Universidad de las Ciencias Informáticas

Facultad 6

Título: **“**Propuesta de algoritmo distribuido de Programación Genética para la generación de modelos predictivos de actividad biológica.”

Trabajo de Diploma para optar por el título de

Ingeniero Informático

Autora: Mayli Medero Cuadrado

Tutor(es): Dr. Ramón Carrasco Velar

Ing. Yania Molina Souto

Junio, 2009

La posibilidad de realizar un sueño es lo que hace que la vida sea interesante.

Paulo Coelho

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____

Mayli Medero Cuadrado

Autora

_____ _____

Ing. Yania Molina Souto

Tutor

Dr. Ramón Carrasco Velar

Tutor

DATOS DE CONTACTO

Tutores:
Ing. Yania Molina Souto
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email: ymolinas@uci.cu
Dr. Ramón Carrasco Velar
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email: rcarrasco@uci.cu

AGRADECIMIENTOS

Quiero agradecerle antes que nada a toda mi familia por la confianza que tuvieron en mi, por el apoyo que me dieron y el sacrificio que han hecho, sepan que de no ser por ustedes este sueño nunca su hubiese hecho realidad.

A mis tutores, por brindarme toda su ayuda y sabiduría.

A la profe Yania por su incondicionalidad en todo momento.

A Andrés, mi chuchy lindo, por hacer uso de toda su paciencia y darme todo su amor.

A Julio Antonio por ser el más sincero de los amigos, a Jesús gracias por su cariño y sus mimos, a Yariel por cuidarme tanto y ser tan especial, sepan que son mis queridos amigos.

A Mary, Taty, Ingrid y Lexy porque siempre estuvieron conmigo cuando no había nadie más.

A Veylys (V), por tener siempre una palabra amable para mí.

A mis tantas amistades porque gracias a ustedes la vida en la UCI se hizo amena con tantas cosas que hicimos y por los recuerdos que perdurarán por siempre.

Gracias a todos.

DEDICATORIA

A mi mami y mi papi, la verdad es que no sé como retribuirle todo lo que han hecho por mí y siguen haciendo.

A mi mamita linda, porque es la mejor de las madres y siempre me ha guiado por los caminos de la vida con una sonrisa y todo su cariño.

A mi papi querido, por ser mi mejor amigo, por ser la persona más cariñosa que he conocido en la vida, por brindarme su mano y sus consejos en todo momento.

A los dos por apoyarme siempre, de no ser por ustedes, hoy no sería posible este logro.

A mis queridas hermanas, por servirme de ejemplo a seguir y estar presente en todos los momentos de mi vida.

A mis tías bellas, por su eterna dedicación para conmigo.

En fin este logro se lo dedico a todas aquellas personas que confiaron en mí, y me aseguraron que yo era capaz de lograr todo lo que me propusiera en la vida.

Mayli

RESUMEN

En el presente trabajo se realiza un estudio minucioso del algoritmo de Programación Genética implementado en la plataforma *bioGRATO* para generar modelos de predicción y predecir actividad biológica en compuestos orgánicos.

Este algoritmo demanda un elevado nivel de cómputo, provocando que las respuestas demoren un tiempo prolongado, en consecuencia a esto es poco práctico realizar los cálculos para generar un modelo de predicción en un único ordenador. Para minimizar los tiempos de respuesta de este algoritmo, se estudió si sería posible una variante distribuida que haciendo uso de la Plataforma de Tareas Distribuidas (PTD), pudiera hacer los cálculos en varias computadoras conectadas a una red.

Palabras Clave: Programación Genética, predicción, actividad biológica, compuestos orgánicos, Plataforma de Tareas Distribuidas.

ÍNDICE

INTRODUCCIÓN

La química computacional es una rama de la química que utiliza computadoras para ayudar a resolver problemas, utilizando los resultados de la química teórica, incorporados en algún software para calcular las estructuras y las propiedades de moléculas y cuerpos sólidos. Estos resultados no experimentales complementan la información necesaria para predecir fenómenos no observados como la actividad biológica de un compuesto.

El desarrollo de esta nueva ciencia en los últimos años ha sido el motor impulsor de industrias como la farmacéutica, ayudando a los especialistas a reducir gastos y acortar el tiempo de investigación-desarrollo de nuevos candidatos a fármacos.

En la actualidad existen numerosos métodos experimentales para determinar la estructura molecular de una sustancia, su actividad y demás características que pudieran ser de interés para su estudio. No obstante a ello, el desarrollo alcanzado por la computación y la química computacional ha propiciado la generación de sistemas que permiten calcular todos estos datos de forma mucho más rápida y además son capaces de generar datos con una amplia aplicación en la investigación experimental, tanto para la interpretación de los resultados obtenidos y la planificación de futuros, como para deducir información no asequible experimentalmente, trayendo consigo que cada día nuestros especialistas se enfrenten a problemas cada vez más complejos, necesitando el procesamiento de grandes volúmenes de información con una elevada demanda de cómputo, que al ser corridos en una computadora demorarían los tiempos de respuesta.

En el año 2005 la Facultad de Bioinformática de la Universidad de las Ciencias Informáticas (UCI) comenzó un proyecto llamado: Plataforma Inteligente para la Predicción de Actividad Biológica de Compuestos Orgánicos. En este proyecto se desarrolló un módulo de Inteligencia Artificial que podía generar modelos de predicción de actividad biológica. A partir de esta idea se desarrollaron varios algoritmos que generaban modelos predictivos a partir de descriptores, fragmentos y la actividad biológica de cada compuesto utilizando diferentes técnicas de Inteligencia Artificial, entre ellas programación genética (PG) escogida por sus demostrados resultados frente a problemas de correlación. El algoritmo que se desarrolló a partir de esta técnica generaba modelos que predecían actividad biológica pero los volúmenes de cálculo eran tan grandes que en ocasiones el algoritmo no generaba respuesta y el usuario no podía obtener los modelos de predicción, obligándolo a trabajar con una cantidad reducida de corridas del algoritmo y con muestras pequeñas. Por lo que se plantea como **problema científico**:

¿Cómo reducir los tiempos de respuesta del algoritmo de PG para la generación de modelos predictivos de actividad biológica?

Para resolver esta situación se necesitaría obtener tecnología de alto nivel o utilizar al máximo los recursos disponibles. En este sentido, en los países del tercer mundo como Cuba, se hace difícil adquirir tecnología de punta por lo que la vía de solución más factible es la conexión de varias computadoras en red, no necesariamente ubicadas en el mismo lugar y que cooperan para resolver un problema común. En esta dirección se destacan los Sistemas de Computación Paralelos y Distribuidos, una solución que brinda alta potencia de cálculo sin necesidad de invertir grandes sumas de dinero, llevando a las grandes instituciones u organizaciones a crear su propia infraestructura de computación.

Un sistema paralelo consiste en múltiples procesadores que se comunican entre ellos, compartiendo memoria para resolver una tarea común. Para utilizar este paradigma se necesita comprender en profundidad el algoritmo secuencial, que da solución al problema para poderlo dividir en subtareas totalmente independientes, y se requieren formas de coordinar los accesos a recursos compartidos.

Un sistema distribuido es una colección de computadoras separadas físicamente y conectadas entre sí por una red de comunicaciones distribuida. Cada máquina posee sus componentes de hardware y software, que el usuario percibe como un solo sistema, ubicadas en cualquier lugar. Estos elementos se interconectan a través de la red de comunicación, cooperan entre sí con el propósito de resolver una tarea común, y se comunican mediante la recepción y el envío de mensajes actuando de forma espontánea y “autónoma”. (1) En estos sistemas es común encontrarse un servidor que monitorea los servicios para llevar a cabo todo el proceso, por lo que el modelo del sistema responde a un Modelo Cliente-Servidor.

En un sistema distribuido, el programador no deberá conocer el código del programa secuencial que se quiera compartir, para reducir los tiempos de cálculos, solo se centrará en ejecutar en los clientes las mismas instrucciones con los respectivos datos, aprovechando los recursos ya existentes. El tamaño del sistema no resultará un problema, ya que este puede ir desde las decenas de computadoras hasta los miles o millones, a esto se le llama escalabilidad.

Teniendo en cuenta que la facultad a la que pertenece el proyecto BIOGRATO posee una Plataforma de Tareas Distribuidas y la demora en obtener respuesta del algoritmo que se va a analizar se plantea como **objeto de estudio**:

- ✓ Los sistemas distribuidos para el procesamiento masivo de información.

El cual se encuentra enmarcado dentro del **campo de acción**:

- ✓ Plataforma de Tareas Distribuidas.

El **aporte práctico** esperado con la realización de este trabajo es una propuesta distribuida del algoritmo de PG de la plataforma BIOGRATO, que permita ampliar el espacio de búsqueda de los usuarios del sistema y minimizar los tiempos de respuesta del algoritmo.

Por lo que se plantea como **objetivo general:**

- ✓ Diseñar una propuesta de algoritmo distribuido de Programación Genética para la generación de modelos de predicción de actividad biológica.

A partir de este objetivo general se desglosaron los siguientes **objetivos específicos**:

- ✓ Analizar la propuesta del algoritmo de PG distribuido.
- ✓ Diseñar la propuesta del algoritmo de PG distribuido.
- ✓ Simular la propuesta del algoritmo de PG distribuido.
- ✓ Analizar los tiempos de respuesta de la simulación del algoritmo distribuido.

Para dar cumplimiento a los objetivos específicos se definieron las siguientes **tareas**:

- ✓ Estudio del algoritmo de programación genética.
- ✓ Análisis del estado del arte de los sistemas distribuidos.
- ✓ Estudio de la plataforma de tareas distribuidas
- ✓ Generación de modelos con muestras de la base de datos de la plataforma BIOGRATO.
- ✓ Estimar los tiempos de respuesta del algoritmo distribuido.

Importancia de la investigación

A pesar de la diversidad en cuanto a sistemas operativos y arquitectura de hardware que pueden llegar a tener las computadoras presentes en una institución, es de gran necesidad contar con un sistema de cómputo que haga uso útil de los distintos recursos computacionales ya sean máquinas dedicadas exclusivamente a un clúster o un conjunto de estaciones de trabajo distribuidas físicamente por la institución que trabajen en conjunto para darle solución a un problema común.

La simulación de un sistema de cómputo distribuido para el algoritmo de PG que se presenta en este trabajo va a permitir utilizar un número mayor de computadoras para dar solución a este problema que demanda de cómputo intenso acelerando la generación de los modelos de predicción y acortando la espera de los usuarios de la Plataforma.

El siguiente trabajo quedará estructurado de la siguiente forma:

Capítulo 1: Fundamentación teórica.

En este capítulo se realiza un estudio sobre los algoritmos evolutivos, haciendo mayor énfasis en la programación genética. Se realiza una investigación sobre los sistemas distribuidos, su aplicación en la rama de la Bioinformática, se muestran ventajas y desventajas de estos sistemas y se detallan los componentes principales de un sistema distribuido y su interacción.

Capítulo 2: Materiales y métodos.

En este capítulo se presentan algunos elementos importantes, como son el lenguaje programación utilizado para desarrollar la aplicación, el entorno de desarrollo en que se realizó, se exponen las características de la Plataforma de Tareas Distribuidas (PTD) y sus componentes, así como el tipo de muestras con las que se simuló la distribución del algoritmo y las características de los ordenadores donde se corrió la aplicación.

Capítulo 3: Resultados y discusión.

En este capítulo se describe brevemente el funcionamiento del algoritmo de programación genética y el inconveniente que representa correr este algoritmo en un solo ordenador, a partir de este problema se plantea la solución propuesta y se presentan algunos resultados experimentales realizados con la

aplicación desplegada en un ordenador, luego en varios ordenadores simulando una distribución para que a modo de comparación se puedan apreciar las diferencias existentes en los tiempos de respuesta de cada una de las variantes.

CAPITULO 1: Revisión bibliográfica

En este capítulo se realiza un estudio sobre los algoritmos evolutivos de forma general, profundizando en la Programación Genética que es el paradigma de algoritmo evolutivo que utiliza el módulo que se quiere distribuir. Se describe como la utilización de sistemas distribuidos representa una solución que aumenta las potencialidades de cómputo de cualquier institución, haciendo un mejor uso de los recursos de una red.

1.1 Algoritmos evolutivos.

Este término es empleado para describir sistemas de resolución de problemas de optimización o búsqueda basados en el ordenador empleando modelos computacionales de algún mecanismo de evolución conocido como elemento clave en su diseño e implementación.

Los Algoritmos Evolutivos (AE) trabajan con una población de individuos, que representan soluciones candidatas a un problema. Esta población se somete a ciertas transformaciones y después a un proceso de selección, que favorece a los mejores. Cada ciclo de transformación y selección constituye una generación, de forma que después de cierto número de generaciones se espera que el mejor individuo de la población esté cerca de la solución buscada. Los algoritmos evolutivos combinan la búsqueda aleatoria, debido a las transformaciones que sufre la población, con una búsqueda dirigida dada por la selección. (2)

Principales Componentes:

- ✓ Población de individuos, que son una representación (no necesariamente directa) de posibles soluciones.
- ✓ Procedimiento de selección basado en la aptitud de los individuos para resolver el problema.
- ✓ Procedimiento de transformación para construir nuevos individuos a partir de los anteriores.

