infraestructura que permite la integración y el uso colectivo de ordenadores de alto rendimiento, redes y bases de datos que son propiedad y están administrados por diferentes instituciones. Puesto que la colaboración entre instituciones envuelve un intercambio de datos y recursos computación, el propósito del grid es facilitar la integración de recursos computacionales. Se lo puede considerar como una nueva forma de computación distribuida, en la cual los recursos pueden ser heterogéneos (diferentes arquitecturas, supercomputadores, clusters) y se conectan mediante redes de área extensa, por ejemplo Internet.

Actualmente, Globus se ha convertido en el estándar de facto para la Grid. Globus consta de tres componentes fundamentales: gestión de recursos, servicio de información y gestión de datos; todos ellos usan el protocolo de seguridad GSI para la comunicación y autenticación.

Condor y Grid Computing son sistemas distribuidos débilmente acoplados.

Las posibilidades de ejecutar el algoritmo genético propuesto en el presente trabajo en ambientes fuertemente acoplados se discute en la sección 4.3.1 mientras que en la sección 4.3.2 se discute la ejecución la ejecución en ambientes débilmente acoplados

4.3.1. Procesamiento en ambientes fuertemente acoplados

Las estrategias de paralelización del fitness mencionadas en la sección 4.2.2.2, son aptas para su ejecución en entornos fuertemente acoplados. Luego resulta de interés adaptar el algoritmo genético propuesto para implementar estas dos estrategias.

La adaptación del algoritmo a un ambiente fuertemente acoplado requiere la modificación del código fuente, con el fin de transferir los datos a cada uno de los procesadores esclavos disponibles y su posterior recuperación. Se emplea en este caso la biblioteca de comunicaciones basada en pasaje de mensajes MPI [50].

La elección responde a que MPI constituye el estándar de facto para el desarrollo de aplicaciones paralelas sobre este tipo de ambientes. El estándar MPI se diseña con el objeto de obtener el más alto rendimiento posible en las numerosas arquitecturas a las que ha sido portado.

Entre las características principales de MPI se destacan:

- Implementación de funciones para comunicación punto a punto. (Sincrónica, Asincrónica)
- Implementación de funciones para comunicación colectiva.

- Capacidad para manejar grupos de procesos.

Si bien el estándar cuenta con más de 120 funciones, para la paralelización del algoritmo propuesto, se utilizan especialmente las funciones *MPI_Send*(), *MPI_Recv*(), *MPI_Bcast ()* y en menor medida la función *MPI_Reduce()*

Las funciones *MPI_Send*() y *MPI_Recv*() permiten la comunicación punto a punto, de manera sincronizada, para el envío y recepción de información entre los distintos procesos [57]. La utilización de estas dos funciones permite implementar de manera sencilla un esquema maestro esclavo como el presentado en la Figura 4.2.

MPI_Bcast () envía información desde un proceso a todos los demás procesos que componen el grupo de procesos. Por último también se utiliza para la paralelización de la función de fitness la función *MPI_Reduce().* La cual reduce los valores provenientes de los distintos procesos que componen el grupo de proceso a un único valor.

En este trabajo se utiliza mpi4py [58], que permite el acceso a las funciones de la biblioteca MPI en Python. En la Figura 4.9 se presenta a modo de ejemplo, una porción del código que implementa la función de fitness que distribuye al conjunto D en la sección 4.2.2.2.

En primer lugar se importa el módulo MPI que contiene las funciones de la biblioteca MPI provistas por mpi4py. Posteriormente se define la función para calcular el valor de fitness de un individuo de forma paralela, donde se hace uso de las funciones MPI, *Get_size()* y *Get_rank()*. Estas funciones informan sobre la cantidad de procesadores disponibles (*nprocs*) y el número identificador que corresponde al procesador que está ejecutando el código (*myrank*), respectivamente.

A partir del número identificador colocado en la variable *myrank* se puede implementar una estrategia maestro esclavo. Para ello se considera al procesador identificado con el número cero, como el procesador maestro y los restantes son considerados procesadores esclavos. Posteriormente se ejecutan distintas porciones de código según el valor del número identificador.

```
import MPI from mpi4py
def fitness(self,indiv):
  myrank = MPI.COMM_WORLD.Get_rank()
  nprocs = MPI.COMM_WORLD.Get_size()
  status= MPI.Status()
  AB=0.0
  # Codigo que ejecutan los nodos esclavos
  if myrank != 0:
    for instance in self.ptrainset.readRecords():
    AB+=self.Distance(instance,indiv)
    MPI.COMM_WORLD.Send(AB,0,1)
  # Codigo que ejecuta el nodo maestro
  if myrank == 0:
    for proc in xrange(1,nprocs):
      AB+=MPI.COMM_WORLD.Recv(None,proc,1,status)
  fitness=(AB)/self.trainset.totalRecords()
  return fitness
```

Figura 4.9: Implementación de la paralelización de la función de fitness utilizando las funciones de MPI, *Send()* y *Recv()*

Los procesadores esclavos calculan las coincidencias en su porción del conjunto de entrenamiento *ptrainset* para el individuo evaluado y transfieren el resultado parcial contenido en la variable *AB* mediante función *Send()* al procesador maestro. El cual espera mediante la función *Recv()* los resultados de los procesadores esclavos. Luego en el procesador maestro se suma cada resultado parcial recibido para obtener el valor de fitness del individuo.

4.3.2. Procesamiento en ambientes débilmente acoplados

Como se menciona en el análisis de concurrencia que se efectúa en la sección 4.2 no se observa dependencia entre las distintas ejecuciones del

algoritmo necesarias para su validación y ajuste.

Las distintas pruebas validatorias del algoritmo poseen algunas características que merecen destacarse.

En primer lugar como se menciona en la sección 4.2 el objetivo de las mismas es determinar si los ajustes realizados al algoritmo son significativos. Durante el tiempo de ejecución de estas pruebas validatorias no resulta necesario ningún tipo de comunicación entre ellas.

En segundo lugar si no se obtienen los resultados de alguna de las pruebas validatorias, no constituye un problema serio y no provoca ningún inconveniente con el resto de las pruebas ejecutadas. Bastará con ejecutar nuevamente la prueba validatoria para así completar las pruebas necesarias.

Las distintas validaciones pueden ser ejecutadas en entornos débilmente acoplados como Condor o Grid Computing, debido a las dos características mencionadas en los párrafos anteriores.

Se discuten a continuación los pasos necesarios para la adaptación de la aplicación de reconocimiento de patrones de tráfico al entorno Condor cuyos resultados preliminares se presentaron previamente en [59]. Una discusión para el caso de Grid Computing puede consultarse en [60].

Cabe destacar que al no contar con un universo Python dentro del entorno Condor, se debe utilizar el universo *vanilla*, el cual no provee muchas de las funcionalidades más interesantes de Condor.

Para poder ejecutar el código de la aplicación en el entorno Condor es necesario satisfacer los siguientes requisitos:

- El código puede ejecutarse en diferentes sistemas operativos y arquitecturas de hardware.
- El código debe transferirse fácilmente al recurso remoto.

El primer requisito se satisface fácilmente, ya que como se menciona en la sección 4.1 se en encuentran implementaciones de la máquina virtual de

Python para la mayoría de los sistemas operativos utilizados en la actualidad.

Con el objetivo de transferir los datos y el código de la aplicación, se utiliza un archivo auto descomprimible que contiene toda la información necesaria para la ejecución de la aplicación en el recurso remoto.

La aplicación es empaquetada en un único archivo usando la herramienta *cx_freeze* [61]. El conjunto de utilidades *cx_freeze* permite incluir todos los módulos que componen la aplicación en un único ejecutable. El ejecutable creado contiene el código para extraer la lista de módulos incluidos, iniciar la máquina virtual de Python y posteriormente pasar el control al módulo principal del programa.

Este archivo se enlaza a su vez contra una biblioteca estática que contiene la máquina virtual de Python. Al final de este proceso se obtiene un ejecutable ubicado en un directorio previamente definido por el usuario.

El lenguaje Python cuenta con un número de módulos dinámicos que no pueden ser enlazados de manera estática al ejecutable previamente creado por *cx_freeze*. Luego, en un segundo paso, estos módulos como así también los archivos de datos requeridos por el programa, se copian al directorio definido por el usuario.

