BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

TESIS PROFESIONAL

Para Obtener El Título De:

LIC. EN CIENCIAS DE LA COMPUTACIÓN

Implementación de un algoritmo genético en paralelo para resolver el problema de la mochila multidimensional

Presenta: **CARLOS BALDERAS POSADA**

Asesor: **DRA. DARNES VILARIÑO AYALA**

Coasesor: **MC. MIREYA TOVAR VIDAL**

Agosto del 2009

Puebla, México

Dedicatorias

Éste trabajo esta dedicado con mucho cariño a mis padres (Sra. Teresa Posada y Sr. Angel Balderas) que me dieron la vida **y** han estado conmigo en todo momento, al igual que mi **hermano** (Eder Daniel Balderas).

Gracias por todo mamá y papá, gracias por darme una carrera para mi futuro, y en especial, por creer en mí, aunque hemos pasado momentos un poco difíciles, como familia, siempre han estado apoyándome y brindándome todo su amor, por todo esto les agradezco de todo corazón el que estén conmigo a mi lado, día a día. Los quiero mucho más de lo que se imaginan. Éste trabajo, que me llevó poco más de un año en terminarlo, es para ustedes, aquí está lo que me han brindaron: amor, desvelos, cariño, preocupaciones, alegrías, regaños, risas, opiniones,…, solamente les estoy devolviendo un poco lo que ustedes me han dado desde que viene a este mundo. Gracias al Todo Poderoso por darme unos padres como ustedes.

Le dedico, además, el presente trabajo a la Dra. Darnes Vilariño Ayala y al Dr. Manuel Martín Ortíz, ya que sin su ayuda, apoyo y enseñanza incondicional, tanto en el ámbito personal como escolar, no hubiera sido posible la conclusión de éste trabajo. Muchas gracias doctores.

Para culminar, también dedico esta investigación a mi amigo, alumno y aprendiz Saúl León Silverio, ya que me recuerda mis inicios de estudiante universitario, lleno de hambre de estudio, con metas a futuro, tratando de ser constante y con ganas de llegar más lejos cada día. Espero, Saúl, que todo lo que te he podido enseñar sea un aliciente para ti, y pronto llegues a ser un gran investigador como lo hemos comentado, cumple ese sueño. Amigo mío, suerte y rompe tus propios obstáculos.

Agradecimientos

Primeramente, quiero agradecer a los profesores de la FCC que han confiado en mí, me han impulsado y han sido un gran ejemplo en mi vida universitaria:

Dra. Darnes Vilariño
Dr. Manuel Martín Ortíz
Dr. Mario Rossainz
Dra. Josefa Somodevilla
Dr. Ivo Pineda
MC Mireya Tovar
MC Hilda Castillo
MC Beatriz Beltrán
MC Rogelio González

Muchas gracias por todo lo que me han brindado hasta la fecha, al igual que su solidaridad y apoyo en momentos críticos, así como todo lo que he podido aprender de cada uno de ustedes. No tengo más palabras para expresar mi respeto y admiración que les tengo.

También quiero agradecer a Lia, Sandy, Roxi, Doña Lulú y Guille, secretarias y contadora de la FCC, por el apoyo y aliento que me han brindado.

Por ultimo, y no por eso menos importante, un agradecimiento más, el cual va dirigido a mi mejor amigo: José de Jesús, así como a mis amigos y alumnos/amigos que me estuvieron acompañando en momentos de alegría, tristeza, enojo, etc. Mario, Germán, Jesús, Adrian, Enrique; mi DIF: Saúl, Eduardo, Emmanuel, Rafael e Inamic, así como a mis otros amigos de generación: Goyri, Yareli, David Sánchez, Fátima y Esaú. Les agradezco todo lo que me han aguantado y soportado. Muchas gracias. No tengo la idea de cómo compensarles todo su cariño y afecto que me han tenido.

Resumen

En la Facultad de Ciencias de la Computación (BUAP) existe un grupo de investigadores, los cuales, han desarrollado o mejorado algunos algoritmos metaheurísticos para dar solución al Problema de la Mochila Multidimensional (MKP por sus siglas en inglés), y también han propuesto diversas aplicaciones que han tratado de dar solución a éste interesante problema. Uno de esos trabajos fue realizado dentro de la Maestría en Ciencias de la Computación de la FCC, se creó una biblioteca para el desarrollo de Algoritmos Genéticos, bajo el lenguaje C++, que opera sólo bajo la plataforma Linux. Esta librería concentra únicamente la creación de objetos de tipo genéticos y métodos que los operan, los cuales deberán ser manejados según requiera la aplicación final y los resultados que se deseen obtener.

Uno de los objetivos más importante del presente trabajo ha sido el desarrollo de una estrategia eficiente de migración de individuos de una población hacia otra población, en aras de paralelizar el algoritmo genético implementado. Se desarrolló tanto un algoritmo genético secuencial para resolver el MKP, como un algoritmo genético paralelo para ofrecer la solución a problemas de este tipo. Para validar los resultados obtenidos se utilizaron los “test problems” expuestos en [27].

Aunque, no para todos los problemas resueltos se obtuvo el óptimo reportado hasta el momento, los algoritmos propuestos llegaron a ser convergentes a éste. En primera instancia se validó la biblioteca desarrollada para resolver un tipo particular de problemas [27] y con ello el grupo de investigación dispone ahora de un software eficiente que será aplicado en el área de Recuperación de Información en la búsqueda de características de individuos en la web.

Índice

Introducción

La optimización y los algoritmos genéticos en la computación

El *Algoritmo Genético* es una técnica de búsqueda basada en la teoría de la evolución de Darwin, que ha cobrado mucha popularidad alrededor del mundo durante los últimos años [1].

Fue en las décadas de 1950 y 1960 cuando varios científicos, de modo independiente, comenzaron a estudiar los sistemas evolutivos, guiados por la intuición de que se podrían emplear como herramienta en problemas de optimización en ingeniería. La idea es evolucionar una población de candidatos a ser solución de un problema conocido, utilizando operadores inspirados en la selección natural y la variación genética natural por cruce.

Con esta idea nace en el año 1993 la computación evolutiva, que retoma conceptos de la evolución y la genética para resolver principalmente problemas de optimización. Esta rama de la inteligencia artificial tiene sus raíces en tres desarrollos relacionados, pero independientes entre sí:

- ***Algoritmos Genéticos***
- ***Programación Evolutiva***
- ***Estrategias Evolutivas***

De estas tres áreas, parten los caminos hacia todos los campos de investigación inspirados en nuestros conocimientos sobre Evolución.

Las **estrategias evolutivas** son métodos computacionales que trabajan con una población de individuos que pertenecen al dominio de los números reales, que mediante los procesos de mutación y de recombinación evolucionan para alcanzar el óptimo de la función objetivo [2]. Entre 1965 y 1973 Rechenberg las introdujo como método para opti-

mizar parámetros reales para ciertos dispositivos. La misma idea fue desarrollada poco después (1975 - 1977) por Schwefel. El campo de las estrategias evolutivas ha permanecido como un área de investigación activa, cuyo desarrollo se produce en su mayor parte, de modo independiente al de los algoritmos genéticos (aunque recientemente se ha visto como las dos comunidades han comenzado a colaborar).

La **programación evolutiva** (PE) es prácticamente una variación de los algoritmos genéticos, donde lo que cambia es la representación de los individuos [3]. En el caso de la PE los individuos son ternas (tripletas) cuyos valores representan estados de un autómata finito. Cada terna está formada por el valor del estado actual, un símbolo del alfabeto utilizado y el valor del nuevo estado. Fogel, Owens y Walsh fueron los creadores en 1966 de la programación evolutiva, una técnica en la cual los candidatos a soluciones a tareas determinadas son representados por máquinas de estados finitos, cuyos diagramas de estados de transición evolucionaban mediante mutación aleatoria, seleccionándose el que mejor aproximara a la meta.

Pasemos ahora a hablar de los **algoritmos genéticos**. La primera mención del término, y la primera publicación sobre una aplicación del mismo, se deben a Bagley en [32]. Éste diseñó algoritmos genéticos para buscar conjuntos de parámetros en funciones de evaluación de juegos, y los comparó con los algoritmos de correlación, procedimientos de aprendizaje modelados después de los algoritmos de pesos variantes de ese periodo. Sin embargo el que es considerado como el creador de los algoritmos genéticos es John Holland [4].

En contraste con las estrategias evolutivas y la programación evolutiva, el propósito original de Holland no era diseñar algoritmos para resolver problemas concretos, sino estudiar, de un modo formal, el fenómeno de la adaptación tal y como ocurre en la naturaleza, y desarrollar vías para extrapolar esos mecanismos de adaptación natural a los sistemas computacionales. En el libro que Holland escribió [4] se presentaba el algoritmo genético como una abstracción de la evolución biológica, y proporcionaba el entramado teórico para la adaptación bajo el algoritmo genético.

El algoritmo genético de Holland es un método para desplazarse, de una población de cromosomas (representados por bits) a una nueva población, utilizando un sistema

similar a la selección natural junto con los operadores de cruces, mutaciones e inversión inspirados en la genética. En este primitivo algoritmo, cada cromosoma consta de genes (bits), y cada uno de ellos es una muestra de un alelo particular (0 ó 1).

Holland adapta los operadores de selección, cruces, mutaciones e inversión a su algoritmo:

- **Selección**: este operador escoge entre los cromosomas de la población aquellos con capacidad de reproducción, y entre estos, los que sean más compatibles, producirán más descendencia que el resto.

- **Cruce**: extrae partes de dos cromosomas, imitando la combinación biológica de dos cromosomas aislados (gametos).

- **Mutación**: se encarga de cambiar, de modo aleatorio, los valores del alelo en algunas localizaciones del cromosoma.

- **Inversión**: invierte el orden de una sección contigua del cromosoma, recolocando por tanto el orden en el que se almacenan los genes.

La mayor innovación de Holland fue la de introducir un algoritmo basado en poblaciones con cruces, mutaciones e inversiones. Es más, Holland fue el primero en intentar colocar la computación evolutiva sobre una base teórica firme. Hasta hace poco, esta base teórica, fundamentada en la noción de esquemas, es la estructura sobre la que se edificaron la mayoría de los trabajos teóricos sobre algoritmos genéticos en las décadas siguientes.

En estos últimos años se ha generado una amplia interacción entre los investigadores de varios métodos de computación evolutiva, rompiéndose las fronteras entre algoritmos genéticos, estrategias evolutivas y programación evolutiva. Como consecuencia, en la actualidad, el término “algoritmo genético” se utiliza para designar un concepto mucho más amplio del que concibió Holland.

Capítulo I: Aspectos Teóricos

1.1 Definición de Algoritmo Genético

Los objetivos que perseguían John Holland eran dos: primero abstraer y explicar rigurosamente el proceso adaptativo de los sistemas naturales; segundo diseñar sistemas artificiales que retuvieran los mecanismos más importantes de los sistemas naturales. En este sentido, podemos dar una definición de los algoritmos genéticos:

*Los **Algoritmos Genéticos** son **Algoritmos de Búsqueda Metaheurística** basados en los mecanismos de selección natural y genética natural. Combinan la supervivencia de los más compatibles entre las estructuras de cadenas, con una estructura de información ya aleatorizada, intercambiada para construir un algoritmo de búsqueda con algunas de las capacidades de innovación de la búsqueda humana [5].*

El tema central en las investigaciones sobre algoritmos genéticos, ha sido la robustez, el equilibrio necesario entre la eficiencia y la eficacia suficiente para la supervivencia en entornos diferentes. Las implicaciones que tiene la robustez en los sistemas artificiales son variadas. Si se puede conseguir que un sistema artificial sea más robusto, se podrán reducir, e incluso eliminar, los costes por rediseños. Y si se es capaz de lograr niveles altos de adaptación, los sistemas podrán desarrollar sus funciones mejor y durante más tiempo. Sin embargo, ante la robustez, eficiencia y flexibilidad de los sistemas biológicos, sólo podemos sentarnos a contemplar, y maravillarnos; mentiríamos si dijéramos que somos capaces de igualarlos.

Podríamos plantearnos la pregunta de porqué deberíamos basarnos en nuestros conocimientos sobre la evolución biológica. La respuesta la encontramos si observamos una constante que se repite en muchos problemas: la búsqueda de soluciones entre una cantidad ingente de candidatos. Veámoslo con un ejemplo, el cálculo de un conjunto de reglas o ecuaciones capaz de regir las subidas y bajadas de un mercado financiero. El modo de llegar a la mejor solución en estas situaciones, pasa por ser capaz de obtener rendimiento de un uso eficaz del paralelismo, que permita explorar diferentes posibilidades de modo simultáneo. Para ello, se precisa, tanto paralelismo computacional (contar

con varios procesadores computando al mismo tiempo), como una estrategia adecuada de búsqueda.

Otra buena razón para adoptar los algoritmos genéticos pasa por los problemas computacionales, estos suelen precisar de un programa adaptativo, capaz de comportarse bien ante cambios en el entorno. Además, la mayoría de estos problemas tienen soluciones complejas, muy difíciles de programar a mano. La computación evolutiva puede ser la respuesta, mediante la selección natural, con variaciones debidas a cruces y/o mutaciones. Su objetivo es el diseño de soluciones de alta calidad para problemas de elevado grado de complejidad, y la habilidad de adaptar esas soluciones de cara a cambios en el entorno.

La evolución, tal y como la conocemos, es básicamente un método de búsqueda entre un número enorme de posibles soluciones. En biología las posibilidades están formadas por un conjunto de secuencias genéticas posibles, y las soluciones deseadas, por organismos capaces de sobrevivir y reproducirse en sus entornos. La evolución puede verse, asimismo, como un modo de diseñar soluciones a problemas complejos, con la capacidad de innovar. Estos son los motivos de que los mecanismos evolutivos sean una fuente de inspiración para los algoritmos de búsqueda. Por supuesto, el buen funcionamiento de un organismo biológico depende de muchos criterios, que además varían a medida que el organismo evoluciona, de modo que la evolución está “buscando” continuamente entre un conjunto cambiante de posibilidades. Por ello, podemos considerarla como un método de búsqueda masivamente paralelo, ya que evalúa y cambia millones de especies en paralelo.

1.2 Algoritmos Genéticos

Para poder entender los conceptos básicos en que se sustentan los Algoritmos Genéticos, es necesario conocer algunos conceptos claves de la teoría de la selección natural. La idea principal en la teoría Neo-Darwinista, es que la compleja y precisa adaptación al entorno que observamos en los seres vivos en el planeta, puede explicarse mediante los siguientes mecanismos:

- **Herencia**: Consiste en la capacidad de un individuo de transmitir sus características a su progenie. Esta información es codificada en todas las especies en los genes.

- **Mutación**: Es un cambio permanente y transmisible en la información genética de un individuo. Puede deberse a un error en el copiado de la información genética durante el proceso de la reproducción, a la exposición a radiaciones, agentes químicos, o a virus. Las mutaciones pueden ser perjudiciales, benéficas o neutras.

- **Selección**: Los individuos de todas las especies pasan por un proceso de selección natural, el cual puede dividirse en dos categorías, selección ecológica, que se refiere a la capacidad del individuo de sobrevivir a su entorno lo suficiente como para reproducirse; y selección sexual, la que ocurre cuando un organismo es sexualmente más atractivo al sexo opuesto favoreciendo que sus características se propaguen con mayor éxito en la población.

- **Adaptación**: Mediante el proceso de selección los organismos se van volviendo mejor adaptados a sus entornos, y a este proceso que incrementa la aptitud de una especie a un entorno determinado se le llama adaptación. La adaptación depende del contexto, una mejora en un entorno determinado puede ser perjudicial en otro. En la naturaleza no hay algo como una meta hacia la que se dirija la evolución, simplemente se responde al entorno mediante cambios adaptativos.

- **Reproducción** (**recombinación**): Asegura que la información genética que codifica a un organismo sea pasado entre distintas generaciones, al tiempo que proporciona un medio de recombinar esa información esparciendo las modificaciones en la población, de acuerdo a la selección natural.

Un Algoritmo Genético, al ser una simulación de la teoría evolutiva, requiere trasladar estos mecanismos biológicos a mecanismos computacionales. Para ello, al formular un algoritmo genético se deben establecer una serie de componentes, los cuales permitirán la evolución de una población de posibles soluciones para un problema determinado.

Básicamente, el Algoritmo Genético funciona como sigue: en cada generación, se crea un conjunto nuevo de “criaturas artificiales” (cadenas) utilizando bits y partes más adecuadas del progenitor. Esto involucra un proceso aleatorio que no es, en absoluto, simple [5]. La novedad que introducen los Algoritmos Genéticos es que explotan eficientemente la información histórica para especular sobre nuevos puntos de búsqueda, esperando un funcionamiento mejorado.

1.2.1 Componentes de los Algoritmos Genéticos

Representación y Codificación de las Soluciones

Los Algoritmos Genéticos no trabajan directamente sobre las soluciones del problema a resolver, sino sobre representaciones de las mismas. Por lo anterior, se debe definir cuál es el dominio de las soluciones al problema, y traducirlo en estructuras manejables por el algoritmo genético.

A esta traducción se le llama *codificación de las soluciones*, siendo el proceso inverso igualmente relevante, al obtener la solución del problema a partir de la estructura genética. Generalmente se usa como representación a cadenas binarias o cadenas de bits, compuestas de ceros y unos (0 y 1), siendo cada cadena la representación de una solución potencial del problema. En el contexto de los Algoritmos Genéticos, a cada una de estas cadenas se les llama cromosomas, individuos o código genético.

La Función de Aptitud (Función Fitness)

Entre los organismos vivos, siempre hay individuos que por sus características son más capaces que otros para sobrevivir y reproducirse, los cuales tienen más probabilidades de propagar sus genes a la siguiente generación. Este nivel de aptitud siempre se mide en función del entorno, es decir, es relativo al resto de la población y depende del entorno en que se ubique el organismo.

De esta misma manera, en los Algoritmos Genéticos es necesario poder medir el nivel de aptitud que tiene un individuo para resolver el problema de que se trate, para ello cada cromosoma de la población recibe un valor que describe dicha aptitud.

A la función que permite asignar este valor a cada individuo de la población, se le llama *Función de Fitness* o *Función de Aptitud*, y su evaluación puede ser lo más costoso del algoritmo, en términos de tiempo y de recursos de cómputo. El diseño de esta función es de fundamental importancia, ya que representa al medio ambiente al que se irá adaptando la población.

Operadores Genéticos más Usuales

Selección

En general, los organismos vivos tienen posibilidades de reproducirse de acuerdo a su nivel de adaptación al medio, y es aquí donde la selección natural entra en juego, favoreciendo la perpetuación de las características que proporcionen alguna ventaja a la especie con respecto al entorno. De manera similar en los algoritmos genéticos una vez que se puede calificar la aptitud de todos los individuos de una población, se debe de contar con alguna manera de favorecer a los individuos con buenas características para incrementar la aptitud de la población para la siguiente generación.

Este operador es la parte más importante de un Algoritmo Genético, ya que éste es el encargado de "Seleccionar" a los candidatos que se reproducirán entre ellos.

Este proceso u operador suele realizarse de forma probabilística, ya que determina de qué manera se elegirán los individuos para recombinarse y generar a un miembro de la siguiente generación.

Las políticas de selección más comunes son:

- *Selección proporcional*: Los individuos son seleccionados aleatoriamente, siendo la probabilidad de ser seleccionados proporcional a la aptitud de cada individuo. El método de este tipo más conocido es el de *selección por ruleta*, propuesto originalmente en Holland 1975 [4].

- *Selección ordinal*: Se elige un grupo de individuos al azar, entre los cuales se establece un orden para determinar quienes se elegirán para reproducirse, usando sus valores de aptitud, un ejemplo de este tipo lo constituye la selección por torneo, propuesta por Wetzel en [14].

- *Selección elitista*: Se utiliza en combinación con alguna de las anteriores, pero manteniendo un cierto número de los mejores cromosomas de generación en generación, o favoreciendo de manera no probabilística a los individuos más aptos sobre los menos aptos. Un ejemplo de este tipo de selección lo constituye la selección de estado uniforme, diseñada por Whitley [15], donde se reemplazan a los individuos menos aptos de la generación, por los más aptos de la nueva generación.

Cruza

La cruza, en los sistemas biológicos, es un proceso (en ocasiones demasiado complejo) que permite la reproducción sexual, lo cual conlleva a que se mezcle material genético de los progenitores (combinación de cromosomas), al tener descendencia.

Cuando se habla de cruzamiento en Algoritmos Genéticos, esto significa que dos individuos mezclan sus cadenas para generar nuevos individuos de la siguiente generación, de esta manera se consigue una buena exploración del espacio de soluciones del problema.

Existen diferentes maneras de llevar a cabo esta mezcla, a continuación se hacen referencias de algunas formas de cruzamiento:

- *Cruza en un punto*: (propuesta en la formulación clásica de Goldberg [12]) En este tipo de cruza, se obtienen los descendientes a partir de los padres, seleccionando al azar un punto de corte en ellos, e intercambiando los trozos de cromosoma para formar a los nuevos individuos.