Ventajas de los algoritmos evolutivos.

Es importante destacar las diversas ventajas que presenta el uso de los algoritmos evolutivos para resolver problemas de búsqueda y optimización: (3)

- ✓ Simplicidad Conceptual.

- ✓ Amplia aplicabilidad.
- ✓ Superiores a las técnicas tradicionales en muchos problemas del mundo real.
- ✓ Tienen el potencial para incorporar conocimiento sobre el dominio y para hibridarse con otras técnicas de búsqueda/optimización.
- ✓ Pueden explotar fácilmente las arquitecturas en paralelo.
- ✓ Son robustas a los cambios dinámicos.

Es importante mencionar que los algoritmos evolutivos, como disciplina de estudio, ha atraído la atención de un número cada vez mayor de investigadores de todo el mundo.

Esta popularidad se debe, en gran medida, al enorme éxito que han tenido los algoritmos evolutivos en la solución de problemas del mundo real de gran complejidad. De tal forma, es de esperarse que en los años siguientes el uso de este tipo de técnicas prolifere aún más.

Los AE se clasifican en:

Estrategias Evolutivas: Técnica desarrollada por Rechenberg y Schwefel y extendida por Herdy, Kursawe, Ostermeier, Rudolph, y otros, fue diseñada inicialmente con la meta de resolver problemas de optimización discretos y continuos, principalmente experimentales y considerados difíciles. Trabaja con vectores de números reales con desviaciones estándar que codifican las posibles soluciones de problemas numéricos. Utiliza recombinación o cruce (crossover aritmético), mutación y la operación de selección, ya sea determinística o probabilística, elimina las peores soluciones de la población y no genera copia de aquellos individuos con una aptitud por debajo de la aptitud promedio. (2)

Programación Evolutiva: Técnica introducida por Fogel y extendida por Burgin, Atmar y otros, inicialmente fue diseñada como un intento de crear Inteligencia Artificial.

La representación del problema se realiza mediante números reales (cualquier estructura de datos), y emplea los mecanismos de mutación y selección. El procedimiento es muy similar a las estrategias evolutivas con la diferencia de que no emplea la recombinación, de tal forma que son denominadas en conjunto algoritmos evolutivos como una manera de diferenciarlas de los algoritmos genéticos. (2)

Algoritmos Genéticos: Modelan el proceso de evolución como una sucesión de frecuentes cambios en los genes, con soluciones análogas a cromosomas. Trabajan con una población de cadenas binarias para la representación del problema, y el espacio de soluciones posibles es explorado aplicando transformaciones a éstas soluciones candidatas tal y como se observa en los organismos vivientes: cruce, inversión y mutación. Como método de selección emplean el mecanismo de la ruleta (a veces con elitismo). Constituyen el paradigma más completo de la computación evolutiva ya que resumen de modo natural todas las ideas fundamentales de dicho enfoque. Son muy flexibles ya que pueden adoptar con facilidad nuevas ideas, generales o específicas, que surjan dentro del campo de la computación evolutiva. Además, se pueden hibridar fácilmente con otros paradigmas y enfoques, aunque no tengan ninguna relación con la computación evolutiva. Se trata del paradigma con mayor base teórica. (2)

Programación Genética: Esta técnica fue creada por John Koza a finales de los 80's, culminando con la publicación de su libro titulado "Genetic Programming: on the Programming of Computers by Means of Natural Selection" (1992). Koza propuso, por medio de esta extensión de lo que originalmente es un algoritmo genético, un método para la evolución de estructuras más complejas como pueden ser estructuras de programas de computadoras o funciones matemáticas. Sobre este paradigma de algoritmo evolutivo se hablará de forma más profunda en el próximo epígrafe. (4) (5)

1.2 Programación Genética.

La Programación Genética (PG) surge como evolución de los algoritmos genéticos (AGs), John R. Koza es quien acuña este término en el año 1962 por lo que es considerado el padre de la Programación Genética. (6)

La PG es un paradigma de algoritmo evolutivo que consiste en la evolución automática de programas usando ideas basadas en la selección natural de Darwin. Además de su utilización para generar programas, también se ha utilizado para generar un amplio rango de soluciones, como fórmulas matemáticas, circuitos electrónicos, entre otros.

El hecho de que muchos problemas prácticos de diferentes dominios de aplicaciones puedan ser formulados como un problema de determinación de un "individuo solución" que produzca una salida deseada cuando se tienen presentes ciertas entradas particulares, hace a la Programación Genética una novedosa línea de investigación.

Esta técnica ha demostrado su capacidad para resolver problemas donde la interrelación entre las variables es desconocida; donde una solución aproximada es aceptable; pero sobre todo donde son muy valoradas las posibles pequeñas mejoras obtenidas y a pesar de su corta edad ya se han reportado resultados de gran envergadura demostrado gran eficiencia en la resolución de problemas complejos en muchas áreas, llegándose a obtener mejores soluciones que las obtenidas por los propios científicos. (6)

1.2.1 Elementos básicos de la Programación Genética.

En la aplicación de la PG a un problema, hay cinco pasos preparatorios importantes a definir (7):

1. El conjunto de terminales.

2. El conjunto de funciones primitivas.

3. La medida de la aptitud.

4. Los parámetros para controlar la corrida.

5. El método para designar un resultado y el criterio para terminar la ejecución del programa.

El conjunto de terminales.

Consiste en identificar el conjunto de terminales. Los terminales son las hojas de los árboles que corresponden a variables o a valores constantes, se pueden ver como las entradas al programa. El conjunto de terminales (junto con el conjunto de funciones) son los ingredientes a partir de los cuales la PG, trata de construir un programa de computador para solucionar total, o parcialmente, un problema.

El conjunto de funciones primitivas.

Consiste en identificar el conjunto de funciones que se van a usar, para generar la expresión matemática que trata de satisfacer la muestra dada finita de datos.

En el conjunto de funciones permitidas pueden incluirse:

- ✓ Operaciones aritméticas (+, -, *, etc.).
- ✓ Funciones matemáticas (seno, coseno, log, etc.).

- ✓ Operadores boléanos (and, or, not, etc.).
- ✓ Operadores condicionales (if-then-else).
- ✓ Funciones que causen iteración (do-until).
- ✓ Funciones que causen recursión.

Una solución (árbol de análisis gramatical) es una composición de funciones del conjunto F de funciones y terminales del conjunto T de terminales.
Cada una de las funciones del conjunto F debe ser capaz de aceptar, como sus argumentos, cualquier valor y tipo de dato que pueda ser retornado por cualquier función del conjunto de funciones, y cualquier valor y tipo de dato que pueda tomar por cualquier terminal del conjunto T.

La medida de la aptitud.
La medida de aptitud nos permite evaluar cómo de bien funciona cada programa de la población en el entorno del problema. Esta medida debe estar completamente definida, debe ser capaz de evaluar cualquier programa de la población. La naturaleza de la medida de aptitud varía con el problema.

Los parámetros para controlar la corrida.
Normalmente, los parámetros que controlan la ejecución de PG son: el tamaño de la población; el número de generaciones; las probabilidades de aplicación de los operadores genéticos de cruce y mutación; y la estrategia de selección de los individuos para la nueva generación.

El método para designar un resultado y el criterio para terminar la ejecución del programa.
La designación del resultado se hace generalmente utilizando el criterio de quedarse con el mejor individuo hasta el momento, es decir se elige el mejor individuo obtenido de las generaciones de una corrida de PG.
El criterio de finalización debe indicar cuándo finalizar la ejecución del algoritmo de PG, lo cual puede ocurrir cuando transcurra el número de generaciones predeterminadas, o bien se halle el individuo que sea solución según la medida de aptitud que se ha proporcionado.

1.2.2 Algoritmo general de la Programación Genética.

La PG incuba programas de computador para la solución de problemas, ejecutando los siguientes pasos: (7)

1. Generar una población inicial de composiciones aleatorias de funciones y terminales del problema (es decir, programas, funciones, árboles de decisión, etc.).

2. Ejecutar iterativamente los siguientes sub-pasos hasta que se satisfaga el criterio de terminación:

   a. Ejecutar cada programa de la población y asignarle un valor de aptitud, de acuerdo a su comportamiento frente al problema.

   b. Crear una nueva población de programas aplicando las siguientes dos operaciones primarias a los programas escogidos, con una probabilidad basada en la aptitud.

   i. Reproducir un programa existente copiándolo en la nueva población.

   ii. Crear dos programas a partir de dos programas existentes, recombinando genéticamente partes escogidas de los dos programas en forma aleatoria, usando la operación cruce aplicada a un punto de cruce, escogido aleatoriamente dentro de cada programa.

   iii. Crear un programa a partir de otro seleccionado aleatoriamente, cambiando aleatoriamente un gen (función o terminal). Es posible que se requiera un procedimiento de reparación para que se satisfaga la condición de clausura.

3. El programa identificado con la mayor aptitud (el mejor hasta la última generación), se designa como el resultado de la corrida de PG. Este resultado puede representar una solución (o una solución aproximada) al problema.

Especificaciones del algoritmo aplicado al problema que se desea resolver, quedaría expuesto en los siguientes pasos:

El algoritmo comienza su funcionamiento generando una población aleatoria inicial de individuos (árboles) que estará compuesto por símbolos terminales (variables y constantes) y por símbolos no terminales (funciones).

Mientras no se cumpla el criterio de terminación:

- ✓ Ejecuta cada individuo de la población generada y obtiene su aptitud.
- ✓ Selecciona los individuos (para reproducción y eliminación), considerando la aptitud de cada uno de ellos.
- ✓ Crea nuevos individuos utilizando operadores genéticos (cruce, mutación, reproducción)

Finalmente devuelve el mejor individuo o mejor conjunto de individuos de la población final según sea el caso.

1.2.3 Generación de la población inicial.

La población inicial en la PG va a estar compuesta por individuos representados por expresiones, donde cada expresión se obtiene generando aleatoriamente un árbol rotulado en cada uno de sus puntos.

Este proceso comienza cuando se selecciona una de las funciones en el conjunto F al azar. Cuando este punto del árbol es rotulado con esta función se dibujan entonces z(f) líneas que se irradian desde ese punto donde z(f) es la aridad de F. Por cada línea, se seleccionará un elemento del conjunto de funciones y de terminales al azar para ser el punto final de esa línea. Si se selecciona una función como rótulo para cualquiera de estos puntos finales, el proceso de generación sigue recursivamente como se hizo anteriormente. Si se selecciona un terminal como rótulo para cualquiera de estos puntos finales entonces ese punto se convierte en hoja del árbol y culmina el proceso de generación para ese punto.

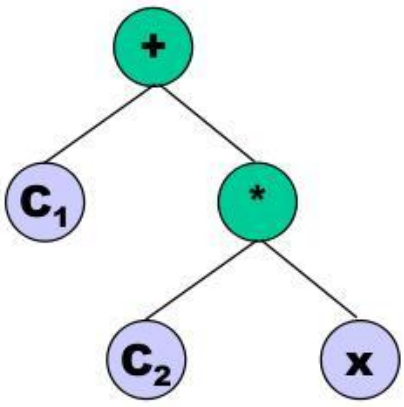


Figura 1. Ejemplo de un individuo generado

El tamaño de los arboles va a variar, pero va a ser menor que un límite permitido. Este límite es tanto en profundidad como en el número de nodos máximo de cada árbol, desechando todos aquellos que lo excedan.

1.2.4 Operadores genéticos.

En la programación genética se utilizan una serie de operadores genéticos cuando se necesita generar nuevos individuos a partir de individuos ya existentes Entre estos operadores se encuentran:

- ✓ Cruce(Crossover)
- ✓ Mutación
- ✓ Reproducción

Cruce

El operador de cruzamiento (crossover) combina el material genético de individuos intercambiando pedazos de dos progenitores para producir dos descendientes. Si los individuos se representan como árboles, se elige al azar un nodo de cada árbol y luego se intercambian los subárboles bajo estos nodos. (8)

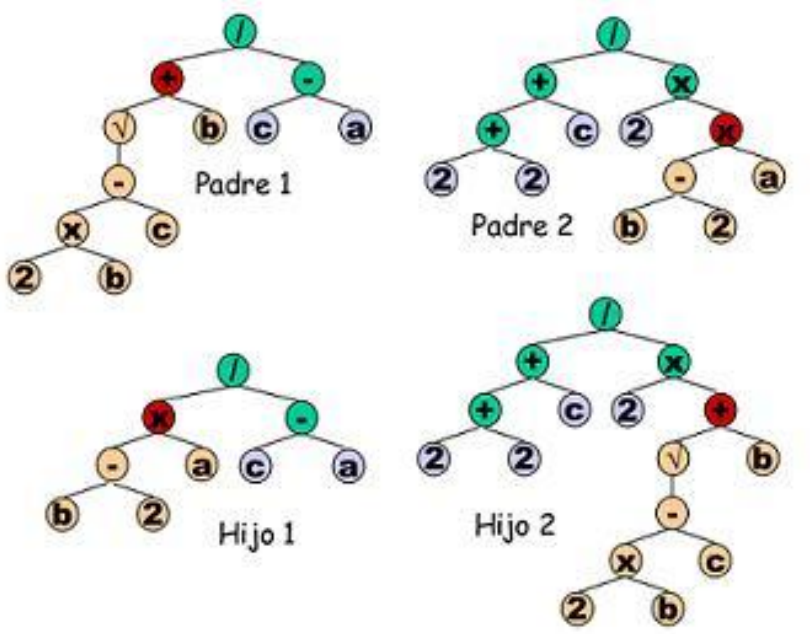


Figura 2. Ejemplo de cruce

Mutación

Actúa sobre un solo individuo, generalmente, uno resultante de un cruce. La mutación actúa con una probabilidad, en general, muy baja y que puede ser definida por el usuario del algoritmo. Un tipo de mutación consiste en escoger en forma aleatoria un nodo del árbol y generar bajo este otro subárbol. (8)

Existen otros tipos de mutaciones, como por ejemplo:

- ✓ Mutación puntual: Donde un solo nodo se intercambia por otro de la misma clase.
- ✓ Permutación: Donde los argumentos de un nodo son permutados.
- ✓ Mutación de subárbol: Donde un subárbol se reemplaza por otro generado al azar. (8)

Reproducción

Esta operación, actúa sobre un solo individuo a la vez y consiste en dos pasos muy simples. Primero, un único individuo se selecciona de la población utilizando algún método de selección, luego se copia sin alteración desde la población actual, hacia la nueva generación.