Finalmente el archivo ejecutable, los archivos de datos y los módulos dinámicos de python se incluyen en un archivo auto descomprimible utilizando la herramienta *makeself.sh* [62]. De esta manera se obtiene un único archivo que puede ser fácilmente transferido a un recurso remoto.

La herramienta *makeself.sh* permite incluir en el archivo auto descomprimible un script que contiene todos los comandos necesarios para realizar la ejecución remota de la aplicación. Se incluyen además las instrucciones necesarias para la creación de los archivos de salida basados en el nombre del recurso remoto y una marca de tiempo. De esta manera se evitan posibles colisiones debido a las múltiples ejecuciones de la aplicación en un mismo nodo.

Para procesar un programa en Condor, se debe ejecutar un trabajo por lotes. Con este fin se necesita crear los archivos que contienen las entradas correspondientes para el programa a ejecutar.

El archivo que se utiliza para la emisión del trabajo en Condor se presenta en la Figura 4.10. Allí se especifica el universo sobre el cual se va a ejecutar

```
Universe=vanilla
executable = gaids.run
should_transfer_files = YES
when_to_transfer_output = ON_EXIT
transfer_output_files = results.output
output=results.output
error=error.output
log=results.log
queue 30
```

Figura 4.10: Archivo que se utiliza para la emisión del trabajo bajo el entorno Condor

la aplicación, como se menciona en párrafos anteriores Condor no cuenta con un universo Python, por lo que se utiliza un universo que solo provee las funcionalidades mínimas. Se indica además el nombre del archivo ejecutable que contiene la aplicación que implementa el algoritmo genético. En este caso se trata del archivo auto descomprimible resultado de ejecutar la aplicación *makeself.sh* (archivo con el nombre gaids.run).

Para posibilitar el mecanismo de transferencia, se colocan dos instrucciones en el archivo de emisión del trabajo: *should_transfer_files*, la cual especifica que Condor debe transferir archivos desde la máquina que emite el trabajo a la máquina remota donde se ejecuta. y *when_to_transfer_output* que especifica cuándo transferir los resultados. Así, Condor transfiere los archivos al nodo, lo ejecuta y transfiere la salida a la máquina que emitió el trabajo.

4.4. Conclusiones

En este capítulo se discuten diferentes alternativas para disminuir el tiempo de proceso del algoritmo genético propuesto. La implementación secuencial

en Python trae como consecuencia un aumento en el tiempo de proceso. Esta situación puede mejorarse mediante la utilización de un compilador en tiempo de ejecución como Psyco. Con sólo incluir el modulo de psyco se puede disminuir en el orden de 2 veces el tiempo de ejecución sin utilizar psyco.

Otras alternativas surgen de las características propias del algoritmo genético propuesto. A partir del análisis de concurrencia del mismo, se concluye que ciertas operaciones del algoritmo puede ser ejecutadas de manera concurrente.

La paralelización de las pruebas validatorias ofrece un buen compromiso entre el speed up que se obtiene y las modificaciones necesarias para su adaptación. Además tiene la ventaja de permitir la ejecución en ambientes débilmente acoplados, lo que implica la posibilidad de utilizar recursos ociosos para su ejecución. Sin embargo existe un límite en la disminución de los tiempos cuando se alcanza un número de procesadores igual al número de pruebas validatorias.

La estrategia de distribución del conjunto P, donde la evaluación de cada individuo de manera concurrente presenta limitaciones debido a las características estacionarias del algoritmo propuesto y la utilización de la técnica de crowding determinístico. Luego no se pueden obtener grandes disminuciones en el tiempo de ejecución.

La alternativa propuesta de distribuir el conjunto de reglas D permite disminuir el tiempo de ejecución y logra una eficiencia cuasi lineal. Sin embargo se necesita modificar el código fuente para implementar un esquema maestro esclavo. En muchos casos esta alternativa de paralelización debe efectuarse sobre entornos fuertemente acoplados como es el caso de los cluster beowulf. Lo que puede constituir una desventaja ya que no siempre es posible contar este tipo de recursos para el procesamiento del algoritmo.

5 Experimentos

En este capítulo se presentan experimentos con el objetivo de determinar la capacidad del algoritmo propuesto para obtener una población de individuos que presenten coincidencias con el mayor número de instancias de tráfico y su capacidad para reconocer anomalías

En la sección 5.2 se discute el comportamiento del algoritmo, principalmente la influencia de la función de fitness propuesta y las funciones de distancias utilizadas en la técnica de crowding. Luego en la sección 5.3 se comprueban los tiempos de ejecución que se obtienen cuando se aplican las diferentes estrategias de paralelización.

5.1. Conjuntos de entrenamiento y prueba

Para los experimentos se utiliza el conjunto de datos recopilado por DARPA durante 1999 [32]. Con este fin DARPA monta una infraestructura de red dedicada, en donde conviven distintos tipos de sistemas operativos, entre los que se destacan máquinas con el sistema operativo Windows, UNIX y Solaris. Sobre esta infraestructura se simulan distintos tipos de ataque. Luego mediante el uso de sniffers que se colocan en puntos estratégicos de la red se recopila información que da origen al conjunto de datos.

El conjunto de datos provisto por DARPA contiene más de 200 instancias de 58 tipos de ataques diferentes a lo largo de 5 semanas de tráfico. El objetivo del mismo es permitir la evaluación de las estrategias que siguen los distintos sistemas de detección de intrusos. El conjunto de datos se ha utilizado en diversos trabajos revisados dentro del estado del arte [14,

44, 2, 15, 31]. Recientemente otros autores han señalado la necesidad de actualizar y/o complementar el conjunto de datos provisto por DARPA [63].

En los experimentos se utilizan 2 conjuntos conformados con instancias de tráfico, para los cuales se emplean subconjuntos de los datos provistos por DARPA. El primer conjunto contiene 9000 entradas de tráfico que representan 4 horas de tráfico normal y se emplea para la fase de entrenamiento. El segundo conjunto se emplea para la fase de prueba y contiene 35000 instancias de tráfico que representan 24 horas de tráfico. El conjunto de prueba está compuesto por instancias de tráfico normal y anómalas. Estas últimas conforman el 1 % de del conjunto de prueba.

En el algoritmo se utiliza una población de 100 individuos, la cual evoluciona a lo largo de 1200 generaciones. Además se utiliza un factor de 0.1 para los operadores de cruzamiento, mutación y descarte. Para este último se emplea, a manera de compromiso, un factor de descarte de condición ($df actor$) de 6.

Se realizan 40 ejecuciones del algoritmo con el objetivo de validar su comportamiento.

Al finalizar el entrenamiento, los individuos obtenidos se utilizan para buscar coincidencias en el conjunto de prueba y se presenta el porcentaje de errores cometidos.

5.2. Comportamiento del Algoritmo Genético Secuencial

En esta sección se estudia el comportamiento del algoritmo propuesto, para lo cual se estudian los resultados obtenidos cuando se aplican las funciones de distancia mencionadas en las sección 3.4.6, para los operadores crowding y crowding determinístico. Además se realiza un estudio de la influencia de la función de fitness por pesos propuesta en la sección 3.4.3 del capítulo 3. Luego, a partir de los conjunto de reglas de clasificación obtenidos, se muestra su eficacia en la detección de anomalías.

5.2.1. Estudio de la influencia de la función de distancia para Crowding y Crowding determinístico

En esta subsección se presentan los resultados para los conjuntos de prueba y entrenamiento. Los mismos se ilustran mediante histogramas de frecuencias relativas en función del error porcentual.

Una comparación de los resultados obtenidos cuando se aplica la función distancia Hamming y la función distancia euclídea por pesos, para el operador de crowding clásico, se muestran en la Figura 5.1 y la Figura 5.2, respectivamente.

La Figura 5.1 muestra el histograma con los resultados en los conjuntos de entrenamiento y prueba cuando se aplica la función distancia de Hamming. El eje de abscisas corresponde al porcentaje del error de clasificación, mientras que el eje de ordenadas corresponde a la frecuencia relativa.

Para el conjunto de entrenamiento en la Figura 5.1 se obtiene el 42 % y el 56 % del error de clasificación para el 55 % y el 25 % de las pruebas de validación procesadas, respectivamente. En el caso del conjunto de prueba se obtiene el 56 % y el 70 % de error de clasificación para el 60 % y el 25 % de las pruebas de validación.