- *Cruza en n-puntos*: En este tipo de mezcla, se eligen *n* puntos de corte en las cadenas padres, y a partir de cada una de esas posiciones el material genético es intercambiado entre las cadenas.

- *Cruzamiento uniforme*: En donde los bits individuales en las cadenas son comparadas entre los dos padres. Los bits son intercambiados con una probabilidad fija, por lo general del 50%.

- *Cruza semi-uniforme*: Exactamente la mitad de los bits no coincidentes son intercambiados; implica calcular inicialmente la distancia de Hamming, (es decir, el número de bits no iguales), este número se divide entre dos y el resultado es el número de bits no coincidentes en ambos padres que serán intercambiados.

- *Cruza cortar y empalmar:* Conlleva un cambio en la longitud de las cadenas hijos, esto se debe a que cada padre elige de forma separada sus puntos de cruce, y en general estos son diferentes. Se elige un punto distinto aleatoriamente en cada uno de los padres a partir de donde se hace el intercambio genético.

Mutación

En los seres vivos el material genético puede sufrir cambios permanentes y transmisibles, ocasionando cambios en los organismos que codifica. Estas modificaciones de los genes pueden ser causados por errores en el copiado o por factores externos; las mutaciones constituyen la forma en que una especie va adquiriendo nuevas características.

En la evolución, una *mutación* es un suceso bastante poco común (sucede aproximadamente una de cada mil replicaciones). En la mayoría de los casos las mutaciones son letales, pero en promedio, contribuyen a la diversidad genética de la especie. En un algoritmo genético tendrán el mismo papel, y la misma frecuencia (es decir, muy baja) [8].

Una vez establecida la frecuencia de mutación, por ejemplo, uno por mil, se examina cada bit de cada cadena cuando se vaya a crear la nueva criatura a partir de sus padres (normalmente se hace de forma simultánea al crossover). Si un número generado aleatoriamente está por debajo de esa probabilidad, se cambiará el bit (es decir, de 0 a 1 o de 1 a 0). Si no, se dejará como está. Dependiendo del número de individuos que haya y del número de bits por individuo, puede resultar que las mutaciones sean extremadamente raras en una sola generación.

1.2.2 Algoritmo Genético Simple

El punto de referencia básico al diseñar algoritmos genéticos fue establecido por David Goldberg en [5], al proponer un algoritmo genético que pudiera servir como punto de partida para cualquiera interesado en utilizar esta técnica. Dada la importancia que ha tenido esta obra en la difusión y popularización de estos algoritmos, hacer mención de este modelo es importante como parte del diseño básico de algoritmos genéticos. La Figura 1 muestra las etapas de ejecución del Algoritmo Genético Simple o AGS.

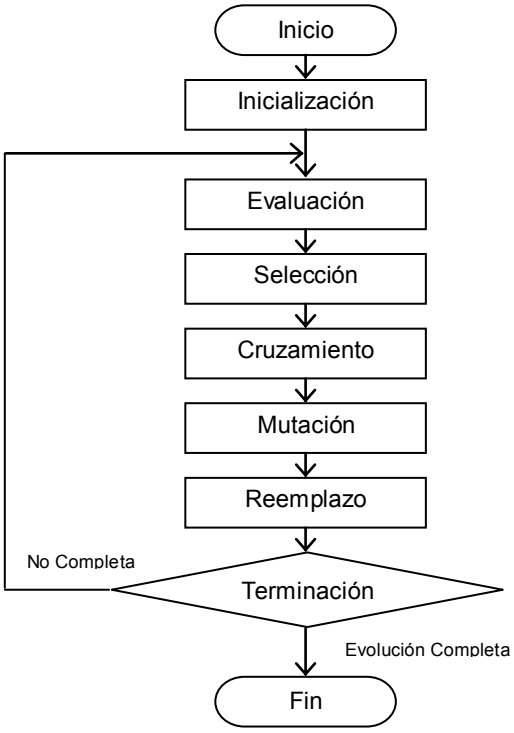


Figura 1: Diagrama de Flujo de un Algoritmo Genético Simple

Pseudocódigo:

El siguiente es el pseudocódigo del Algoritmo Genético Simple, también denominado Canónico [9].

```
BEGIN
  Generar una población inicial
  Computar la función evaluación de cada individuo

  // Producir nueva generación si no se ha terminado de evolucionar

  WHILE NOT Terminado_evolucion DO BEGIN
   /*Ciclo reproductivo*/
   FOR Tamaño población/2 DO T BEGIN

     Seleccionar dos individuos de la generación anterior,
     para el cruce (probabilidad de selección proporcional
     a la función de evaluación del individuo)

     Cruzar con cierta probabilidad los dos individuos
     obteniendo dos descendientes

     Mutar los dos descendientes con cierta probabilidad

     Computar la función de evaluación de los dos
     descendientes mutados

     Insertar los dos descendientes mutados
     en la nueva generación

  END /*end for*/

  IF la población ha convergido THEN
   Terminado := TRUE

END /*end while*/

END
```

1.2.3 Ventajas y Desventajas de lo Algoritmos Genéticos vs. Otras Técnicas de Búsqueda

Ventajas

- No necesitan conocimientos específicos sobre el problema que intentan resolver.

- Operan de forma simultánea con varias soluciones, en vez de trabajar de forma secuencial como las técnicas tradicionales.
- Cuando se usan para problemas de optimización (maximizar una función objetivo) resultan menos afectados por los máximos locales (falsas soluciones) que las técnicas tradicionales.

- Resulta sumamente fácil ejecutarlos en las modernas arquitecturas masivas en paralelo.

- Usan operadores probabilísticos, en vez de los típicos operadores determinísticos de las otras técnicas.

Desventajas

- Pueden tardar mucho en converger, o no converger en absoluto, dependiendo en cierta medida de los parámetros que se utilicen: tamaño de la población, número de generaciones, etc.

- Pueden converger prematuramente [6].

1.3 Algoritmos Genéticos Paralelos

1.3.1 Computación Paralela

Un **programa es paralelo** si en cualquier momento de su ejecución puede ejecutar más de un proceso. Para crear programas paralelos eficientes hay que poder crear, destruir y especificar procesos así como la interacción entre ellos [16]. Básicamente existen tres formas de paralelizar un programa:

- *Paralelización de Grano Fino*: la paralelización del programa se realiza a nivel de instrucción. Cada procesador hace una parte de cada paso del algoritmo (selección, cruce y mutación) sobre la población común.

- *Paralelización de Grano Medio*: los programas se paralelizan a nivel de bucle. Esta paralelización se realiza habitualmente de una forma automática en los compiladores.

- *Paralelización de Grano Grueso*: se basan en la descomposición del dominio de datos entre los procesadores, siendo cada uno de ellos el responsable de realizar los cálculos sobre sus datos locales.

La **computación paralela** se ha convertido en una parte fundamental en muchas áreas del cálculo científico, ya que permite la mejora del rendimiento simplemente con la utilización de un mayor número de procesadores, núcleos, memorias y la inclusión de elementos de comunicación que permitan a los procesadores trabajar conjuntamente para resolver un determinado problema.

Arquitecturas paralelas

Los **sistemas de cómputo paralelos** constituyen un conjunto de procesadores interconectados, ya sea en un sólo equipo o distribuidos en varios. El tipo de esta interconexión y la forma de operar de este conjunto de procesadores ha dado lugar a diferentes tipos de clasificaciones de sistemas paralelos, siendo una de las más aceptadas la taxonomía de arquitecturas de computadoras propuesta por Flynn [16]. Esta clasifica a las

computadoras en 4 tipos, basándose en el número de instrucciones y en los flujos de datos que es capaz de manejar la arquitectura. Los modelos descritos por Flynn son:

- **SISD** (Single Instruction Single Data): Una única instrucción y un único flujo de datos: Una computadora que no utiliza paralelismo ni en las instrucciones ni en los datos, un ejemplo es la computadora personal con un sólo procesador.

- **MISD** (Multiple Instruction Single Data): Múltiples instrucciones y un único flujo de datos: Una arquitectura inusual dado que múltiples instrucciones por lo general requieren múltiples datos sobre los cuales operar, su aplicabilidad es muy limitada por lo que no se han producido en gran escala.

- **SIMD** (Single Instruction Multiple Data): Una única instrucción y un flujo múltiple de datos): Las arquitecturas de este tipo son conocidas como arreglos de procesadores, los cuales ejecutan el mismo conjunto de instrucciones sobre datos diferentes.

- **MIMD** (Multiple Instruction Multiple Data): Múltiples instrucciones y flujo múltiple de datos: Esta arquitectura es la más utilizada, los sistemas distribuidos se ejecutan sobre este modelo de computadora, ya sea utilizando el modelo de memoria compartida o de memoria distribuida. En este caso, los procesadores son autónomos en cuanto a las operaciones que realizan, y procesan distintos datos cada uno.

Existe otra forma de clasificar a los sistemas paralelos es de acuerdo a su arquitectura de memoria, de acuerdo a este criterio existen dos tipos de sistemas:

- **Computadoras paralelas de memoria compartida,** en las que los procesadores acceden a la misma memoria global, accesible a todos, por lo que los mecanismos de sincronización entre procesadores deben establecerse de forma explícita. Estos sistemas tienen la ventaja de que la comunicación entre procesadores es muy rápida, pero un problema potencial es que el bus de datos común para acceder a la memoria es un cuello de botella potencial, para evitar esto se suelen utilizar memorias caché, para que cada procesador pueda recuperar con rapidez los datos que utilice con mayor frecuencia; esto trae como consecuencia que haya que mantener dichos cachés sincronizados, para evitar inconsistencias en los datos. Estas consideraciones hacen que los circuitos de este tipo de computadoras sean

muy complejos y que el número de procesadores en paralelo esté limitado en la práctica por dicha complejidad.

- **Computadoras paralelas de memoria distribuida**, en los cuales cada procesador tiene su propia memoria particular, y los procesadores para comunicarse entre sí, utilizan el envío de mensajes. Este esquema puede admitir a cientos, hasta miles de procesadores interconectados, con el inconveniente de que los tiempos de comunicación son más lentos que los del esquema de memoria compartida.

Herramientas para procesamiento paralelo

Casi todas las herramientas existentes para paralelizar un algoritmo son del tipo de paso de mensajes, dado que este método de comunicación es el usado en sistemas de memoria distribuida, los cuales son los de uso más extendido. El paradigma de paso de mensajes consiste en que los procesadores se comuniquen entre ellos mediante el envío/recepción de mensajes enviados por un medio dedicado o no dedicado. Las instrucciones básicas en este modelo son el *send*, que define un espacio de memoria a ser enviado y el destinatario, y el *receive*, el cual define el espacio de memoria en el que se almacenará el dato recibido.

Al nivel más bajo de paso de mensajes tenemos a la interfaz de **sockets BSD**, las cuales permiten establecer una comunicación en ambos sentidos, con TCP o con UDP, con la desventaja de que la programación es propensa a errores, y requiere comprensión por parte del programador de la red a bajo nivel.

- **Parallel Virtual Machine** (PVM): Es un sistema de software que, permite a un conjunto de computadoras heterogéneas comportarse como una sola computadora paralela distribuida. Sus principales ventajas son su interoperabilidad y su tolerancia a fallos. Fue diseñado bajo el paradigma de proporcionar una máquina virtual al usuario.

- **La interfaz de paso de mensajes MPI** (Message Passing Interface): es un estándar definido por un grupo de personas pertenecientes a organizaciones gubernamentales, académicas e industriales, el cual especifica una serie de instrucciones para la comunicación de sistemas paralelos sin detallar como deben ser imple-

mentados para una plataforma en particular. Se ha vuelto el estándar de facto para la comunicación entre procesadores distribuidos en paralelo, debido a su eficiencia, pero también por ser producto del esfuerzo conjunto de múltiples organizaciones y por la disponibilidad de distintas implementaciones libres. En este trabajo se utiliza esta herramienta en su calidad de estándar en las investigaciones sobre algoritmos evolutivos en paralelo.

1.3.2 Los Algoritmos Genéticos y la Paralelización

Se puede decir que los Algoritmos Genéticos tienen una estructura que se adapta perfectamente a la paralelización. De hecho la evolución natural es en si un proceso paralelo, ya que evoluciona utilizando varios individuos. Los principales métodos de paralelización de Algoritmos Genéticos consisten en la división de la población en varias subpoblaciones. El tamaño y distribución de la población entre los distintos procesadores será uno de los factores fundamentales a la hora de paralelizar el algoritmo.

Existen varias *formas de paralelizar un Algoritmo Genético*. A continuación se enuncian de forma informal algunas realizaciones:

- La primera y más intuitiva es la global, que consiste básicamente en paralelizar la evaluación de los individuos manteniendo una población.

- Otra forma de paralelización global consiste en realizar una ejecución de distintos AG's secuenciales simultáneamente (estas dos formas de paralelización no cambian la estructura del algoritmo utilizado). El resto de las aproximaciones sí cambian la estructura del algoritmo y dividen la población en subpoblaciones que evolucionan por separado e intercambian individuos cada cierto número de generaciones. Si las poblaciones son pocas y grandes, tenemos la paralelización de grano grueso. Si el número de poblaciones es grande y con pocos individuos en cada población tenemos la paralelización de grano fino.

- Por último, existen algoritmos que mezclan propiedades de estos dos últimos y que se denominan mixtos.

Además de conseguir tiempos de ejecución menor, al paralelizar un Algoritmo Genético se está modificando el comportamiento algorítmico, y esto hace que se puedan obtener otras soluciones y experimentar con las distintas posibilidades de implementación y los distintos factores que influyen en ella. Estas poblaciones van evolucionando por separado para detenerse en un momento determinado e intercambiar los mejores individuos entre ellas.

Técnicamente hay 3 características importantes que influyen en la eficiencia de un algoritmo genético paralelo:

- La topología que define la comunicación entre subpoblaciones.
- La proporción de intercambio: número de individuos a intercambiar.
- Los intervalos de migración: periodicidad con que se intercambian los individuos.

1.3.3 Algoritmos Genéticos Paralelos

Llevar un algoritmo genético a una versión paralela del mismo tiene como objetivo mejorar su rendimiento, así como la calidad de la búsqueda. Para realizar esto existen diferentes maneras de dividir las tareas del algoritmo. Para la clasificación de los algoritmos genéticos en paralelo, se toman en cuenta 3 conceptos fundamentales: el modelo paralelo, la distribución de la población y la implementación.

Pueden ser clasificados en tres categorías; aquellos que tienen un modelo maestro-esclavo o globales, un modelo de isla (o Algoritmo Genético Distribuido) o también conocido como de población múltiple de grano burdo[1] y un modelo de células o de población única de grano fino.

- Los **algoritmos globales:** Utilizan el mismo Algoritmo Genético Secuencial, pero se dividen las evaluaciones de los individuos en varios procesadores y probable-

[1] El "tamaño de grano" en paralelismo se refiere a la razón entre el tiempo empleado en cálculos y el tiempo empleado en comunicación. Cuando mayor es tiempo gastado en cálculos que en comunicación) el procesamiento es llamado de grano burdo o coarse grained; cuando es pequeña (mucha comunicación, poco procesamiento), al procesamiento se le nombra de grano fino o fine grained.

mente, la aplicación de los operadores genéticos en la población, es decir, los cromosomas a evaluar se reparten entre el número de procesadores disponibles. Los resultados se reportan al procesador maestro que realiza todas las otras operaciones del AG. Este esquema aporta bastante beneficio en speedup[2] porque generalmente la evaluación de la función objetivo es la parte del algoritmo que consume más tiempo de procesador. Sólo la evaluación se distribuye entre los diferentes procesadores, las operaciones restantes se efectúan de manera centralizada.

- **Modelo de células** o Algoritmos Genéticos de grano fino: El tamaño de las subpoblaciones es más pequeño, pero existe mayor comunicación entre los procesadores. Normalmente se utiliza un individuo por procesador, aunque dependiendo del número de procesadores pueden insertarse varios en un sólo procesador. Cada procesador se relaciona con ciertos procesadores, formando un vecindario, el cual delimita la subpoblación donde se aplican los operadores de cruce y selección. Los vecindarios pueden traslaparse, por lo que existe intercambio de información genética entre ellos. Este modelo requiere de computadoras masivamente paralelas.

- **Modelo distribuido** o los Algoritmos Genéticos de grano burdo**:** Buscan disminuir el tiempo de convergencia del algoritmo sin hacer uso intensivo del canal de comunicación. El Modelo de Islas Paralelo evalúa la población en paralelo y busca dividir la población total en subpoblaciones de acuerdo al número de procesadores que se tengan disponibles en igual proporción y dividir por igual el número de cadenas que forman la población total. Cada una de estas subpoblaciones puede entonces ejecutar el AG de manera normal en un nodo distinto y evolucionar de manera independiente; pero ocasionalmente, a intervalos de tiempo determinados o número de evaluaciones, tal vez cada 5 generaciones más o menos, las subpoblaciones comparten o intercambian unas cuantas cadenas que siempre son los mejores individuos de cada subpoblación. Esto es conocido como el operador de migración. Para que un elemento migre se convierte el individuo a una codificación neutra independiente tanto de la plataforma como del propio algoritmo. Este paso

[2] Speedup es un término común en computación paralela, se refiere a qué tanto un algoritmo paralelo es más rápido que su equivalente secuencial. En español, seria algo como “aceleramiento”.

es muy importante, no sólo por el hecho de que se tengan codificaciones distintas del genoma, sino porque máquinas distintas pueden tener distintas formas de codificar los datos. En este caso la búsqueda en cada subconjunto es un poco diferente, debido a que las poblaciones iniciales imponen una tendencia de muestreo que causa una trayectoria diferente por el espacio de búsqueda. Al introducir la migración, el Modelo de Isla es capaz de explotar las diferencias en los diferentes subconjuntos; si un gran número de cadenas migra en cada generación, ocurre una mezcla global y las diferencias locales entre islas serán expulsadas. Si la migración no es frecuente, puede ser insuficiente para prevenir de cada subconjunto pequeño una convergencia prematura. Este modelo también es eficiente para un sólo procesador. Para afinar el grado de acoplamiento entre las subpoblaciones se pueden usar dos mecanismos importantes:

- **Reducción incestuosa durante la migración**: De entre los inmigrantes sólo puede cruzarse aquel inmigrante con distancia mayor del mejor de la población de destino; los restantes se descartan. En el caso de codificaciones binarias se puede emplear la distancia de Hamming. Este sistema premia la inmigración de los distintos, y sigue un mecanismo de xenofilia además de que aumenta el acoplamiento entre las subpoblaciones.

- **Migración de chusma**: Se le conoce también como migración de multitud. Este mecanismo hace exactamente lo contrario al que se acaba de describir ya que migra el más cercano por una distancia prefijada. Se correspondería con subpoblaciones xenofóbicas, ya que de entre los migrantes solamente aceptan aquel que mas se parece a la población de destino.

Después de la migración, el AG mezclará a los inmigrantes con el resto de los individuos de la subpoblación destino. Entonces en los AG distribuidos, una vez que la población es dividida, el número de individuos de cada subpoblación es pequeño, lo que provoca que el algoritmo converja rápidamente.

Este modelo de islas, con una tasa de migración razonable y con un criterio de emigración también razonable (tanto el criterio de “el mejor” como el de selección

"por torneo" funcionan perfectamente) tiene como ventajas acelerar el algoritmo de búsqueda y evitar la convergencia genética prematura.

1.4 Ejemplos de aplicación de Algoritmos Genéticos

Los algoritmos genéticos tienen múltiples aplicaciones en la investigación y en el diseño industrial, en esta sección se muestran interesantes aplicaciones reales de esta técnica en la investigación y en la industria.

Turbinas para motor a reacción

En Holland 1992 [17], se describe un algoritmo genético desarrollado en conjunto por la General Electric y el Rensselaer Polytechnic Institute. Dicho algoritmo arrojó un diseño de una turbina de un motor a reacción, tres veces más eficiente que la diseñada por ingenieros humanos, y 50% mejor que otra solución obtenida con un sistema experto que recorrió 10,387 posibles soluciones para brindarla. Este tipo de proyectos de diseño pueden durar hasta 5 años, mientras el algoritmo genético obtuvo esta solución en dos días. Aunque, para ser justos, no se menciona el tiempo de desarrollo y diseño del propio algoritmo.

Optimización de circuito integrado

Texas Instruments optimizó la ubicación de los componentes de un circuito integrado, utilizando un algoritmo genético, según se reporta en Begley 1995 [18]. El algoritmo encontró una configuración de las conexiones que permitió una reducción del 18% del espacio requerido.

Diseño de brazo espacial

En Keane y Brown 1996 [19], los autores describen la utilización de un algoritmo genético para diseñar un brazo de carga para ser utilizado en el espacio, con satélites y estaciones espaciales. Como resultado se obtuvo una estructura con la misma cantidad de material que un diseño convencional, pero con características muy superiores en lo que se refiere a amortiguar vibraciones perjudiciales. Los investigadores hacen notar que

el algoritmo sólo se ejecutó durante 10 generaciones (por lo costoso de la simulación), y la población aún no había convergido.