1.2.5 Métodos de selección.

Existen diversos métodos de selección que se pueden utilizar durante la selección de progenitores y de sobrevivientes. Algunos de ellos son:

Selección elitista: se garantiza la selección de los k miembros más aptos de cada generación.

Selección proporcional a la aptitud: los individuos más aptos tienen más probabilidad de ser seleccionados, pero no la certeza.

Selección por ruleta por bondad: una forma de selección proporcional a la aptitud en la que la probabilidad de que un individuo sea seleccionado es proporcional a la diferencia entre su aptitud y la de sus competidores. Conceptualmente, esto puede representarse como un juego de ruleta donde el individuo obtiene una sección de la ruleta, pero los más aptos obtienen secciones mayores que las de

los menos aptos. Luego la ruleta se hace girar y en cada ocasión se elige al individuo que posea la sección en la que se asiente la bola que viaja por la ruleta.

Selección por ruleta por orden: los individuos se ordenan según su aptitud, y los números de orden resultantes se utilizan como probabilidades de selección para formar la muestra. La selección se basa en este ranking, en lugar de las diferencias absolutas en aptitud. La ventaja de este método es que puede evitar que individuos muy aptos ganen dominio al principio a expensas de los menos aptos, que reduciría la diversidad genética de la población y podría obstaculizar la búsqueda de una solución aceptable.

Selección por torneo: se eligen subgrupos de individuos de la población, y los miembros de cada subgrupo compiten entre ellos. Sólo se elige un individuo de cada subgrupo para la reproducción.

Selección por torneo binario: un caso particular del método de selección anterior, en el cual se eligen aleatoriamente *k* pares de individuos de la población base, y se constituye la muestra seleccionando el mejor de cada par. (6)

1.3 Descriptores.

Los métodos QSAR han demostrado que las relaciones entre la estructura molecular y las propiedades físico-químicas de los compuestos se pueden cuantificar matemáticamente a partir de parámetros estructurales simples, conocidos como descriptores.

Los descriptores pueden ser clasificados en:

- ✓ Índices basados en propiedades químico-físicas: Son aquellos, como lo dice su nombre, que se derivan de la determinación experimental de una determinada propiedad químico-física de la sustancia.
- ✓ Índices mecánico-cuánticos: Son aquellos que se determinan por cálculos mecánico-cuánticos.
- ✓ Índices topológicos y topográficos: Son aquellos que se determinan a partir del grafo químico, o sea de la estructura química. (9)

Al lograr representar de la misma forma la estructura y la actividad en un compuesto, la generación de un modelo QSAR se reduce a establecer una correlación entre dos conjuntos de números, relacionados por una expresión algebraica (un conjunto de números representan las propiedades y el

otro representa las estructuras moleculares de las moléculas que se estudian). El modelo obtenido a partir del algoritmo que se está analizando es un ejemplo de ello, se busca la relación existente entre la actividad biológica de un compuesto y los descriptores que describen numéricamente sus características estructurales, químico-cuánticas y/o físico-químicas.

A pesar de la investigación teórica y experimental en este campo, no existe acuerdo acerca de aquel conjunto de descriptores óptimo, y dado que diferentes descriptores codifican distinta información, la estrategia consiste en aplicar aquellos más relevantes según la particularidad del caso en estudio. (10)

En el cálculo y selección de descriptores existe básicamente un compromiso entre su eficacia y eficiencia. La eficacia se entiende como la bondad de un descriptor en términos de diferenciar entre moléculas diferentes, mientras que la eficiencia hace referencia a la velocidad de cálculo asociada al descriptor (10). Este tema solo es analizado para ubicar al lector y que pueda entender la naturaleza y la información que aportan los datos de entrada del algoritmo.

1.4 Sistemas Distribuidos.

El diseño de algoritmos es una parte fundamental en las tecnologías de la información, como se ha visto reflejado por el gran número de libros y artículos relacionados con el tema. La gran mayoría de los cuales, tratan los algoritmos en el contexto de una programación secuencial, donde el flujo de control va siempre en una misma dirección y en cada paso del algoritmo se realiza una única acción. El gran auge del concepto de redes de comunicación, junto con los avances en la metodología de programación ha conseguido que el concepto de comunicación y de diseño de algoritmos distribuidos surja como un nuevo aspecto en las técnicas de desarrollo del software.

La computación desde sus inicios ha sufrido muchos cambios, desde los grandes ordenadores que permitían realizar tareas en forma limitada y de uso un tanto exclusivo de organizaciones muy selectas, hasta los actuales ordenadores ya sean personales o portátiles que tienen las mismas y eventualmente mayores capacidades que los primeros los cuales están cada vez más introducidos en el quehacer cotidiano de una persona.

Los mayores cambios se atribuyen principalmente a dos causas, que se dieron desde las décadas de los setenta:

- ✓ El desarrollo de los microprocesadores, que permitieron reducir en tamaño y costo a los ordenadores y aumentar en gran medida las capacidades de los mismos y su acceso a más personas.

- ✓ El desarrollo de las redes de área local y de las comunicaciones que permitieron conectar ordenadores con posibilidad de transferencia de datos a alta velocidad.

Es en este contexto aparece el concepto de Sistemas Distribuidos (SD) que son definidos como:

Una colección de computadores autónomos conectados por una red, y con el software distribuido adecuado para que el sistema sea visto por los usuarios como una única entidad capaz de proporcionar facilidades de computación. (11)

El primer logro en este sentido fue en los años 80 y el marco que intentaba definir la forma de integrar diversos elementos que posibilitaran dicha solución se denominó Entorno de Informática Distribuida (DCE). DCE brindaba una infraestructura completa para la implementación de sistemas que son el resultado de ejecutar funciones en un conjunto de nodos distribuidos en una red local o global. Este marco se denominó middleware o tecnología posibilitadora debido a que estaba en un nivel intermedio entre el sistema operativo y la aplicación distribuida que se ejecutaba. El organismo que regulaba este conjunto de componentes era el Open Software Foundation, que luego se convirtió en el Open Group y tenía como objetivo brindar los servicios necesarios para llevar a cabo la implementación de los sistemas distribuidos.

Los SD están concebidos para fortalecer la capacidad de cálculo en una red de computadoras y tienen como principio dividir las aplicaciones en sub-tareas que son resueltas concurrentemente, aunque no suelen ser completamente rápidos, sus elementos están diseñados para actuar de forma independiente, por lo que si falla uno, no implica que falle el sistema completo, estos pueden estar separados por centenares de metros o kilómetros.

Permiten además, la aparición de recursos o su desaparición, incluso en tiempo de ejecución. Las instrucciones se pueden ejecutar en arquitecturas heterogéneas, y se admiten varias aplicaciones a la vez. (12)

1.4.1 Ventajas y desventajas de los sistemas distribuidos.

Los sistemas distribuidos tienes diversas ventajas que permiten su utilización en cualquier institución ya que brinda una solución que aumenta las potencialidades de cómputo y tiempo de respuestas cuando se trabaja con grandes volúmenes de información. Esto se evidencia en la eficiencia y flexibilidad que brindan estos sistemas, dando respuestas rápidas, permitiendo la ejecución concurrente de procesos en varias computadoras y el empleo de técnicas de procesamiento distribuido. Entre otras ventajas se pueden citar:

- ✓ Balanceo en la carga de trabajo: Son capaces de balancear la carga de trabajo entre varias computadoras conectadas en red.
- ✓ Confiabilidad, disponibilidad y tolerancia a fallas: Son más confiables porque un fallo en un nodo no tiene porque representar un fallo en todo el sistema
- ✓ Modularidad y heterogeneidad: Los sistemas distribuidos muestran mayor flexibilidad.
- ✓ Crecimiento incremental: El tamaño del sistema puede incrementar a medida que se añadan nodos a la red en la que se está distribuyendo.
- ✓ Reducción de tiempos: La respuesta puede obtenerse de una forma más rápida al ser distribuido el trabajo entre varios nodos.
- ✓ Bajo costo: El precio de los ordenadores de consumo y la disponibilidad de redes hacen que crear un sistema distribuido sea mucho más barato.

Sin embargo es de conocimiento que estos sistemas poseen ciertas dificultades entre las que figuran requerimientos de mayor control de procesamiento y administración más compleja. Se pueden apreciar otras desventajas como:

- ✓ Uso ineficiente de los recursos distribuidos.
- ✓ Capacidad reducida para administrar apropiadamente grupos de procesadores y memoria localizada en distintos sitios.
- ✓ Enorme dependencia del desempeño de la red y de la confiabilidad de la misma.

- ✓ Debilitamiento de la seguridad.
- ✓ Mayor complejidad en su construcción.

1.4.2 Aplicación de los sistemas distribuidos a la resolución de problemas en la Bioinformática.

El desarrollo de la Bioinformática ha ido estrechamente relacionado con el de las tecnologías experimentales y esta última ha ido pareja a su popularidad en el ámbito científico y sobre todo a su repercusión e impacto social y económico. Este desarrollo supone un nivel de complejidad adicional ya que cada día los científicos se enfrentan a problemas mucho más complejos, sustentado sus investigaciones con la ayuda de algoritmos que necesitan el procesamiento de enormes cantidades de datos para arribar a una respuesta. El reto actual de todas las áreas de la Bioinformática es responder a este cambio en la demanda de las necesidades. Los cambios han sido progresivos y a medida que los métodos de resolución en uso se iban quedando cortos, esta se centra en buscar soluciones a ello.

El primer paso consistió en la instauración de sistemas de clústers y adaptación de algoritmos y procedimientos para su ejecución paralela a pequeña escala. Estas soluciones, si bien permiten desarrollar y probar nuevos algoritmos y soluciones, resultan insuficientes para el tratamiento de datos experimentales masivos. Las necesidades básicas se pueden expresar como una mayor demanda de capacidad de almacenamiento y tratamiento de información, y un gran crecimiento de la capacidad de cálculo. En estas condiciones, la única solución viable consiste en la compartición de recursos entre grupos. Los primeros resultados de estos esfuerzos están poniendo de manifiesto un problema adicional: la comprensión de las grandes cantidades de información disponibles empieza a precisar de nuevos sistemas de minería de datos.

La forma más efectiva en costo y recursos para resolver los problemas actuales y por extensión responder a las crecientes demandas sociales, es recurrir al uso de sistemas distribuidos

La aplicación de estos sistemas en la Bioinformática no se justifica solamente por el crecimiento de las demandas computacionales. Precisamente por sus características, los problemas tratados se proyectan perfectamente sobre una estructura computacional distribuida.

Estas características son fundamentalmente dos:

- ✓ Problemas gruesos: la mayoría de los problemas son divisibles en grandes subprocesos de intensa demanda computacional y ejecutable de forma independiente y en paralelo.

- ✓ Experimentos de muy alto rendimiento: las grandes colecciones de datos demandan un tratamiento uniforme (análisis de estructuras, etc.) y por tanto son tratables como una enorme colección de problemas independientes.

1.4.3 Características de un problema para poder ser distribuido.

Para lograr la distribución de un problema que requiera grandes prestaciones de cómputo hay que tener en cuenta una serie de características que debe tener todo problema para encontrarle una buena solución distribuida. La característica más importante que debe tener es la posibilidad de dividir el problema en unidades más pequeñas que puedan ser procesadas de forma individual. En este sentido existen dos formas de dividir el problema (13):

- ✓ Particionamiento de los datos o descomposición del dominio. En este particionado, el centro de atención es la partición de los datos. De ser posible, estos son separados en pequeñas partes de aproximadamente el mismo tamaño. Luego, se particionan los cálculos a ser realizados, asociándolos con los datos sobre los cuales actúan.