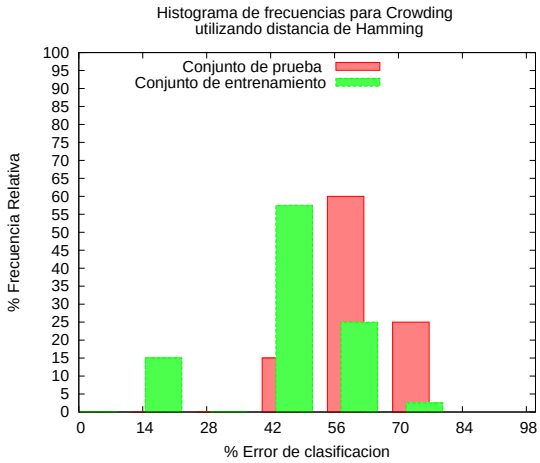

Figura 5.1: Histograma de frecuencias relativas para crowding utilizando distancia de Hamming para 1200 generaciones

En la Figura 5.2 se presentan los resultados obtenidos mediante la función de distancia euclídea con pesos. Se observa que el valor más alto en el histograma para el conjunto de entrenamiento ahora ha descendido a valores entre el 14 % y el 42 % de error de clasificación. En el conjunto de prueba se alcanzan valores entre el 42 % y el 56 %.

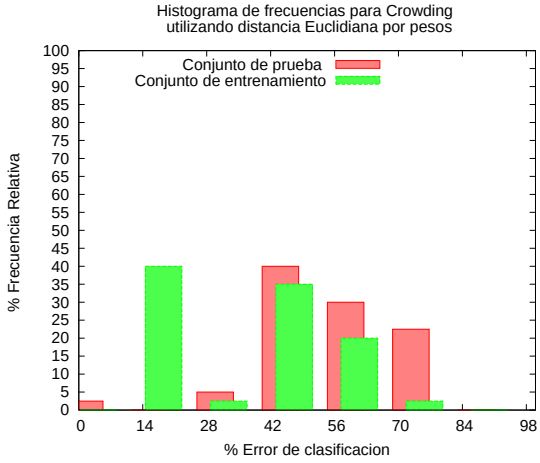

Figura 5.2: Histograma de frecuencias relativas para crowding utilizando distancia euclídea para 1200 generaciones

Como surge de la comparación de las Figuras 5.1 y 5.2 la distancia euclídea por pesos muestra mejores resultados que la distancia de Hamming, tanto para el conjunto de prueba como para el conjunto de entrenamiento. Sin embargo, como se observa en la Figura 5.2, aún en este caso se obtienen errores de clasificación altos (42 % y 56 %) para un número de casos importantes.

Las figuras 5.3 y 5.4 muestran los resultados obtenidos para el caso de crowding determinístico cuando se aplican las distancias de Hamming y euclídea por pesos, respectivamente. De las mismas surge que los resultados mejoran considerablemente a los obtenidos con crowding.

En la figura 5.3 se muestran los resultados al aplicar distancia de Hamming. Se observa que alrededor del 80 % de las validaciones ejecutadas han obtenido un error de clasificación para el conjunto de entrenamiento entre el 14 % y el 28 %. Para el conjunto de prueba los valores más altos del histo-

grama se encuentran entre el 14 % y 28 % del error de clasificación para el 25 % y el 40 % de las validaciones realizadas, respectivamente.

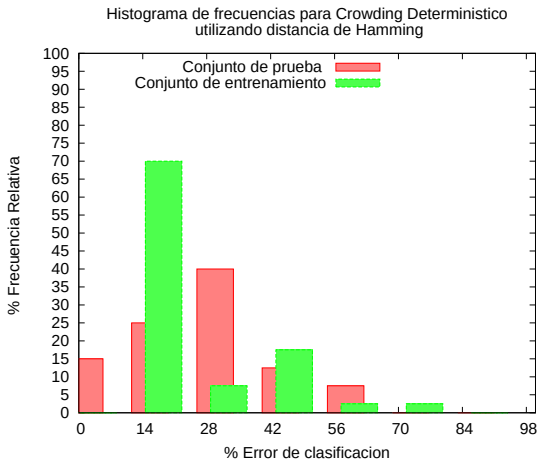

Figura 5.3: Histograma de frecuencias relativas para crowding determinístico utilizando distancia de Hamming para 1200 generaciones

En la Figura 5.4 se muestran los resultados al aplicar la función de distancia euclídea. Puede verse que para el conjunto de entrenamiento alrededor del 60 % de las 40 pruebas de validación ejecutadas finalizan con un error de clasificación cercano al 15 %. Para el conjunto de prueba el histograma esta más distribuido ya que se obtiene aproximadamente un 14 % de error para el 25 % de las validaciones, un 28 % de error para el 40 % de las validaciones y un 42 % para el 25 % de las validaciones.

En este último caso los porcentajes en la frecuencia relativa resultan similares a los de la Figura 5.3 tanto para el conjunto de entrenamiento como para el conjunto de prueba. Luego se desprende que la utilización de la propuesta de función de distancia euclídea basada en pesos no ofrece ventajas significativas sobre la función de distancia de Hamming al utilizar crowding determinístico.

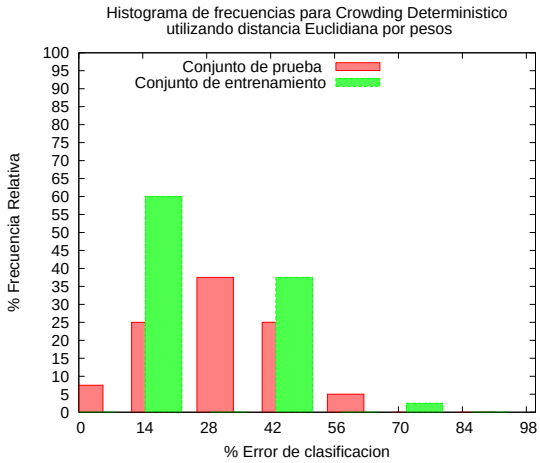

Figura 5.4: Histograma de frecuencias relativas para crowding determinístico utilizando distancia euclídea para 1200 generaciones

5.2.2. Estudio de la influencia de la función de peso en la función de optimización.

En esta subsección se comparan los resultados obtenidos para las funciones de peso α y α' discutidas en las sección 3.4.3.

Las reglas obtenidas por el algoritmo genético al aplicar la función de peso α en la función de optimización se muestran en el Cuadro 5.1. Las tres primeras celdas indican el número de regla y las instancias de tráfico que fueron exitosamente reconocidas en el conjunto de entrenamiento (IRE) y conjunto de prueba (IRP). Se observa en este caso que el algoritmo ha convergido a una única solución. Esta solución se debe a que una gran cantidad de individuos no han sido capaces de encontrar coincidencias en al menos una instancia de tráfico del conjunto de entrenamiento y consecuentemente, fueron penalizados y eventualmente descartados.

Cuadro 5.1: Reglas obtenidas utilizando la función α

	IRP	IRE	HH	MM	SS	Proto	sPort	sPort	sIP	sIP	sIP	sIP	dIP	dIP	dIP	dIP
1	7620	4120	0	0	1	5	-1	80	172	16	-1	-1	207	-1	-1	-1

Los individuos obtenidos al finalizar la ejecución del algoritmo utilizando la función de peso α' se muestran en el Cuadro 5.2, que en este caso confor-

man 16 reglas. Se observa que muchas de las reglas no han sido capaces de encontrar coincidencias en ninguna instancia de tráfico del conjunto de entrenamiento y el conjunto de prueba. Este resultado responde por un lado a las características de la función de peso α' que ha favorecido a individuos con atributos con alta frecuencia de aparición y por otro a la heurística detallada en la sección 3.4.9 que selecciona a los individuos más generales que presentan mayores diferencias entre sí. De esta manera se ha podido encontrar reglas que podrían ser capaces de clasificar instancias de tráfico no presentes en el conjunto de entrenamiento.