Predicción financiera

Se reporta en [20], la comparación de un algoritmo genético contra una red neuronal en la predicción del comportamiento de 1600 acciones. Al final el algoritmo genético produjo un beneficio del 5.47% contra el 4.4% de la red neuronal.

En este mismo campo, se tienen reportes de que las empresas de correduría bursátil utilizan algoritmos genéticos para la toma de decisiones de inversión. Concretamente en Coale 1997 [21], el autor escribe que First Quadrant una empresa de inversiones estadounidense, con inversiones superiores a los 2,200 millones de dólares, utiliza algoritmos genéticos en este sentido. Al igual que LBS Capital Management, otra empresa estadounidense, que administra fondos de pensiones con la ayuda de algoritmos genéticos [22].

Tácticas militares

También se han aplicado los algoritmos genéticos a situaciones militares. Kewley y Embrechts 2002 [23], reportan haber utilizado algoritmos genéticos para desarrollar planes tácticos para batallas militares. En este caso se utilizó la coevolución para mejorar la calidad de las soluciones, la estrategia evolucionaba conforme los planes del enemigo también lo hacían. Para medir la calidad de las soluciones, se realizaron comparaciones con estrategias diseñadas por expertos militares. El estudio concluyó que los planes generados por el algoritmo, tenían una efectividad significativamente mayor que los generados por los expertos.

Investigación de operaciones

United Distillers and Vinters, la mayor y mas rentable distribuidora de licores del mundo, utiliza un algoritmo genético para la distribución y almacenamiento de su inventario, según se reporta en Lemley 2001 [24].Gracias al programa creado por el algoritmo la eficiencia del almacenamiento casi se ha duplicado.

Los algoritmos genéticos no son una panacea para resolver cualquier tipo de problema, tienen aplicación en múltiples áreas, lo cual es evidente con los ejemplos mostrados, pero, estos también pueden ser resueltos bajo otras técnicas heurísticas, ya sea con: *Recosido Simulado*, *Algoritmo de Balas*, *Colonia de Hormigas*, etc.

1.5 Problema de la Mochila Multidimensional

Dentro de la programación matemática, los problemas que involucran que el dominio de las variables sean binarios, aparecen con mucha frecuencia. La formulación más general es la que se muestra en la ecuación (1):

$$\begin{aligned} &\min(\max) \quad z = c^t x \\ &sa: Ax \le b \\ &A \in M_{mxn}, b \in R^n, x \in (0,1)^n, c \in R^n \end{aligned} \tag{1}$$

Si los coeficientes de la matriz A son todos positivos, se le denomina el Problema de la Mochila Multidimensional (MKP), profundamente abordado por la comunidad de investigadores, ya que los algoritmos exactos para resolver este problema pertenecen a la clase NP.

Se han propuesto algoritmos exactos, uno de los más relevantes ha sido el algoritmo de Balas, quien en [33, 34, 35] propuso un algoritmo aditivo para resolver esta clase de problemas.

Se han diseñado e implementado también algoritmos que aprovechan las características de la matriz de restricciones, tal es el caso de Hillier [36], Kochenberg, McCarl y Wymannn [37], Senju y Toyoda [38] y Zanakis [39].

Fontanari en [40], realizó un análisis estadístico del MKP, investigó la dependencia de la función objetivo con las capacidades de la mochila y su número de restricciones, en el caso específico cuando a los n objetos le son asignados el mismo valor de ganancia y los coeficientes de la función objetivo son distribuidos uniformemente sobre el intervalo.

También el problema MKP ha sido abordado usando algoritmos heurísticos, entre los autores más destacados en heurísticas basadas en límites se encuentran Balas y

Martín [34], los cuales usaron la programación lineal para obtener cotas y disminuir las restricciones. En 1968 Senju y Toyoda [37] publicaron la heurística dual, la cual inicia con una solución que contiene sólo 1's; entonces de acuerdo a reglas heurísticas, a diferentes variables, se les van asignando 0's sucesivamente, para así obtener una solución factible.

Pirkul [41], presentó un algoritmo heurístico el cual hace uso de la dualidad sustituta. Las *m* restricciones de la mochila son transformadas en restricciones únicas, usando multiplicadores sustitutos. Volgenant y Zoon [42] extendieron la heurística de Magazine y Oguz en dos maneras: la primera en cada paso se calculan los valores de los multiplicadores de Lagrange, al final del procedimiento, se calcula el límite superior para cada solución evaluada, cambiando algunos de los valores de ciertos multiplicadores.

En lo referente a las heurísticas de búsqueda Tabú, se pueden mencionar un gran número de publicaciones como las hechas por Dammeyer y Voss [43], las cuales se basan en la eliminación de reversa, o las de Aboudi y Jörnsten [44] que combinaron la búsqueda Tabú con la heurística de pivote de Balas y Martín [35, 34], en una heurística para la programación entera 0-1. Battiti y Tecchiolli [45] presentaron una heurística basada en la búsqueda Tabú reactiva.

Este problema ha sido también abordado usando algoritmos genéticos, los resultados más importantes han sido introducidos por Rudolph y Sprave [46, 47], donde la selección de los padres no está restringida, como en los algoritmos genéticos estándar, pero se restringe para las soluciones vecinas. Las soluciones no factibles son penalizadas como lo establecen Khuri, Bäck, y Heitkötter [48]. Otro tipo de heurísticas incluyen las investigaciones de Glover [49] que mejora la heurística dual de Senju y Toyoda, tiene una calidad superior en sus resultados para la generación aleatoria de problemas con tamaños mayores para m=20 y n=100. Hanafi, Freville y Abedellaoui [50] presentaron un algoritmo simple de multiniveles (SMA) en el cual se pueden usar un número de procedimientos de búsqueda local.

Los últimos resultados en la aplicación de algoritmos genéticos los tiene P.C. Chu y J.E. Beasley en [26, 25], que presentaron una heurística basada en algoritmos genéticos. En general las componentes de su algoritmo son comparadas a las usadas en los algoritmos genéticos estándar; la diferencia radica en que los operadores incorporados

incluyen técnicas de aprendizaje, las cuales garantizan que todas las soluciones descendientes sean factibles. En este trabajo también se demostró que dicha heurística es capaz de obtener soluciones muy cercanas al óptimo para problemas de diferentes características; y requieren una pequeña cantidad de esfuerzo computacional para llegar a la solución.

En lo que concierne a Algoritmos Metaheuristicos Paralelos que dan solución al problema MKP se tiene el trabajo desarrollado por Stefka Fidanova [51], en el que propone el diseño paralelo de un algoritmo basado en colonia de hormigas, arrojando muy buenos resultados al ser comparados con algoritmos secuenciales, la única limitante presentada fue en el cómputo de las función objetivo.

Otro trabajo importante es el desarrollado por Puchinger [52], donde se propone un algoritmo cooperativo memético con poda y corte, el algoritmo planteado se ejecuta de forma paralela y continua en un intercambio bidireccional de información, utilizando además un algoritmo de codificación binaria directa de candidatos y de reparación de soluciones, donde se mejora la búsqueda, sin embargo la implementación realizada fue utilizando concurrencia y no paralelismo completamente, obteniendo de esa manera resultados limitados, aún se encuentra mejorando su aplicación.

Otro de los algoritmos paralelos desarrollados ha sido el presentado por Miyagi, Hayato [53], donde se paralelizó un algoritmo genético secuencial para poder conocer qué tipo de topología (comunicación entre nodos) es la más idónea para resolver el problema MKP. En ese trabajo sólo se analizó la calidad de la solución del problema MKP contra la idoneidad de la topología de comunicación. Las topologías analizadas fueron: la estrella, lineal y de tipo árbol binario balanceado. La conclusión obtenida en este trabajo arrojó que la topología en línea (o conocida como bus) puede mantener una variedad de cromosomas (variedad de soluciones factibles), dando esto una mejor calidad de soluciones, debido a que esta topología tiene mayor distancia entre el nodo raíz al nodo hoja.

Capítulo II: Definición del Problema y Algoritmos Propuestos.

2.1 Planteamiento del Problema

En nuestra Facultad de Ciencias de la Computación (BUAP) se tiene un grupo de investigadores que desarrollan algoritmos paralelos. Se han desarrollado diversas aplicaciones, en particular para darle solución al problema de la mochila multidimensional, por ejemplo, se dispone de una implementación del algoritmo de Balas siguiendo un esquema de comunicación Maestro-Esclavo [25], sin embargo los resultados obtenidos no han sido muy buenos, ya que para problemas de pequeña dimensión se agota la memoria, por las propias características del algoritmo.

Chu and Beasley [26] han desarrollado una metaheurística muy eficiente, en la cual se reporta por primera vez el concepto de reparación de una solución, en aras de llegar a soluciones factibles rápidamente. Este trabajo ha reportado soluciones muy buenas para diversos “test problem” reportados en [27], sin embargo los problemas que han sido abordados se han considerado problemas de hasta medianas dimensiones, es decir menos de 1000 variables.

Como parte del proceso de investigación se ha desarrollado una librería paralela para algoritmos genéticos y se ha trabajado en el desarrollo de un algoritmo en paralelo para resolver el problema en enteros mixtos lineal [12].

En el presente trabajo se pretende utilizar la librería de Algoritmos Genéticos desarrollada en un proyecto anterior (llamada GALib) [12] y así crear un Algoritmo Genético Paralelo capaz de resolver problemas de tipo MKP. Dicha librería no posee ningún método que permita la migración de los mejores individuos de un nodo a otro, este es el objetivo fundamental de este trabajo; pues es necesario desarrollar de manera eficiente la estrategia de migración, para este trabajo se usa un esquema de comunicación maestro-esclavo, que aprovecha al máximo las características del conjunto de soluciones factibles.

Se propone entonces, implementar un Algoritmo Genético Básico bajo la librería ya mencionada, para resolver el problema MKP, para de ahí, tomar como base ese algoritmo e implementarlo de forma paralela ocupando la estrategia de migración diseñada, y probar dicho algoritmo con los problemas reportados en [27] y con problemas reales.

2.2 La importancia de la investigación en algoritmos genéticos

El poder de la evolución como teoría científica es asombroso. Definitivamente una teoría capaz de explicar la vida, así como la increíble diversidad y nivel de adaptación de los seres vivos a su entorno, es un logro científico de enorme magnitud. Antes de la formulación de la evolución darwinista, el origen de nuestra existencia era quizás el más grande misterio; gracias a la explicación ofrecida por Darwin este misterio parece haber sido resuelto.

Ciertamente estamos rodeados por maravillas de diseño y especialización biológicas, muchos modelos para nuestra tecnología provienen precisamente de la observación y la imitación de lo que observamos en la naturaleza. Si la evolución es el probable artífice de nada menos que nuestra existencia, entender y aplicar este proceso a la solución de problemas de cualquier tipo, es de gran importancia. Estamos hablando de aplicar el proceso de diseño más poderoso observado por el hombre, y con la aparición de las computadoras, la evolución simulada y artificial aparece como una de las técnicas de búsqueda y diseño más eficaces y útiles que se conocen.

Aunque la evolución tiene, gracias a la evidencia a su favor, categoría científica, esto no significa que su funcionamiento sea completamente entendido. Esto es particularmente cierto en lo que se refiere a la evolución simulada, en donde se busca soluciones para problemas específicos, en lugar del genérico de la naturaleza, resumido como la continuada existencia de los genes.

2.3 C++, GALib, OOMPI y LAM/MPI

Para la implementación del sistema se utilizó el entorno de programación **C++**, este lenguaje fue elegido por su potencia, por ser orientado a objetos y porque las librerías GALib[3] y OOMPI[4] estan escritas bajo este lenguaje. Esta última característica es fundamental, por la facilidad que brinda este paradigma de programación para escribir, extender, mantener y reutilizar código.

Es importante mencionar que en el diseño del algoritmo se utilizaron conceptos inspirados en la biblioteca GALib creada en [34]. Algo importante que se debe mencionar, es que la biblioteca GALib es una biblioteca de funciones en C++ que proporciona al programador un conjunto de objetos para el desarrollo de algoritmos genéticos. Usando GALib es posible resolver problemas de optimización mediante la construcción de un algoritmo genético, usando estructuras de datos y operadores estándar o específicos (crear nuevos modelos basados en lo estandard) de selección, cruce y mutación. El programador debe crear una función llamada *Fitness*, la cual permite que los genomas se adapten al resultado que se desea encontrar.

Para las comunicaciones paralelas se eligió a la Interfaz de Paso de Mensajes (**MPI**, por sus siglas en inglés *Message Passing Interface*), el cual es el protocolo de comunicaciones paralelas más utilizado en la actualidad. MPI es un conjunto de rutinas que pueden ser utilizadas en programas escritos en C++, Fortran y Ada, para implementar comunicaciones de procesamiento paralelo. Una gran ventaja de MPI es que es un estándar diseñado por más de 60 personas provenientes de 40 organizaciones interesadas en la computación paralela. En especial, en la creación de este proyecto, se uso la versión OOMPI, que es la versión de MPI orientada a objetos.

Object Oriented MPI[5] (**OOMPI**) es un a librería de la clase de especificación que encapsula la funcionalidad de MPI dentro de una clase funcional jerárquica, para proporcionar una simple, flexible e intuitiva interfaz.

[3] Para el desarrollo de este proyecto se uso la versión 2.4.6 de la librería GALib y modificada en [12].
[4] La versión utilizada de OOMPI en el desarrollo de esta aplicación es OOMPI 1.0.4
[5] Ver archivo PDF o PS de referencia para OOMPI[30]

Una de las implementaciones más extendidas de MPI es **LAM** (Local Area Multicomputer), desarrollado en sus orígenes por el Ohio Supercomputer Center y mantenido en la actualidad por el Open Systems Laboratory (OSL) en la Universidad de Indiana, Estados Unidos. Actualmente puede obtenerse, además de en su página web, como parte de las distribuciones de Linux, por ejemplo Fedora. Aunque existen otras implementaciones de MPI, como MPICH, en nuestro caso, nos encontramos trabajando en la implementación mediante LAM. La disponibilidad de implementaciones de las librerías MPI y la facilidad de su puesta en funcionamiento son factores que inclinan la decisión hacia esta plataforma.

2.4 Algoritmo de Simplificación

Como se busca darles solución a problemas de grandes dimensiones, es conveniente analizar las características del problema a resolver, para fijar variables en 0 y en 1, eliminar restricciones de ser posible y detectar sin gran esfuerzo computacional si el problema no tiene solución factible, es por ello que apoyados en criterios propuestos por Crowder, Johnson y Padberg [54], se propone el algoritmo de simplificación, el cual propone un tratamiento exhaustivo del sistema de restricciones, mediante la aplicación de diferentes criterios.

Descripción del Algoritmo: Cada restricción del problema puede separarse de la forma que muestra la ecuación (2):

$$\sum_{j \in N^+} a_j x_j + \sum_{j \in N^-} a_j \leq a_0 \qquad (2)$$

donde N^+ denota el conjunto de índices con variables que tienen coeficiente positivo y N^- conjunto de índices de variables que tienen coeficiente negativo en cada restricción.

Paso 1: Inicialización (fijar solución ($\hat{f}$) = false).

Paso 2: Análisis de factibilidad:

Si para alguna restricción se cumple que $\sum a_j > a_0, j \in N^-$, entonces
  Ir al paso 5.
Si no, entonces
  Ir al paso 3.

Paso 3: Análisis de inactividad:

Si para alguna restricción se cumple que $\sum a_j \leq a_0, j \in N^+$, entonces
Eliminar la restricción.

Si fj = false, entonces
ir al Paso 4,
Sino, entonces
Ir al Paso 6.

Paso 4: Fijar Solución:

En cero:
Si se cumple que $a_j > a_0 - \sum a_k$ para $j \in N^+, k \in N^-$, entonces

fijar x_j en cero,
fj = true.
En uno:
Si se cumple que $-a_j > a_0 - \sum a_k$ para $j \in N^-, k \in N^-$, entonces

fijar x_j en uno,
fj = true.

Si fj = true entonces
Ir al Paso 2
Si no, entonces
Ir al Paso 6.

Paso 5: El problema no tiene solución factible. Ir al Paso 6.

Paso 6: Fin del algoritmo

2.5 Algoritmo Genético Secuencial

A continuación se describe el Algoritmos Genético Secuencial (AGS) que se desarrolló, utilizando la librerìa GALIB.

Paso 1: Crear el objeto que contiene el problema MKP a resolver

Paso 1.1: Leer el archivo que contiene al problema

Paso 1.2: Simplificar el problema (Algoritmo de Simplificación)

Paso 1.2.1: Verificar la factibilidad de solución del problema

Paso 1.2.2: Verificar la inactividad de restricciones

Paso 1.2.3: Fijar en ceros y unos el esquema de solución

Paso 2: Crear el objeto que lleva la información del genoma

Paso 3: Crear el objeto que contiene al algoritmo genético

Paso 3.1: Indicar cuantas poblaciones existirán
Paso 3.2: Ingresar el número de individuos que habrá por población
Paso 3.3: Establecer cuantas generaciones se considerarán en cada evolución
Paso 3.4: Fijar la tasa de mutaciones en la evolución

Paso 4: Inicializar aleatoriamente las poblaciones

Paso 5: Evolucionar

Paso 5.1: Ingresar a la función *Fitness*
Paso 5.2: Realizar la cruza del genoma actual (que lleva el algoritmo genético) con el esquema solución obtenido en el Paso 1.
Paso 5.3: Verificar si el genoma resultante del Paso 5.2 es una solución factible En caso de serlo, ingresar el genoma a la lista de las soluciones factibles.
Paso 5.4: Mientras no se concluya el proceso de evolución, regresar al Paso 5.1

Paso 6: Mostrar la solución del problema.

2.6 Algoritmo Genético Paralelo

En esta sección se muestra el Algoritmo Genético Paralelo desarrollado en éste proyecto, se puede observar que tal algoritmo tiene de base al Algoritmo Genético Secuencial creado. Cabe recordar que el esquema de comunicación, empleado en el algoritmo paralelo, es maestro-esclavo y la migración de la población de un nodo a otro se realiza utilizando a los nodos vecinos. El algoritmo es el siguiente

Paso 1: Leer el archivo que contiene el problema MKP.

Paso 2: Aplicar Algoritmo de Simplificación al problema.

Paso 3: Fijar los parámetros necesarios para la aplicación del algoritmo genético: tamaño de la población, número de generaciones, porcentaje de mutación y el tipo de cruza.

Paso 4: Generar la población inicial utilizando el Objeto *CGAlgorithm* de la librería.

Paso 5: Mientras el algoritmo genético no concluya:

Se combina el genoma actual de la evolución y el esquema de la solución obtenida, mediante el algoritmo de simplificación

Cada 50 iteraciones (migrar):

- Cada nodo esclavo i envía su genoma al nodo esclavo i+1.
- Cada nodo esclavo i recibe su genoma del i-1, una vez recibido el genoma, lo mezcla entre su población.

Paso 6: Enviar la mejor solución al nodo 0.

Paso 7: El nodo maestro recibe las soluciones de los nodos esclavos.

Paso 8: Mostrar el resultado del problema MKP, incluyendo el vector solución y el valor de la función objetivo.

Capítulo III: Análisis del Problema

Para el análisis y diseño del sistema se ha utilizado el Proceso Unificado de desarrollo de Software, confeccionando el diagrama de casos de uso para la captura de requerimientos, en la vista de análisis, los diagramas de clases, los diagramas de actividades del sistema, los diagramas de secuencia que muestran como se realiza el envío y la recepción de mensajes en la vista de diseño y el diagrama de paquetes y de nodos en la vista de implementación.

3.1 UML

UML (*Lenguaje Unificado de Modelado*), es ante todo un proceso de desarrollo de Software, es decir, es el conjunto de actividades necesarias para transformar los requisitos de un usuario (análisis) en un sistema Software (implementación).

UML está basado en componentes. Los verdaderos aspectos definitorios de UML como proceso unificado se resumen en tres fases clave:

- Proceso unificado dirigido por caso de uso
- Proceso unificado centrado en la arquitectura
- Proceso unificado iterativo e incremental

3.1.1 Visión General de UML

UML, es un lenguaje estándar de modelado para Software, es decir, es un lenguaje para la visualización, especificación, construcción, documentación de los componentes de sistemas en los que el Software juega un papel importante, en una palabra: “Diseño”. UML permite a los desarrolladores visualizar los resultados de su trabajo en esquemas y diagramas estandarizados, no obstante, tras esta notación grafica, UML especifica una semántica, es decir, su significado.