- ✓ Particionamiento funcional o descomposición funcional. La descomposición funcional es una estrategia diferente para enfrentar los problemas. El punto principal en este esquema es el cómputo a realizar, en lugar de los datos manipulados. Si uno tiene éxito dividiendo las funciones del programa en grupos disjuntos, luego puede proceder a estudiar los requerimientos de datos de estos grupos funcionales. Si se da el caso de que la partición de datos resultante es también disjunto, se dice que la partición funcional es completa. En este caso se procede al revés que en el caso anterior. Aquí se divide primero el cómputo y luego se evalúa la posibilidad de descomponer los datos.

A partir de que se compruebe que el problema puede ser dividido de una de las dos formas expuestas anteriormente, se procede a la implementación del sistema.

El sistema esta basado en el modelo Cliente-Servidor .Los componentes esenciales son el servidor, el cliente y la interfaz de administración. La función del servidor es almacenar los datos y el algoritmo que

va a procesarlos, dividir el problema en sub-problemas llamados unidades de trabajo, y repartir las unidades de trabajo entre los clientes, siempre que estos se reporten como disponibles. Por otra parte, el cliente será el encargado de realizar la petición de una unidad de trabajo al servidor para procesar los datos, y una vez concluida la tarea se encargará de devolver el resultado al servidor para volver a pedir una nueva unidad de trabajo. La interfaz de administración se encargará de gestionar las funcionalidades del servidor como: administración de cuentas (adicionar y/o eliminar usuarios, grupos, etc.), administración de problemas, así como su ejecución y monitoreo; obtener los resultados de las ejecuciones finalizadas y configuración del servidor, entre otras.

En el análisis realizado se evaluaron algunos métodos para predecir actividad biológica, como la programación genética, que es la técnica a utilizar en este caso, exponiendo los componentes principales de la misma así como el funcionamiento del algoritmo de manera general, lo cual permitió valorar la complejidad del mismo para la realización de tratamiento de grandes volúmenes de datos. Esto condujo a su vez al estudio de los sistemas distribuidos como posible solución a estos problemas, los cuales demandan altos niveles de cómputo.

CAPÍTULO 2: Programas y metodologías

En este capítulo se exponen las características de las herramientas y metodologías empleadas en este trabajo. Se describe la herramienta de Programación Genética que hace uso de ese algoritmo que se desea distribuir, así como el lenguaje de programación y entorno de desarrollo en que fue desarrollada. Se presenta la Plataforma de Tareas Distribuidas como alternativa actual para resolver problemas que demanden gran potencia de cálculo así como sus componentes principales y características. Se hace una breve descripción de los recursos computacionales utilizados, las características de la red existente en la institución, y un breve análisis de las muestras utilizadas para ejecutar el algoritmo.

2.1 Herramienta de Programación Genética.

La herramienta que se desarrolló es una aplicación de escritorio, basada en el estándar de ventanas. El tipo de letra que se utilizó para su diseño es MS San Serif de estilo regular y tamaño 11. Los botones tienen las mismas dimensiones y los íconos utilizados son los usados en la plataforma. El sistema cuenta con una barra de menú en la parte superior izquierda de la ventana, con las funcionalidades que brinda la aplicación. El diseño es sencillo y fácil de usar con el objetivo de que el usuario no se desoriente dentro de la herramienta.

Fue concebida para el uso de cualquier persona con conocimientos mínimos de computación y con conocimientos apropiados en química que le permitan entender los resultados que arroje el sistema.

El especialista, luego de ejecutar la aplicación, debe suministrar una muestra de compuestos y definir los parámetros de entrada de la aplicación y el sistema como respuesta, le devolverá los modelos matemáticos que describen entre la muestra entrada por el usuario y cierta actividad biológica que se esté estudiando. Con estos modelos se podrá predecir en muestras que aun no han sido analizadas, si está presente o no dicha actividad.

Para el uso de la herramienta se debe disponer de sistemas operativos Linux, Windows 95 o superior. Debe tenerse instalado el Java Runtime Environment (JRE) versión 1.5 o superior. Como requisitos mínimos para el uso de la herramienta las máquinas deben tener:

- ✓ Procesador Pentium 3 o superior.
- ✓ 256 Mb de RAM.

- ✓ 50 Mb de espacio en disco duro.

2.2 JAVA como lenguaje de programación.

Java es un lenguaje de programación orientado a objetos, de plataforma independiente que fue desarrollado por la compañía Sun Microsystems con la idea original de usarlo para la creación de páginas web. A continuación se hará una breve explicación de las razones que se tuvieron en cuenta para elegir este lenguaje para el desarrollo de la aplicación.

Java tiene muchas similitudes con el lenguaje C y C ++, aunque es mucho más simple y de rápido aprendizaje. Fue diseñado orientado a objetos desde el principio. Es distribuido debido a que proporciona una colección de clases para su uso en aplicaciones de red que permiten establecer y aceptar conexiones con servidores o clientes remotos. Fue diseñado para crear software altamente fiable, pues para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución, dada su naturaleza distribuida, donde las applets se bajan desde cualquier punto de la red.

Como la seguridad se impuso como una necesidad de vital importancia, se implementaron barreras de seguridad en el lenguaje y en el sistema de ejecución en tiempo real. Una de las características de este lenguaje que lo hace muy importante es que es indiferente a la arquitectura, ya que está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, sobre arquitecturas distintas y con sistemas operativos diversos. Para lograr esto, se creó un formato intermedio diseñado para transportar el código eficientemente a múltiples plataformas, hardware y software.

Es un lenguaje muy extendido por el mundo y cada vez cobra más importancia tanto en el ámbito de Internet como en la informática en general. Es portable, multihilo, dinámico y permite realizar aplicaciones para todo tipo de entornos, ya sea para Web, dispositivos móviles, aplicaciones de escritorio, servidor, etc.

2.3 Entornos de desarrollo.

Los IDEs (Integrated Development Environment) son un conjunto de herramientas para el programador, que suelen incluir en una misma suite, un buen editor de código, administrador de proyectos y archivos, enlace transparente a compiladores y debuggers e integración con sistemas controladores de versiones o repositorios.

En el uso del lenguaje de programación Java para el desarrollo de aplicaciones se encuentran diversos entornos de desarrollo integrado tales como NetBeans, JBuilder, JDeveloper, BlueJ y Eclipse. Cada uno de ellos posee sus ventajas y desventajas, contribuyendo siempre al enriquecimiento del propio lenguaje.

2.3.1 Eclipse

Se utilizó como entorno de desarrollo Eclipse, debido a que este posee una serie de ventajas y aplicaciones que lo hacen el entorno de desarrollo ideal para implementar la solución propuesta. Entre estas características se pueden mencionar:

- ✓ Integra diferentes aplicaciones.
- ✓ Utiliza JAVA como lenguaje de programación, aunque también permite, a través de la incorporación de plug-ins, programar para otros lenguajes.
- ✓ Soporta la programación orientada a objetos, la depuración y compilación de código.
- ✓ Con su uso, la implementación de aplicaciones en Java resulta sencilla.
- ✓ Es una herramienta de código abierto y se ejecuta en una gran cantidad de sistemas operativos.
- ✓ Es soportado por múltiples arquitecturas y posee capacidad de control de versiones con Subversión.

2.4 Plataforma de Tareas Distribuidas.

El funcionamiento de la Plataforma de Tareas Distribuidas (PTD) se realiza sobre una arquitectura Cliente-Servidor, en la que las computadoras que se comunican durante el cómputo para procesar los

datos representan a los clientes, y el computador donde está montada la plataforma constituye el servidor, el que cual se encarga de monitorear todo el proceso.

Para poder definir un cálculo distribuido sobre esta plataforma, sólo se requiere heredar de dos clases de Java, la clase DataManager y la clase Task, en las cuales el desarrollador debe redefinir diferentes métodos para lograr establecer su aplicación distribuida.

2.4.1 Clase DataManager.

La clase DataManager pertenece al servidor y su propósito es generar todas las unidades de trabajo que serán enviadas a los clientes, procesar los resultados, supervisar y ajustar el tamaño de las unidades, brindar información del estado de la ejecución del problema en un momento determinado y encargarse además, de cerrar todos los recursos utilizados una vez concluido el trabajo distribuido.

La clase padre DataManager consta de una serie de métodos que se deben redefinir por el desarrollador para que el problema sea aceptado por el sistema. Una vez ejecutado el mismo se le asigna un directorio que podrá usar durante el cómputo.

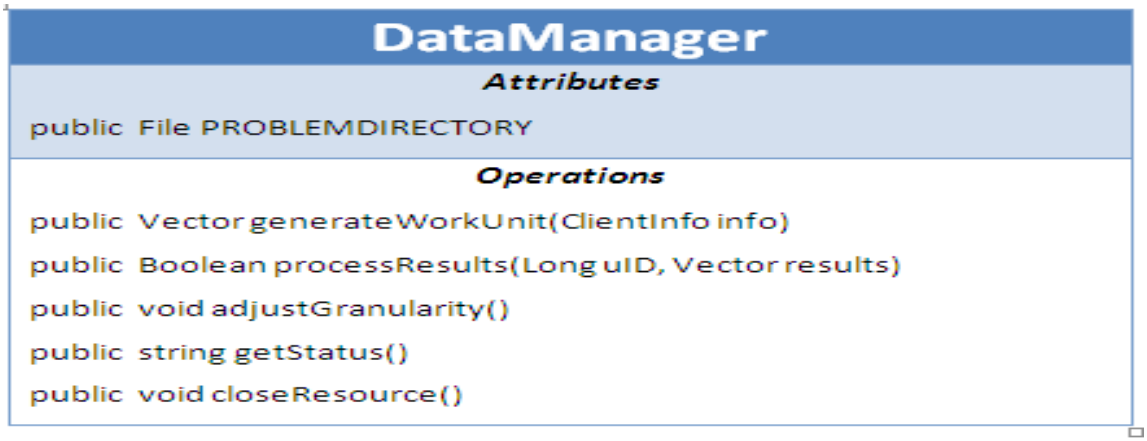


Figura 3: UML de la clase DataManager

Este directorio se utiliza para crear archivos, almacenar datos temporalmente, datos relacionados con la ejecución y el historial de resultados. Se establece en el servidor cuando se ejecuta un problema y se accede a él a través de la constante PROBLEMDIRECTORY que se encuentra en la clase DataManager.

Al finalizar el cálculo, este directorio se comprime y se coloca a disposición del usuario para que pueda facilitar su descarga. Los métodos de la clase DataManager deben tener un tiempo de ejecución breve, debido a que cualquier tarea en el DataManager que se demore mucho deberá ser implementada a través de un hilo.

El programador debe implementar un constructor por defecto al heredar de la clase DataManager. Cualquiera de las excepciones incapturables o errores será enviada de nuevo al servidor por medio de la cláusula *throws*.

Método generateWorkUnit (ClientInfo info).

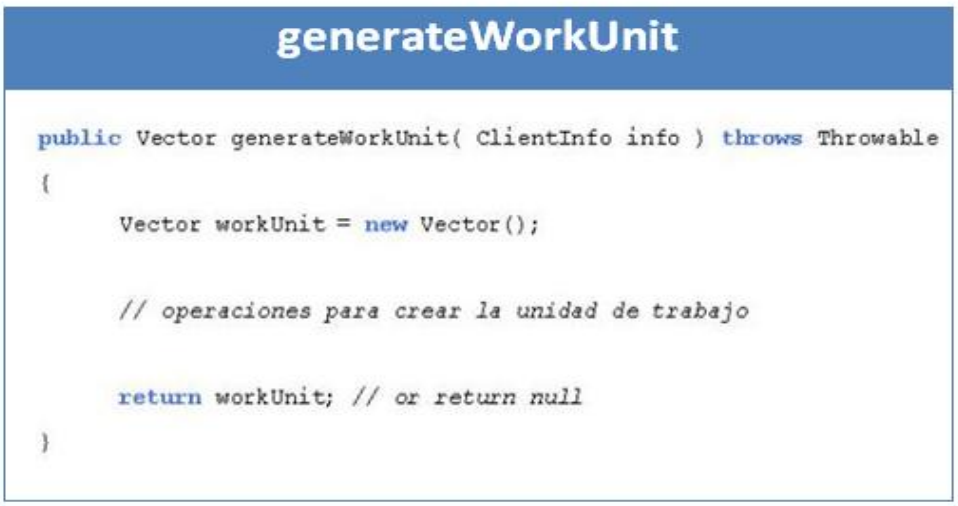


Figura 4: Declaración del método generateWorkUnit()

Este método es llamado por el sistema cada vez que un cliente solicita una unidad de trabajo para procesar y el tipo de datos que devuelve es un Vector. El algoritmo que está ejecutándose en los clientes recibe entonces este vector, que no es más que la unidad de trabajo que solicitó para procesar. Cuando todos los elementos contenidos en este vector son enviados, deben ser casteados a su tipo de datos original al llegar al Task. Si no hubiesen unidades de trabajo disponibles para procesar, esta función retornara null, esto le indica al servidor que no hay unidades de trabajo en ese momento.