Cuadro 5.2: Reglas obtenidas cuando se utiliza la función α'

	IRP	IRE	HH	MM	SS	Proto	sPort	dPort	sIP	sIP	sIP	sIP	dIP	dIP	dIP	dIP
1	7184	2912	0	0	1	5	-1	80	172	16	117	-1	-1	-1	-1	-1
2	5086	2669	0	0	1	5	-1	80	172	16	116	-1	-1	-1	-1	-1
3	5592	2658	0	0	1	5	-1	80	172	16	115	-1	-1	-1	-1	-1
4	5004	0	0	0	1	5	-1	80	172	16	112	-1	-1	-1	-1	-1
5	3491	0	0	0	1	5	-1	80	172	16	113	-1	-1	-1	-1	-1
6	0	0	0	0	1	5	-1	80	172	16	60	-1	-1	-1	-1	-1
7	0	0	0	0	1	5	-1	80	172	16	118	-1	-1	-1	-1	-1
8	0	0	0	0	1	5	-1	80	172	16	101	-1	-1	-1	-1	-1
9	0	0	0	0	1	1	-1	80	172	16	116	-1	-1	-1	-1	-1
10	0	0	0	0	1	1	-1	80	172	16	115	-1	-1	-1	-1	-1
11	0	0	0	0	1	7	-1	80	172	16	117	-1	-1	-1	-1	-1
12	0	0	0	0	1	5	-1	84	172	16	117	-1	-1	-1	-1	-1
13	0	0	0	0	1	5	-1	16	172	16	117	-1	-1	-1	-1	-1
14	0	0	0	0	1	4	-1	80	172	16	116	-1	-1	-1	-1	-1
15	0	0	0	0	1	7	-1	80	172	16	116	-1	-1	-1	-1	-1
16	0	0	0	0	1	5	-1	88	172	16	116	-1	-1	-1	-1	-1

Las figuras 5.5 y 5.6 muestran la evolución del error del mejor resultado obtenido, para las 40 validaciones efectuadas, utilizando las funciones α y α', respectivamente.

Como se observa en la Figura 5.5 el algoritmo que utiliza la función α converge a una solución a partir de la generación 800 y obtiene resultados apenas inferiores al 60 % de error de clasificación en el conjunto de entrenamiento y muy cercanos al 80 % en el conjunto de prueba.

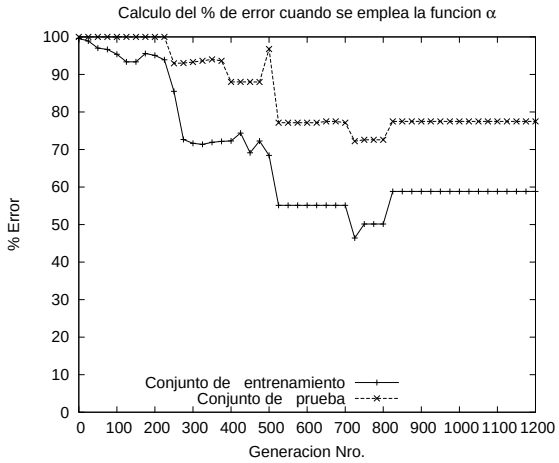

Figura 5.5: Porcentaje de error en los conjuntos de prueba y entrenamiento utilizando la función α definida en la ecuación (3.2) para 1200 generaciones

La Figura 5.6 muestra los mejores resultados obtenidos con el algoritmo cuando se emplea la función α' definida en la ecuación 3.3. En este caso la convergencia del algoritmo se observa a partir de la generación 800 con una solución cercana al 15 % de error de clasificación. Mientras que en el conjunto de prueba recién a partir de la generación 900 se converge hacia una solución cercana al 20 % de error de clasificación.

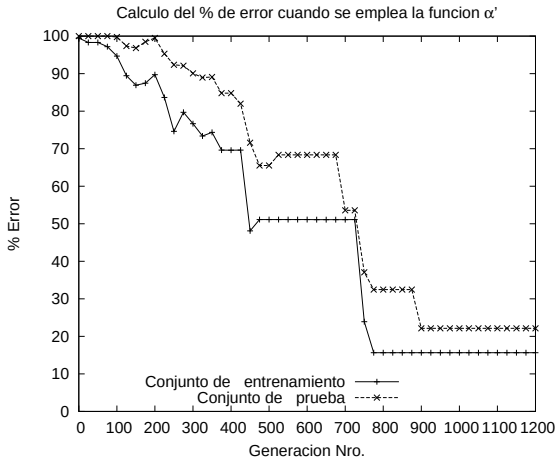

Figura 5.6: Porcentaje de error en los conjuntos de prueba y entrenamiento cuando se utiliza la función α' definida en la ecuación (3.3) para 1200 generaciones

5.2.3. Resultados obtenidos con 1800 generaciones

Como se señaló en los capítulos 3 y 4 resulta de interés procesar en entornos distribuidos el algoritmo genético propuesto con el fin de disminuir los requerimientos computacionales. Los tiempos obtenidos se detallan en la sección 5.3. En este apartado se analiza la calidad de los resultados obtenidos en ambientes distribuidos para 1800 generaciones.

En la Figura 5.7 se muestran los resultados del algoritmo durante 1800 generaciones utilizando distancia de Hamming. Puede verse que para el conjunto de entrenamiento alrededor del 90 % de las 40 pruebas de validación ejecutadas finalizan con un error de clasificación entre 15 % y el 28 %. Para el conjunto de prueba se obtiene aproximadamente un 14 % de error para el 65 % de las validaciones y un 42 % de error para el 30 % de las validaciones. Lo que resulta una mejora significativa respecto a los resultados obtenidos con 1200 generaciones.

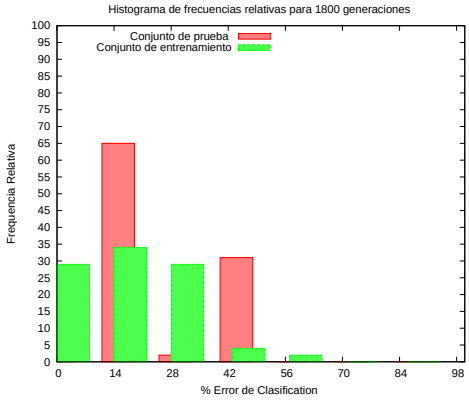

Figura 5.7: Histograma de frecuencias relativas para crowding determinístico utilizando distancia hamming para 1800 generaciones

La Figura 5.8 muestra los mejores resultados obtenidos con el algoritmo cuando se emplea la función α' definida en la ecuación 3.3. En este caso la convergencia del algoritmo se observa a partir de la generación 1100 con una solución cercana al 15 % de error de clasificación. Mientras que en el conjunto de prueba recién a partir de la generación 1500 se converge hacia una solución cercana al 5 % de error de clasificación.

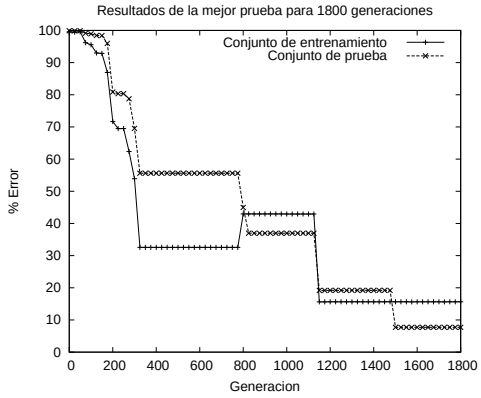

Figura 5.8: Porcentaje de error en los conjuntos de prueba y entrenamiento cuando se utiliza la función α' definida en la ecuación para 1800 generaciones

En el Cuadro 5.3 se muestra una comparación de los resultados obtenidos en 1200 y 1800 generaciones.

Cuadro 5.3: Comparación de los errores de clasificación obtenidos para los conjuntos de entrenamiento y prueba en 1200 y 1800 generaciones.

Generación	% Error Entrenamiento	% Error Prueba
1200	15	20
1800	15	5

Se observa que en el algoritmo ejecutado a lo largo de 1800 generaciones se obtiene el mismo error de clasificación para el conjunto de prueba. Sin embargo se obtiene una mejora en el conjunto de prueba del 75 % respecto al resultado obtenido luego de 1200 generaciones. Este comportamiento se debe a la heurística elegida para la selección de las mejores reglas que componen la población.

5.2.4. Detección de anomalías

En el Cuadro 5.4 se muestran los porcentajes de falsos positivos (instancias de tráfico normal clasificadas como anomalías) y falsos negativos (instancias anómalas clasificadas como normales) que se obtienen con los conjunto de reglas resultantes del algoritmo propuesto luego de 1200 y 1800 generaciones.

Al finalizar las 1200 generaciones el algoritmo clasificó erróneamente como anomalías al 20 % de las instancias de tráfico normales, sin embargo el algoritmo fue capaz de clasificar correctamente el 100 % de las instancias de tráfico anómalas. A la luz de estos resultados puede ser de interés utilizar técnicas complementarias para investigar el contenido del 20 % del tráfico no reconocido correctamente.