3.1.2 El vocabulario de UML

UML proporciona a los desarrolladores un vocabulario que incluye tres categorías:

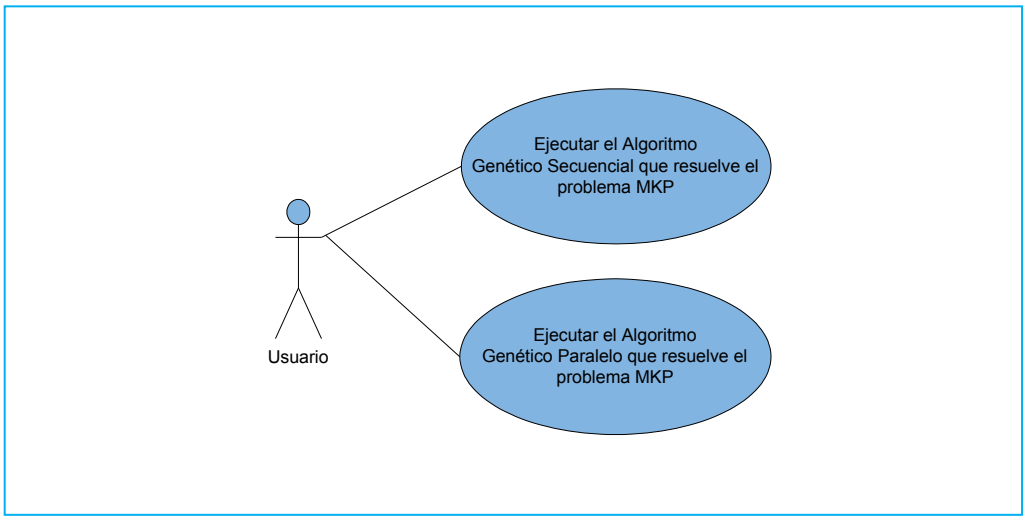
- Elementos: Hay cuatro tipos de elementos:
    - Estructurales*
    - De comportamiento*
    - De agrupación
    - De notación
- Relaciones: hay tres tipos de elementos:
    - De dependencia
    - De asociación*
    - De generalización*
- Diagramas: hay nueve tipos de diagramas:
    - Caso de Uso*
    - Clases*
    - Objetos
    - Secuencia*
    - Colaboración
    - Estados
    - Actividad*
    - Componentes*
    - Despliegue

(*) Estos son los elementos gráficos de UML que se utilizaran en el análisis y diseño del sistema Software que es creado como parte de este proyecto.

3.2 Diagramas de Casos de Uso

A continuación se explican los diagramas de casos de usos que muestran la interacción del sistema con su entorno.

3.2.1 Diagrama de Caso de Uso General



Descripción:

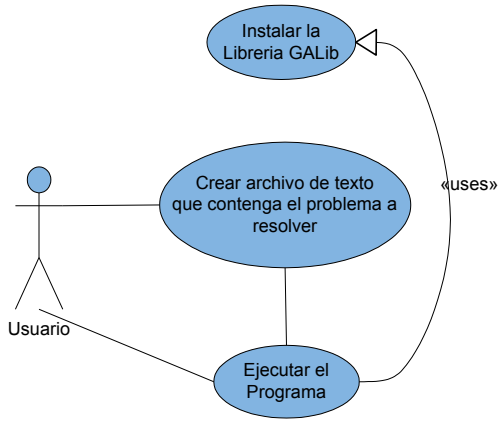
El Usuario puede resolver su problema con las siguientes formas:

- Ejecutar el Programa que contiene el Algoritmo Genético Secuencial que resuelve el problema MKP, el cual corre en una PC con Windows o Linux.

- Ejecutar el Programa que contiene el Algoritmo Genético Paralelo que resuelve el problema MKP, el cual sólo corre en una red de PC's que tengan Linux, LAM/MPI y OOMPI.

3.2.2 Diagramas de Caso de Uso Específicos

Caso de Uso
Ejecutar el Algoritmo Genético Secuencial que resuelve el problema MKP



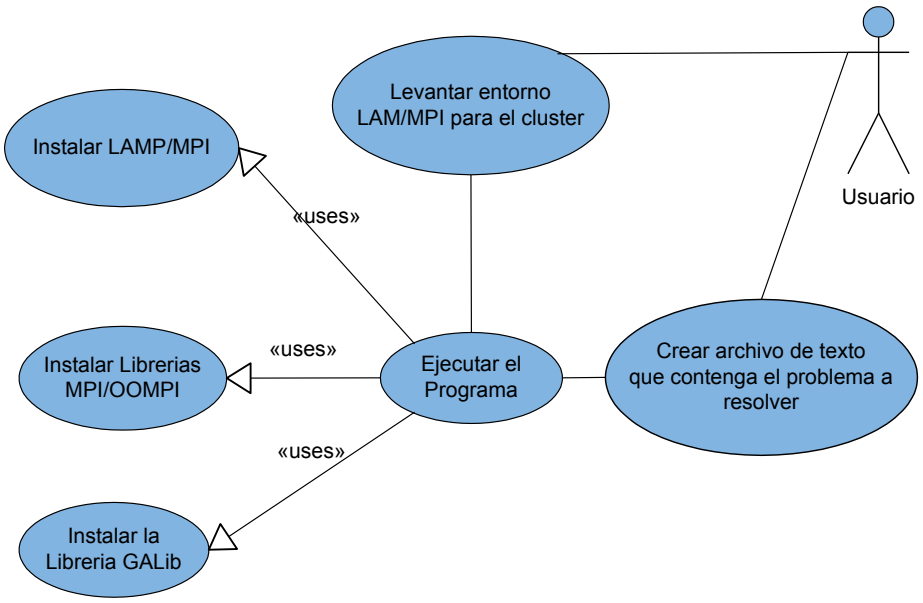
Descripción:

Si el usuario desea modificar partes del código fuente del programa debe instalar la Liberia GALib, para poder compilar el programa con sus cambios hechos.

El usuario debe antes de ejecutar el programa para resolver su problema MKP, crear un documento, el cual almacena la información necesaria que el programa debe leer, para dar una solución al mismo.

Ya teniendo el archivo de texto que contiene el problema a resolver, se procede a ejecutar el programa, pasándole como argumento el nombre del archivo de texto.

Caso de Uso
Ejecutar el Algoritmo Genético Paralelo que resuelve el problema MKP



Descripción:

El usuario debe tener instalado en su máquina una versión de Linux que tenga LAMP/MPI, para poder ejecutar la aplicación, en caso de no tenerlo debe de instalarlo.

Si el usuario desea modificar partes del código fuente, debe instalar las librerías siguientes: OOMPI, MPI y GALib, para poder compilar correctamente.

El usuario debe antes de ejecutar el programa para resolver su problema MKP, crear un documento, el cual almacena la información necesaria que el programa debe leer, para dar una solución al problema.

Ya teniendo el archivo de texto que contiene el problema MKP a resolver, se procede a inicializar el entorno LAM/MPI, para luego ejecutar con mpirun el mismo, pasándole como argumento el archivo que contiene el problema.

3.3 Modelado de Clases del Sistema

En el modelado del sistema se ha considerado la biblioteca de clases, la cual hace uso de las librerías GALib y OOMPI las cuales son la base de los algoritmos desarrollados, tanto secuencial, como paralelo.

3.3.1 Diagrama general de clases

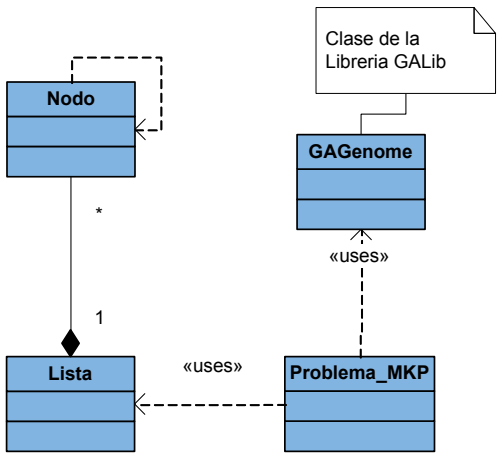
Un *diagrama de clases* sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de contenimiento.

Un diagrama de clases está compuesto por los siguientes elementos:

- Clase: atributos, métodos y visibilidad.
- Relaciones: Herencia, Composición, Agregación, Asociación y Uso.

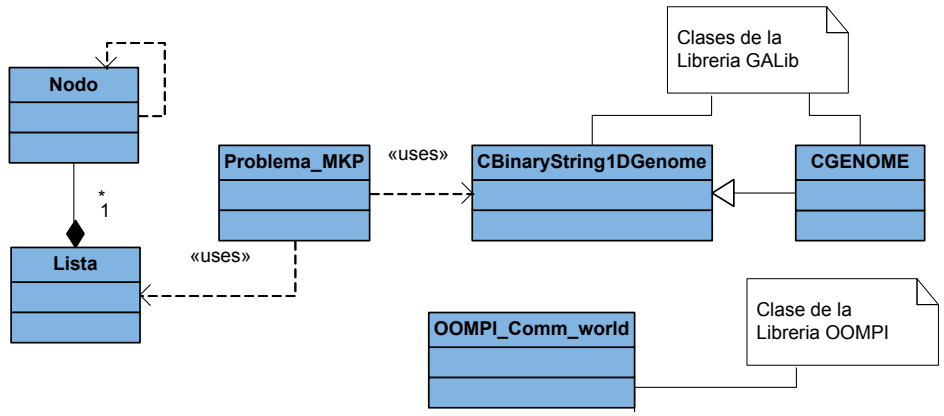
El diagrama general de las clases más importantes se muestra a continuación con la notación UML.

Diagrama de Clases para el Algoritmo Genético Secuencial que resuelve el problema MKP



En este diagrama se puede apreciar a grandes rasgos la estructura del sistema que se usa para la creación del algoritmo secuencial.

Diagrama de Clases para el Algoritmo Genético Paralelo que resuelve el problema MKP



En este diagrama se muestra la estructura del sistema que se usa para la creación del algoritmo paralelo.

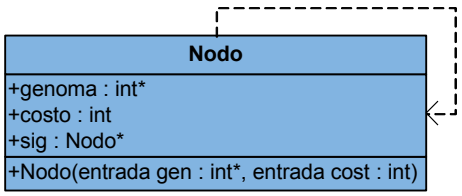
Capítulo IV: Diseño del Sistema

En este capítulo se aborda el diseño del sistema. Se describen las clases, que fueron creadas, una por una y también se muestran los diagramas de secuencia, actividades y de nodos.

4.1 Descripción detallada de las clases

Como se pudo observar anteriormente, las clases creadas para este sistema (que no se contemplan en GALib y OOMPI) son la clase Nodo, Lista y Problema_MKP. Estas se describirán a continuación:

Clase **Nodo**

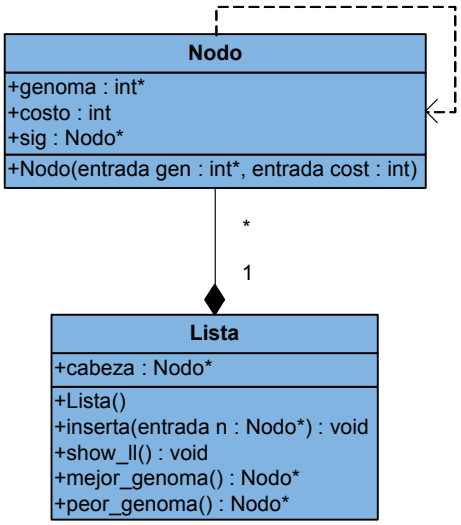


Descripción:

Esta clase es la encargada de crear objetos de tipo Nodo, los cuales son la parte más importante para la creación de la lista ligada que se usa para almacenar los resultados del problema.

Elemento	Descripción
genoma : int*	Apuntador a un arreglo que representa al mejor genoma factible para la solución del problema, su dominio de valores es de uno (1) y cero (0)
costo : int	Guarda el costo real del genoma, por el que se mide la factibilidad de la solución
sig : Nodo*	Apunta a un nodo siguiente
Nodo(entrada gen : int*, entrada cost : int)	Constructor de la clase Nodo, inicializa las variables del objeto, recibe el arreglo que representa el genoma y el costo de ese genoma

Clase **Lista**

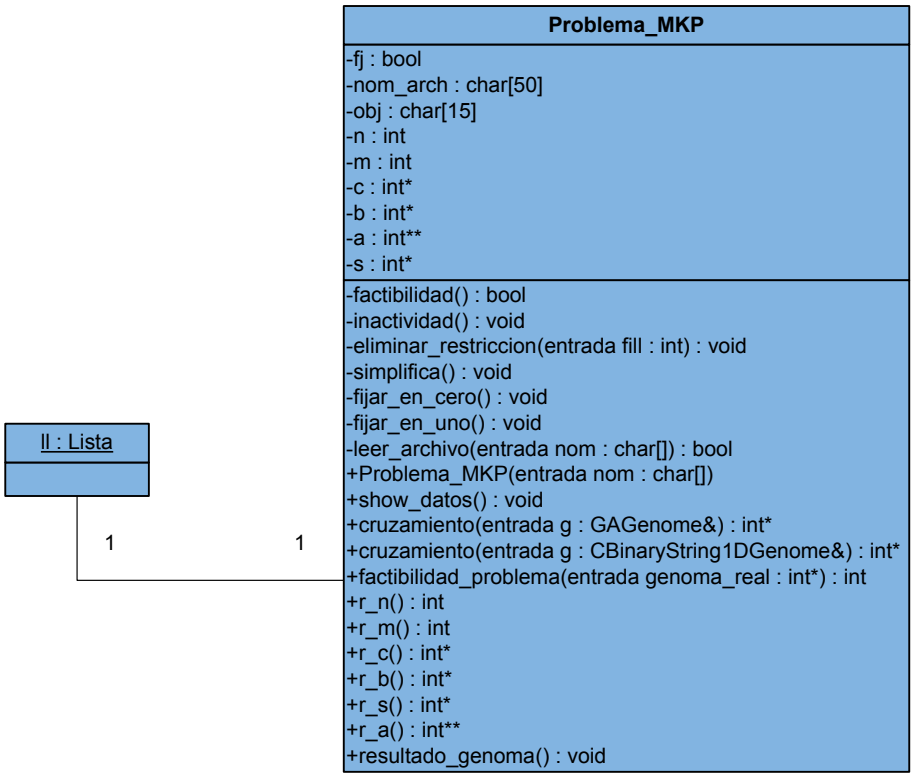


Descripción:

Esta clase es la encargada de crear la lista ligada y mantener los nodos que son soluciones factibles del problema. Como se podrá observar, esta clase necesita de objetos de tipo Nodo para poder existir.

Elemento	Descripción
cabeza : Nodo*	Apuntador que lleva el inicio de la lista ligada.
Lista( )	Constructor de la clase, inicializa la cabeza en NULL
inserta(entrada n : Nodo) : void	Inserta el nodo n en la lista ligada. Al momento de insertar el nodo, este se va insertando de forma ordenada tomando como factor ordenable el costo; es decir los nodos que tengan el mayor costo se insertarán mas cerca de la cabeza de la lista, y en caso contrario se insertarán al final de la misma.
show_ll( ) : void	Imprime en pantalla toda la lista ligada
mejor_genoma( ) : Nodo*	Regresa un apuntador al nodo que representa al mejor genoma factible para la solución del problema (el primer nodo). Usado para obtener el resultado de la maximización de la función objetivo en el problema MKP.
peor_genoma( ) : Nodo*	Regresa un apuntador al nodo que representa al peor genoma factible para la solución del problema (el último nodo). Usado para obtener el resultado de minimizar la función objetivo en el problema MKP.

Clase **Problema_MKP**



Descripción:

Esta clase es la encargada de leer el archivo que contiene el problema resolver y de guardar los datos en las variables correspondientes, minimizar el problema, verificar si el problema tiene solución y poder seguir con la ejecución del Algoritmo Genético Secuencial, a su vez también es la encargada de mostrar el resultado obtenido.

Esta clase necesita de un objeto Lista para poder guardar las soluciones factibles del problema y en una de sus funciones recibe un objeto de tipo GAGenome o CBinaryString1DGenome, el cual lleva el genoma actual del proceso evolutivo, y será verificado para saber si éste constituye una solución factible del problema.

Elemento	Descripción
fj : bool	Sirve al algoritmo de simplificación para detener el proceso de simplificación del problema leído en el archivo, es decir el problema original.
nom_arch : char[50]	Guarda el nombre del problema
obj : char[]15	Guarda que tipo de objetivo que desea alcanzar el problema MKP, ya sea Max para Maximizar el problema, o Min para minimizar
n : int	Guarda el número de variables del problema MKP.
m : int	Almacena el número de restricciones que posee el problema a resolver.
c : int*	Almacenar la función objetivo del problema MKP. Al momento de leer n, se crea un vector de tamaño n y es en esta variable donde se almacena al vector
b : int *	Se encarga de almacenar el valor de los términos independientes del problema a resolver. Al momento de leer m, se crea un vector de tamaño m y es en esta variable donde se almacena ese vector
a: int **	Almacena las restricciones del problema MKP. Al momento de leer n y m, se crea una matriz de tamaño m x n y esa matriz es almacenada en esta variable
s : int*	Guarda un vector de tamaño n, el cual será un esquema del genoma solución del problema, resultado de la aplicación del algoritmo de simplificación.
Problema_MKP(entrada nom : char[50])	Constructor de la clase, recibe como entrada el nombre del archivo a leer. En su interior se crea un objeto de tipo Lista
leer_archivo(entrada nom : char []) : bool	Encargada de leer el archivo que contiene al problema MKP y almacenar cada parte del problema en sus respectivas variables. Regresa *true* si se leyó correctamente el archivo o *false,* si hubo un problema al leer el mismo.
show_datos() : void	Muestra en pantalla la información leída del archivo correspondiente al problema MKP a resolver
simplifica( ) : void	Realiza la simplificación del problema, manda a llamar las funciones que componen el algoritmo de simplificación: factibilidad( ), inactividad( ), fijar_en_cero( ) y fijar_en_uno( ). Este proceso se encuentra descrito en [31].

factibilidad( ) : bool	Se encarga de verificar si el problema MKP tiene una solución factible. Regresa *true* si tiene solución factible o *false* si no tiene solución factible
inactividad( ) : void	Verifica si una condición es inactiva para la solución del problema, si es inactiva una restricción es eliminada de la matriz de restricciónes por medio de la función siguiente
elimina_restriccion(entrada fil : int) : void	Recibe que fila de la matriz de restricciones debe ser eliminada y la borra de ella, por ser una restricción que no elimine soluciones del problema original.
fijar_en_cero( ) : void fijar_en_uno( ) : void	Son las encargadas de fijar un esquema de solución para el problema, este esquema, creado por éstas funciones, es almacenado en la variable s.
cruzamiento(entrada g: CGenome&) : int* cruzamiento(entrada g: CBinaryString1DGenome&) : int*	Recibe un genoma, como resultado de la aplicación de la función fitness, realizando la cruza del mismo con el esquema de solución del problema, retornando el vector resultante.
factiblididad_problema(entrada genoma_real : int*) : int	Recibe el genoma que fue el resultado de la aplicación de la función de cruzamiento. Se encarga de verificar si ese genoma es una solución factible, (posible solución) para el problema MKP, si lo es, se almacena ese genoma junto con su costo en la lista ligad. Si el genoma es factible se retorna el costo, en caso contrario la constante NO.
resultado_genoma( ) : void	Se encarga de obtener de la lista ligada el nodo que contiene la mejor solución del problema. Si el objetivo es Maximizar se extrae el nodo que se encuentra en la cabeza de la lista, si es Minimizar, se extrae el nodo que se encuentra al final de la lista ligada.
r_n( ) : int	Regresa el valor de la variable n
r_m( ) : int	Regresa el valor de la variable m
r_c( ) : int*	Regresa al arreglo c
r_b( ) : int*	Regresa al arreglo b
r_a( ) : int**	Regresa a la matriz a
r_s( ) : int*	Regresa el arreglo s

4.2 Diagramas de secuencia

Los *diagramas de secuencia* muestran el intercambio de mensajes (es decir la forma en que se invocan y que es lo que recibe cada función del objeto especificado) en un momento dado. Los diagramas de secuencia ponen especial énfasis en el orden y el momento en que se envían los mensajes a los objetos.

A continuación se mostrarán los diagramas de secuencia correspondientes a este proyecto.