El parámetro ClientInfo pasado al método generateWorkUnit (ClientInfo info) representa la información del cliente que solicita la unidad de trabajo. Esto permite generar cómputos específicos para determinados recursos computacionales.

Método processResults (Long uID, Vector results).

processResults

```
public boolean processResults (Long unitID, Vector results) throws
Throwable
{
      // procesar el conjunto de resultado

      if ( <problema ha finalizado> )
      {
            return true;
      }
      else
      {
            return false;
      }
}
```

Figura 5: Declaración del método processResults()

Este método es llamado cuando un cliente retorna un conjunto de resultados. Esta función consta de dos parámetros: el primero es el ID, que identifica a la unidad de trabajo generada por el método generateWorkUnit () y el segundo es el conjunto de resultados (vector) que devolvió el algoritmo al ser ejecutado en el cliente.

Cuando el cálculo distribuido culmina, este método retorna *true.* Si esto sucede, el servidor eliminará del sistema de cómputo el problema y comprimirá el directorio de trabajo, colocándolo disponible para su descarga.

Método adjustGranularity ().

adjustGranularity

```
public void adjustGranularity (int percent) throws Throwable
{
      // asumiendo que la granularidad es controlada por una variable
     // nombrada granularity

      granularity = granularity + ((int)(granularity * percent)/100);
}
```

Figura 6: Declaración del método adjustGranularity()

Este método es llamado periódicamente en el servidor, recibiendo un parámetro correspondiente al porcentaje (positivo o negativo) de cuanto la granularidad paralela debe ajustarse de forma que la medida del tiempo de procesamiento coincida con el tiempo de procesamiento óptimo. Este porcentaje se basa en el tiempo promedio de procesamiento de anteriores conjuntos de resultados ya calculados, utilizando el promedio exponencial del movimiento de carga de las funciones. Por lo tanto, el problema se ajusta constantemente para responder a los cambios en la dinámica y heterogénea red de los clientes puesta a disposición del sistema.

Método getStatus ().

getStatus

```
public String getStatus() throws Throwable
{
      String s = // algún dato de interés sobre el progreso de la
                 // ejecución
      return s;
}
```

Figura 7: Declaración del método getStatus()

Este método se llama cuando un usuario solicita información acerca del estado de un problema determinado al servidor. La información que devuelve es de tipo *string*, y se espera que contenga información útil acerca del estado actual del cálculo.

Método closeResources ().

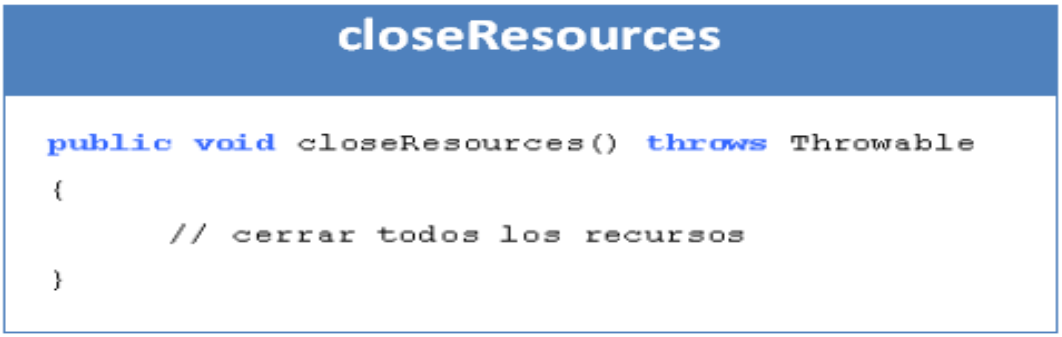


Figura 8: Declaración del método closeResources()

Este método es invocado justo antes de que el problema se elimine del sistema. El se encarga de cerrar cualquier recurso utilizado, por ejemplo: archivos, directorios, entre otras. El usuario debe implementar el código necesario para cerrar cualquier recurso, de manera que los archivos de cualquier ejecución puedan ser comprimidos y posteriormente eliminados del disco del servidor.

2.4.2 Clase Task

La clase Task se ejecuta en el cliente y se encargará de procesar todas las unidades de trabajo generadas por el método generateWorkUnit () de la clase DataManager del servidor.

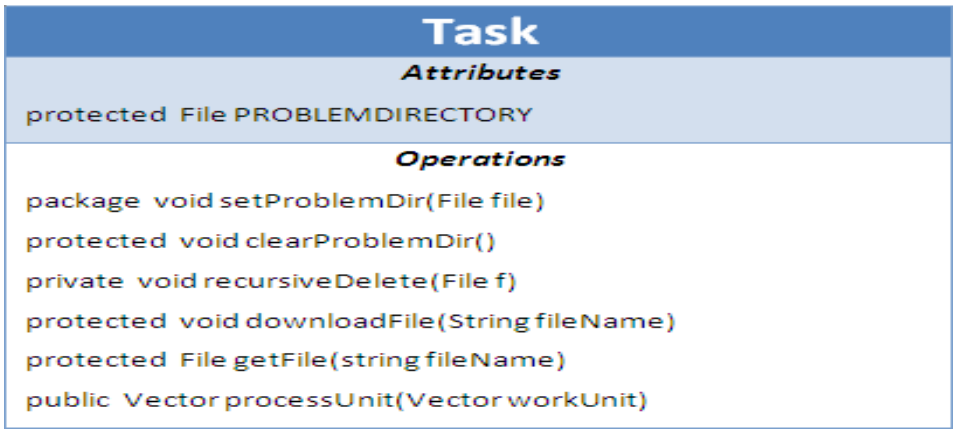


Figura 9: UML de la clase Task

En esta clase solo se redefine el método processUnit (). Un algoritmo puede estar compuesto de una serie de funciones para procesar diferentes tipos de unidades de trabajo generadas por el DataManager.

Método processUnit (Vector workUnit).

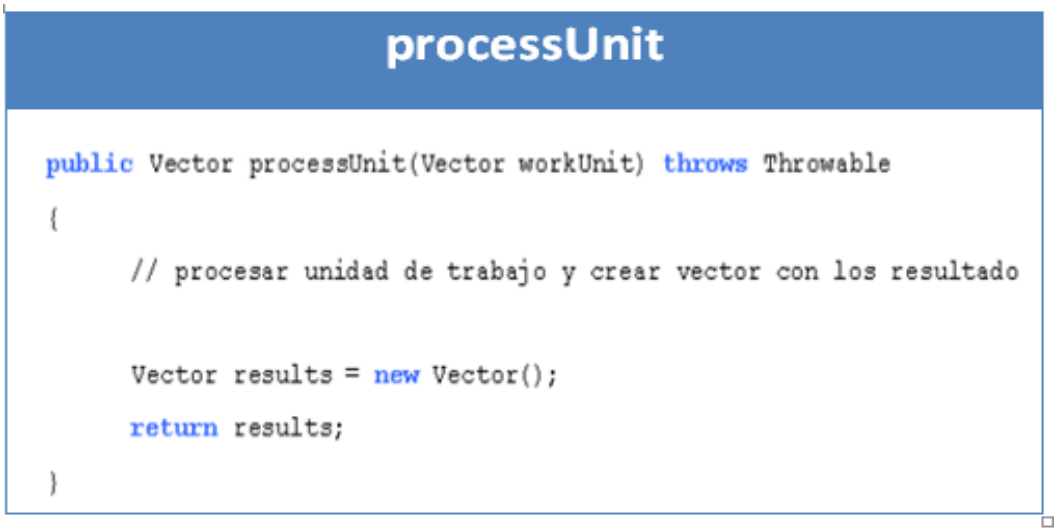


Figura 10: Declaración del método processUnit().

Este método recibe como parámetro una unidad de trabajo (vector) que se genera en el método generateWorkUnit () de la clase DataManager. Todos los tipos de datos de los elementos del vector son casteados a su tipo original de datos a través de los mecanismos de Java. Los resultados de las unidades de trabajo son devueltos en un vector y son regresados al método processResults () en el DataManager

Si una unidad de trabajo provoca reiterados errores durante su procesamiento, entonces la ejecución se elimina del sistema.

2.5 Recursos computacionales. Hardware y Software.

La elección de los recursos computacionales a utilizar en un conjunto de computadoras para realizar cálculos intensivos puede definirse primeramente en base a conocer la magnitud del problema o problemas que se quieren resolver. También está en dependencia del hardware, los recursos económicos y los humanos con los que se cuentan.

Estaciones de trabajo.

En total se utilizaron 20 estaciones de trabajo con diferentes características de hardware y software (Tabla 1).

Tabla 1: Rasgos de hardware y software de las estaciones de trabajos utilizadas.

Tipo	Procesador (CPU)	Memoria (RAM)	Sistema Operativo
1	Intel (R) Pentium (R) 4. Velocidad de CPU 3.0 GHz.	1 Gb	WindowsXP
2	Intel (R) Pentium (R) 4. Velocidad de CPU 3.0 GHz.	1 Gb	Ubuntu

La diferencia principal entre las computadoras de tipo 1 y tipo 2 radica en el sistema operativo, y los software que tienen instalados.

Las estaciones de trabajo utilizadas no solo tienen instalado el software necesario para realizar la investigación sino que cuentan con otras aplicaciones usadas por los usuarios de la institución para su trabajo diario.

Para lograr obtener una heterogeneidad se utilizaron estaciones de trabajos con dos sistemas operativos diferentes:

- ✓ Ubuntu; esta distribución de GNU/Linux es bastante amigable y brinda facilidades de trabajo para los usuarios con poco dominio de Linux.
- ✓ Windows XP y todos los programas necesarios con los cuales se trabaja en la institución.

2.6 Características de las redes.

El canal de comunicación es uno de los subsistemas más importantes en la construcción de un conglomerado de computadoras, pues de este dependen todo tipo de comunicaciones que haya entre los nodos. Cuando no se tiene el conocimiento sobre la importancia de este aspecto, y se descuida, el

tener la última tecnología en cuanto a procesadores no ayuda mucho al sistema distribuido, pues la comunicación definida en éste, echa por la borda el trabajo de cooperación que pudiera tener de una forma eficiente.

En esta propuesta de sistema distribuido se utiliza la red de tipo Ethernet a 100 Mbps de velocidad que estaba instalada en la institución donde se realizaron las pruebas.

Aunque una red de este tipo no siempre satisface las necesidades de comunicación en este trabajo, fue suficiente para cumplir con los objetivos propuestos. Además, por su bajo costo relativo y la facilidad de uso e implementación, las redes Ethernet son las que más abundan en las instituciones cubanas.

2.7 Muestras utilizadas.

Las muestras utilizadas en la investigación fueron tomadas de la base de datos de fragmentos perteneciente al proyecto *bioGRATO*. De forma general se trabajó con 5 muestras:

Tabla 2: Descripción de las muestras utilizadas.

Tipo de muestra	Tamaño de la muestra
Ciclo_5	531
Cluster_3_Camino de longitud 4	8 043
FragCamino_2	10 155
FragCamino_4	20 913
FragCamino_6	29 010

El tamaño de las muestras fue escogido cuidadosamente, para analizar el trabajo de la aplicación y su velocidad con juegos de datos de tamaños diversos.

CAPITULO 3: Resultados y Discusión

En este capitulo se exponen las características del algoritmo de programación genética que se quiere distribuir. Se presentan resultados experimentales de este algoritmo ejecutado en un ordenador y a partir de aquí, se brinda la solución propuesta a este problema y se exponen resultados experimentales haciendo uso de la solución brindada de forma distribuida. Finalmente, se analizan los tiempos obtenidos en las corridas experimentales del algoritmo con las diferentes muestras utilizadas.

3.1 Breve explicación del algoritmo de Programación Genética utilizado.

Para poder desarrollar un algoritmo de PG es necesario tener una buena definición del problema. En este epígrafe se describe de forma detallada las características propias del algoritmo a distribuir.

Generación de la población inicial.

La población inicial se genera de forma aleatoria haciendo uso del conjunto de funciones, terminales y constantes del problema, hasta llegar al número de individuos especificados por el administrador del sistema a través de la interfaz visual.