Luego de las 1800 generaciones del algoritmo se clasificó erróneamente como anomalías solo al 5 % de las instancias de tráfico normal y se clasificaron correctamente al 100 % de las instancias de tráfico anómalas.

Cuadro 5.4: Porcentaje de falsos positivos y falsos negativos en conjunto de prueba.

Generación	% Falsos positivos	% Falsos negativos	% anomalías reconocidas
1200	20	0	100
1800	5	0	100

En ambos casos se reconocen correctamente el 100 % de las instancias de tráfico del conjunto de prueba que contienen anomalías. Este comportamiento se debe a que el conjunto de prueba contiene un número bajo de instancias de tráfico anómalas. Sin embargo se considera que el algoritmo ejecutado a lo largo de 1800 generaciones obtiene un conjunto de reglas que representa mejor el tráfico normal de la red y en consecuencia podría reconocer potencialmente un porcentaje mayor de anomalías.

5.3. Procesamiento distribuido

En este apartado se discuten los tiempos de ejecución cuando se utilizan las diferentes estrategias de paralelización discutidas en el capítulo 4. En la sección 5.3.1 se analiza el tiempo de la ejecución ejecución secuencial, luego en las secciones 5.3.2 y 5.3.3 se lo compara con los tiempos obtenidos en ambiente fuerte y débilmente acoplados respectivamente.

5.3.1. Tiempo secuencial

La ejecución de la implementación en Python con *psyco* del algoritmo genético para 1200 generaciones en un P4 HT 3000Mhz y 1GB de RAM es de 20.05 minutos. Si se tienen en cuenta las 40 pruebas validatorias el tiempo se extiende a aproximadamente 802 minutos

En el Cuadro 5.5 se muestran los tiempos de ejecución de las distintas etapas del algoritmo genético propuesto. Se indica además la variable correspondiente a las métricas de la sección 4.2 del capítulo 4, según corresponda.

Cuadro 5.5: Tiempos en segundos de las diferentes etapas del algoritmo secuencial

Etapas del algoritmo	Variables	Tiempo (s)
Fitness (un individuo)	t_f	0.3
Fitness Población Inicial	$t_f(P)$	30.5
Fitness en Crowding	$t_f k_c$	0.61
Mutación		0.01
Cruzamiento		0.01
Descarte		0.01
Fitness (promedio)	$t_f \tilde{k}_m$	0.36

El tiempo que se obtiene para el cálculo del fitness de un individuo t_f es de 0.3 segundos. Consecuentemente se observa que para el cálculo del fitness inicial, donde se evalúan a todos los individuos de la población, se requieren 30 segundos. Esta operación sólo resulta significativa al inicio de la ejecución del algoritmo.

Durante las siguientes iteraciones del algoritmo sólo se necesita calcular el fitness de los individuos que han sido modificados por los operadores de mutación y descarte de condición, lo que demanda 0.36 segundos en promedio. Además se requiere el cálculo del fitness de los nuevos individuos que se obtienen de la operación de cruzamiento, operación que requiere 0.6 segundos.

Como se desprende del Cuadro 5.5 las etapas de cruzamiento, mutación y descarte demandan el 3 % del tiempo de cada iteración del algoritmo, por lo que no resultan significativas para calcular el tiempo total secuencial, mientras el fitness demanda el 97 % del tiempo de ejecución. Luego se puede aplicar la ecuación para determinar el tiempo total de ejecución secuencial que se propuso en la sección 4.2:

$$\begin{aligned} T_s &\approx [t_f(k_m + k_c)]\, G\, v \\ T_s &\approx (0{,}6 + 0{,}6) * 1200 * 40 = 57600\, seg = 960\, min \end{aligned}$$

La estimación del tiempo secuencial T_s teórico de acuerdo a la métrica propuesta es de 960 minutos. La diferencia respecto del 20 % respecto del tiempo que se obtiene en la práctica se debe a que la ecuación (4.12) siempre se considera que el tiempo $t_f k_m$ para 2 individuos. Sin embargo en la práctica existen algunos casos donde se seleccionan los individuos que superan el máximo factor de descarte, por lo que no son agregados a la población. Una posibilidad para acercar el valor teórico a la práctica es utilizar $t_f \tilde{k}_m$, factor que considera el efecto anterior. Luego el tiempo teórico estimados es:

$$\begin{aligned} T_{sc} &\approx [t_f(\tilde{k_m} + k_c)]\, G\, v \\ T_{sc} &\approx (0{,}36 + 0{,}6) * 1200 * 40 = 46032\, seg = 767\, min \end{aligned}$$

Si se utiliza el tiempo promedio para calcular el fitness $t_f\, \tilde{k}_m$, el resultado teórico es de 767 minutos, el cual presenta un 5 % de error respecto al obtenido en la práctica.

En el Cuadro 5.6 se muestran los resultados estimados y reales. Del mismo surge que T_{sc}, pese a su simplicidad, es una métrica apropiada.

Cuadro 5.6: Comparación de los tiempos teóricos y reales para el algoritmo secuencial

Tiempo	Tiempo real (min)	Tiempo teórico (min)	Tiempo teórico corregido (min)
Secuencial	802	960(20 %)	767(5 %)

5.3.2. Resultados de los Tiempos en ambientes fuertemente acoplados

En esta sección se presentan los resultados que se obtienen al aplicar las estrategias de paralelización discutidas de la sección 4.2.2 sobre ambientes fuertemente acoplados

La infraestructura disponible para ejecutar los experimentos consiste en el cluster beowulf *Reloaded,* el cual se compone de una estación de trabajo maestra que actúa como front end y de 10 estaciones de trabajo esclavas. Las estaciones poseen un procesador P4 HT 3.0 GHz, 1 MB de Cache L2, 1 GB de RAM, disco rígido SATA de 80 GB. La estación de trabajo maestra y las esclavas están interconectadas mediante una red gigabit Ethernet.

El tiempo de ejecución total que se obtiene al aplicar la estrategia de distribución de la población P entre los N procesadores disponibles, es de 12.4 minutos. Si se tienen en cuenta las 40 pruebas validatorias el tiempo se extiende a 496 minutos. Luego, si bien se logra disminuir el tiempo total en un 61 %, no se ha podido obtener un speed up lineal debido a que sólo se pudo distribuir P entre 2 procesadores de los 10 disponibles, aspecto que se había adelantada en el capítulo 4.

En el Cuadro 5.7 se muestran los tiempos que se obtienen en la práctica para cada una de las etapas del algoritmo propuesto cuando se divide a la población P.

Se observa que el tiempo para calcular el fitness inicial de todos los individuos de la población se ha reducido linealmente. La ejecución sobre 10

Cuadro 5.7: Tiempos en segundos de las distintas etapas del algoritmo paralelo que divide a la población P

Etapas del algoritmo	Variables	Tiempo (s)
Fitness (un individuo)	t_f	0.3
Fitness Población Inicial	$t_f(P)$	3.5
Crowding	$t_f k_c$	0.3
Mutación		0.001
Cruzamiento		0.001
Descarte		0.00.1
Fitness (promedio)	$t_f \tilde{k}_m$	0.24

procesadores ha permitido obtener un tiempo de 0.3 segundos, valor diez veces menor al que se obtiene en el algoritmo secuencial.

El tiempo requerido para calcular el fitness de los individuos durante la aplicación del operador de crowding determinístico se ha reducido a 0.3 segundos frente a los 0.6 del algoritmo secuencial. Sin embargo el tiempo promedio para calcular el fitness de los individuos obtenidos por el operador de mutación y descarte apenas se ha reducido un 33 % del tiempo obtenido en el algoritmo secuencial.

Es posible calcular el tiempo paralelo de acuerdo a la ecuación (4.13) presentada en el capítulo 4. Luego la estimación del tiempo paralelo teórico es:

$$\begin{aligned} T_p &\approx (\quad \frac{t_f \tilde{k}_m}{N} + \frac{t_f k_c}{N}) \, G \, v \\ T_p &\approx (\quad 0{,}24 + 0{,}3) * 1200 * 40 = 25920 \, seg = 432 \, min \end{aligned}$$

Como se puede observar el tiempo teórico estimado con las métricas propuestas es de 432 minutos, que resulta una buena aproximación al tiempo que se obtiene en la práctica. El error del 12 % que se obtiene responde a la diferencia que existe entre el tiempo de fitness promedio $t_f \tilde{k}_m$ y los tiempos que se obtienen en la práctica. También deben considerarse los tiempos de comunicación que por motivos de simplicidad no se han tenido en cuenta.