Nombre: **Diagrama de secuencia del Algoritmo Genético Secuencial**

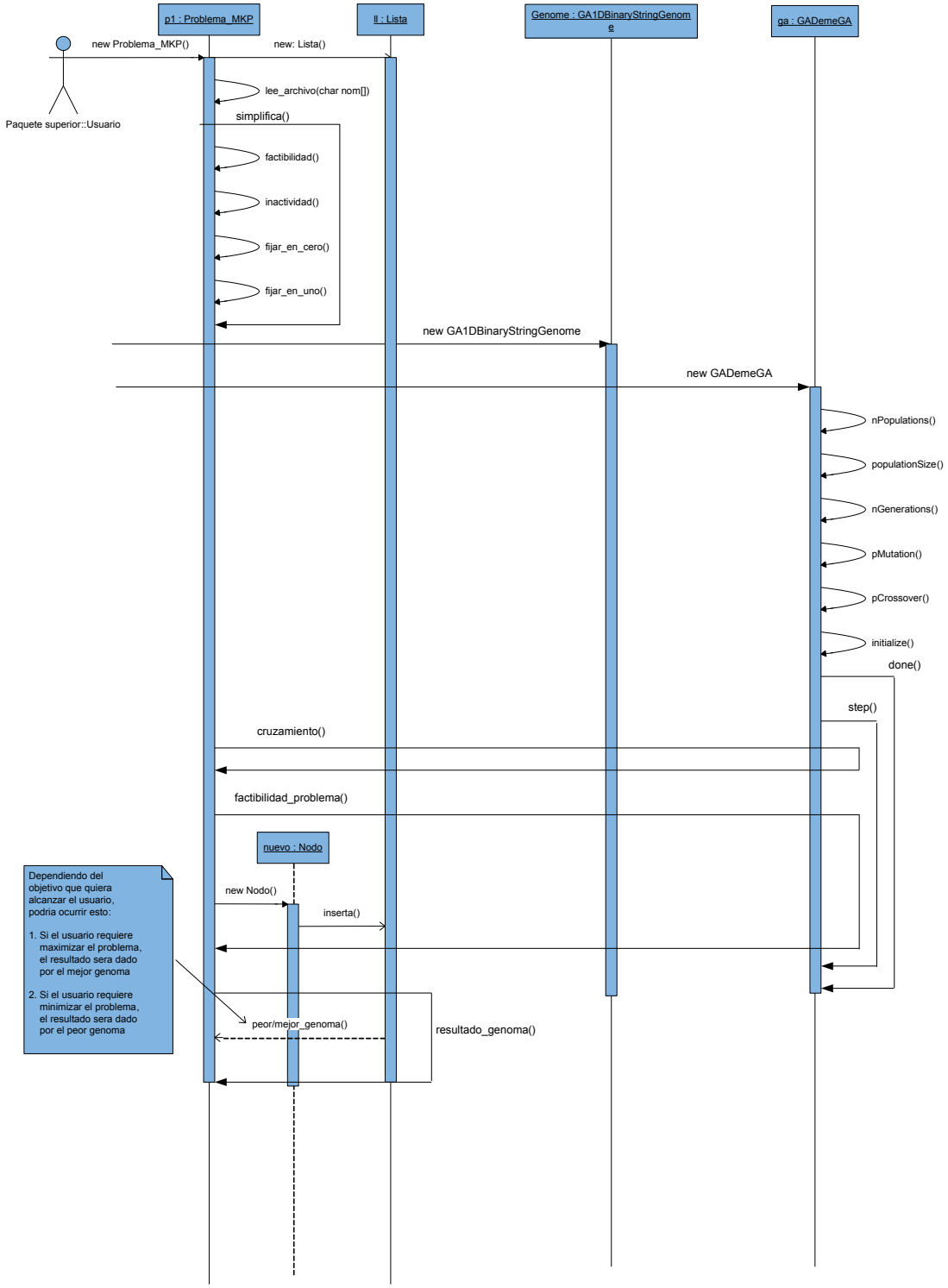
Descripción:

Este diagrama representa la secuencia que lleva el programa que ejecuta el Algoritmo Genético Secuencial que resuelve el problema MKP.

Objetos involucrados:

Nombre	Tipo/clase	Descripción
p1	Problema_MKP	Contiene el problema a resolver y las funciones que son necesarias para leer el problema, simplificar el problema, cruzar los genomas que se crean en el proceso de la evolución con el esquema de solución y mostrar el resultado.
ll	Lista	Lleva la lista ligada que contiene los genomas factibles para ser solución del problema MKP a resolver.
genome	GA1DBinaryStringGenome	Genoma, el cual lleva en cada paso de la evolución un elemento diferente, se usa en la función fitness y dentro de esa función se cruza este objeto con el esquema de solución, el cual se encuentra dentro del objeto p1, con la finalidad de encontrar una solución factible al problema MKP.
ga	GADemeGA	Contiene al algoritmo genético encargado de evolucionar al genoma en cada paso de la evolución. Este objeto es muy especial, porque no sólo contiene al algoritmo evolutivo, sino que, dentro de él, se crean varias poblaciones de genomas, contiene los métodos necesarios para la cruza y mutación de éstos.
nuevo	Nodo	Es creado cada vez que se encuentre un genoma factible para la solución del problema, y es enlazado en el objeto ll.

Diagrama de Secuencia del Algoritmo Genético Secuencial para resolver el problema MKP



Nombre: **Diagrama de secuencia del Algoritmo Genético Paralelo**

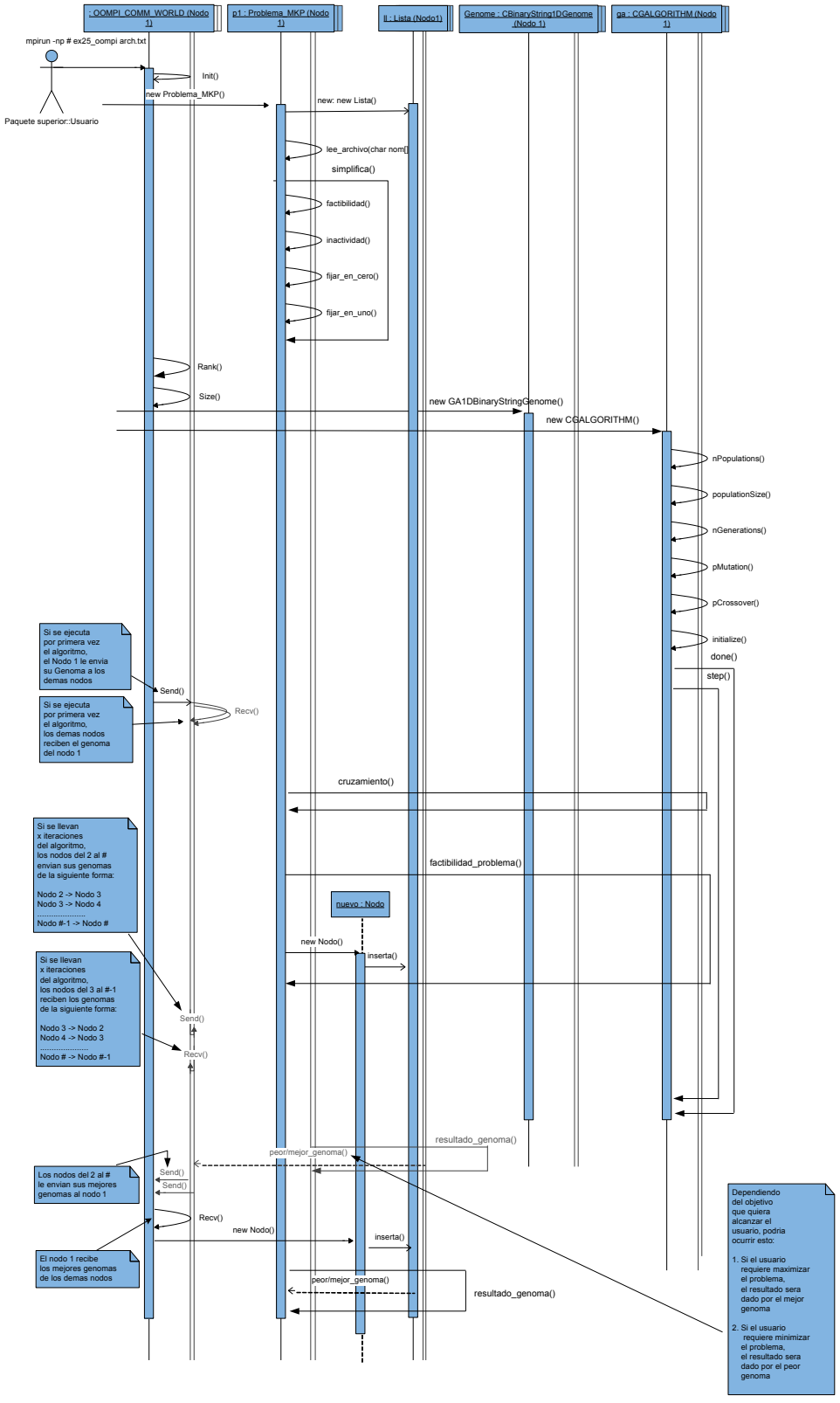
Descripción:

Este diagrama representa la secuencia en el intercambio de información que se realiza al ejecutar el Algoritmo Genético Paralelo, que resuelve el problema MKP.

Objetos involucrados:

Nombre	Tipo/Clase	Descripción
	OOMPI_COMM_WORLD	No existe un objeto de tipo OOMPI_COMM_WORLD, simplemente se usan variables y funciones estáticas de clase. Esta clase nos permite inicializar el entorno MPI, enviar y recibir genomas entre los nodos que ejecutan al programa en paralelo
p1	Problema_MKP	Almacena el problema a resolver y las funciones que son necesarias para leer el mismo, simplifica el problema, cruza los genomas que se crean en el proceso de la evolución con el esquema de solución y muestra el resultado.
ll	Lista	Almacena la lista ligada que contiene los genomas factibles del problema a resolver.
genome	CBinaryString1DGenome	Genoma, el cual lleva en cada paso de la evolución un elemento diferente, se usa en la función fitness y dentro de esa función se cruza este objeto con el esquema de solución, el cual se encuentra dentro del objeto p1, con la finalidad de encontrar una solución factible del problema.
ga	CGALGORITHM	Contiene al algoritmo genético encargado de evolucionar al genoma en cada paso de la evolución. Este objeto es muy especial, porque contiene el algoritmo evolutivo básico y los métodos necesarios para la cruza y mutación de genomas.
nuevo	Nodo	Es creado cada vez que se encuentre un genoma factible para la solución del problema, y es enlazado en el objeto ll.

Diagrama de Secuencia del Algoritmo Genético Paralelo para resolver el problema MKP



4.3 Diagramas de actividad

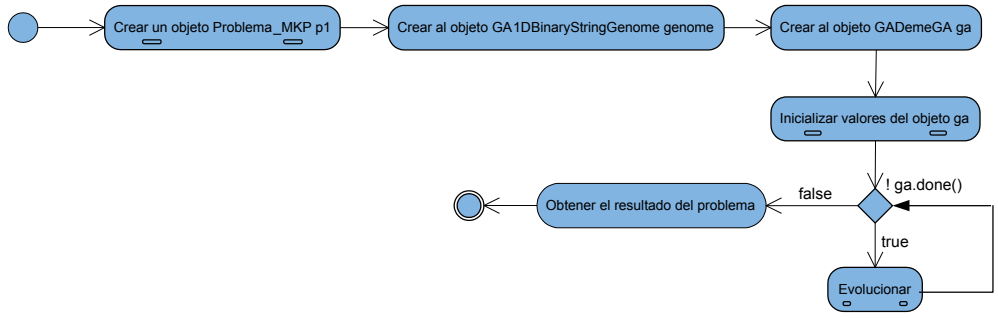
En UML un *diagrama de actividad* es utilizado para mostrar la secuencia de actividades. Los diagramas de actividades muestran el flujo de trabajo desde el punto de inicio hasta el punto final, detallando muchas de las rutas de decisiones que existen en el progreso de eventos contenidos en la actividad. Estos también pueden usarse para detallar situaciones donde el proceso paralelo puede ocurrir en la ejecución de algunas actividades.

A continuación se mostrarán los diagramas de actividad correspondientes a este proyecto.

Nombre: **Diagrama de actividad del Algoritmo Genético Secuencial**

Descripción:

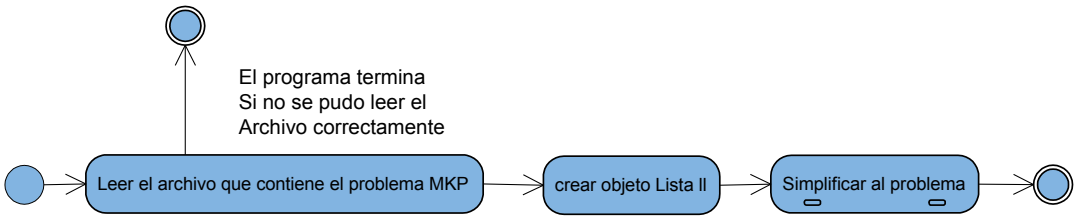
Este diagrama representa las actividades más generales que se realizan al momento de ejecutar al algoritmo genético secuencial.



Nombre: **Diagrama de actividad de Crear un Objeto Problema_MKP**

Descripción:

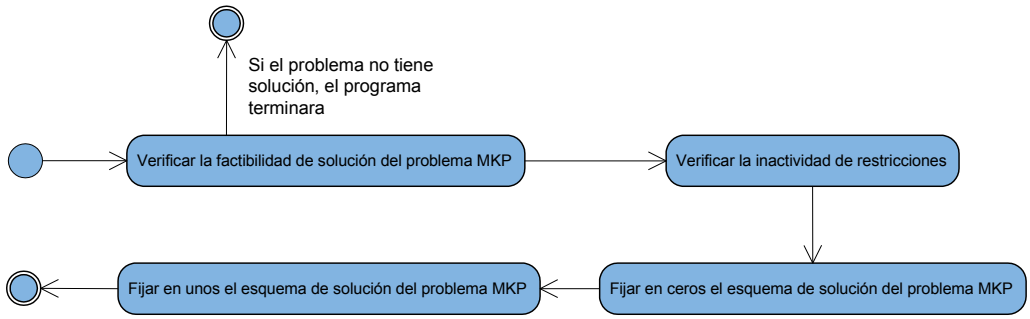
Este diagrama representa que actividades internas se realizan al crear el objeto Problema_MKP p1.



Nombre: **Diagrama de actividad al momento de Simplificar al problema**

Descripción:

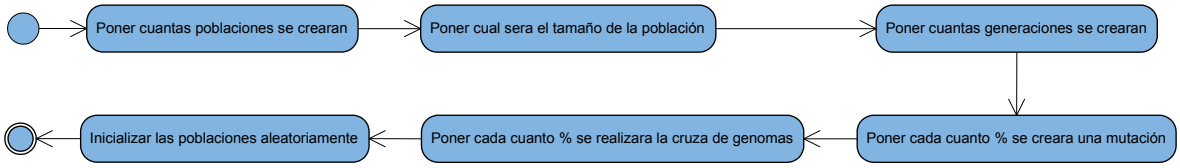
Este diagrama representa que actividades internas se realizan al simplificar el problema leído del archivo (problema original).



Nombre: **Diagrama de actividad de inicializar valores del objeto ga**

Descripción:

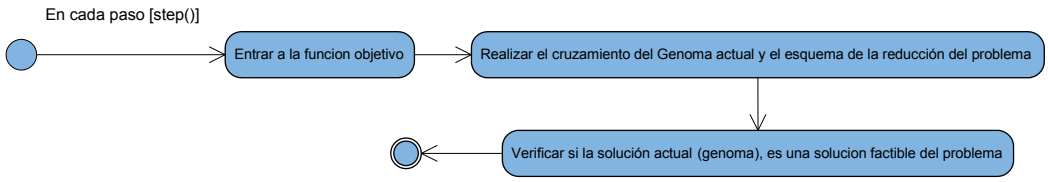
Este diagrama representa las actividades internas que se realizan al asignar los valores requeridos del objeto que contiene el Algoritmo Genético.



Nombre: **Diagrama de actividad de evolucionar**

Descripción:

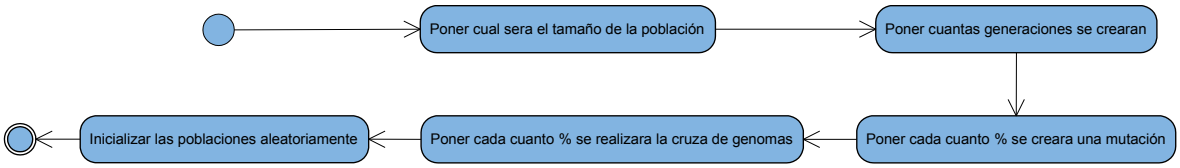
Este diagrama representa qué actividades internas se realizan al momento de hacer la evolución.



Nombre: **Diagrama de actividad para inicializar valores del objeto ga (para el diagrama de actividad en paralelo)**

Descripción:

Este diagrama representa que actividades internas se realizan al asignar los valores requeridos del objeto que contiene el Algoritmo Genético.

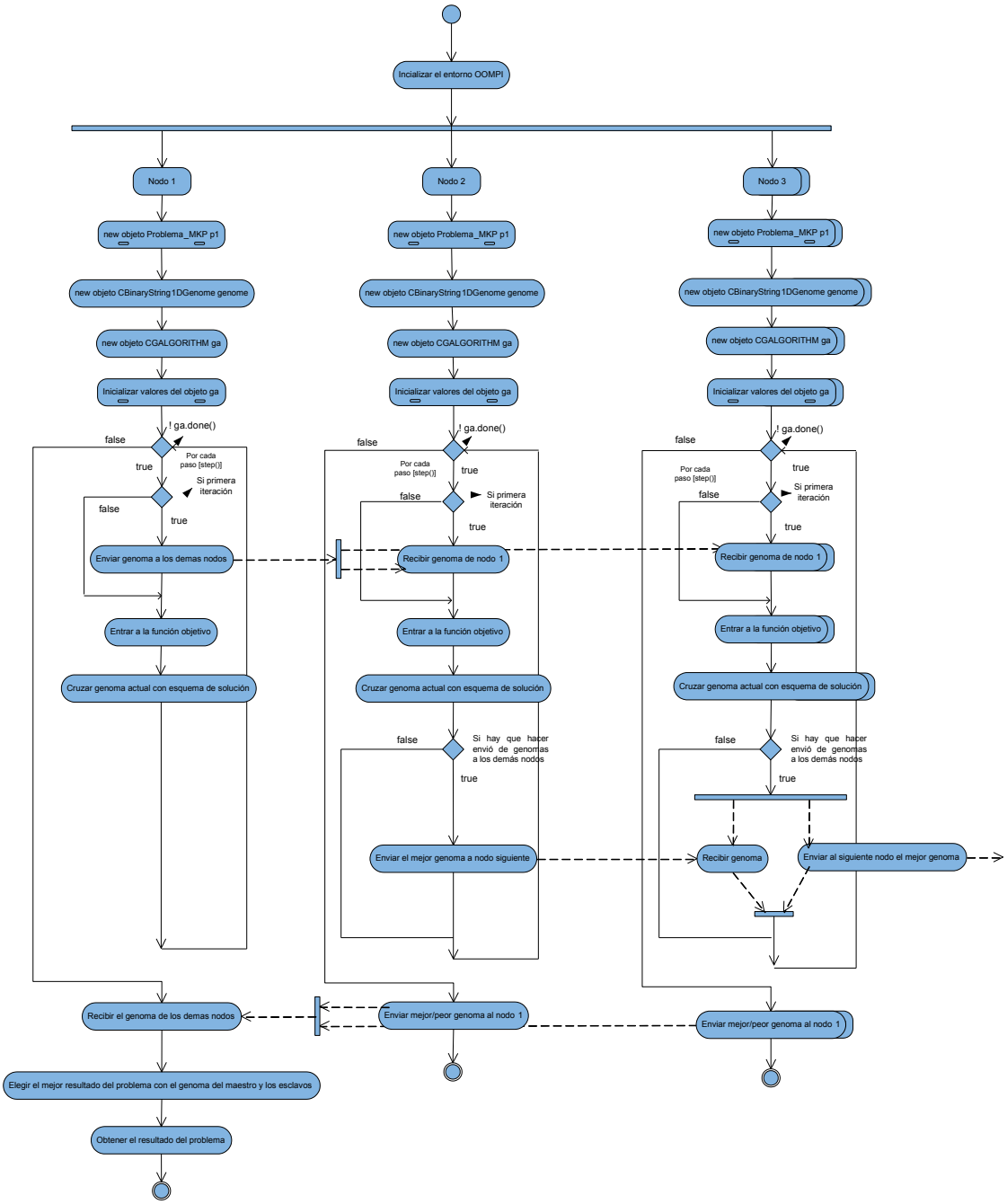


Nombre: **Diagrama de actividad del Algoritmo Genético Paralelo**

Descripción:

Este diagrama representa que actividades se hacen al momento de ejecutar el algoritmo genético paralelo. Como se puede observar en el mismo, se crea un entorno MPI, para poder realizar el envío y recepción de genomas a los diferentes nodos. De manera general el algoritmo propuesto se define a continuación:

Diagrama de Actividad del Algoritmo Genético Paralelo para resolver el problema MKP



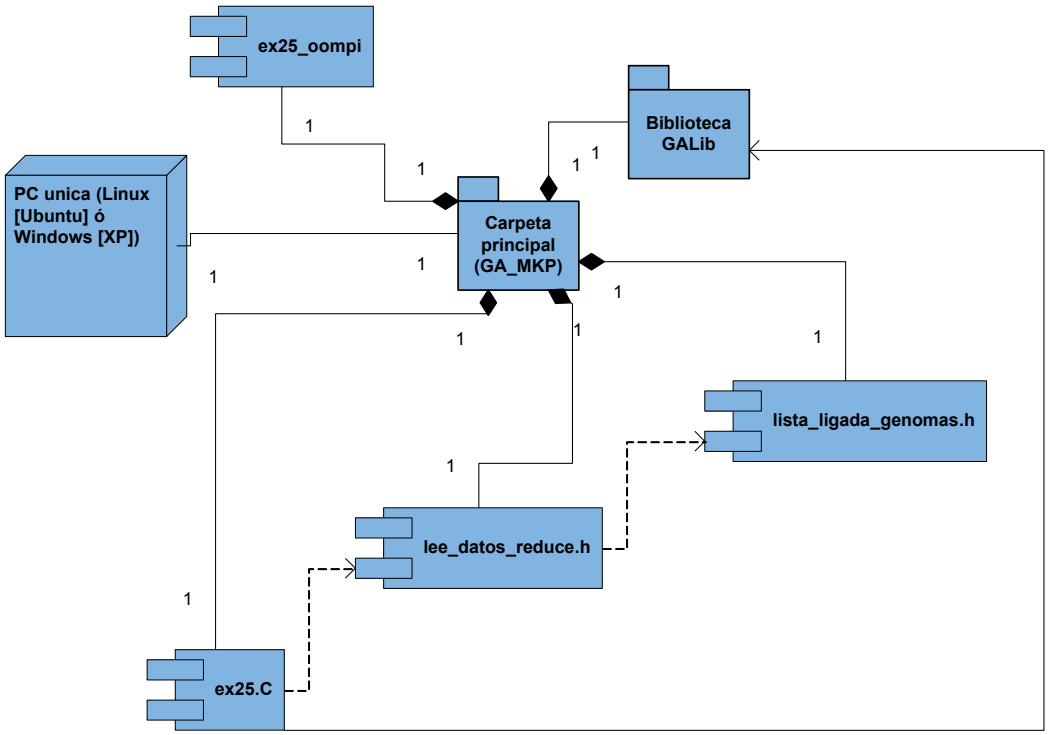
4.4 Diagramas de componentes

Un *diagrama de componentes* muestra las organizaciones y dependencias lógicas entre componentes *software*, sean éstos componentes de código fuente, binarios o ejecutables. Desde el punto de vista del diagrama de componentes se tienen en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del *software*, la reutilización, y las restricciones impuestas por los lenguajes de programación y las herramientas utilizadas en el desarrollo.