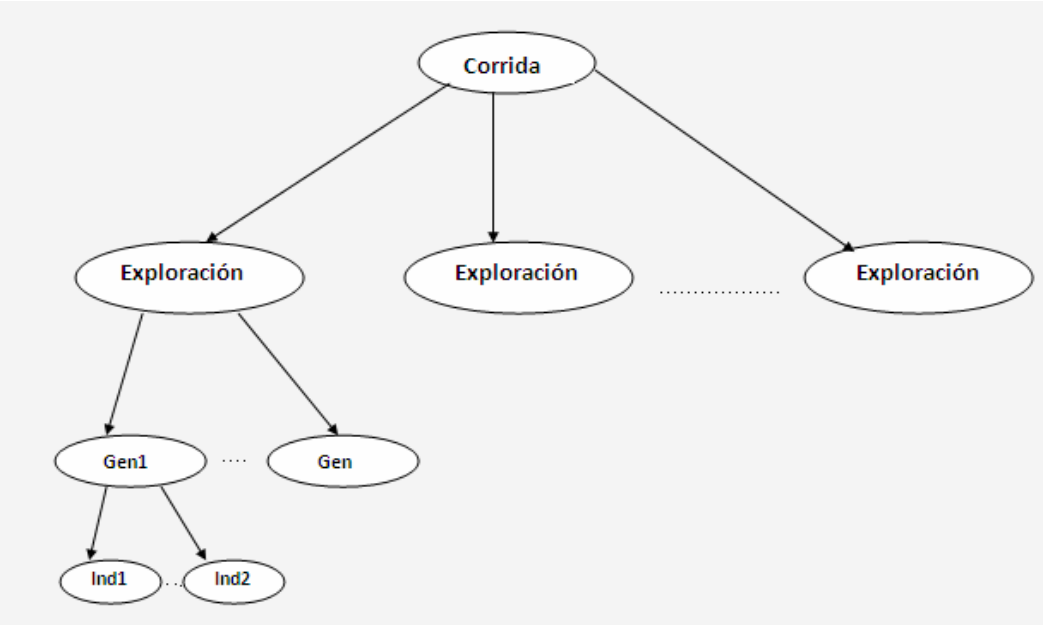


Figura 11: Ejecución del algoritmo por corridas, exploraciones y generaciones.

En el caso particular del algoritmo que se estudia, la población de individuos está formada por los modelos matemáticos que describen la actividad biológica estudiada. Estos modelos corresponden a funciones matemáticas, por esta razón el conjunto de funciones permitidas sólo abarca:

- ✓ Operaciones aritméticas (+, -, *, etc.).
- ✓ Funciones matemáticas (seno, coseno, log, etc.)

El usuario define el número de corridas, de exploraciones, de generaciones y la cantidad de individuos con el que desea ejecutar el algoritmo. Cada corrida posee un número de exploraciones que a su vez posee un número de generaciones, y esta última un número de individuos que se cumplirá con seguridad cada 5 generaciones de cada exploración.

Una vez generada aleatoriamente la primera generación, de la primera exploración, se les aplica una serie de operadores genéticos a los individuos (cruzamiento, mutación, reproducción) dando lugar a nuevos individuos. De la nueva población se eliminan todos aquellos que hayan sido repetidos y todos los que sobrepasen el tamaño definido por el usuario, el resto pasa a ser la próxima generación dentro de la misma exploración. Cada 5 generaciones se escoge el mejor individuo, utilizando selección elitista, este será copiado sin alteración en la generación siguiente (por ejemplo, si el usuario define 10 individuos por generación y se está en la quinta generación se copia a la sexta el mejor individuo encontrado hasta el momento que será el de mayor aptitud, generando nueve individuos aleatoriamente hasta completar nuevamente la cantidad especificada por el usuario, desechando el resto de la población que se tenía hasta ese momento y volviendo a cruzar, mutar y eliminar individuos formando una nueva generación, repitiéndose el ciclo hasta cumplirse el número de generaciones definido por el usuario). Los mejores individuos de cada 5 generaciones se van comparando y quedándose en memoria. El mejor encontrado en la última generación de la exploración resultará ser el mejor de esa exploración, luego se comparan los individuos por exploraciones y de este modo se obtiene el mejor individuo de la corrida en general que a su vez será comparado con los mejores individuos del resto de las corridas hallándose la mejor solución, siendo éste, el modelo de mejor ajuste para la actividad biológica analizada.

3.2 Resultados experimentales ejecutando el algoritmo de PG en una PC.

En este epígrafe se analizarán algunos resultados experimentales obtenidos al utilizar el sistema para generar modelos predictivos donde se variaron parámetros, como la cantidad de corridas, el número de exploraciones y el número de generaciones así como el tamaño de las muestras. Para lograr estos resultados se midió la hora de inicio (HI), la hora de fin (HF) y el tiempo que demoró el algoritmo (TD).

Resultado experimental número 1

Se utilizó una muestra (ciclo_5), con 531 compuestos, La interfaz visual de la aplicación se cargó con los siguientes parámetros de entrada:

Máxima cantidad de nodos en un árbol: 23
Número de individuos por generación: 10
Probabilidad de Mutación: 0.95
Probabilidad de que un nodo sea hoja: 0.4
Número de Exploraciones: 50
Número de Generaciones por Exploración: 50
Número de Corridas: 1
Operadores que se utilizaron: +, -, /, *, √, log, exp.

HI: 10.53 am

HF: 10.59 am

TD: 6 minutos

La generación del modelo obtenido solo tarda 6 minutos definiendo el usuario 50 generaciones, con 50 exploraciones y una sola corrida del algoritmo. Teniendo en cuenta que el tamaño de la muestra es relativamente pequeño y que solo se corre una vez el algoritmo no tiene sentido la distribución del mismo.

Resultado experimental número 2

Se utilizó una muestra (ciclo_5), con 531 compuestos, La interfaz visual de la aplicación se cargó con los siguientes parámetros de entrada:

Máxima cantidad de nodos en un árbol: 23

Número de individuos por generación: 10
Probabilidad de Mutación: 0.95
Probabilidad de que un nodo sea hoja: 0.4
Número de Exploraciones: 50
Número de Generaciones por Exploración: 50
Número de Corridas: 50
Operadores que se utilizaron: +, -, /, *, √, log, exp.

En este experimento hay que tener en cuenta la variación de exploraciones y de corridas, ahora en total se van a realizar 2500 exploraciones en 50 corridas del algoritmo.

HI: 11.26 am

HF: 5.09 pm

TD: 5 horas y 43 minutos (543 min., 57.2 veces más)

Se puede ver la diferencia de tiempo de este experimento con el experimento anterior, ambos fueron realizados con la misma muestra aumentando el número de corridas del algoritmo arrojando como resultado, que el tiempo aumente de ejecución para la obtención del modelo aumente en más de 50 veces.

Resultado experimental número 3

Se utilizó una muestra cluster3cam_4 con 8043 compuestos. La interfaz visual de la aplicación se cargó con los siguientes parámetros de entrada:

Máxima cantidad de nodos en un árbol: 23
Número de individuos por generación: 10
Probabilidad de Mutación: 0.95
Probabilidad de que un nodo sea hoja: 0.4
Número de Exploraciones: 50
Número de Generaciones por Exploración: 50
Número de Corridas: 10
Operadores que se utilizaron: +, -, /, *, √, log, exp.

En este experimento se varió el tamaño de la muestra y la cantidad de corridas del algoritmo, manteniendo constantes los parámetros de exploraciones y generaciones como en el experimento

anterior, esto devuelve un tiempo de respuesta de 9 horas y 50 minutos, demostrando que el usuario necesitaría más de una jornada laboral para obtener el modelo.

Resultado experimental número 4

Se utilizó una muestra FragCaminos_2 con 10155 compuestos. La interfaz visual de la aplicación se cargó con los siguientes parámetros de entrada:

Máxima cantidad de nodos en un árbol: 23
Número de individuos por generación: 10
Probabilidad de Mutación: 0.95
Probabilidad de que un nodo sea hoja: 0.4
Número de Exploraciones: 40
Número de Generaciones por Exploración: 50
Número de Corridas: 20
Operadores que se utilizaron: +, -, /, *, √, log, exp.

Se varía el tamaño de la muestra y además se varían la cantidad de exploraciones por corridas del algoritmo, manteniendo constantes la cantidad de generaciones, esto devuelve un tiempo de respuesta de 23 horas y 40 minutos.

Resultado experimental número 5

Se utilizó una muestra FragCaminos_4 con 20913 compuestos. La interfaz visual de la aplicación se cargó con los siguientes parámetros de entrada:

Máxima cantidad de nodos en un árbol: 23
Número de individuos por generación: 10
Probabilidad de Mutación: 0.95
Probabilidad de que un nodo sea hoja: 0.4
Número de Exploraciones: 20
Número de Generaciones por Exploración: 50
Número de Corridas: 25
Operadores que se utilizaron: +, -, /, *, √, log, exp.

Se varía el tamaño de la muestra, el número de corridas y de exploraciones manteniendo constante la cantidad de generaciones como en el experimento anterior, esto devuelve un tiempo de respuesta de 25 horas y 10 minutos.

Resultado experimental número 6

Se utilizó una muestra FragCaminos_6 con 29010 compuestos. La interfaz visual de la aplicación se cargó con los siguientes parámetros de entrada:

Máxima cantidad de nodos en un árbol: 23
Número de individuos por generación: 10
Probabilidad de Mutación: 0.95
Probabilidad de que un nodo sea hoja: 0.4
Número de Exploraciones: 20
Número de Generaciones por Exploración: 50
Número de Corridas: 35
Operadores que se utilizaron: +, -, /, *, √, log, exp.

Se varía el tamaño de la muestra y la cantidad de corrida manteniéndose constantes el número de exploraciones y generaciones, esto devuelve un tiempo de respuesta de 37 horas y 40 minutos.

Análisis de los resultados de forma secuencial.

Tomando en cuenta los resultados experimentales realizados anteriormente, se decide tomar las muestras con las que se ha venido trabajando y fijar los parámetros que hasta este momento se han variado, como son la cantidad de corridas, el número de generaciones y el número de exploraciones, para ver como los tiempos de respuesta del algoritmo ascienden a medida que se necesita procesar muestras cada vez mayores.

Cantidad de Corridas: 1
Cantidad de Exploraciones: 20
Cantidad de Generaciones: 50

Figura 12: Resultados de muestras contra tiempo secuencialmente.

En el gráfico anterior se puede observar que a medida que se aumenta el tamaño de la muestra aumentan significativamente los tiempos de respuesta del algoritmo, además se debe resaltar que a medida que el usuario va aumentando el espacio de búsqueda del algoritmo al aumentar el número de corridas, exploraciones o generaciones el tiempo del algoritmo asciende considerablemente, lo cual puede comprobarse en la Fig. 13 en la que se graficaron simultáneamente algunas de las variables utilizadas en los experimentos.

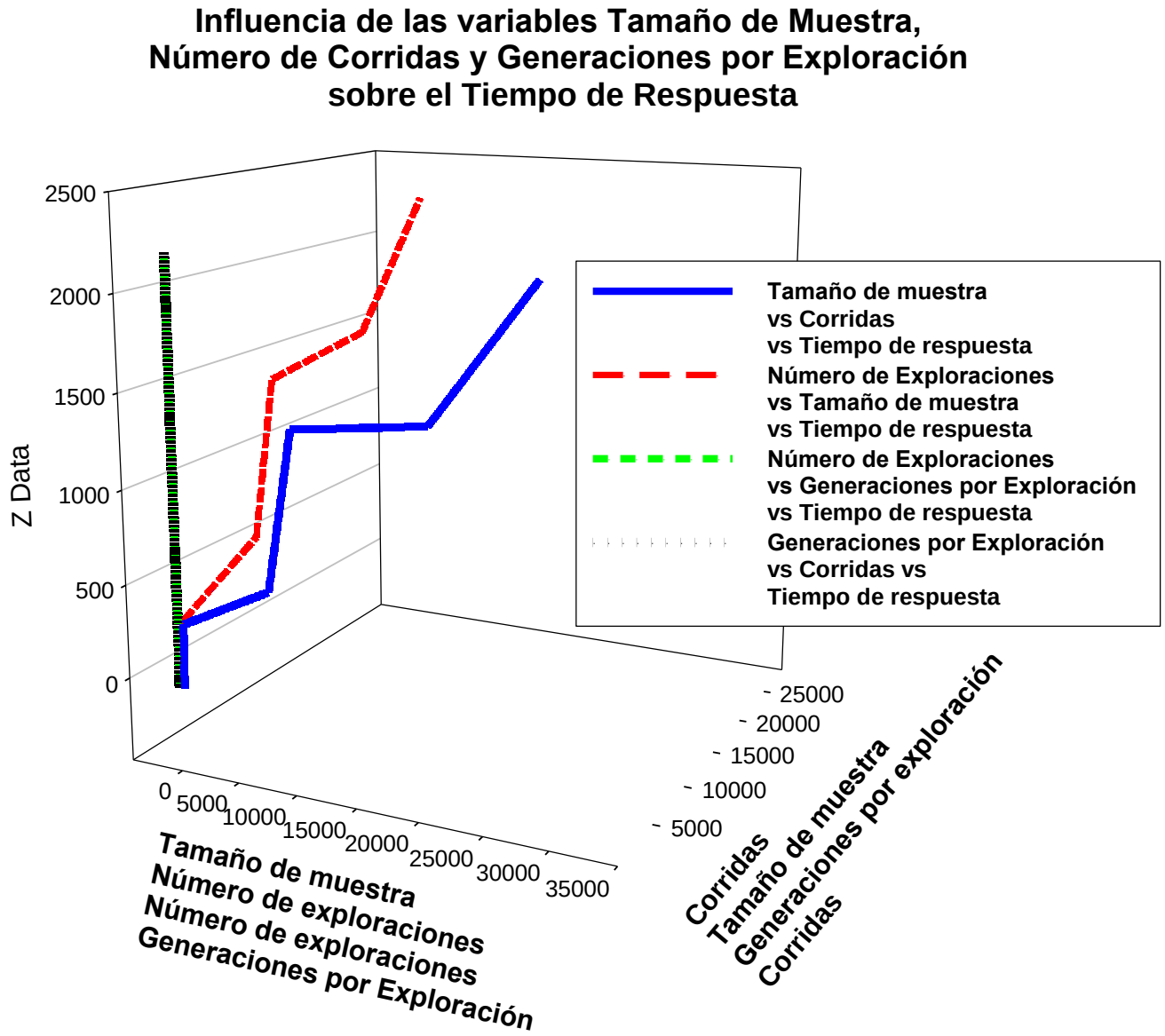


Figura 13. Influencia de diferentes variables utilizadas en el tiempo de respuesta del algorito de Programación Genética

3.3 Solución propuesta.

En el algoritmo de PG explicado anteriormente existe una cuestión importante a analizar. Se trata de las estadísticas de ejecución, lo que significa que el algoritmo al ser ejecutado con muestras muy grandes y al aumentar los espacios de búsqueda, aumentando el número de exploraciones y generaciones, los tiempos de respuesta del algoritmo se vuelven prolongados e incluso se puede dar

la situación de que nunca devuelva una respuesta debido a la gran cantidad de cálculos que se generan.