Cuando se aplica la estrategia de paralelización que divide el conjunto de

entrenamiento D entre los N procesadores disponibles se obtiene un tiempo total de ejecución de 3.41 minutos. Si se ejecutan las 40 pruebas validatorias el tiempo total asciende a 136 minutos. Esta estrategia ha permitido obtener una disminución del 85 % respecto al tiempo secuencial.

En el Cuadro 5.8 se muestran los tiempos de las distintas etapas del algoritmo genético cuando se divide el conjunto de entrenamiento.

Cuadro 5.8: Tiempos en segundos de las distintas etapas del algoritmo paralelo que divide al conjunto de entrenamiento D

Etapas del algoritmo	Variables	Tiempo (s)
Fitness (un individuo)	t_f	0.03
Fitness Población Inicial	$t_f(P)$	3.2
Crowding	$t_f k_c$	0.065
Mutación		0.001
Cruzamiento		0.001
Descarte		0.001
Fitness (promedio)	$t_f \tilde{k}_m$	0.065

La principal ventaja que ofrece esta estrategia se observa en el cálculo del fitness de un individuo, el cual es de 0.03 segundos. Luego se alcanza una considerable disminución del tiempo de las etapas en donde se calcula la misma.

En el cálculo del fitness de la población se obtiene un tiempo de 0.3 segundos, el cual es un valor similar al que se obtiene cuando se utiliza la paralelización que distribuye al conjunto P.

Por otra parte se observan diferencias durante la aplicación del operador de crowding determinístico, donde se obtiene un tiempo de 0.065 segundos, lo que representa una disminución proporcional respecto al algoritmo secuencial.

El tiempo requerido para calcular el fitness de los individuos que se obtienen por el operador de mutación y el operador de descarte es de 0.065, el cual disminuye en el orden del 98 % respecto al tiempo del algoritmo secuencial.

Cuando se aplica la ecuación (4.16) discutida en el capítulo 4, para calcular el tiempo paralelo teórico que considera la distribución del conjunto de entrenamiento D se obtiene:

$$\begin{aligned} T_p &\approx \frac{t_f}{N}(k_m + k_c)\, G\, v \\ T_p &\approx (0{,}065 + 0{,}065) * 1200 * 40 = 62400seg = 103min \end{aligned}$$

Donde el tiempo total de ejecución paralelo es de 104 minutos, la diferencia del 23 % respecto a los 136 minutos obtenidos en la práctica responde en gran parte a la utilización del tiempo promedio del fitness $t_f \tilde{k}_m$y los tiempos de comunicación.

En el Cuadro 5.9 se muestra una comparación de los tiempos teóricos y reales obtenidos con las dos estrategias de paralelización utilizadas en este apartado y el tiempo secuencial. Se observa que la distribución del conjunto de entrenamiento D es la más eficiente. La misma logra reducir el tiempo de ejecución a 136 minutos lo que constituye una disminución del 84 % respecto del tiempo secuencial.

Cuadro 5.9: Comparación de los tiempos teóricos y los tiempos obtenidos en la práctica para el algoritmo secuencial y los algoritmos ejecutados en forma paralela.

Tipo de implementación	Tiempo teórico (min)	Tiempo real (min)
Secuencial (promedio)	768	802
Distribución P	432	480
Distribución D	104	136

5.3.3. Resultados de los Tiempos en ambientes débilmente acoplados

En esta sección se presentan los resultados que se obtienen al aplicar las estrategias de paralelización discutidas de la sección 4.2.2 sobre ambientes débilmente acoplados como el entorno Condor.

La infraestructura disponible para ejecutar trabajos de Condor consiste en una estación maestra que se comporta como front end y 10 estaciones de trabajo esclavas. Estas estaciones conforman lo que se conoce como pool en la nomenclatura de Condor. Las estaciones poseen un procesador P4 HT 3.0 GHz, 1 MB de Cache L2, 1 GB de RAM, disco rígido SATA de 80 GB, están interconectadas mediante una red Gigabit Ethernet.

El tiempo que se obtiene al ejecutar el algoritmo genético propuesto sobre la infraestructura disponible es de 80.2 minutos.

El tiempo total de ejecución bajo esta estrategia de paralelización puede calcularse de forma teórica si se utiliza la ecuación (4.9) que se define en el capítulo 4.

$$\begin{aligned} T_s &\approx [t_f(k_m + k_c)]\, G\, \frac{v}{N} \\ T_s &\approx [(0{,}36 + 0{,}6)] * 1200 * \frac{40}{10} = 4603\, seg = 76{,}8\, min \end{aligned}$$

Se observan que los resultados teóricos presentan una diferencia del 5 % a los obtenidos en la práctica.

En el Cuadro 5.10 se muestra una comparación de los tiempos teóricos y reales obtenidos con esta estrategia de paralelización y el tiempo secuencial. Se observa que la distribución de las pruebas validatorias (v) logra reducir el tiempo de ejecución a 80 minutos lo que constituye una disminución del 90 % respecto del tiempo secuencial.

Cuadro 5.10: Comparación de los tiempos teóricos y los tiempos obtenidos en la práctica para el algoritmo secuencial y los algoritmos ejecutados en forma paralela.

Tipo de implementación	Tiempo teórico (min)	Tiempo real (min)
Secuencial	768	802
Distribución v	76.8	80.2

Es importante resaltar que la estrategia de paralelización de las validaciones es competitiva con la paralelización del fitness. En este ejemplo los tiempos son incluso menores.

De esta forma se manifiesta la potencia de cálculo que se puede alcanzar con entornos tipo Grid , como Condor en este caso. En el grupo de trabajo del autor se están llevando a cabo experiencias con mayor número de máquinas en condiciones de trabajo muy similares a las de producción.

5.4. Conclusiones

En este capítulo se han clasificado exitosamente patrones de tráfico normal mediante el algoritmo genético propuesto.

El algoritmo genético para el reconocimiento de patrones de tráfico normal observa resultados prometedores en el caso de estudio planteado. Se generan reglas con atributos que permiten encontrar coincidencias del orden del 85 % para el conjunto de entrenamiento y alrededor de un 95 % para el conjunto de prueba. El algoritmo fue capaz de reconocer el 100 % de las anomalías presentes en el conjunto de prueba.

La utilización de la función de peso α' definida en la ecuación (3.3) constituye una mejora significativa a la función de fitness. Como se ha señalado en la sección 5.2.2 se han podido encontrar reglas de clasificación que, potencialmente son capaces de clasificar instancias de tráfico no presentes en el conjunto de entrenamiento.

De las técnicas de crowding evaluadas, crowding determinístico ha presentado los mejores resultados a costa de no aumentar la complejidad computacional del algoritmo. Crowding determinístico muestra no ser sensible a la función de distancia utilizada, situación que no se presenta en otras variantes de crowding.

El procesamiento en entornos distribuidos ha permitido disminuir sensiblemente los tiempos de ejecución de 802 minutos a aproximadamente 80, con

un speed up y eficiencia cercanos a 10 y 1, respectivamente para la paralelización de las validaciones mientras que para la paralelización del fitness el tiempo es de 136 minutos.

Consecuentemente se ha podido extender a 1800 las generaciones del algoritmo y por lo tanto, mejorar la calidad de los resultados obtenidos.

Las métricas aproximadas propuestas en el capítulo 4 han predecido razonablemente bien los tiempos de proceso pese a la simplicidad de las mismas.

También es importante destacar que el procesamiento en entornos débilmente acoplados de las pruebas validatorias muestran las ventajas potenciales de Grid Computing y middleware similares a Condor.

La disponibilidad de mayores recursos computacionales, mediante la Grid Regional Mendoza, permitirá investigar la paralelización del fitness en conjunto con las validaciones estadísticas.

De esta manera el poder computacional disponible permitirá contemplar el uso e investigación de algoritmos más costosos como el generacional o variantes de algoritmos genéticos paralelos como son los algoritmos genéticos descentralizados.

Desde el punto de vista de la computación distribuida puede resultar de interés simular en entornos débilmente acoplados el cálculo de fitness para lo cual se puede aprovechar la experiencia del grupo de trabajo del autor [64].