Nombre: **Diagrama de componentes de la implementación del algoritmo genético secuencial que resuelve el problema MKP**

Descripción:

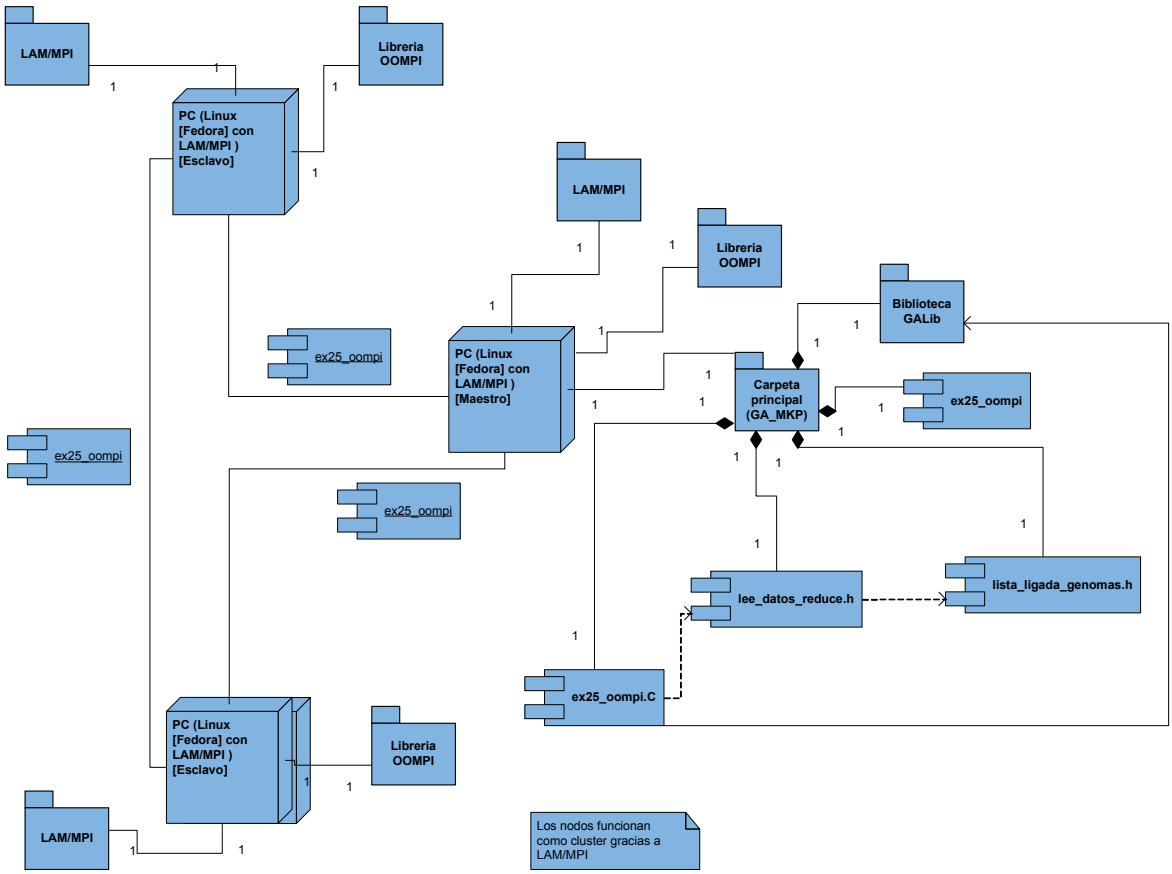
Este diagrama representa como está almacenado el programa en el nodo y sus dependencias con otras librerías.



Nombre: **Diagrama de componentes de la implementación del algoritmo genético paralelo que resuelve el problema MKP.**

Descripción:

Este diagrama representa como está almacenado el programa en cada nodo, sus dependencias con otras librerías y la comunicación que hay entre los nodos que intervienen en el proceso de solución del problema.



Capítulo V: Implementación y Uso

En éste capítulo, se describen los elementos necesarios para ejecutar el Algoritmo Genético, ya sea en modo Secuencial o en Paralelo, para resolver el Problema de la Mochila Multidimensional. Primero se crea un archivo que contiene los datos del MKP a tratar, posteriormente se ejecuta el programa que contiene un Algoritmo Genético y finalmente se interpretan los resultados.

5.1 Archivo de Lectura

Para resolver un problema de tipo MKP con el Algoritmo Genético Secuencial o Paralelo, los datos necesarios para la ejecución del mismo se deben almacenan en un archivo de texto plano. El archivo posee la siguiente estructura:

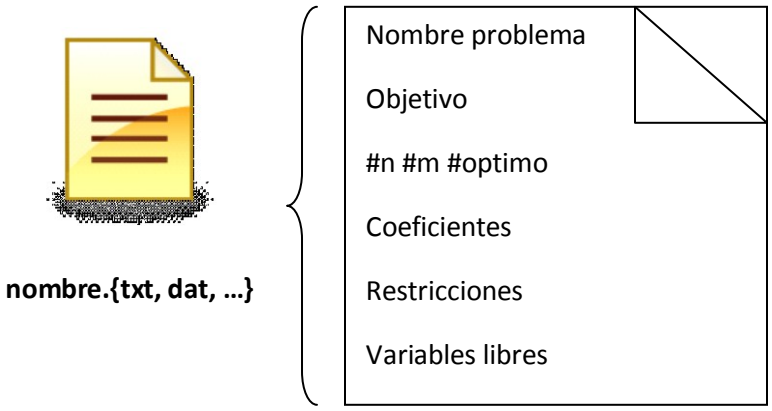


Figura 2: Contenido del archivo a leer

donde:

- *Nombre problema*: le da nombre al problema a resolver.

- *Objetivo*: indica qué tipo de objetivo se desea alcanzar en la solución del problema MKP. El objetivo puede ser de dos tipos:

    - *maximizar:* indica que se debe de maximizar la función objetivo.
    - *minimixar:* indica que se debe de minimizar la función objetivo.

- *#n:* tamaño del problema.

- *#m:* número de restricciones del problema.

- *#optimo:* un valor diferente de cero (0) si se conoce un valor óptimo al problema, en caso de no ser así, el valor será cero (0).

- *Coeficientes:* función objetivo: $\sum_{j=1}^{n} p_j$

- *Restricciones:* matriz de restricciones: $\sum_{j=1}^{n} w_{ij};\ \ i = 1,...,m$

- *Términos Independientes*: $c_i,\ \ i = 1,...,m$

Un ejemplo de archivo se muestra en la Figura 3:

Ejemplo.txt

```
ejemplo
maximizar
4 1 0
1750000 2000000 1500000 1900000
500000 600000 400000 550000
1000000
```

Figura 3: Ejemplo de un archivo

Donde, interpretando el archivo, se tiene:

Maximizar: $1750000\ x_1 + 2000000\ x_2 + 1500000\ x_3 + 1900000\ x_4$

Sujeta a: $500000\ x_1 + 600000\ x_2 + 400000\ x_3 + 550000\ x_4 \leq 1000000$

Tal que $x_1, x_2, x_3, x_4 \in \{0,1\}$

5.2 Ejecución del programa con el Algoritmo Genético

Antes de pasar a resolver algún problema de tipo MKP con los programas creados, uno deberá asegurarse de los siguientes puntos:

- Para el programa que contiene al Algoritmo Genético Secuencial:
  - Plataforma Linux:

    Tener instalado el paquete *galib246.rar* y el compilador *GNU C/C++* (en caso de que se desee modificar partes del código). Para instalar deberá seguir las instrucciones de la distribución de Linux pertinente para instalar un archivo **make**

  - Plataforma Windows 2000/XP/Vista:

    Tener instalado el paquete *galib-2.4.6-gcc3.4.4-2siomek*. Para instalar deberá contar con el compilador *GNU C/C++*, el software *Dev-C++* (en caso de que se desee modificar partes del código) y tener actualizado el *path* de Windows con el compilador GNU para poder usar gcc/g++ en MS-DOS.

- Para el programa que contiene al Algoritmo Genético Paralelo:
  - Plataforma Linux:

    Tener instalado la plataforma *LAM/MPI*, la biblioteca *OOMPI,* el compilador GNU C/C++ (en caso de que se desee modificar partes del código) y el paquete *Mibrary.rar*. Para instalar deberá seguir las instrucciones de la distribución de Linux pertinente para instalar un archivo **make**.

  - Plataforma Windows 2000/XP/Vista:

    No aplica el software en plataformas Windows

Ya que las plataformas tengan todo lo anteriormente descrito en orden, se podrá dar solución a un problema MKP, para ello, se debe contar con el archivo de texto descrito en la sección anterior, respetando la estructura. En caso de no contar con el archivo, deberá de crearse uno.

Si la estructura del archivo es la correcta, se pasa a ejecutar el programa

- Para el programa que contiene al Algoritmo Genético Secuencial:
  - Plataforma Linux:
    1. Entrar a línea de comandos de Linux (BASH).
    2. Ir a la carpeta *galib246/*, su ubicación depende de cómo y en dónde fue instalada.
    3. Ir a la carpeta *examples/*.
    4. Ejecutar el programa de la siguiente forma:
       ./ags-mkp <nombre_archivo>.<ext>
       (`./ags-mkp ejemplo.txt`)
  - Plataforma Windows 2000/XP/Vista:
    1. Entrar a línea de comandos (MS-DOS).
    2. Ir a la carpeta *galib246/*, su ubicación depende de cómo y en dónde fue instalada, por lo general se podrá encontrar dentro del volumen C:
    3. Ejecutar el programa de la siguiente forma:
       C:>ags-mkp.exe <nombre_archivo>.<ext>
       (`C:>ags-mkp.exe ejemplo.txt`)
- Para el programa que contiene al Algoritmo Genético Paralelo:
  - Plataforma Linux:
    1. Entrar a línea de comandos de Linux (BASH).

2. Ir a la carpeta *Mibrary/*, su ubicación depende de cómo y en dónde fue instalada.
3. Ir a la carpeta *ejemplos/*.
4. Iniciar la plataforma LAM/MPI a través del comando *lamboot* con los parámetros necesarios para crear el entorno paralelo.
5. Ejecutar el programa de la siguiente forma:
   ./mpirun –np # agp-mkp < nombre_archivo>.<ext>

   Donde –np # indica cuantos nodos son los que usará la aplicación en paralelo.

   (./mpirun –np 5 agp-mkp ejemplo.txt)

- Plataforma Windows 2000/XP/Vista:

  No aplica el software en plataformas Windows

5.3 Interpretación de Resultados

Al terminar la ejecución del programa, en cualquiera de sus modalidades, la pantalla de la consola muestra la siguiente información, ésta contiene: el vector solución del genoma resultante (con dominio de ceros y/o unos), el valor resultante de la maximización/minimización del problema MKP, y el tiempo que le tomó al programa encontrar la solución mostrada. Esto se puede observar en la Figura 4

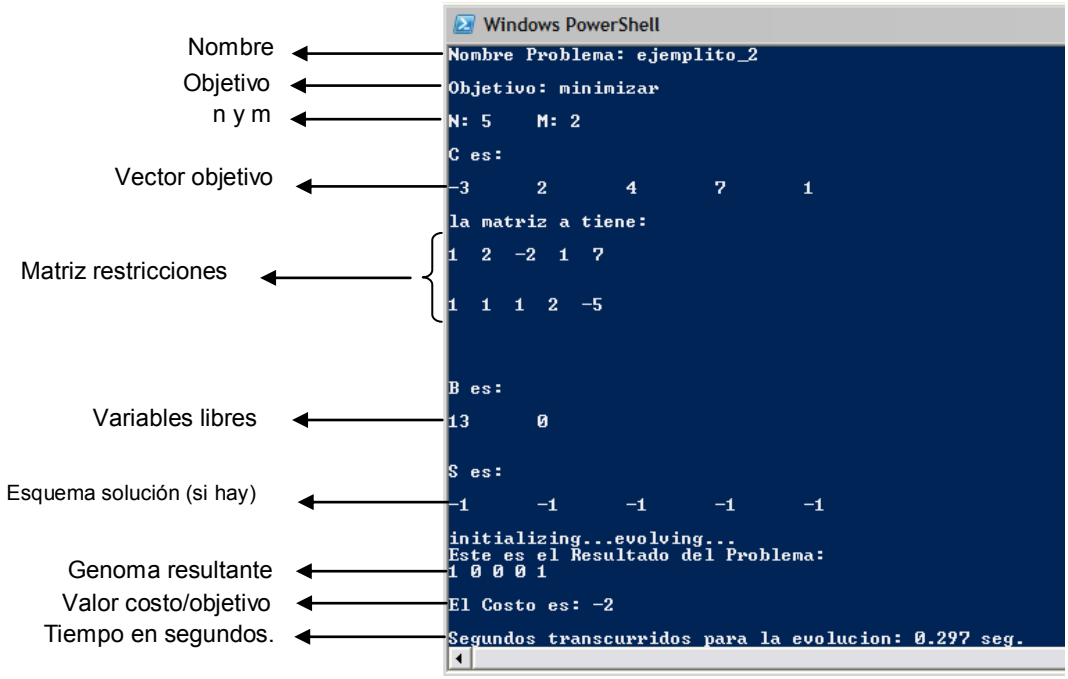


Figura 4: Ventana de Shell, mostrando la ejecución del programa

Capítulo VI: Pruebas y Análisis de Resultados

En este capítulo se analizan los resultados arrojados por los sistemas creados en este proyecto: Algoritmo Genético Secuencial (AGS) y Algoritmo Genético Paralelo (AGP) que resuelven el problema MKP.

Antes de iniciar el análisis de los gráficos, se enuncian algunos aspectos básicos con respecto al Programa que resuelve el MKP.

Los problemas que sirvieron para validar el software desarrollado son los test problems que se encuentran en [27], estos son:

Nombre del Problema	Número de Variables	Número de Restricciones	Mejor Solución Reportada
mkp_01	6	10	3800
mkp_03	15	10	4015
mkp_04	20	10	6120
mkp_05	28	10	12400
mkp_06	39	5	10618
mkp_07	50	5	16537
mk_gk01b	100	15	3674
mk_gk02b	100	25	3869
mk_gk03b	150	25	5506
mk_gk04b	150	50	5610
mk_gk05b	200	25	7344
mk_gk06b	200	50	7481
mk_gk07b	500	25	18578
mk_gk08b	500	50	18285
mk_gk09b	1500	25	58085
mk_gk10b	1500	50	57292
mk_gk11b	2500	100	95231

Como podrá observarse, en [27], solo dan a conocer la mejor solución para cada problema, pero no se menciona cuales son las variables que se toman o se dejan en cada resultado. En este capítulo, no solo se analizó y probó si el programa que ejecuta el Algoritmo Genético llega a la mejor solución, también se da a conocer que variables son las que se toman o dejan para cada problema, en este caso, se le llamará el genoma solución.

Equipo de cómputo

Para la ejecución de los programas que contienen el AGS y el AGP, se tiene que:

- ✓ Descripción de las características de la PC con las que se realizaron los experimentos iniciales:
  - ➢ Procesador Intel Core Duo a 1.60 Ghz, 512 Mb de RAM, Windows XP con ServicePack.
  - ➢ Librería GALib Versión 2.4.6-gcc3.4.4, preparada para ser usada en Dev-C++ con MinGW.

- ✓ Descripción del Clúster: en este sistema se corrió una instancia del programa AGP que resuelve el problema MKP
  - ➢ 6 Procesadores AMD Opteron duales, 2 GB de RAM, Fedora Core 5.
  - ➢ Librería GALIb, LAM/MPI, las bibliotecas OOMPI y compiladores para C++.

Al momento de realizar las pruebas, estas fueron hechas de la siguiente forma, cada archivo que contenía un problema MKP, uno por uno, fue resuelto 10 veces con cada Programa, el que ejecuta el AGS y el AGP, y se midió el tiempo de ejecución consumido en cada caso. Al finalizar se reporta el tiempo promedio y los diferentes genomas obtenidos, así como el valor de la función objetivo para cada uno de ellos.

Tabla de Resultados:

A continuación se muestran algunas tablas con los resultados obtenidos en la ejecución de los Algoritmos Genéticos.

Algoritmo Genético Secuencial

de poblaciones: 100
de individuos por población: 100
de generaciones: 50

Nombre del Problema	N	M	Mejor solución obtenida por el algoritmo desarrollado	Tiempo promedio (seg.)
mkp_01	6	10	3800	0.962
mkp_03	15	10	4015	1.593
mkp_04	20	10	6120	2.014
mkp_05	28	10	12400	2.544
mkp_06	39	5	10618	4.937
mkp_07	50	5	-	-
mk_gk01b(*)	-	-	-	-
mk_gk02b(*)	-	-	-	-
mk_gk03b(*)	-	-	-	-
mk_gk04b(*)	-	-	-	-
mk_gk05b(*)	-	-	-	-
mk_gk06b(*)	-	-	-	-
mk_gk07b(*)	-	-	-	-
mk_gk08b(*)	-	-	-	-
mk_gk09b(*)	-	-	-	-
mk_gk10b(*)	-	-	-	-
mk_gk11b(*)	-	-	-	-

(*) Estos problemas no pudieron ser resueltos en la PC, por falta de capacidad de memoria, provocando que se aborte la ejecución del programa.

Resultado del Genoma en el AGS

Nombre del Problema	Resultado Genoma
mkp_01	(a) 1 0 1 0 1 1
mkp_03	(a) 1 1 0 1 0 1 1 0 1 1 0 0 0 1 1
mkp_04	(a) 1 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1
mkp_05	(a) 1 1 1 0 0 0 0 0 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1
mkp_06	(a) 1 1 0 1 0 1 0 1 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1
mkp_07	-
mk_gk01b	-
mk_gk02b	-
mk_gk03b	-
mk_gk04b	-
mk_gk05b	-
mk_gk06b	-
mk_gk07b	-
mk_gk08b	-
mk_gk09b	-
mk_gk10b	-
mk_gk11b	-

Algoritmo Genético Paralelo

de individuos por población: Si n ≤ 100 → 250 elementos, o si n > 100 → n*4 elementos
de generaciones: Si n ≤ 100 → 180 generaciones, o si n > 100 → (n*3)/2) generaciones
de nodos en el cluster: 2, 4 y 6

Nombre del Problema	N	M	Mejor solución obtenida por el algoritmo desarrollado	Tiempo promedio (seg.)
mkp_01	6	10	3800	4.211
mkp_03	15	10	4015	36.02
mkp_04	20	10	6120	52.31
mkp_05	28	10	12400	49.85
mkp_06	39	5	10618	106.80
mkp_07	50	5	16483	109.97
mk_gk01b(*)	100	15	3674	320.48
mk_gk02b(*)	100	25	3869	425.41
mk_gk03b(*)	150	25	5506	510.74
mk_gk04b(*)	150	50	5610	586.37
mk_gk05b(*)	200	25	7344	622.40
mk_gk06b(*)	200	50	7481	685.51
mk_gk07b(*)	500	25	18578	893.32
mk_gk08b(*)	500	50	18285	902.36
mk_gk09b(*)	1500	25	56036	1858.23
mk_gk10b(*)	1500	50	55741	2978.36
mk_gk11b(*)	2500	100	93199	7896.24

(*)Los problemas *mk_gk**nb*** (para ***n*** = 01, 02,…, 11) fueron procesados en el cluster. A continuación se muestran los resultados obtenidos.