Teniendo en cuenta la gran cantidad de ordenadores conectados en red con los que cuenta la UCI, se debe pensar en un sistema que aproveche estas ventajas y ayude a disminuir el tiempo de respuestas del algoritmo antes expuesto.

Debido a lo anterior, se analizó la posibilidad de diseñar una propuesta para la distribución del algoritmo que hiciera uso de la Plataforma de Tareas distribuidas y que minimizara los tiempos de respuesta.

Se estudió cada una de las clases del algoritmo así como las dependencias entre ellas. Se detectaron 4 clases fundamentales:

- ✓ Corrida.
- ✓ Exploración.
- ✓ Generación.
- ✓ Individuo.

Las 4 clases fueron analizadas y se detectó que existía una estrecha dependencia entre las clases exploración, generación e individuo, debido a que cada 5 generaciones se escoge el mejor individuo, este se copia sin alteración en la generación siguiente (por ejemplo, si el usuario define 10 individuos por generación y estamos en la quinta generación se copia a la sexta el mejor individuo encontrado hasta el momento que será el de mayor aptitud, se generan aleatoriamente nueve individuos hasta completar de nuevo la cantidad especificada por el usuario y esta será la respuesta devuelta por la exploración que se esté ejecutando en ese momento. En los casos de las clases *corrida* y *exploración*, no se encontró dependencia ninguna dando la idea de que el algoritmo pudiera ser distribuido a partir de este nivel, por lo tanto se dirigió el trabajo a distribuir el algoritmo al nivel de exploraciones.

Teniendo en cuenta lo anterior se diseñó un nuevo algoritmo que distribuye las exploraciones y hace uso de la Plataforma de tareas distribuidas para distribuir el trabajo que antes se realizaba en un ordenador, en varios clientes conectados a un servidor que funciona como un sistema de gran transparencia, es decir, este sistema va a comportarse de manera ideal sin que el usuario perciba en ningún momento los posibles problemas que puedan presentarse en él.

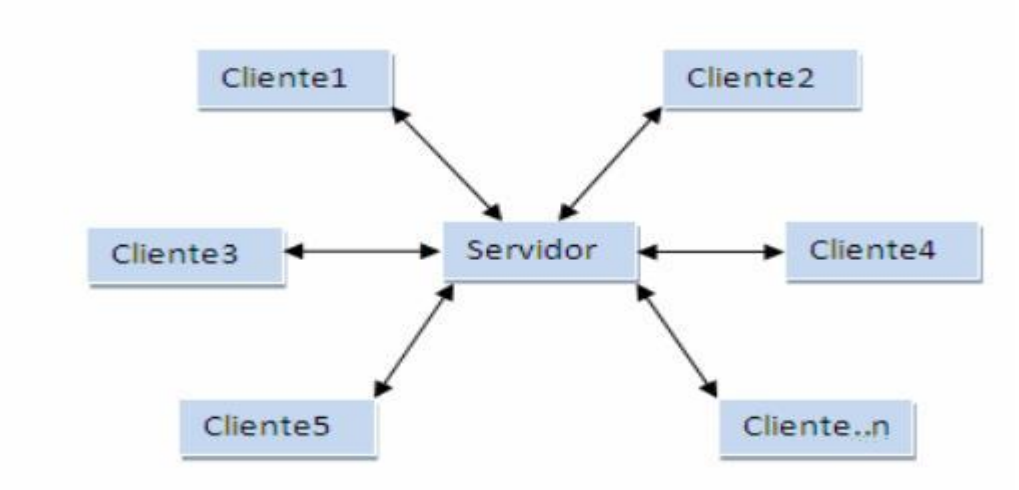


Figura 14: Interacción del servidor con los clientes.

El funcionamiento de este sistema consiste en un servidor que se encargará de almacenar el algoritmo y los datos que se van a procesar. Almacenará específicamente, el algoritmo de programación genética y el fichero de la muestra. Posteriormente, los clientes se reportan disponibles y hacen una petición de unidad de trabajo, el servidor se encarga entonces de repartir las unidades de trabajo entre los clientes y estos a la vez se encargan de procesarlas y enviárselas de vuelta al servidor para volver a pedir otra unidad de trabajo. Por último, el servidor se encarga de integrar todos los resultados que ha obtenido de los clientes para devolver el resultado final del algoritmo luego de haber procesado la muestra completa.

Para calcular el tiempo de distribución del algoritmo, se realizó una simulación de un sistema distribuido capaz de distribuir *n* exploraciones deseadas por el usuario sobre una muestra determinada. El cálculo del tiempo que demora el algoritmo en un sistema distribuido se determina de la siguiente forma:

$$t_d = \sum_{i=1}^{CV} t_{md} + t_g$$

$$CV = \begin{cases} CE / CPC_s & \text{si} \quad CE \,\%\, CP_c = 0 \\ CE / CPC_s + 1 \end{cases}$$

donde:

t_d = Tiempo de distribución.

CV = Cantidad de veces a distribuir.

t_{md} = Tiempo de la exploración que mas demore en terminar.

t_g = Tiempo de gestión (tiempo que demoran las comunicaciones y procesamiento de información entre el servidor y los clientes).

CE = Cantidad de exploraciones que se van a distribuir.

CPC_s = Cantidad de computadoras personales donde se va a distribuir.

3.4 Resultados experimentales del algoritmo de PG distribuido.

En esta sección se analizarán algunos resultados experimentales obtenidos al utilizar una simulación de un sistema distribuido para generar modelos donde se variaron parámetros, operaciones y muestras con el fin de analizar los tiempos de respuesta que devuelve el algoritmo de programación genética. Estos experimentos a realizar son similares a los se realizaron en un ordenador, ahora en varios, simulando una distribución. De esta forma se podrá observar la diferencia entre los tiempos de respuesta del algoritmo comparando las dos vías.

Resultado experimental número 1

Se utilizó una muestra (ciclo_5), con 531 compuestos.

En este experimento se van a realizar 50 corridas con 50 exploraciones para cada una de ellas, en total quedarían 2500 exploraciones a distribuir, las mismas se van a distribuir en 20 PC, con el objetivo de ver el tiempo de respuesta del algoritmo de forma distribuida.

La cantidad de exploraciones a distribuir dividido entre la cantidad de máquinas que se tienen disponibles arroja que se debe distribuir 125 veces para cubrir la solución completa.

Los tiempos que demora cada distribución oscilaron entre 9 segundos y 16 segundos, para devolver un tiempo de respuesta del algoritmo distribuido de:

TD: 25 minutos y 40 segundos

Resultado experimental número 2

Se utilizó una muestra cluster3cam_4 con 8043 compuestos.

En este experimento se van a realizar 10 corridas con 50 exploraciones para cada una de ellas, en total quedarían 500 exploraciones a distribuir, las mismas se van a distribuir en 20 PC, con el objetivo de ver el tiempo de respuesta del algoritmo de forma distribuida.

La cantidad de exploraciones a distribuir dividida entre la cantidad de máquinas que se tienen disponibles arroja que se debe distribuir 25 veces para cubrir la solución completa.

El tiempo que demoró cada distribución osciló entre 35 y 38 segundos, para devolver un tiempo de respuesta del algoritmo distribuido de:

TD: 14 minutos y 10 segundos

Resultado experimental número 3

Se utilizó una muestra FragCaminos_2 con 10155 compuestos.

En este experimento se van a realizar 20 corridas con 40 exploraciones para cada una de ellas, en total quedarían 800 exploraciones a distribuir y las mismas se van a distribuir en 20 PC, con el objetivo de ver el tiempo de respuesta del algoritmo de forma distribuida.

La cantidad de exploraciones a distribuir dividida entre la cantidad de máquinas que se tienen disponibles arroja que se debe distribuir 40 veces para cubrir la solución completa.

El tiempo que demoró cada distribución osciló entre 40 y 45 segundos, para devolver un tiempo de respuesta del algoritmo distribuido de:

TD: 28 minutos y 5 segundos

Resultado experimental número 4

Se utilizó una muestra FragCaminos_4 con 20913 compuestos.

En este experimento se van a realizar 25 corridas con 20 exploraciones para cada una de ellas, en total quedarían 500 exploraciones a distribuir y las mismas se van a distribuir en 20 PC, con el objetivo de ver el tiempo de respuesta del algoritmo de forma distribuida.

La cantidad de exploraciones a distribuir dividida entre la cantidad de máquinas que se tienen disponibles arroja que se debe distribuir 25 veces para cubrir la solución completa.

El tiempo que demoró cada distribución osciló entre 74 y 77 segundos, para devolver un tiempo de respuesta del algoritmo distribuido de:

TD: 31 minutos y 25 segundos

Resultado experimental número 5

Se utilizó una muestra FragCaminos_6 con 29010 compuestos.

En este experimento se van a realizar 35 corridas con 20 exploraciones para cada una de ellas, en total quedarían 700 exploraciones a distribuir y las mismas se van a distribuir en 20 PC, con el objetivo de ver el tiempo de respuesta del algoritmo de forma distribuida.

La cantidad de exploraciones a distribuir dividida entre la cantidad de máquinas que se tienen disponibles arroja que se debe distribuir 35 veces para cubrir la solución completa.

El tiempo que demoró cada distribución osciló entre 76 y 80 segundos, para devolver un tiempo de respuesta del algoritmo distribuido de:

TD: 45 minutos y 34 segundos

Análisis de los resultados de forma distribuida.

Para analizar los resultados experimentales realizados anteriormente, se decide tomar una de las muestras con las que se ha venido trabajando y fijar los parámetros que hasta este momento se han variado, como son la cantidad de corridas, el número de generaciones, el número de exploraciones, para ver cómo se comportan los tiempos de respuesta del algoritmo a medida que se incrementan los recursos computacionales (computadoras).

Muestra: FragCaminos_6 con 29010 compuestos.

Cantidad de corridas: 1

Cantidad de exploraciones: 20

Cantidad de generaciones: 50

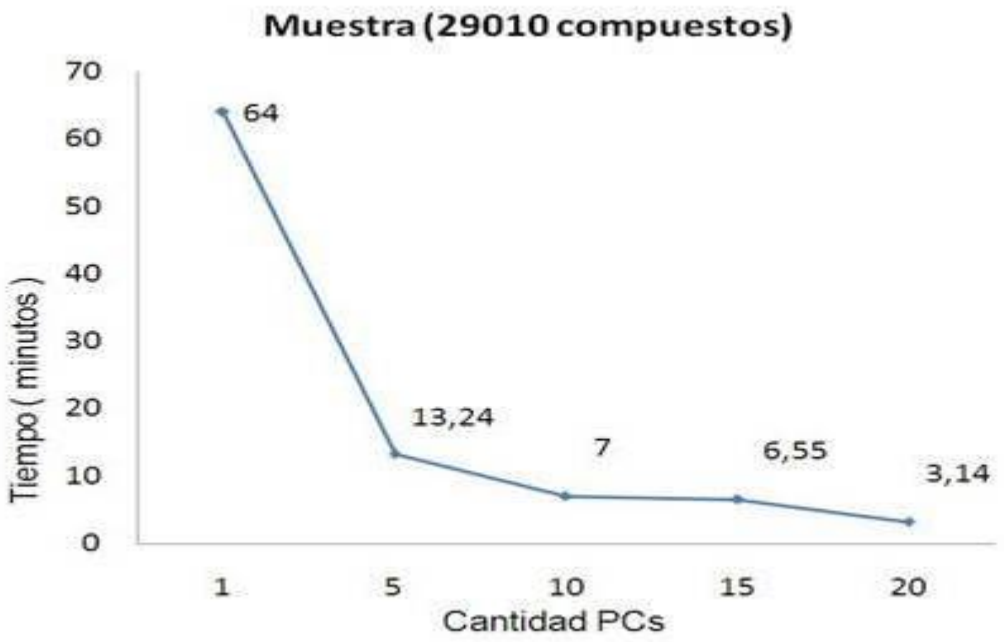


Figura 15: Resultados del algoritmo distribuido con una muestra de 29010 compuestos.