Finalmente permanece abierto el estudio de la disminución de tiempo que se puede obtener cuando se utiliza una combinación de ambientes fuerte y débilmente acoplados con el fin de paralelizar simultáneamente fitness y validaciones.

6 Conclusiones y trabajos futuros

En este capítulo se presentan las conclusiones finales del presente trabajo, como así también se proponen algunas líneas de trabajo para el futuro.

6.1. Conclusiones

En la presente tesis se ha presentado una arquitectura para un sistema de detección de intrusos. De los módulos que componen la misma, el núcleo principal lo constituye el módulo de aprendizaje y la vez objetivo general de este trabajo. Dicho módulo se basa en un algoritmo genético de tipo estacionario cuyo objetivo es obtener un conjunto de reglas que representen las instancias normales de tráfico y que constituye una de las características más destacables del presente trabajo.

Entre los componentes del algoritmo genético propuesto, se destaca el empleo de la función de fitness basada en pesos con el objeto de favorecer algunos atributos que por experiencia disciplinar presentan mayor relevancia para la detección de intrusos. El empleo de la misma ha permitido encontrar reglas de clasificación que, son potencialmente capaces de clasificar instancias de tráfico no presentes en el conjunto de entrenamiento.

Se ha profundizado en las alternativas para la implementación de las técnicas de nicho, las cuales no se discuten en detalle en los trabajos disponibles del estado del arte. Luego, se ha propuesto una variante de crowding determinístico basado en una función de distancia euclídea, la cual considera los atributos más relevantes de acuerdo a la experiencia disciplinar.

Con el fin de estudiar el comportamiento del algoritmo propuesto se han diseñado experimentos. En los mismos se observaron resultados prometedores para el caso de estudio planteado. Se han obtenido reglas con atributos que han permitido encontrar coincidencias del orden del 85 % para el conjunto de entrenamiento y alrededor de un 95 % para el conjunto de prueba. Dichas reglas fueron capaces de reconocer el 100 % de las anomalías incluidas en el conjunto de prueba.

Debido a los costos computacionales del algoritmo propuesto, se estudiaron alternativas para disminuir los tiempos de ejecución del mismo. Con este fin se establecieron métricas simples para estimar el tiempo de proceso requerido. A partir del correspondiente análisis de concurrencia, se discutieron e implementaron distintas estrategias de paralelización.

La paralelización de las pruebas validatorias ofrecen un buen compromiso entre el speed up obtenido y las modificaciones necesarias para su adaptación. Además tiene la ventaja de permitir la ejecución en ambientes débilmente acoplados, lo que implica la posibilidad de utilizar recursos ociosos para su ejecución. Sin embargo existe un límite en la disminución de los tiempos cuando se alcanza un número de procesadores igual al número de pruebas validatorias.

En el caso de la paralelización de la función de fitness, la alternativa propuesta de distribuir el conjunto de reglas D permite disminuir el tiempo de ejecución y logra una eficiencia cuasi lineal. Sin embargo se necesita modificar el código fuente para implementar un esquema maestro esclavo.

El procesamiento en entornos distribuidos ha permitido disminuir sensiblemente los tiempos de ejecución, consecuentemente se ha podido extender a 1800 las generaciones del algoritmo, con lo que se ha podido mejorar la calidad de los resultados obtenidos.

6.2. Trabajos futuros

Para este trabajo se empleó un conjunto de datos provisto por DARPA, el cual dada su antigüedad puede no representar convenientemente el tipo

de tráfico de las redes actuales. Luego, resulta importante comprobar los resultados del algoritmo propuesto sobre una infraestructura de red real.

La arquitectura propuesta permitirá experimentar variantes para cada uno de los módulos del sistema. Se estima que la captura de datos desde distintas fuentes permitirá obtener información adicional respecto de la que se obtiene en un punto de captura único [65]. Luego se contempla investigar la posibilidad de emplear una arquitectura distribuida, con módulos de captura y procesamiento ubicados en diferentes puntos de la red.

En el módulo de aprendizaje resulta de interés estudiar el comportamiento de diferentes variantes del algoritmo propuesto, como resulta la aplicación de algoritmos genéticos distribuidos. Estos algoritmos permiten acelerar la convergencia de las soluciones a la vez que exploran distintos espacios de soluciones. También puede ser importante realizar una comparación con otras técnicas de aprendizaje de máquina como redes neuronales, reglas de asociación o clasificadores bayesianos por ejemplo.

En el módulo de evaluación y reconocimiento de anomalías se emplea una heurística simple, por lo que puede ser de interés investigar otras heurísticas que podrían mejorar la calidad de los resultados. El empleo de técnicas que incluyan lógica difusa puede ser una alternativa a considerar.

Desde el punto de vista de la computación distribuida cabe la posibilidad de llevar a cabo el cálculo de fitness entornos débilmente acoplados, con el fin de evaluar su factibilidad.

Finalmente permanece abierto el estudio de la disminución de tiempo que se puede obtener cuando se utiliza una combinación de ambientes fuerte y débilmente acoplados con el fin de paralelizar simultáneamente fitness y validaciones.

Bibliografía

[1] C. Sinclair, P. Lyn, and S. Matzer, “An application of machine learning to network intrusion detection.,” in *15th Annual Computer Security Applications Conference*, 1999.

[2] W. Li, “A genetic approach to network intrusion detection,” in *Proceedings of the United States Department of Energy Cyber Security Group 2004 Training Conference*, 2004.

[3] M. Roesch, “Snort: Lightweight intrusion detection for networks,” in *LISA*, pp. 229–238, 1999.

[4] B. Mukherjee, L. T. Heberline, and K. Levitt, “Network instruction detection.,” *IEEE Network*, vol. 8, pp. 26–41, 1994.

[5] W. Lee, *A data mining framework for constructing features and models for intrusion detection systems*. PhD thesis, Columbia University, 1999.

[6] A. K. Ghosh, A. Schwartzbard, and M. Schatz, “Learning program behavior profiles for intrusion detection,” in *Proceedings 1st USENIX Workshop on Intrusion Detection and Network Monitoring*, pp. 51–62, Apr. 1999.

[7] D. Endler, “Intrusion detection applying machine learning to solaris audit data,” in *Proc. of the 1998 Annual Computer Security Applications Conference (ACSAC’98)*, (Scottsdale, AZ), pp. 268–279, IEEE Computer Society Press, 1998.

[8] S. Staniford, J. A. Hoagland, and J. M. McAlerney, “Practical automated detection of stealthy portscans,” *J. Comput. Secur.*, vol. 10, no. 1-2, pp. 105–136, 2002.

[9] J. Luo, “Integrating fuzzy logic with data mining methods for intrusion detecion,” Master’s thesis, Mississippi State University, 1999.

[10] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo, “A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data,” in *Data Mining for Security Applications, 2002.* (D. Barbara and S. Jajodia, eds.), Kluwer, 2002.

[11] T. Brugger, “Data mining methods for network intrusion detection,” tech. rep., University of California, Davis, 2004.

[12] K. A. D. Jong, W. M. Spears, and D. F. Gordon, “Using genetic algorithms for concept learning,” *Machine Learning*, vol. 13, pp. 161–188, 1993.

[13] J. R. Quinlan, *C4.5 : programs for machine learning*. D, Morgan Kaufmann Publishers Inc., 1993.

[14] R. Gong, M. Zulkernine, and P. Abolmaesmumi, “A software implementation of a genetic algorithm based approach to network intrusion detection.,” in *Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks (SNDP/SWAN’05)*, vol. 0, pp. 246–253, 2005.

[15] W. Lu and I. Traore., “Detecting new forms of network intrusion using genetic programming.,” *Computational Intelligence,*, vol. 20, pp. 475–494, 2004.

[16] C. Yin, S. Tian, H. Huang, and J. He, “Applying genetic programming to evolve learned rules for network anomaly detection,” in *LNCS ICNC 2005*, vol. 3612, pp. 323–331, Springer-Verlg, 2005.

[17] M. Crosbie and E. H. Spafford, “Applying genetic programming to intrusion detection,” in *Working Notes for the AAAI Symposium on Genetic Programming* (E. V. Siegel and J. R. Koza, eds.), (MIT, Cambridge, MA, USA), pp. 1–8, AAAI, 10–12 1995.