Resultado del Genoma en el AGP

Nombre del Problema	Resultado Genoma
mkp_01	(a) 0 1 1 0 0 1 (b) 1 0 1 0 1 1
mkp_03	(a) 1 1 0 1 0 1 1 0 1 1 0 0 0 1 1
mkp_04	(a) 1 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1
mkp_05	(a) 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 (b) 1 1 1 0 0 0 0 0 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1
mkp_06	(a) 1 1 0 1 0 1 0 1 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1
mkp_07	(a) 0 0 0 1 0 1 0 1 1 1 1 0 1 0 1 1 1 0 1 0 0 0 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1
mk_gk01b	(a) 0 1 0 0 0 1 1 1 0 1 1 0 1 0 1 0 0 1 0 1 1 1 1 0 0 0 1 1 0 0 0 0 1 0 1 1 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 1 1 1 1 0 1 1 1 0 0 1 1 1 1 1 0 1 0 1 0 1 0 0 1 0 0 0 0 1 1 0 1 1 0 1 1 1 0 1 1 1 0 1 0 0 0 0
mk_gk02b	(a) 1 1 1 0 1 1 1 1 0 0 1 1 1 0 1 0 1 0 1 0 0 0 0 1 1 0 0 1 1 1 0 0 0 1 0 1 0 1 0 1 1 1 0 0 0 0 0 1 1 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 0 1 0 1 0 0 0 1 1 1 1 0 0 1 0 0 1 1 0 0 1 0 1 0 1 0 0 0 0 1 1 1 0 0 0 0 1 1
mk_gk03b	(a) 1 1 0 1 1 0 0 0 1 1 0 1 1 1 0 0 1 1 1 1 0 1 0 0 0 0 0 0 0 0 1 0 1 1 0 0 1 1 1 0 1 1 0 0 1 0 0 0 1 1 0 0 0 1 0 1 1 1 0 1 0 0 0 1 0 0 0 1 1 1 0 0 0 1 0 0 0 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 0 1 0 0 0 0 1 1 1 0 1 0 1 0 0 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 1 0 0 0
mk_gk04b	(a) 0 1 0 0 1 0 1 1 0 1 1 0 0 0 0 0 1 1 1 0 1 0 0 1 1 0 0 1 1 0 1 0 1 0 0 0 0 1 1 0 1 1 1 1 0 1 0 1 1

	1 1 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 0 0 0 1 1 1 0 1 0 1 0 1 1 0 1 1 0 0 0 1 0 1 1 1 1 0 1 0 0 0 0 0 0 1 1 1 1 0 1 1 0 0 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 1 0 1 0 0 0
mk_gk05b	(a) 1 1 1 0 1 1 1 1 0 1 0 1 0 0 0 1 1 1 1 0 1 0 0 0 1 0 1 1 0 0 1 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1 1 0 0 1 0 1 1 1 0 1 1 1 1 1 0 0 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 1 1 0 1 0 1 0 1 1 1 1 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 1 1 1 1 0 0 1 0 0 1 1 1 1 0 0 1 0 0 1 1 1 1 1 0 0 1 0 1 1 0 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 1 1 0 1 0
mk_gk06b	(a) 1 1 1 0 0 1 1 0 0 1 1 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 1 1 1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 1 0 1 0 1 0 0 1 1 1 0 1 1 0 1 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 0 1 1 0 1 1 1 0 0 0 0 1 1 0 1 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 1 1 0 1 0 0 0 0 0 1 0 1 1 0 1 1 0 0 1 0 1 0 1 1 1 0 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1 0 1 1 0 0 1
mk_gk07b	(a) 1 1 1 1 0 1 0 1 1 0 0 0 1 0 0 1 1 1 0 1 1 1 1 0 0 1 1 0 0 0 0 1 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 0 1 0 1 0 0 1 0 0 0 1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 0 0 1 0 1 1 1 1 0 1 0 0 0 0 1 1 1 0 0 0 1 0 0 1 0 1 0 1 0 1 0 0 1 0 1 1 1 0 1 1 0 0 0 1 0 1 1 1 0 0 0 0 0 1 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 0 1 0 0 1 1 1 0 0 0 0 1 1 0 0 0 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0 0 0 0 1 1 0 1 1 0 1 0 1 0 0 1 1 0 1 1 1 0 1 0 0 0 0 1 0 1 0 1 0 0 0 1 1 1 0 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0 1 1 1 0 1 1 0 0 1 0 1 1 0 0 0 0 0 1 0 1 1 1 1 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 1 0 0 0 1 1 1 1 0 0 1 1 0 0 1 0 1 1 1 0 1 0 0 1 1 0 1 0 0 1 0 1 1 0 0 0 1 0 0 1 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 1 1 1 0 0 0 1 0 1 1 0 1 1 1 0 0 0 0 1 1 1 1 0 1 1 1 0 1 1 1 0 0 1 1 0 1 0 1 0 0 0 1 1 1 1 0 1 1 1 0 0 1 1 0 1 0 0 0 0 1 0 0 1 1 0 1 0 0 0 1 0 1 1 1 0 1 1 1 0 1
mk_gk08b	(a) 1 1 0 0 0 1 0 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1 0 1 0 1 1 0 0 1 0 1 0 0 0 1 0 1 1 1 1 0 1 0 0 1 1 0 0 1 1 0 0 1 0 1 0 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1 0 1 1 0 0 0 0 0 0 1 1 0 1 0 0 0 1 0 1 1 0 1 0 1 1 0 0 0 0 0 1 0 1 1 0 1 0 1 1 1 0 0 0 0 0 1 1 1 0 1 0 0 0 1 0 1 1 0 1 0 1 1 0 1 1 1 0 0 0 0 1 1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 0 0 1 1 1 0 0 1 0 0 0 0 0 1 1 1 1 0 0 1 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 1 1 1 0 1 0 0 1 0 0 0 1 1 1 1 0 0 1 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 1 0 0 1 1 1 0 0 1 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 1 0 0 1 0 1 0 1 1 1 0 1 1 0 1 0 0 1 0 0 0 0 1 1 0 1 0 1 1 0 0 1 1 1 0 1 1 0 1 1 1 0 0 1 0 1 0 1 1 1 0 1 1 0 1 0 0 0 0 0 0 0 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 0 0 1 0 0 0 0 0 1 0 1 1 1 0 0 0 0 1 0 1 1 1 0 1 0 0 0 0 0 0 0 0 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 0 1 0 1 1 1 0 1 0 1 0 0 1 1 0 0 1 0
mk_gk09b	(a) 0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1 0 1 1 1 1 1 0 1 0 0 1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 1 0 1 1 0 0 1 0 0 1 0 1 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 0 0 1 1 1 1 1 1 0 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 0 1 0 1 0 1 0 0 0 0 1 1 1 0 1 0 0 0 0 0 0 0 1 0 1 1 1 0 0 0 1 0 1 1 0 0 0 0 1 1 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 0 1 0 1 0 0 1 1 1 0 0 1 1 1 0 0 0 0 1 1 1 0 1 0 1 0 0 1 0 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 0 1 1 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 0 1 0 1 1 1 0 0 1 0 1 0 0 0 0 1 1 0 1 0 1 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 1 1 1 1 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 0 1 1 1 0 0 1 0 0 0 1 0 0 1 0 1 1 1 0 1 1 1 1 0 1 0 0 1 0 1 1 1 0 1 1 1 1 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 0 0 1 1 0 0 1 1 0 0 0 0 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 0 1 0 1 1 0 1 0 0 0 1 1 1 1 0 1 0 0 1 1 0 0 1 0 0 0 1 1 1 1 0 1 0 0 1 1 1 0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 1 0 1 1 1 1 1 1 1 0 1 1 0 1 0 0 1 0 0 0 1 1 0 1 0 0 1 0 0 0 0 1 1 1 0 1 0 0 1 0 1 1 1 0 1 1 0 0 0 0 1 1 1 1 0 0 1 1 0 0 0 1 0 0 1 0 1 0 1 1 1 0 1 1 1 0 1 1 0 0 1 0 0 0 1 1 0 1 0 1 0 0 0 1 0 1 1 1 0 0 1 1 0 0 1 0 0 1 1 0 0 0 0 0 1 1 1 1 0 1 0 0 1 1

	0011110001010111000011101010011010110101011010 0011110100111001000001010010001111101111111100010 1000111001100001011010011100001011001001000000 1110101011001110000000110110101010011001010100 0 1101110010010000011000010111111101000010011011 0 1001101100000010110100000101010110001110110001 1 0010111000010011100000110000011011010011001101 1 0011100110011001101001111111011000100010000011 1 1101010111111011011001011010001100100011000000 1 1011000100101110010010101000011110101001010110 0 1110111010110010001101111101011000011111011110 0 0011011001101100101100111111010101100001000111 1 0001000111010100110100011110110100010001010100 0 0000010010110100011100110110001101001001111
mk_gk10b	(a) 0001101110111100000101011010110011110110000111 0000100100110011001010011111111001110101001010 0 1001010100001101111000110110011101110001100 1000 1000011100101011110110101001101001101010110 1000 0101110000001011001010001001010001000100010110 0 1010101011010100100000111010110111111101111011 0 0001010110011010111011100110110010010001101 0011101110000000110101000101011100101011110110 01 1111100110001100101101110100110001111011010110 1 0100000101100111011111111100101101110001100001 11 1001100100011101111110110001101001000000100111 0 1010111110100011001011101000101100000110011110 1 0011101110000010100110100100011110100011110010 00 1000010010011010110000110010011100001011010 0010 1000101100001111001101000011001011000010101 1000 1000001101100110111110010011101111110010001 1011 1011001100001101000111100010000000100111101 1100 1000000000100001111000001001101100110000001 01001 1010010101111000010001010101100001011100011 0000 0100001100100110011010110100111110011110010 0011 1010001001100111100010001111000111101001001 1101 0001011100011100011000011011011001000100101 1000 0110100100101000110011101011000011101111110 0001 1110111011100110000101010001110000110010001 1010 0101101100000011001110000101101110101000101 1110 1110100110010001011010111011111011011001001 1111 0110010100101011001001111010110101000000111 1101 0101101110110110011111010011001100110101011 1111 1110010010110000101100101110101000010000011 1010 1000111111111110100010100000101000111101110 1100 0001100000111110101000111101100111011000000 1111 1100001010111100110001101010101110100001000
mk_gk11b	(a) 0100111111010100110001110001110110101101101 1101 1011111011100000000111001001010011110001010 1100 0011111101000100110001011101111100001010101 1111 1000110001011011110010111011111010000001011 1101 0011010100100101010010111101010011100010011 0110 0101110001010010000001111111111111011101000 0001 0010110011001000100101111101101001000111000 1001 1110011011100000010110110010101010111001001 0000

0 1 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0 1 1 1 1 0 1 0 0 1 1 0 1 1 0 1 0 0 0 0 1 0 1 0 1 0 1 0 1 0 0
0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 1 0
0 0 0 1 1 0 1 1 1 0 1 1 1 1 0 1 0 1 0 0 1 1 0 1 0 1 1 0 0 1 1 0 1 1 0 1 0 1 1 1 1 0 0 1 0 1 0
0 1 1 1 0 0 1 0 1 0 0 0 1 1 1 0 0 1 0 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 1 0 0 1 0 1 0
1 1 1 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1 0 1 1 0 1 0 0 1 1 1 1 1 0 0 0 0
1 1 0 0 1 0 0 0 1 1 1 1 0 0 1 0 1 1 0 1 1 0 0 1 0 0 1 0 0 0 0 1 1 1 0 0 1 0 1 0 1 1 1 1 1 1 1
1 0 0 0 0 1 1 1 0 1 0 0 0 1 0 0 1 0 0 0 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1 0 1 1 0 1 0 1 1 0 0 0 1
0 0 1 0 0 0 0 1 1 1 0 0 1 1 1 0 1 0 0 0 0 1 1 1 0 0 1 0 1 1 1 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 1 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1 1 0
1 1 1 0 0 1 0 1 1 1 1 1 0 0 1 1 0 0 0 0 0 0 0 1 1 1 1 0 1 1 1 0 0 1 0 0 1 1 1 0 0 0 0 1 0 0 0
0 1 0 0 0 1 0 1 0 0 0 0 1 1 0 0 0 1 0 1 1 1 1 0 1 0 1 0 1 0 1 1 1 0 0 1 0 0 1 1 0 0 0 0 1 0 0
0 0 0 1 0 0 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 0 1 1 0 1 0 0 1 0
1 1 0 1 1 0 0 0 1 0 0 1 1 0 0 1 1 1 0 0 0 1 1 0 0 0 1 1 1 1 0 1 0 1 1 0 0 1 0 1 0 1 0 0 0 1 1
1 1 1 1 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 0 0 0 1 0 1 0 1 1 1 0 0 1 1
0 0 1 1 1 1 0 1 0 0 0 0 1 1 1 1 1 0 0 0 0 1 0 0 1 0 1 1 0 1 1 1 1 0 0 0 0 0 1 0 0 1 1 1 1 1 0
1 1 0 1 1 0 0 0 0 0 0 1 1 0 0 0 1 1 0 0 0 0 1 0 1 1 0 1 0 1 0 0 1 0 0 0 0 1 0 1 1 0 1 0 1 1 1
1 1 1 0 0 1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 0 0 1 1 1 0 1 1 0 1 1 0 0 0 0 0 1 1 1 1 0 0 1 1 0 0 0
0 1 1 1 1 0 0 1 0 1 1 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 1 1 0 0 1 1 0 1 1 1 1 0 0 1 1 0 1 1 1 1
0 1 1 1 0 1 0 1 1 0 1 1 1 0 0 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 0 0 1 1 0 0 0 1 1 0 1 0 0 0 1
0 1 1 1 0 1 0 0 0 0 0 0 0 1 1 0 0 0 1 1 0 0 1 1 0 0 1 1 0 1 0 1 0 0 1 1 1 1 1 1 1 1 0 0 1 0 1
1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 1 0 0 1 0
0 1 1 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 0 1 0 1 1 0 1 1 1 0 1 0 1 0 0 0 0 0
1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 0 0 1 0 0 1 0 0 0 1 1 1 1 1
1 1 1 0 0 1 1 1 0 0 1 0 1 0 0 0 1 1 0 1 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 1 0
1 0 1 1 0 0 0 0 0 0 1 1 1 0 1 0 1 1 0 0 0 1 0 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0 0 1 1 0 0 0 0 1 1
0 1 0 1 0 0 1 1 1 0 0 0 1 0 0 0 1 0 1 1 0 0 1 0 1 1 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 1 1 1 1 1 1
0 1 0 1 0 1 1 1 1 1 0 1 0 1 0 1 1 0 0 1 0 1 1 1 0 1 1 0 0 1 1 0 1 0 0 0 0 0 1 0 0 0 1 1 1 0 0
0 0 1 1 1 0 0 1 0 0 1 1 0 1 0 0 1 1 1 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 1 1 1 0 0 1 1 1 0 1 0 0 1
0 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 1 0 1 0 1 0 0 0 1 0 1 1 1 0 1 0 1 1 0 1 0 0 1 1 0 0 0 1 1 1
1 1 1 0 0 1 0 1 1 0 1 0 1 0 1 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1
0 0 0 0 0 1 1 1 1 0 1 1 0 0 1 0 1 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 1 0 1 0 1 0 0 1 1 0 1
1 0 1 1 0 1 1 0 1 1 0 1 0 1 0 1 1 0 1 1 0 1 1 0 1 0 1 0 0 1 1 1 1 0 1 1 0 1 0 1 0 1 0 0 0 0 0
0 0 0 1 0 1 0 0 1 1 0 0 0 1 0 0 1 0 1 1 1 1 1 1 1 0 1 1 0 0 1 0 0 0 1 0 0 0 0 1 1 0 0 1 0 1 1
0 1 0 1 1 0 1 0 1 1 0 0 0 1 1 0 1 1 1 1 0 1 1 1 0 0 1 0 0 0 1 1 1 0 0 0 1 0 0 0 0 1 1 0 0 1 1
0 1 1 1 0 1 1 0 1 0 1 1 1 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 0 0 1 0 0 1 1 0 0 1 1 0 0 0 1 0 0 0
0 0 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 0 1 0 0 0 1 1 1 1 1 0 1 0 1 1 0 1 0 0 0 1 0 1 1 0 1 1
1 1 1 0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 0 0 0 1 1 0 0 1 0 0 0 0 0 1 1 0 0 1
0 1 1 0 1 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 1 1 0 1 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 0 1 1 1 0 1 1 0
0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 0 1 0 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1
1 0 1 1 0 0 0 1 0 0 0 0 1 1 1 0 1 0 0 0 1 1 0 1 1 0 1 1 1 1 1 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1 0
0 1 0 1 0 1 0 0 1 1 0 0 0 1 0 1 1 0 0 0 1 1 0 1 1 0 1 1 1 0 1 0 0 1 0 0 0 1 0 1 1 1 0 1 1 0 1
1 1 1 0 1 1 0 0 0 1 0 0 1 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 1 0 1 1 0 1 0 0 1 0 1 1 0 0
0 1 1 0 1 0 0 1 1 0 0 1 0 0 1 0 1 0 0 1 0 1 1 1 1 0 1 0 1 0 1 1 1 1 0 0 0 1 0 0 1 1 0 0 1 0 1
0 1 1 1 1 1 0 0 1 1 1 0 0 1 0 1 0 1 0 1 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0
1 0 0 1 1 1 1 1 0 0 0 1 0 1 1 1 1 0 0 0 1 1 1 1 1 1 1 0 0 1 0 1 1 1 0 1 0 1 0 0 1 1 0 0 0 0 1
1 0 1 1 0 1 0 0 1

La siguiente tabla muestra una comparación de los resultados reportados [27] contra los resultados obtenidos por el AGS y por el AGP, asi como el tiempo empleado en encontrar la solución[6].

Problema MKP	Solución Reportada [27]	Solución AGS	Tiempo (segs.)	Solución AGP	Tiempo (segs.)
mkp_01	3800	3800	0.962	3800	4.211
mkp_03	4015	4015	1.593	4015	36.02
mkp_04	6120	6120	2.014	6120	52.31
mkp_05	12400	12400	2.544	12400	49.85
mkp_06	10618	10618	4.937	10618	106.80
mkp_07	16537	-	-	16483	109.97
mk_gk01b	3674	-	-	3674	320.48
mk_gk02b	3869	-	-	3869	425.41
mk_gk03b	5506	-	-	5506	510.74
mk_gk04b	5610	-	-	5610	586.37
mk_gk05b	7344	-	-	7344	622.40
mk_gk06b	7481	-	-	7481	685.51
mk_gk07b	18578	-	-	18578	893.32
mk_gk08b	18285	-	-	18285	902.36
mk_gk09b	58085	-	-	56036	1858.23
mk_gk10b	57292	-	-	55741	2978.36
mk_gk11b	95231	-	-	93199	7896.24

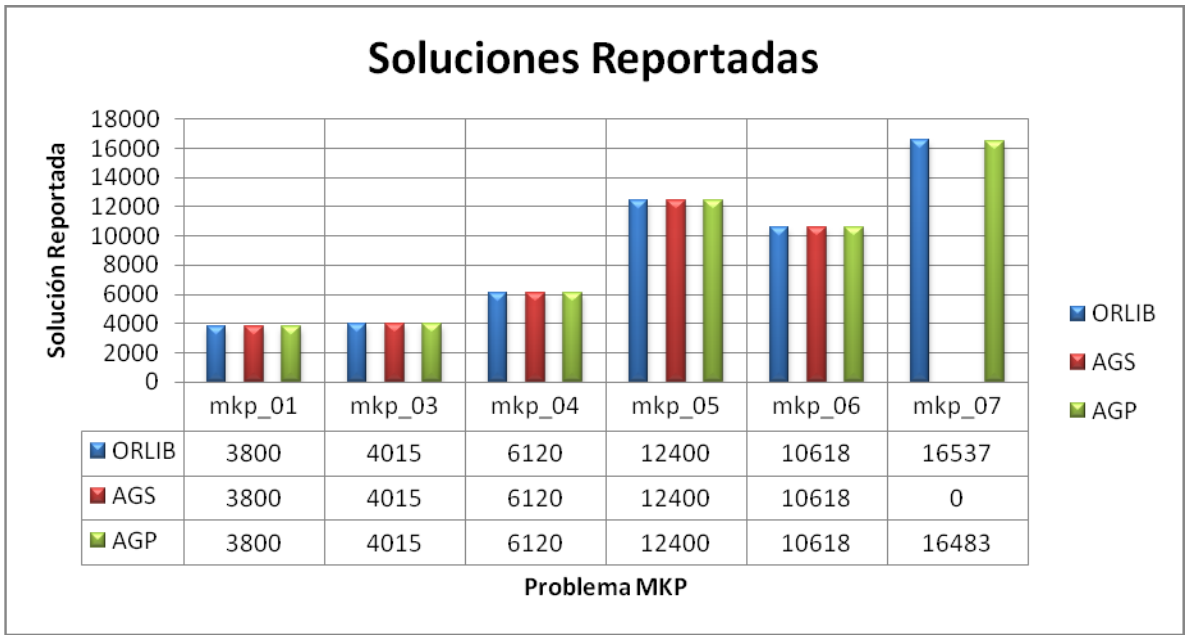


Gráfico 1: Gráfico que compara la solución que se encuentra en ORLIB contra los resultados obtenidos en el AGS y el AGP propuestos

[6] Hay que tomar en cuenta que estos resultados son variables, dependiendo la capacidad de procesamiento del equipo de cómputo utilizado para ejecutar los algoritmos, también, si es que el sistema se encuentra realizando otras tareas al mismo tiempo que se ejecutan los mismos.