3.5 Comparación entre los resultados secuenciales y distribuidos.

En la Tabla 4 se muestra un resumen comparativo entre los resultados obtenidos de forma secuencial y de forma distribuida.

Tabla 4: Comparación de los tiempos del algoritmo de forma secuencial y distribuido.

Tamaño de muestra	Parámetros fijados.	Tiempo secuencial.	Tiempo distribuido.
531	Corridas:50 Generaciones:50 Exploraciones:50	5 horas y 43 minutos.	25 minutos y 40 segundos

8043	Corridas:10 Generaciones:50 Exploraciones:50	9 horas y 50 minutos.	15 minutos y 10 segundos.
10 155	Corridas:20 Generaciones:50 Exploraciones:40	23 horas y 40 minutos	28 minutos y 5 segundos.
20 913	Corridas:25 Generaciones:50 Exploraciones:20	25 horas y 10 minutos.	31 minutos y 25 segundos.
29 010	Corridas:35 Generaciones:50 Exploraciones:20	37 horas y 40 minutos	45 minutos y 34 segundos.

En la tabla 4 puede apreciarse como los tiempos de ejecución disminuyen considerablemente al simular la propuesta distribuida del algoritmo de PG para generar los modelos.

CONCLUSIONES

- ✓ Se propone un algoritmo de programación genética distribuido como resultado del análisis, diseño y simulación del mismo.
- ✓ Se demostró que el algoritmo de programación genética aumenta el tiempo de respuesta al incrementarse el número de corridas, así como la cantidad de exploraciones, de generaciones por exploración y del tamaño de la muestra.

RECOMENDACIONES

- ✓ Implementar la propuesta del algoritmo de programación genética distribuida.

BIBLIOGRAFÍA

1. **Coulouris, George.** Sistemas Distribuidos. Madrid : s.n., 2001.

2. **Andaluz, A.M.** *Algoritmos evolutivos y algoritmos genéticos.* Madrid : s.n., 2005.

3. **Fogel, D. B.** *Evolutionary Computation,Toward a New Philosophy of Machine Intelligence.* New York : The Institute of Electrical and Electronic Engineers, 1995.

4. **Cubias, Doménike, Ortega Martínez y Boris E. Turcios.** *Algoritmos genéticos.* s.l. : Universidad Don Bosco, 2005.

5. **Poli R., Rowe, J. y McPhee, N.** *"Markov chain models for GP and variable-length GAs with homologous crossover". En Genetic ProgrammingProceedings of the Genetic and Evolutionary Computation Conference.* San Francisco, California : s.n., 2001.

6. **Molina Souto, Yania y Mejias Cesar, Yuleidys.** *"Módulo de predicción de actividad biológica anticancerígena de compuestos orgánicos, partiendo de fragmentos, utilizando Programación Genética".* Ciudad de la Habana : Universidad de las Ciencias Informáticas (UCI), 2007.

7. **Koza J.** Genetic Programming, on the programming of computers by means of natural selection. [En línea] 1992. [Citado el: 26 de febrero de 2009.] http://www.ru.lv/~peter/zinatne/ebooks/MIT%20-%20Genetic%20Programming.pdf.

8. **Aranda, Jorge Baier.** PUC Programación Genética. *PUC Programación Genética.* [En línea] [Citado el: 26 de febrero de 2009.] http://www2.ing.puc.d/~jabaier/iic2622/gp.pdf.

9. **Carrasco-Velar, R.** Introducción al diseño de Fármacos. *Introducción al diseño de Fármacos.* [En línea] [Citado el: 22 de enero de 2009.] http://revistas.mes.edu.cu/elibro/libros/tecnologia/9789591606471.pdf.

10. **Gracia, O. R.** *Herramientas de cribado virtual aplicadas a inhibidores de tirosina quinasas. Contribución al desarrollo del programa PRALINS para el diseño de quimiotecas combinatorias.* s.l. : Escola Tècnica Superior IQS.

11. **Colouris, G, Dollimore, J. And Kindberg, T.** *Distributed Systems: Concepts and Design.* Massachussets : Addison-Wesley Publishing Company. Reading, 1989.

12. **Santoro, Nicola.** DESIGN AND ANALYSIS OF DISTRIBUTED ALGORITHMS. Ottawa, Canada : s.n., 2007.

13. **Grupo de Bioinformática .** *Manual del Desarrollador: Plataforma de Tareas Distribuidas.* Ciudad de la habana : s.n.

14. **Goldberg, D. E.** *Genetic Algorithms in Search, Optimization and Machine Learning.* Massachussets : Addison-Wesley Publishing Co., 1989.

15. Berkeley Open Infrastructure for Network Computing. [En línea] http://boinc.berkeley.edu/..

16. **Martín Martín, Jorge y Morate G, D.** Seminario: algoritmos genéticos Valladolid. [En línea] [Citado el: 5 de mayo de 2009.] http://www.infor.uva.es/~calonso/IAI/TrabajoAlumnos/memoriaAG.pdf..

17. **Koza, J. R.** Sitios oficiales de California. [En línea] [Citado el: 22 de mayo de 2009.] http://www.genetic-programming.com..

18. **Poli R.,** *"Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover".* s.l. : Genetic Programming and Evolvable Machines., 2001.

19. **Kubinyi, H.** QSAR: Hansh analysis and approaches. 1993.

20. **CK., Ian Foster.** The Grid: Blueprint for a New Computing Infrastructure. [aut. libro] Morgan Kaufmann. 1998.

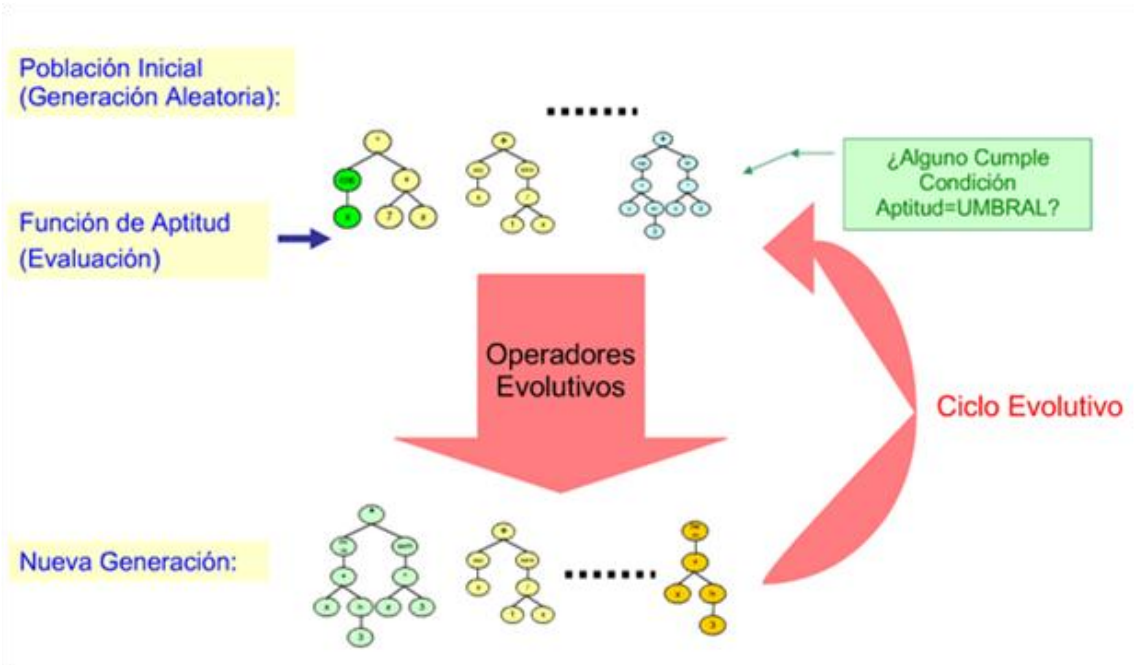
21. **Diudea., M.** QSPR/QSAR Studies by Molecular Descriptors. s.l. : Nova Science Publishes Inc., 2001.

22. **Fernández Casaní, Álvaro.** *ARQUITECTURAS GRID orientadas a la gestión de recursos.* 2004.

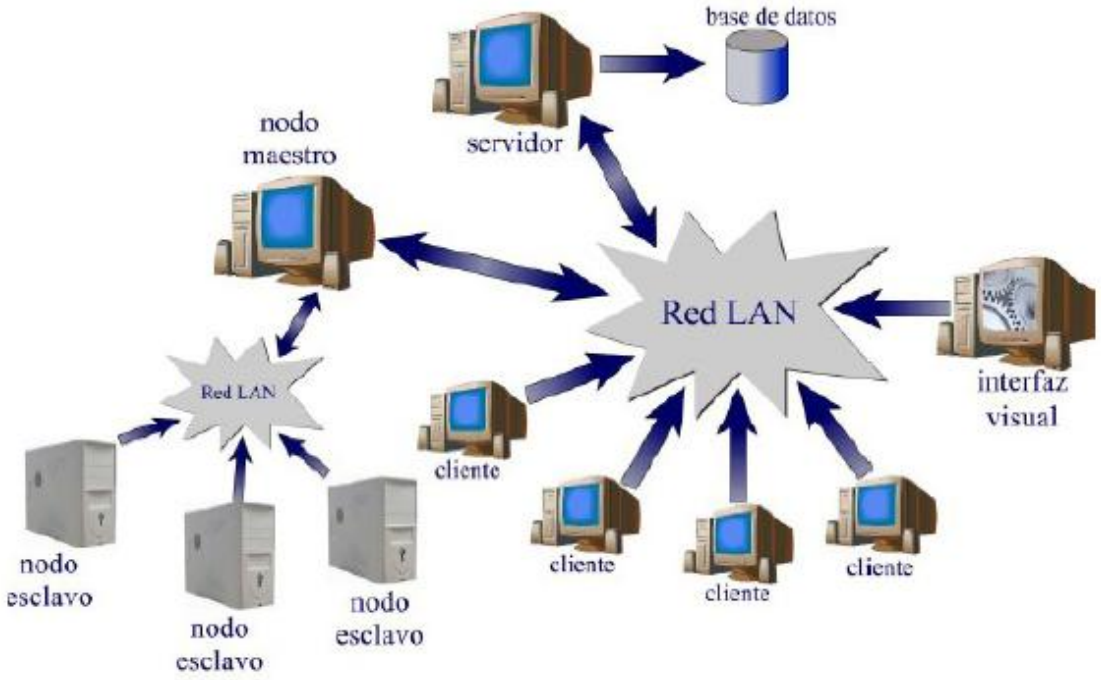
23. **Keane T.A.,** General-Purpose Heterogeneous Distributed Computing System. s.l. : National University of Iriland Maynooth., 2004.

24. **Tanenbaum A.** Steen MV.Distributed Systems: Principles and Paradigms.Prentice Hall. s.l. : Pearson Education, 2002.

ANEXOS



Anexo 1: Ciclo evolutivo del algoritmo de programación genética.



Anexo 2: Vista General del Sistema de Tareas Distribuidas.

GLOSARIO DE TÉRMINOS

Descriptor: Número que describe la estructura química o una propiedad de la molécula o fragmento de esta.

Índices: Números obtenidos mediante determinados algoritmos matemáticos que contienen información relacionada con la forma molecular, el grado de ramificación, tamaño molecular y la flexibilidad estructural.

Plug-in: Es una aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica, generalmente muy específica, como por ejemplo servir como driver en una aplicación, para hacer así funcionar un dispositivo en otro programa.

Compuestos orgánicos: Los compuestos o moléculas orgánicas son sustancias formadas principalmente por átomos de carbono e hidrógeno, y por otros elementos representativos como oxígeno, nitrógeno, azufre, fósforo y halógenos.

Moléculas: Una molécula es una partícula formada por un conjunto de átomos ligados por enlaces covalentes.

Átomos: Es la entidad química más pequeña, el mismo está compuesto de electrones, protones y neutrones.

Predicción: Se trata de análisis racional de lo que va a suceder…. Acción de anunciar por revelación, ciencia o conjetura [algo que ha de suceder].

Cómputo: Procedimiento que determina el valor de una cantidad por medio de operaciones matemáticas

Bioinformática: Es la aplicación de los ordenadores y los métodos informáticos en el análisis de datos experimentales y simulación de los sistemas biológicos.

Sistemas Distribuidos: Sistemas en que existen varias CPU conectadas entre sí, las cuales trabajan de manera conjunta.

Química computacional: es una rama de la química que utiliza computadores para ayudar a resolver problemas químicos.

Química medicinal: es una disciplina de la ciencia, intersección de la química, la bioquímica y la farmacología.

Clúster: se denomina a los conjuntos o conglomerados de computadoras construidos mediante la utilización de componentes de hardware.

Applet: es un componente de una aplicación que se ejecuta en el contexto de otro programa, por ejemplo un navegador web.

Directorio: es una agrupación de archivos de datos, atendiendo a su contenido, a su propósito o a cualquier criterio.

Aridad: se define la aridad de un operador matemático o de una función como el número de argumentos necesarios para que dicho operador o función se pueda calcular.

Actividad biológica: Propiedad que caracteriza el comportamiento biológico de compuestos químicos (Molécula o Fragmento).