[18] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[19] M. Mahoney and P. Chan, “Learning rules for anomaly detection of hostile network traffic,” in *3rd IEEE Int’l Conf. on Data Mining. (2003)*, 2003.

[20] J. J. Ciarlante, “Red privada virtual sobre mensajeria instantanea,” Master’s thesis, Universidad de Mendoza Facultad de Ingenieria, Argentina, 2005.

[21] J. M. Winett, “Rfc 147, the definition of a socket.” Disponible en : `ftp://ftp.rfc-editor.org/in-notes/rfc147.txt`, 1971.

[22] D. Goldberg, *Genetic Algorithms in search Optimization and Machine Learning.* Addison Wesley, 1989.

[23] M. Mitchell, *An Introduction to Genetic Algorithms (Complex Adaptive Systems)*. MIT Press, 1991.

[24] J. Holland, *Adaptation In Natural and Artificial Systems*. Ann Arbor: The University of Michigan Press, 1975.

[25] M. Wong and K. Leung, *Data Mining Using Grammar-Based Genetic Programming and Applications*. Kluwer Academic Publishers Norwell, 2000.

[26] B. Miller and M. Shaw, “Genetic algorithms with dynamic niche sharing for multimodalfunction optimization.,” Tech. Rep. 95010, University of Illinois at Urbana-Champaing, 1995.

[27] S. Mahfoud, “Crowding and preselection revisited.,” Tech. Rep. 92004, University of Illinois at Urbana-Champaing, 1992.

[28] S. Bridges and R. Vaughn, “Fuzzy data mining and genetic algorithms applied to intrusion deteccion,” in *National Information Systems Security Conference*, (Disponible en: `http://csrc.nist.gov/nissc/2000/proceedings/papers/005.pdf`), 2000.

[29] J. Gomez, D. Dasgupta, F. Gonzalez, and O. Nasraoui, “Complete expression trees for evolving fuzzy classifier systems with genetic algorithms and application to network

intrusion detection.," in *Annual Fuzzy Information Society*, pp. 469–474, 2002.

[30] A. Sung and S. Mukkamala, "The feature selection and intrusion detection problems," in *Advances in computer science ASIAN 2004*, vol. 3321/2004, pp. 468–482, 2004.

[31] T. Shon, Y. Kim, C. Lee, and J. Moon, "A machine learning framework for network anomaly detection using svm and ga," in *6th IEEE Information Assurance Workshop*, 2005.

[32] R. Lippmann, J. W. Fried, D. J. Korba, and K. Das, "The 1999 darpa off-line intrusion detection evaluation," *Computer Networks*, vol. 34, pp. 579–595, 2000.

[33] G. Dimitris, T. Ioannis, and D. Evangelos, "Feature selection for robust detection of distributed denial-of-service attacks using genetic algorithms," in *Methods and Applications of Artificial Intelligence*, vol. 3025/2004, pp. 276–281, 2004.

[34] A. Chittur, "Model generation for an intrusion detection system using genetic algorithms." Disponible en: `http://www1.cs.columbia.edu/ids/publications/`, 2001.

[35] "Tcpdump - libpcap." Disponible en : `http://www.tcpdump.org/`.

[36] J. Postel, "Rfc 768, user datagram protocol." Disponible en: `ftp://ftp.rfc-editor.org/in-notes/rfc768.txt`, 1980.

[37] J. Postel, "Rfc 792, internet control message protocol." Disponible en: `ftp://ftp.rfc-editor.org/in-notes/rfc792.txt`, 1981.

[38] J. Postel, "Rfc 791, internet protocol." Disponible en : `ftp://ftp.rfc-editor.org/in-notes/rfc791.txt`, 1981.

[39] C. Catania and C. G. Garino, "Una propuesta de reconocimiento de patrones en el tráfico de red basada en algoritmos genéticos," in *ninth Argentinian Symposium on Artificial Intelligence, ASAI* (D. Godoy and A. Maguitman, eds.), pp. 174–185, 2007.

[40] C. Catania and C. G. Garino, "Reconocimiento de patrones en el tráfico de red basada en algoritmos genéticos," *Inteli-*

gencia Artificial, Revista Iberoamericana de Inteligencia Artificial, vol. Special Issue: 9th Argentinian Symposium on Artificial Intelligence, 2007. Aceptado para su publicación.

[41] D. Whitley, “The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best,” in *Proceedings of the Third International Conference on Genetic Algorithms* (J. D. Schaffer, ed.), (San Mateo, CA), Morgan Kaufman, 1989.

[42] B. Sareni and L. Krahenbuhl., “Fitness sharing and niching methods revisited.,” *IEEE Transactions on Evolutionary computation*, vol. 2, September 1998.

[43] C. Oei, D. Goldberg, and S.-J. Shang, “Tournament selection, niching, and the preservation of diversity,” Tech. Rep. 91011, University of Illinois at Urbana-Champaign, 1991.

[44] E. Leon, O. Nasraoui, and J. Gomez, “Anomaly detection based on unsupervised niche clustering with application to network intrusion detection.,” in *IEEE Congress on Evolutionary Computation*, 2004.

[45] “The python programming language.” Disponible en: `http://http://www.python.org`.

[46] A. Rigo, “Psyco, the python specializing compiler.” Disponible en: `http://psyco.sourceforge.net/psyco.ps.gz`, 2001.

[47] E. C. Paz, “Desiging eficient master-slave parallel genetic algorithtms,” Tech. Rep. 97004, University of Illinois at Urbana-Champaing, 1997.

[48] E. C. Paz, “A summary of research on parallel genetic algorithms.,” Tech. Rep. 95007, University of Illinois at Urbana-Champaing, 1995.

[49] E. Alba and J. Troya, “A survey of parallel distributed genetic algorithms.,” *Complexity Volume*, vol. 4, no. 4, 1999.

[50] “Message passing interface library.” Disponible en: `http://http://www-unix.mcs.anl.gov/mpi`.

[51] K. Rasheed and B. D. Davison, “Effect of global parallelism

on the behavior of a steady state genetic algorithm for design optimization," in *Proceedings of the 1999 Congress of the 1999 Congress on Evoluationary Computation, 1999. CEC 99.*, pp. 534–541, 1999.

[52] "Beowulf.org." Disponible en : `http://www.beowulf.org/overview/index.html`.

[53] "Globus toolkit 4." Disponible en: `http://www.globus.org`.

[54] M. Litzkow, M. Livny, and M. Mutka, "Condor - a hunter of idle workstations," in *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[55] "Myrinet overview." Disponible en : `http://www.myricom.com/myrinet/overview/`.

[56] "Infiniband overview." Disponible en : `http://www.infinibandta.org/events/past/it_roadshow/overview.pdf`.

[57] P. S. Pacheco, *Parallel Programming with MPI*. D, Morgan Kaufmann Publishers, Inc., 1997.

[58] L. Dalcín, R. Paz, and M. Storti, "Mpi for python," *J. Parallel Distrib. Comput.*, vol. 65, no. 9, pp. 1108–1115, 2005.

[59] P. Martinez, C. Catania, C. G. Garino, and J. Diaz, "Reconocimiento de patrones de tráfico de red en un ambiente condor," in *Anales del CACIC 2007, XIII Congreso Argentino de Ciencias de la Computación*, pp. 1288–1299, 2007.

[60] C. Catania, J. Monetti, C. G. Garino, S. Salinas, P. Martinez, and O. León, "Grid enabled tool for network traffic classification.," tech. rep., LAPIC, Mendoza, 2007.

[61] C. C. Ltd., "cx_freeze, a set of utilities for freezing python scripts into executables." Disponible en: `http://python.net/crew/atuining/cx_Freeze/`.

[62] S. Peter, "makeself, make self-extractable archives on unix." Disponible en: `http://www.megastep.org/makeself/`.

[63] M. Rehák, M. Pechoucek, D. Medvigy, M. Prokopová, J. Tozicka, and L. Foltýn, "Agent methods for network intrusion detection and response," in *HoloMAS*, pp. 149–160, 2007.

[64] J. Monetti, C. G. Garino, and O. León, “Diseño y programación de aplicaciones para grid computing,” in *Anales del Congreso ENIDI 2006* (S. R. y Jorge Nuñez McLeod, ed.), Fac. de Ingenieria, UNCuyo, 2006.

[65] W. Lee and D. Xiang, “Information-theoretic measures for anomaly detection,” in *Proc. of the 2001 IEEE Symposium on Security and Privacy*, pp. 130–143, May 2001.