En el gráfico 1 se muestran los valores reportados en la librería ORLIB y los obtenidos por los algoritmos desarrollados en el presente trabajo. Se observa, que a partir del problema MKP_07, el AGS ya no encontró solución al problema MKP. El AGP si encontró una solución que se acerca bastante al mejor valor reportado hasta el momento.

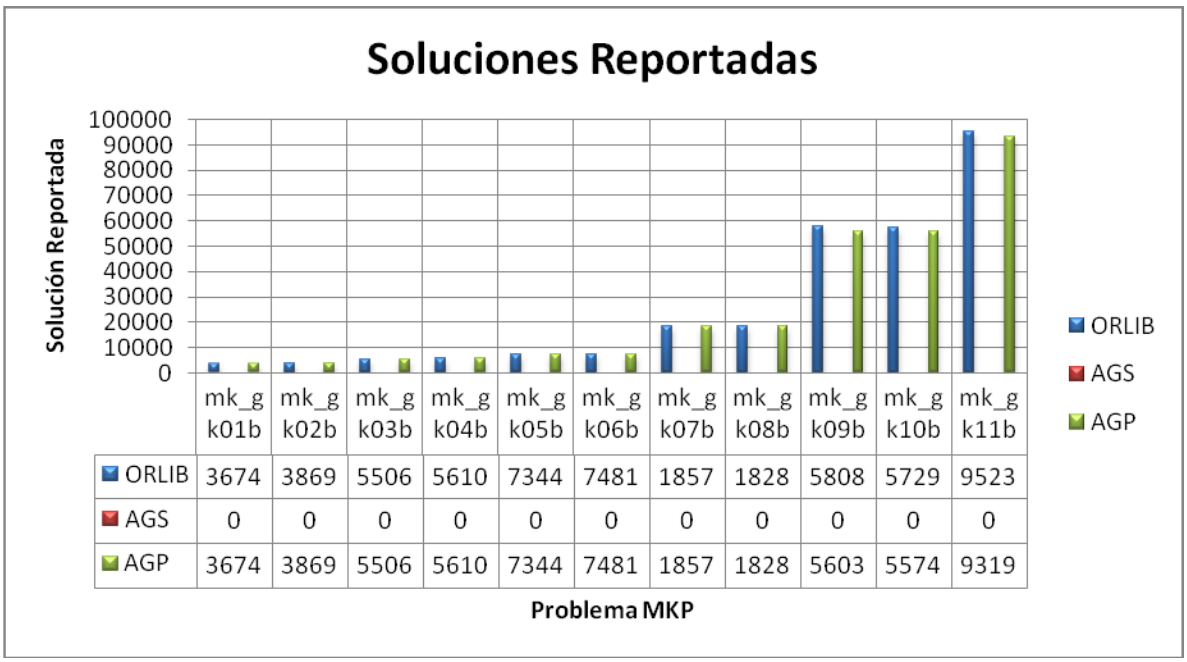


Gráfico 2: Gráfico que compara la solución de los problemas MKP más grandes que se encuentra en ORLIB contra los resultados obtenidos en el AGS y el AGP propuestos

Como se observa en el gráfico 2, para los problemas más grandes de la librería ORLIB el AGS no encontró solución alguna, como puede apreciarse el AGP en los 8 primeros problemas encontró el valor reportado, no así en los últimos 3 problemas, aunque el valor de la función objetivo obtenida se acerca bastante al mejor valor reportado hasta el momento.

En la siguiente tabla, se muestra el tiempo que le tomó al AGP en encontrar la solución al problema MKP planteado, bajo 2, 4 y 6 nodos del cluster.

Problema MKP	Tiempo en Seg. (2 nodos)	Tiempo en Seg. (4 nodos)	Tiempo en Seg. (6 nodos)
mkp_01(*)	**4.21**	**5.55**	**5.55**
mkp_03	41.12	36.02	42.12
mkp_04	51.58	51.58	52.31
mkp_05	57.96	49.85	51.96
mkp_06	106.80	109.48	103.48
mkp_07	104.85	109.97	103.85
mk_gk01b(*)	**302.92**	**327.92**	**320.48**
mk_gk02b	448.05	425.41	403.05
mk_gk03b	510.74	515.49	517.49
mk_gk04b	595.73	586.37	618.73
mk_gk05b	607.51	632.51	622.40
mk_gk06b	610.21	685.51	664.21
mk_gk07b	893.32	890.60	808.60
mk_gk08b	904.14	902.36	902.14
mk_gk09b	1564.56	2438.56	1858.23
mk_gk10b	3487.31	2978.36	2243.31
mk_gk11b(*)	**7896.24**	**7919.86**	**8726.86**

(*) Estos problemas son los que se exponen en el gráfico que se mostrará a continuación.

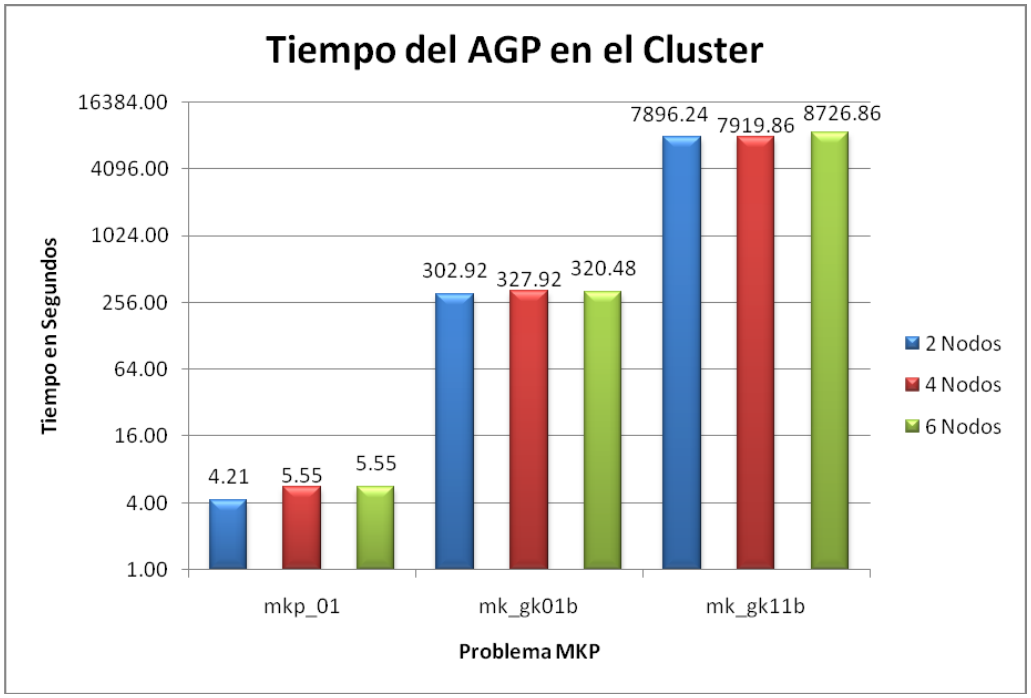


Gráfico 3: Gráfico que compara los tiempos de ejecuciòn de los problemas màs representativos de ORLIB bajo 2, 4 y 6 nodos

Por los resultados obtenidos, puede apreciarse que, cuando el AGP alcanza el mejor valor reportado hasta el momento lo obtiene sobre 4 nodos del cluster. Esto es lógico, porque con 2 solamente se busca el mejor valor a partir de 2 poblaciones iniciales, sin embargo con 4 nodos, al realizar la migración y mezclar los mejores individuos de una población con otra, permite enriquecer el espacio de búsqueda, y por tanto con mayor rapidez se encuentra la solución final. Por otra parte con 6 nodos, el intercambio de mensajes entre ellos aumenta, y con ello, también el tiempo de ejecución.

Capítulo VII: Conclusiones y Recomendaciones

1. En el presente trabajo se cumplieron satisfactoriamente los objetivos planteados, se desarrolló tanto un algoritmo genético secuencial, como un algoritmo genético paralelo, probando así la librería desarrollada en la Facultad.

2. Se desarrolló un esquema de migración de los mejores genomas de una población a otra, esto permite agrandar el espacio de búsqueda y mejorar la calidad de los individuos encontrados, tratando con ello de encontrar el óptimo del problema, en un tiempo razonable.

3. Se desarrolló un algoritmo genético paralelo, permitiendo que sobre cada nodo se trabaje con una población inicial diferente, e incluyendo el esquema de migración desarrollado, para resolver este tipo particular de problemas de optimización.

4. El AGS desarrollado fue probado sobre la PC descrita con anterioridad y se puede concluir que sobre la misma, sólo pueden resolverse problemas de a lo sumo 50 variables y 25 restricciones. El objeto ga desarrollado en la blibliotéca utilizada permite simular la ejecución del algoritmo con más de una población inicial. Por los experimentos desarrollados se concluye que para encontrar una solución óptima en un tiempo razonable es conveniente a lo sumo utilizar 100 poblaciones, y cada población con 100 genomas (individuos) diferentes.

5. Se programó además el algoritmo de simplificación, el cual permite reducir el tamaño del problema original o detectar a priori si es posible o no que exista solución factibles. Se pudo apreciar que sólo se logra la reducción para problemas del tipo MKP donde no existe ninguna relación entre los coeficientes de la función objetivo y el de la matriz de restricciones; sin embargo sólo logró reducir el tamaño del problema y fijar variables en uno de los *test problems*, pues como es conocido este tipo de problema está construidos de tal manera que existe relación entre los coeficientes de las variables y de las restricciones que intervienen en los mismos.

6. Para la ejecución del AGP se utilizó el cluster de la Facultad. Se realizaron experimentos con 2, 4 y 6 nodos respectivamente.

7. El AGP desarrollado permite que cada nodo comience con una población generada de manera aleatoria, si el número de variables del problema (*n*) es mayor que 100, el número de individuos de cada población es: (*n*4*), y el número de generaciones es de *(n*3)/2*.

8. El AGP permitió resolver todos los test problems, aunque en 4 de ellos se acercó al mejor valor reportado, no llegó al óptimo.

9. Por los experimentos realizados puede concluirse que el AGP mostró mejor comportamiento utilizando solamente a 4 nodos del cluster

10. Para los experimentos realizados se utilizó la cruza en un punto, la mutación base de la bliblioteca. Una recomendación importante sería realizar los mismos experimentos utilizando los diversos tipos de cruzamiento que ofrece la librería utilizada.

Referencias

[1]. Coello C., Carlos A. *Introducción a la Computación Evolutiva (Notas de Curso)*. México, D.F. 2000 - 2005.

[2]. Beyer, H. G. *The Theory of Evolution Strategies*. Natural Computing Series. Springer, Berlin 2001.

[3]. Michalewicz, Zbigniew. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1996.

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

[5]. Goldberg, D. *Genetics Algorithms in Search, Optimization and Machine Learning*. Addison Wesley. 1989

[6]. Mitchell, Melanie. *An Introduction to Genetic Algorithms*. The MIT Press. Cambridge, Massachusett. 1996

[7]. Acevedo Gómez, Sylvia. *Biología*. Editorial Norma. Colombia 1998.

[8]. Merelo Guervos, Juan Julian. *Informática evolutiva: Algoritmos genéticos*. http://geneura.ugr.es/~jmerelo/ie/ags.htm

[9]. http://eddyalfaro.galeon.com/geneticos.html. *Algoritmos Genéticos*. La Página de Eddy.

[10]. Hidalgo Pérez, José Ignacio. *Una revisión de los Algoritmos Evolutivos y sus aplicaciones*. www.cesfelipesegundo.com/revista/Articulos2004b/Articulo5.pdf

[11]. Hoff, A., Løkketangen, A., and Mittet, I. *Genetic Algorithms for 0/1 Multidimensional Knapsack Problems*. Norsk Informatikkonferanse. 1996. http://home.himolde.no/~arildh

[12]. Mirón Enríquez, Aquiles José Manuel. *Una biblioteca de clases para la implementación de algoritmos genéticos paralelos con MPI y C++*. Puebla, Pue. Enero 2008

[13]. Holland, J. H. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI. 1975

[14]. Wetzel, A. *Evaluation of Efectiveness of genetic algorithms in combinational optimization*. University of Pittsburgh, Pittsburgh (unpublished). 1983

[15]. Whitley, D. *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*. Pág. 116–121. San Mateo, California, July. Morgan Kaufmann Publishers. 1989

[16]. Flynn, M.. *Some Computer Organizations and Their Effectiveness.* IEEE Trans. Comput., Vol. C-21, pp. 948. 1972

[17]. Holland, John. *Genetic algorithms*. Scientific American. Pág. 66-72. Julio de 1992,

[18]. Begley, Sharon y Gregory Beals. *Software au naturel*. Newsweek, 8 de mayo de 1995, p.70.

[19]. Keane, A.J. y S.M. Brown. *The design of a satellite boom with enhanced vibration performance using genetic algorithm techniques. En Adaptive Computing in Engineering Design and Control.* Proceedings of the Second International Conference, I.C. Parmee, p.107-113. University of Plymouth, 1996.

[20]. Mahfoud, Sam y Ganesh Mani. *Financial forecasting using genetic algorithms*, *Applied Artificial Intelligence.* vol.10, no.6, p.543-565. 1996.

[21]. Coale, Kristi. *Darwin in a box*. Wired News, 14 de julio de 1997.

[22]. Naik, Gautam. *Back to Darwin: In sunlight and cells, science seeks answers to high-tech puzzles*. The Wall Street Journal, 16 de enero de 1996, p. A1

[23]. Kewley, Robert y Mark Embrechts. Computational military tactical planning system. IEEE Transactions on Systems, Man and Cybernetics, Part C. Applications and Reviews, vol.32, no.2, p.161-171 (mayo de 2002)

[24]. Lemley, Brad. *Machines that think*. Discover, enero de 2001, p.75-79.

[25]. Gutiérrez Bavastrom, Dianelys. *Instrumentación Computacional del algoritmo de Balas con Analizador Sintáctico*. 1994

[26]. Chu, P., J.E.Beasly. *A Genetic Algorithm for the Multidimensional Knapsack Problem*. Journal of Heuristics. Pág. 463-486. 1988

[27]. *Librería de Test problems para el MKP*. http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html

[28]. M.J. Blesa, Ll. Hernàndez, F. Xhafa. *Tabu search for 0-1 multidimensional knapsack revisited: choosing internal heuristicsand fine tuning of parameters*. (TextBook). http://tracer.lcc.uma.es/problems/multi01knap/knap.htm

[29]. Pinacho Rodríguez, Erick. *Una plataforma para el desarrollo de aplicaciones en paralelo usando JAVA*. Puebla, Pue. 2007

[30]. http://www.osl.iu.edu/download/research/oompi/oompi.pdf

[31]. *Leal Bando, L. Diseño e Implementación de un Algoritmo Paralelo Maestro-Esclavo para resolver el problema de la Mochila Multidimensional*. Octubre 2006. Puebla, Pue. México.

[32]. Bagley, J.D. *The behavior of adaptive systems which employ genetic and correlation algorithms*. Dissertation Abstracts International. 1967. Vol. 28 No.12.

[33]. Balas, E. 1965. *An Additive Algorithm for Solving Linear Programs with Zero-One Variables*. Operations Research 13, 517-546.

[34]. Balas, E. 1974. *Disjunctive Programming: Properties of the Convex Hull of Feasible Points*. MSRR, No. 348, Carnegie Mellon University (Pittsburg, P.A).

[35]. Balas, E. and C.H. Martin. *Pivot and Complement-A heuristic for 0-1 Programmming*. Management Science 26. Pág. 86-96. 1980

[36]. Hillier, F.S. *Efficient Heuristics Proccedures for Integer Linear Programming with an Interior*. Operations Research 17. Pág. 600-637. 1969.

[37]. Kochenberger, G.A., B.A. McCarl, and F.P. Wymann. *A Heuristic for General Integer Programming*. Decisions Sciences 5, Pág 36-44. 1974.

[38]. Senju, S. and Y.Toyoda. *An Approach to Linear Programming with 0-1 Variables*. Management Science 15, 196-207. 1968.

[39]. Zanakis, S.H. *Heuristic 0-1 Linear Programming: An Experimental Comparison of Three Methods*. Management Science 24, 91-104. 1977.

[40]. Fontanari, J.F. *A Statistical Analysis of the Knapsack Problem*. Journal of Physics A Mathematical and general. 1995.

[41]. Pirkul, H. *A Heuristic Solution Procedure for the Multiconstraint Zero-One Knapsack Problem*. Naval Research Logistics 34, Pág. 161-172. 1987.

[42]. Volgenant, A. and J.A. Zoon. *An Improve Heuristic for Multidimensional 0-1 Knapsack Problems*. Journal of the Operational Research Society 41, Pág. 963-970. 1990.

[43]. Dammeyer, F. and S.Voss. *Dynamic Tabu List Management Using Reverse Elimination Method*. Annals of Operations Research 41, Pág. 31-46. 1993.

[44]. Aboudi, R. and K.Jörnsten. *Tabu Search for general Zero-One Ingeger Programs Using the Pivot and Complement Heuristic*. ORSA Journal on Computing 6, Pág. 82-93. 1994.

[45]. Battiti, R. and G.Tecchiolli. *Local Search with Memory: Benchmarking RTS*. OR Spectrum 17, Pág. 67-86. 1995.

[46]. Rudolph, G. and J.Sprave. *Significance of Locality and Selection Pressure in the Grand Deluge Evolutionary Algorithm*. In H.M. Voigt, W. Ebeling, I. Rechenberg, and H.P. Schwefel (eds.), Parallel Problem Solving from Nature IV. Proceedings of the International Conference on Evolutionary Computation. Lecture Notes in Computer Science, Springer, Pág. 686-694. 1996

[47]. Schaffer, J.D.; Caruana, R.A.; Eshelman, L.J. y Das, R. *A study of control parameters affecting online performance of genetic algorithms for function optimization*. Proc. of 3th Intl. Conf. on Genetic Algorithms.J.D. Shaffer(Ed.) Morgan Kaufmann, Los Altos, CA. Pág. 51-60. 1989

[48]. Khuri, S.,T.Back, and J. Heitkotter. *The zero / one Multiple Knapsack Problem and Genetic Algorithms*. Proceedings of the 1994 ACM Symposium on Applied Computing (SAC' 94), ACM Press, Págs. 188-193. 1994

[49]. Glover, F. *Optimization by Ghost Image Processes in Neural Networks*. Computer and Operations Research 21, Págs. 801-822. 1994

[50]. Hanafi, S., A. Freville and A.El. Abedellaoui. *Comparison of Heuristics for the 0-1 Multidimensional Knapsack Problem*. In I.H. Osman and J.P. Kelly (eds.), Meta-Heuristics: Theory and Applications. Kluwer Academic Publishers, Págs.449-465. 1996

[51]. Stefka Fidanova. *ACO Algorithm for MKP Using Various Heuristic Information*. Numerical Methods and Applications. Springer Berlin / Heidelberg 438-444. 2003.

[52]. Puchinger, Jakob. *Cooperating Memetic and Branch-and-Cut Algorithms for Solving the Multidimensional Knapsack Problem*. MIC2005. Vienna, Austria. http://www.ads.tuwien.ac.at/publications/bib/pdf/puchinger-05a.pdf. 2005

[53]. Miyagi, Hayato. *A Parallel Genetic Algorithm and Its Variance Analisis for a New Multiple Knapsack Problem*. ITC-CSCC 2008. Japan. http://www.ieice.org/explorer/ITC-CSCC2008/pdf/p157_C2-5.pdf. 2008.

[54]. Crowder, H., E. Johnson, M. Padberg. *Solving large scale zero-one linear Programing Problems*. Operations Research, Vol 31, No.5, Págs. 803-834. 1983.